SOFTWARE VERIFICATION RESEARCH CENTRE

SCHOOL OF INFORMATION TECHNOLOGY

THE UNIVERSITY OF QUEENSLAND

Queensland 4072
Australia

TECHNICAL REPORT

No. 00-17

Safety Assurance of Commercial-Off-The-Shelf Software

Peter Lindsay and Graeme Smith

May 2000

Phone: +61 7 3365 1003
Fax: +61 7 3365 1533

Safety Assurance of Commercial-Off-The-Shelf Software

Dr Peter Lindsay & Dr Graeme Smith
Software Verification Research Centre
University of Queensland, Brisbane, Qld 4072, Australia
email: pal@svrc.uq.edu.au, smith@svrc.uq.edu.au

Abstract: Commercial-Off-The-Shelf (COTS) software is increasingly being suggested for use in systems development, for reasons including cost, functionality, useability, testedness, availability of support and upgrades. At the same time, standards for use of software in safety-related systems are becoming increasingly stringent. This paper examines the issue of providing safety assurance for systems involving COTS software, and surveys what international standards say about the issue. The strengths and weaknesses of a number of different approaches are discussed. The paper suggests that "whole life-cycle" costing estimates, including estimates of the cost of developing and maintaining safety assurance, be made prior to deciding whether to use COTS components in safety-related applications.

1 Introduction

1.1 Why COTS?

System developers are under increasing pressure to use Commercial-Off-The-Shelf (COTS) software components: that is, software that was originally designed by someone else, for other (possibly more general) purposes, and which is not intended to be internally modified by the system developer. COTS software includes for example application packages, operating systems, user-interface builders and graphics packages, software embedded in COTS hardware components, and so on.

Arguments typically advanced for the use of COTS products are:

- **Cost**: COTS products are generally cheaper because of economies of scale, whereby development and maintenance costs are amortised over a large user base;
- **Functionality**: COTS products may provide a high percentage of the desired functionality, plus a lot more besides;
- **Useability**: operators and end users often feel more comfortable with user interfaces based on or resembling ones they use at home or in the office;
- **Testedness**: because of the wide user base, the product has generally been thrashed pretty hard, and where problems are found they often get reported and may get fixed in future releases;
- **Support**: again because of the wide user base, training and in-service support are often

available, perhaps even from multiple sources if the user base is sufficiently large;

- **Future proof**: as technology moves on, the product may also evolve to stay up to date and to move to new platforms.

As attractive as these arguments seem at first sight, however, there are many traps to be wary of, and some especially difficult – and expensive – problems to be overcome before using COTS products in safety-related systems. By **safety-related system** we mean one whose failure could have safety implications. This includes systems that control or include physical devices, including those displaying information or advice to operators, which may cause injury or death [Std5679].

This paper surveys the issues and the options available, and makes recommendations for organisations considering using COTS software in safety-related systems. Although the paper focuses on safety issues, many of the lessons extend to other properties of critical systems where assurance of system dependability is a concern. It also applies to COTS-like components, such as Non-Development Items (NDIs), Government Off-The-Shelf (GOTS) products (also known as Government Furnished Equipment), and legacy systems. The term Software of Unknown Pedigree (SOUP) has recently been coined to cover this more general class of software. Many of the issues apply to hardware as well as software.

In what follows, we follow [McDermid98] in using **COTS** to refer to software components which are readily available from commercial sources, for general applications and not easily modified. (Access to source code is generally denied.) Typical characteristics of COTS components are that a number of different configurations may be available, and upgrades may occur either during system development or while it is in service, or both.

1.2 Overview of paper

Section 2 overviews the requirements of current and emerging safety standards for systems involving COTS software. Section 3 examines the purported benefits of COTS software when viewed from a safety perspective and outlines the fundamental requirements of safety assurance for COTS components. Section 4 describes approaches for providing safety assurance for systems with COTS components and outlines the approaches' strengths and weaknesses. Section 5 summarises our findings and makes recommendations for organisations considering using COTS software in safety-related systems.

2 Safety assurance and standards

2.1 Recent developments in safety standards

The last couple of years have seen the emergence of new international safety standards for systems involving software [Waben99]. These include

- The long-awaited release of international (and now Australian) standard IEC 61508 "Functional safety of electrical/electronic/programmable electronic safety-related systems" [Std61508], which is likely to have a profound influence on civilian safety systems;

- New CENELEC standards in the railways sector, and emerging standard IEC 61511 in the industrial control sector; and
- A new Australian defence standard, Def(Aust) 5679 “The procurement of computer-based safety critical systems” [Std5679]

A new, substantially revised version of the US defence standard MIL_STAN 882 has been released [Std882D], and debate is reopening about the aerospace standard DO-178B [Std178B]. See Section 6 for a broader list of system safety standards; several good surveys are available [Waben99,WWW].

It is difficult to generalise about all these new standards, but some general trends – and common requirements – are becoming apparent. Most standards now require, in one form or another, that a **Safety Case** be developed, which presents documented evidence that the system is safe to operate. For software components, there generally needs to be an assessment of the software’s required contribution to safety, in the form of a **Safety Integrity Level (SIL)** that indicates how much the system relies on the software for safety [Lindsay97].

The standards typically recommend that certain activities (typically safety-oriented V&V activities, but also important infrastructure activities such as configuration management) be undertaken during development of safety-related software, and require that the rigor and independence with which the activities are carried out be commensurate with the Safety Integrity Level. (Note that the definition of SILs and the requirements for activities differ from standard to standard.) This might be termed a process-oriented approach, and reflects the degree of consensus currently available. Many commentators and evaluators would prefer to see more product-oriented evidence, but consensus on exactly what – and how much – should be required has been harder to reach.

Because of their focus on provision of assurance through development processes, however, standards generally have less guidance to offer to system developers and safety evaluators when it comes to COTS components. The rest of this section outlines what some of the standards say in relation to COTS software. It turns out that there are limited practical options available, and that while most standards allow concessions to be made, the preferred solution is to require equivalent rigor for the assurance for COTS and bespoke items. In Section 3 we return instead to fundamentals to develop some useful guidance for safety assurance of COTS software.

2.2 IEC 61508

IEC 61508 [Std61508] requires that, if standard or previously used components are to be used, they shall be clearly identified and their suitability justified. Justification may be derived from operation in a similar application or evidence of suitable verification and validation procedures. However, the previous environment(s) should also be evaluated to establish that any operational assumptions are satisfied by the new environment.

2.3 MIL-STD-882D

US system-safety defence standard MIL-STD 882D [Std882D] notes the difficulties presented by COTS products. It recommends tailoring the safety program to incorporate safety management and summary of assessment only for small COTS products. For larger COTS products, a formal safety plan is recommended, along with a safety working group and safety requirements/criteria analysis. General consideration is given to the assessment of any documentation or operational evidence and to performing additional hazard analyses as necessary.

2.4 Def(Aust) 5679

Australian defence standard Def(Aust) 5679 [Std5679] provides requirements and guidance for both COTS systems and COTS components (referred to as Non-Development Items).

Components developed in accordance with other safety standards may be assigned a level up to SIL S_2 if the Auditor approves and evidence is provided that component specifications meet the derived component safety requirements. To be assigned a SIL of S_3 or higher, COTS components require the application of full design and implementation assurance. Otherwise, components may only be rated S_0.

COTS systems must be developed to a safety standard. The processes of Def(Aust) 5679 must still be followed as far as is possible, including production of a Safety Case meeting the intent of Def(Aust) 5679. Presented safety arguments and processes can be approved at the discretion of the safety Auditor and Evaluator.

2.5 STANAG 4452

Analysis Task 8 of NATO standardisation agreement STANAG 4452 [Std4452] on munitions-related systems requires that commercial or government-furnished software be analysed and tested unless specifically excluded by the Managing Activity. The level of analysis and test required is not indicated.

2.6 UK defence standards

UK system-safety defence standard Def Stan 00-56 [Std00-56] requires the production of a Safety Case for COTS products. Detailed guidance on the retrospective application of the standard is given in Annex D of part 2. In particular, a Safety Programme Plan, a Project Quality Plan, a Project Configuration Management Plan and a Hazard Log should be established. Existing safety analysis information, including service histories, should be examined for deficiencies, and evaluated against the Def Stan 00-56 requirements.

In safety-critical software standard Def Stan 00-55 [Std00-55], the use of previously developed software in a new or modified system must be shown to not adversely affect the safety of the new system. Reverse engineering, verification and validation activities are required for any software not produced according to the standard; this is likely to require access to source code. The extent of reverse engineering may be reduced if other assurance activities have been conducted or the in-service history of the software is appropriate, and detailed guidance is given on this matter. In particular, quantified error rates and failure probabilities for the software may be taken into account.

Safety-critical hardware standard Def Stan 00-54 [Std00-54] notes the widespread use of

COTS products in hardware development. A COTS product may only be used if evidence of its integrity can be gathered from the process used in its design and production, its source of supply, and its service history. The argument used is qualitative. The COTS product must have been supplied with a comprehensive specification. Safety analysis is required to show that the item is not used outside the limits documented in the specification. Evidence is required of comprehensive testing that the item operates in accordance with its specification. Any failures of the COTS product must be recorded in the Safety Records Log, together with measures taken to prevent further occurrence of the fault. Any modifications to COTS products must be made in accordance with the standard. The configurations of all COTS products must be recorded.

2.7 Avionics standards

The Society of Automotive Engineer's Aerospace Recommended Practice ARP4754 [Std4754] considers COTS-like products in relation to the modification of aircraft. In particular, the problems of altering a legacy system and integrating a system with a different aircraft type are examined. In general, the certification data necessary to support the safety assessment are required. Credit may be sought for previous assurance activities if the aircraft or subsystem is traceable to the certification data. Otherwise, the applicant should identify and substantiate the assumptions necessary to support the assessment. If it is unavailable, certification data may be generated by reverse engineering or from an analysis of the service history.

The widely used aerospace-software development standard DO-178B [Std178B] discusses the use of existing software in new aircraft and software whose data does not satisfy the guidelines of the standard. Certification data should be reviewed and supplemented where necessary to satisfy the safety assessment and verification activities. Reverse engineering may be employed if data is not available. The service history may be used provided the configuration can be identified and an analysis confirming relevance of the service history can be provided. Some estimate of the software reliability is also required based on length of service period and records of observed failures.

3 Safety assurance requirements for COTS components

Having seen that standards offer little practical guidance on development of safety assurance of COTS software, we return instead to fundamentals and focus on three general criteria for the assurance of COTS components:

1. Verifying specified behaviour, including the elimination of unspecified behaviour;
2. Validating the safety of specified behaviour in the new operational context; and
3. Ensuring safety under change.

These are explained in more detail below.

3.1 Verifying specified behaviour

Assurance needs to be provided that the item behaves in accordance with specified functionality. The form of this assurance will vary with each strategy but must satisfy two conditions:

- Specified behaviour is satisfied with a level of confidence commensurate with the claimed integrity level; and
- No unspecified behaviour or influence can arise through undocumented functionality, dormant code or use of common resources. This requirement is often overlooked and can be difficult to achieve [Kohl98].

3.2 Validating safety of item behaviour

Even if the behaviour of an item can be trusted, it is also necessary to demonstrate that the behaviour is safe in the application environment. Safety is defined with respect to a particular operational environment so, if the environment is changed, “safeness” of the specified behaviour must be re-evaluated. (In particular, Safety Integrity Levels have no force outside the context in which a component’s safety assurance was developed.) Furthermore, the item assurance must be qualified for the new operational demands as well as the environmental conditions such as stress, temperature, humidity and electromagnetic interference.

Demonstrating safety of an item’s behaviour in a new application can be treated independently of assuring the behaviour itself. Some methods that have been proposed include:

- **Hazard analysis**. In most cases, it will be necessary to perform at least some hazard analysis to identify the hazardous conditions of the application in its operating environment. For COTS components, it is also possible to refine the analysis through the system architecture to define safety requirements that the item must satisfy [McDermid98].
- **Testing**. One proposal for certification of items is to use black-box testing together *with system-level fault injection*, to determine the effect on the system if the component fails, and *operational system testing*, to determine the effect on the system if the component functions properly [Voas98]. This has the benefit of using the real system but it can be time consuming and often relies on a sophisticated test environment. Of course, it is also difficult to achieve certainty of correctness with testing alone.
- **Modelling and simulation**. A similar approach is to model the item, together with bespoke components, and demonstrate that emergent safety requirements derived elsewhere in the safety case are met. The complexity of this task is reduced if the proofs can be discharged automatically by a model-checking (exhaustive simulation) tool as in the proposed approach of the Systems Assurance Group at DERA [O’Halloran99]. Where behaviour of an item is uncertain, this approach models the item pessimistically to allow for worst-case scenarios. Similarly, it incorporates fault injection to allow for errors in the COTS and bespoke components.

3.3 Retaining assurance under change

Like any software system, COTS items are likely to change. In particular, they frequently undergo modifications which are not always backwards compatible. A significant problem is maintaining assurance in the face of modification. To begin, we must be certain that the item configuration in operation corresponds to the assurance evidence of the safety case, even in the face of change. If the item changes, the assurance must be regenerated. Reacquiring assurance when items are modified can be costly, particularly when the scope of modifications are outside our control or even unknown.

3.4 The benefits of COTS measured up against safety goals

Before turning to possible approaches to providing safety assurance for COTS components in Section 4, it is worth revisiting the perceived benefits of COTS as described in Section 1, to see how they measure up:

- **Cost**: There is considerable debate as to whether COTS components really do reduce costs when the whole life-cycle, including maintenance, of the developed system is considered [McDermid98]. Some of the main reasons are given in the bullet points that follow.
- **Functionality**: As discussed, one of the main safety assurance activities is concerned with eliminating undesirable behaviour, including unexpected, unanticipated or unpredictable behaviour. It is thus important to be able to tightly constrain functionality, and to eliminate un-needed functionality. This is especially important where such functionality can have side-effects that are not easy to determine or to control. The unconstrained nature of many COTS applications, and the trend to add functionality in upgrades (often without notification), are major problems for developers of critical systems. (The Ariane 5 maiden flight is one example where un-needed functionality caused a system to fail.)
- **Useability**: While safety is generally increased by improving the display of information to operators, there is also a danger that operators will assume too much – a case of familiarity breeding contempt. Subtle differences in user interface functionality can have major undesired effects, as users of many applications packages will know.
- **Testedness**: Having a broad user base definitely can lead to better test coverage, at least at "black box" level. However, the environment in which the COTS component operates strongly influences its behaviour. Testing may not have taken place in a similar environment, or the same kind of requirements might not have been tested. Also, it has often been observed that many software failures occur in areas of functionality less tested. In recognition of this, software safety standards typically require evidence of high coverage of all possible "paths" through the software, especially where related to safety critical functionality. Such evidence cannot be collected if source code and detailed designs are not available.

  There is also an unfortunate commercial reality that time-to-market is often more important for COTS suppliers than correctness. Software is often released with known bugs included.

- **Support**: An effect of broadening the user base is that you may now be a small fish in a large pond, and have little or no contractual leverage to get problems fixed. The COTS package may evolve, leaving your version unsupported.
- **Future proof**: From an assurance viewpoint, upgrades can be problematical. Any change to a component may mean the whole system needs to be retested, which can be expensive and time-consuming. If the component changes substantially, it may mean redesigning and re-implementing system interfaces. These costs can outweigh any initial savings achieved for the component [McDermid98].

4 Providing safety assurance for COTS components

Four general approaches for generating safety assurance are examined below:

1. Transferring assurance from other development processes. This is particularly effective where the assurance processes are similar to those required by the safety standard being used or can be argued to be equivalently rigorous.
2. Isolate COTS components using appropriate design. Some appropriate design techniques are discussed, including wrappers and redundancy.
3. Post-development safety case construction. This is the approach favoured by most standards but can be prohibitively expensive.
4. Using empirical evidence of operational use. While this may offer a low-cost alternative, it is unlikely that sufficient accurate data collected under the same operational conditions and using the same version of the COTS item will be available to support the claims of required integrity. Arguments will therefore need to be made based on assumptions.

4.1 Transferring Assurance

When items are developed under standards other than that being used, it may be possible to transfer evidence into the safety case. Ideally, this would involve little or no technical evaluation and would be based on review of the kinds of supporting evidence provided under the other standard.

The major difficulty with this approach is relating levels of assurance between standards. The assurance tasks applied by different standards, and even the principles behind them can vary substantially. In general, it is not possible to provide a straightforward mapping between standard assurance levels and a more detailed comparison of processes is required. In some cases, the relationship between process is clear, for example the application of formal proof under UK Def Stan 00-55 or IEC 61508. However, justifying equivalence between different assurance processes, for example test and formal proof, will be very difficult.

Of course, it is still necessary to validate the claimed assurance in the context of the new application. This in itself can require some technical analysis. Hence the approach is more acceptable within particular domains, e.g., it is proposed as a strategy for COTS components used in nuclear reactors [Preckshot95, Scott95]. When this is not the case, additional evidence needs to be provided.

4.2 Restricting Components Through Design

It may be possible to utilise the functionality of low integrity COTS components within a high integrity design by restricting the influence of the component on the rest of the system. This is an attractive option where the functionality of an item is desirable and trust in item reliability is impossible or costly to achieve.

However, ensuring that an item is adequately restricted can also be costly and sometimes requires detailed knowledge of the item design. In particular, all possible influences of COTS components need to be established, including possible interference from dormant code and use of common resources.

One way of restricting the influence of COTS components is by isolating them using encapsulation mechanisms such as *wrappers* [Brown98, O'Halloran99]. Wrappers provide two roles: they can prevent certain inputs from reaching the COTS component and thus prevent it from performing particular functions, and they can check outputs from the COTS component and ensure they meet certain requirements [Voas98]. This can ensure that the COTS component does not adversely impact the rest of the system.

Effective use of wrappers requires that the COTS component's interface is well-understood. Any unexpected outputs may be able to pass through the wrapper. Lack of detailed and accurate documentation is therefore a problem when using this approach. Black box testing of COTS components with wrappers installed, and *fault-injection testing*, i.e., testing with random errors introduced, have been suggested as a means of increasing the assurance of the effectiveness of wrappers [Voas98]. However, testing may not be able to provide the high levels of confidence required. Another problem with wrappers is that they can be quite complex. It is possible that the wrapper can be larger than the COTS component it is isolating [McDermid98]. In such cases, it may be more effective to develop a new component than use the COTS component.

Another approach to restricting the influence of COTS components is by using a redundant architecture. Replication and majority voting are effective approaches for dealing with hardware. However, similar techniques cannot be effectively applied to software where faults are systematic: for the same inputs, replicated software components will produce the same results. *N-version programming*, e.g., using COTS products with the same functionality from different vendors, has been suggested, but has debatable reliability [Leveson95].

An alternative form of redundancy is to partition the system into a high-performance portion and a high-assurance kernel (also called a safety kernel [Std5679, Std4452]). COTS components can be used in the high-performance portion and, if they fail, the high-assurance kernel takes over and provides the required functionality (typically less efficiently). One such example is the Simplex Architecture [Sha98] and the approach is used in the nuclear and process control industries by separating control and protection functionality. Unfortunately, this approach suffers the same weaknesses as using wrappers: the kernel may not catch unexpected functionality, and the kernel may be prohibitively complex.

4.3 Post Development Safety Case Production

In principle, a safety case may be produced from documentation after development. This is a strategy favoured by all standards but it is often expensive and, in some cases, not practical since the COTS item cannot be subsequently altered. Faults, should they be found, cannot necessarily be fixed since the modification of COTS items is not controllable. This is not acceptable if the fault discovered could be the source of a critical hazard. Furthermore, the design implementation techniques used, e.g., the choice of programming language, may not conform to what the standard requires or be amenable to the required analysis techniques.

A primary difficulty with this strategy is that it requires access to source code and other documents which may not be readily available or too expensive to procure. (Thanks to Y2K, Australian law now allows a certain amount of reverse engineering of design, for purposes of testing functionality and integration of components into systems.) However, even if all such documentation is available, it may not accurately represent the actual COTS item's configuration.

Where documentation is available, it is not always feasible to reverse engineer a safety case due to the complexity of the task. In fact, it can be much more complex than developing a safety case for bespoke software [McDermid98]. This is particularly the case if the system has been developed without assessment in mind.

An alternative proposal for post-development certification of COTS items is to use black-box testing together with system-level fault and operational system [Voas98]. However, the effectiveness of black-box testing to achieve high levels of integrity is questionable and more rigorous testing generally requires knowledge of design and source code structure [Scott]. A particular problem is determining the total functionality of the COTS item to test [McDermid98].

4.4 Using Empirical Evidence

Some standards such as DO-178B [Std178B] and IEC 61508 [Std61508] allow assurance to be generated from empirical evidence such as operational use. Where it is applicable, this is a low-cost option for gaining assurance but much care needs to be taken to justify validity of the evidence.

Since empirical evidence is based on operational use, strong justification is required to show that the evidence will transfer into the new environment. In particular, it is difficult to reuse evidence of reliability when the operational demands differ significantly. For this reason, this strategy suffers the same weaknesses as transferring assurance and is similarly more acceptable in domains with stable operational profiles, e.g., it is also proposed as a strategy for dealing with COTS components used in nuclear reactors [Preckshot95, Scott95].

Even if operational profiles are similar, a very large body of evidence is required to provide sufficient assurance for safety-critical functions. Thus, operational evidence is mostly proposed to complement, rather than replace, other assurance methods [Preckshot95]. In fact, the number of operational hours required to demonstrate that the COTS item satisfies high levels of integrity is probably orders of magnitude greater than the COTS item's lifetime. For this reason, the approach is more applicable to particular COTS items such as real-time operating systems, which tend to be very stable and are used in hundreds of thousands of applications, than more general COTS items which are continuously upgraded [McDermid98].

Providing accurate records of operational use can also be an issue. Operational profiles are rarely recorded and difficult to estimate. Accurate incident logs are also scarce so the data on which to base evidence is unlikely to be valid, or even available.

5 Conclusions

We conclude with a brief summary of our findings, and a list of issues that need to be considered in selecting, acquiring and integrating COTS software into safety-related systems, over and above usual system-level safety assurance activities.

5.1 Summary

Four general approaches for generating safety assurance are examined above:

1. Transferring assurance from other development processes. Ideally, this involves little or no technical evaluation. However, relating levels of assurance between standards can be difficult and the claimed assurance must be validated in the new operating environment.
2. Restricting the influence of COTS components using appropriate design. This is particularly attractive where a COTS component with the desired functionality exists but its reliability is impossible or costly to ascertain. However, proposed approaches, such as wrappers and redundant architectures, require a good understanding of the component's interface, in particular a good understanding of un-needed functionality provided by the component, and may be prohibitively complex.
3. Post-development safety case construction. This is favoured by most standards but requires access to source code and other documents of the correct configuration, and can be more complex than developing a safety case for bespoke components. The inability to modify the COTS product to fix faults or use an implementation technique compliant with the standard is also an issue.
4. Using empirical evidence of operational use. This can be a low-cost option for gaining assurance. However, it is necessary to justify that the evidence is applicable in the new operating environment and generally a very large body of accurate evidence is required.

5.2 Issues in selecting COTS

- Where possible, choose components which have been used in similar applications, with a good record of freedom from failure, and for which a safety case has already been developed.
- The general consensus currently seems to be that COTS should not be used for components at Safety Integrity Levels above IEC61508 SIL2.
- Consider full life-cycle costs, and determine strategy with regard to upgrades.

5.3 Issues in acquiring COTS

- Where a safety case has been developed, be sure to acquire it too.
- Failing that, try to get access to design and testing information (including source code), as well as in-service history (with full details of the environment in which the component was used).
- At very least, get a full component specification (including a detailed interface specification) and information about the component's known failure modes and (if available) their observed frequency. Component specifications should include *all* component functionality, not just desired functionality.
- Negotiate an ongoing support agreement, particularly if a no-upgrades strategy is to be followed. If upgrades are to be incorporated, ensure configuration and version identification will be accurate and specifications will be kept up to date, and determine a system upgrade and regression testing strategy in advance.

5.4 Issues in integrating COTS components into system

- Use protective design techniques such as wrappers to isolate the component and/or safety kernels to monitor its state and maintain a safe system state.
- In addition to usual analysis (including failure modes analysis), analyse for possible effects of invocation of un-needed functionality.

5.5 Research issues

Further evaluation of the strengths and weaknesses of the four strategies we have presented needs to be carried out and criteria for the selection of the most effective approach in particular situations needs to be determined. In addition, a number of issues relating to the use of individual approaches need to be researched.

To use the approach of transferring assurance, guidance on the relationship between integrity levels of different standards needs to be provided. Also, techniques for transferring assurance into different operating environments need to be developed.

In order to use the approach of restricting components, there is a need to identify robust architectural design techniques that would allow low integrity components to be used in higher integrity environments. In addition, methods for identifying and analysing dormant code and un-needed functionality need to be developed. This approach also needs to be incorporated into existing safety standards by the provision of guidance on acceptable SIL reductions for protective architectures.

6 References

6.1 Software Safety Standards

[Std5679] Australian Department of Defence, Def(Aust) 5679, The Procurement of Computer-Based Safety Critical Systems, 1998.

[Std178B] Radio Technical Commission for Aeronautics, Software Considerations in Airborne Systems and Equipment Certification. RTCA/DO-178B, 1992.

[Std4754] Society of Automotive Engineers. Aerospace Recommended Practice 4754: Certification Considerations for Highly-Integrated or Complex Aircraft Systems, 1996.

[Std882D] US Department of Defense, MIL-STD-882D Standard Practice: System Safety, 1999.

[Std50128] European Committee for Electrotechnical Standardization, CENELEC prEN 50128, Railway Applications: Software for Railway Control and Protection Systems, draft 1998.

[Std61508] International Electrotechnical Commission, IEC 61508, Functional Safety of electrical/electronic/programmable electronic safety-related systems, 1998.

[Std61511] International Electrotechnical Commission, IEC 61511, Functional safety: Safety instrumented systems for the process industry sector, draft 2000.

[Std00-54] UK Ministry of Defence, Interim Defence Standard 00-54: Requirements for Safety Related Electronic Hardware in Defence Equipment, 1999.

[Std00-55] UK Ministry of Defence, Defence Standard 00-55: The Procurement of Safety Critical Software in Defence Equipment, 1995.

[Std00-56] UK Ministry of Defence, Defence Standard 00-56: Safety Management Requirements for Defence Systems, 1991.

[Std4452] North Atlantic Treaty Organisation. NATO STANAG 4452: Safety Assessment of Munition-Related Computing Systems, September 1996

6.2 Research papers, books and resource material

[Brown98] M.L. Brown. *Commercially Developed and Non-Development Items in Safety Critical Systems.* ***Slide presentation***, 1998.

[ESA96] European Space Agency. ARIANE 5: Flight 501 Failure. http://www.esrin.esa.it/tidc/Press/Press96/ariane5rep.html

[Kohl98] R. Kohl. *V & V of COTS Dormant Code: Challenges and Issues.* ***Slide presentation***, 1998.

[McDermid98] N. Talbert. *Interview with John McDermid: The Cost of COTS.* IEEE Computer, 31(6):46-52, June 1998.

[O'Halloran99] C. O'Halloran. *Assessing Safety Critical COTS Systems.* Journal of the System Safety Society, 35(2), 1999.

[Leveson95] N.G. Leveson. *Safeware: System Safety and Computers.* Addison-Wesley, 1995.

[Lindsay97] P.A. Lindsay and J.A. McDermid, *A Systematic Approach To Software Safety Integrity Levels*, in: Proceedings 16th Internat Conf on Computer Safety, Reliability and Security (SAFECOMP'97), York 1997, Springer Verlag, 1997. http://svrc.it.uq.edu.au/Bibliography/svrc-tr.html?97-04

[Preckshot95] G.G. Preckshot and J.A. Scott. *A Proposed Acceptance Process for Commercial Off-The-Shelf (COTS) Software in Reactor Applications.* Lawrence Livermore National Laboratory, Report UCRL-ID-122526, September 1995.

[Scott] J.A. Scott and J.D. Lawrence. *Testing Existing Software for Safety-Related Applications.* Lawrence Livermore National Laboratory, Report UCRL-ID-117224, Revision 7.1.

[Scott95] J.A. Scott, G.G. Preckshot and J.M. Gallagher. *Using Commercial-Off-the-Shelf (COTS) Software in High-Consequence Safety Systems*. Lawrence Livermore National Laboratory, Report UCRL-JC-122246, November 1995.

[SCS] Safety critical systems e-mail forum. High Integrity Systems Engineering group, University of York. http://www.cs.york.ac.uk/hise/sclist/sclist.html

[Sha98] L. Sha, J.B. Goodenough and B. Pollack. *Simplex Architecture: Meeting the Challenges of Using COTS in High-Reliability Systems.* Crosstalk, April 1998.

[Voas98] J.M. Voas. *Certifying Off-the-Shelf Software Components.* IEEE Computer, 31(6):53-59, June 1998.

[Waben99] A.K. Wabenhorst and B. Atchison, *A survey of international safety standards*, Software Verification Research Centre technical report 99-30, 1999. http://svrc.it.uq.edu.au/Bibliography/svrc-tr.html?99-30

[WWW] WWW Virtual Library – safety critical systems. http://www.comlab.ox.ac.uk/archive/safety.html