A FRAMEWORK FOR DETERMINING THE SUFFICIENCY OF SOFTWARE SAFETY ASSURANCE

R.D. Hawkins, T.P. Kelly

Department of Computer Science, The University of York, Deramore Lane,
York, YO10 5GH, UK
{rhawkins|tim.kelly}@cs.york.ac.uk

Keywords: Software, Safety, Assurance, Suppliers, Sufficiency.

Abstract

In this paper we present a framework for ensuring software suppliers provide the necessary information about their software in order to support an overall platform safety case. The framework has been developed particularly for use on defence projects utilising a range of both bespoke and previously developed software. The framework aims to provide detailed guidance on *what* is expected from the software supplier (to avoid ambiguity, inconsistency and uncertainty), but not to unnecessarily constrain the supplier by detailing *how* that should be achieved (to facilitate the use of previously developed software and a wide supplier base).

The framework defines a set of five core software safety assurance principles. These principles must be shown to be addressed for all software that may contribute to hazards of the platform. The framework also defines the criteria by which the sufficiency of the evidence provided by the supplier against these principles is determined.

1. Requirements for the software safety assurance framework

The primary aim of the software safety framework described in this paper is to ensure the overall platform meets the requirements of Defence Standard 00-56 Issue 4 [1]. The framework has been developed particularly for use on a complex defence project utilising a large and diverse range of both bespoke and previously developed software products (both in-house and external). This means that there are a number of different suppliers, each having responsibility for developing and assuring their own software. In addition the software suppliers may utilise a range of software assurance and development standards and processes.

What is therefore required is a software safety framework that ensures that the suppliers provide the necessary information about their software in order to support the overall platform safety case. The framework must make it clear how the sufficiency of this information will be determined, without being overly prescriptive. Ultimately the aim of the framework is to provide detailed guidance on what is expected from the software supplier (to avoid ambiguity, inconsistency and uncertainty), but not to unnecessarily constrain the supplier by detailing how that should be achieved (to facilitate the use of previously developed software and a wide supplier base).

1.1. Challenges of existing approaches to software safety assurance

The type of projects we are particularly considering in this paper – highly integrated with a large number of suppliers who often provide previously developed or COTS products - are very common, perhaps the norm, for large military systems. Previous experience with projects of this type has shown that the integrator is often required to carry out large amounts of expensive additional work (in addition to that done by the software supplier) in order to make an overall safety case for the platform. This results from:

- Inconsistency across the suppliers in what is considered an acceptable software safety approach.
- Too little software safety evidence being provided by the supplier in order to provide the necessary assurance.
- Lack of understanding by the suppliers of the role of their software in the occurrence of platform hazards.
- Supplier's claiming to have developed 'safe software' but failing to provide sufficient demonstration and justification.

This approach presented in this paper aims to reduce project risk by providing a framework that addresses these problems.

1.2. What is required from software suppliers?

00-56 requires the production of a safety case consisting of "a structured argument, supported by a body of evidence, that provides a compelling, comprehensible and valid case that a system is safe for a given application in a given environment." For the software aspects of the system, this requires an argument regarding the risk contribution of that software (a "risk argument"), i.e. how is the software contribution to the hazards of the overall platform controlled?

00-56 also requires that the argument and evidence should be commensurate with the risk posed by the system. For the software aspects of the system, this requires an argument documenting the reasons why there is sufficient confidence in the argument of the risk contribution of the software (a "confidence argument"), i.e. the confidence in the software

risk argument is appropriate for the criticality of the software contribution to platform hazards.
The idea of splitting a safety argument into a risk argument and associated confidence argument was first proposed in [2]. The framework developed here uses this idea as a basis for clearly defining the expectations on the software suppliers.

1.3. What is required by software suppliers?

In order to provide a software risk argument, suppliers must correctly understand how, and to what extent, their software can contribute to the hazards of the overall platform. This requires that the following information be defined for all software under consideration:

- Specific contributions that the software may make to the identified platform hazards, either in the form of potential failure behaviour of the software (that the software is required to avoid), or defined functionality or properties that the software must provide. These can be identified from the hazard analysis activities performed as part of a standard platform hazard analysis process (for example an FFA or FTA).
- The criticality of those contributions, based upon the severity of the hazardous outcome and the degree of contribution of the software to the overall hazard. This recognises the fact that the software is normally just one element of the overall contribution of a system to a platform hazard.

There are a number of similar approaches to categorising the criticality of software contributions to system hazards. A commonly used approach is that defined by US MIL Std 882C [3] that determines a software hazard risk index based on the severity category of the hazard, and the degree of control that the software has over the hardware aspects of the system. Our approach categorises the criticality of each software contribution to a platform hazard in a similar manner. Since we are determining the software contributions for each equipment, the levels of hardware redundancy provided by *other equipment* at the platform level is also taken into consideration, as well as the degree of control of the software within that equipment. We define four levels of criticality for equipment hazardous software contributions: Critical, High, Medium, Low.
The information described above is the minimum information required by software suppliers in order to be able to provide a software risk argument. It defines how their software affects safety at the platform level, taking into account the particular hazards of the platform and the interactions of their equipment with other systems. Without this, although a software provider may be able to reason about the behaviour and potential failure modes of their software, they are unable to understand how that will affect the overall platform hazards. This means that this information must be determined early on such that it can be passed onto the suppliers.

2. Overview of the Framework

The framework defines in detail the expectations on software suppliers with respect to demonstrating the safety of the software as part of the platform. The framework addresses both the software risk argument and the confidence argument.
To address the software risk argument, the framework defines a set of five core software safety assurance principles. These principles must be shown to be addressed through the provision of sufficient evidence for all software that may contribute to hazards of the platform. The software safety assurance principles are described in section 2.1. These principles are fundamental to any software risk argument and, as such, are independent of the criticality of the software contribution to hazards or any other particular features of the software under consideration.
To address the confidence argument, the framework defines the criteria by which the sufficiency of the evidence provided to demonstrate the achievement of the core principles is determined. Unlike the software safety assurance principles, the confidence criteria vary according to the criticality of the software contribution to the platform hazards. Where the software contribution is of higher criticality, more confidence is required that the assurance principles have been addressed. The framework splits the confidence criteria into the following areas:

- The criteria to be used to assess whether the confidence achieved in each of the core principles is commensurate with the criticality. See section 2.2.
- The types of safety assurance evidence that may be used to address each of the core principles and the limitations that influence the confidence that can be gained from each evidence type. See section 2.3.
- Criteria for determining the trustworthiness of each evidence item with respect to the criticality. See section 2.4.

2.1. Software Safety Assurance Principles

In this section we describe the core software safety assurance principles that must be addressed for any software that may contribute to platform hazards. These principles have largely been extracted from previous work undertaken by the authors [4], [5]. The principles have been found to be valid in a number of platforms, including those reported in [6].
These principles are:

- **Software Safety Requirements validity (Valid)** – Demonstrate that the defined software safety requirements provide an appropriate means of addressing the identified software contributions to platform hazards.
- **Software Safety Requirements satisfaction (Satisfaction)** – Demonstrate satisfaction of all the identified software safety requirements.
- **Software Safety Requirements Decomposition (Decomposition)** – Demonstrate that the requirements and design are appropriately allocated, decomposed, apportioned and interpreted throughout the decomposition of the software design and through to implementation.
- **Assessment of Hazardous Software Failure Behaviour (Hazardous Behaviour) –** Demonstrate that potential

hazardous failure behaviour of the software has been assessed and addressed.

- **Absence of Development Errors (Errors) -** Demonstrate that potentially hazardous errors have not been introduced through the software development and implementation process. For example, it must be demonstrated that the software is free from intrinsic errors (e.g. buffer overflows and divide-by-zero errors).

The principles should be demonstrated to hold at all levels of abstraction in the software design. These core principles are always valid, and therefore do not vary according to the criticality of the software contribution to hazards. What does vary according to criticality is the confidence with which those principles are demonstrated, as described below.

2.2. Confidence Criteria for the Software Safety Assurance Principles

Each of the software safety assurance principles must be demonstrate with sufficient confidence. Confidence is gained that the principle is addressed through the provision of evidence, and also an argument as to how that evidence demonstrates that principle. The confidence provided in each principle must be commensurate with the criticality of the software contribution to the platform hazard. That is to say that where software makes a more critical contribution to platform hazards, more confidence must be provided that the principles have been met. Below, we define the criteria for determining whether the assurance is commensurate with the each criticality level.
These criteria have been defined by characterising what is meant by demonstrating different levels of confidence in each of the software safety assurance principles. These definitions build on work undertaken by Reinhart and McDermid [7], who provide more general definitions for different levels of assurance.

Critical (1)

Satisfaction: Absolute assurance provided that the defined software safety requirements will always be met as required by the executed software in a defined execution and operational context. There is no uncertainty in the achievement of the software safety requirement.

Decomposition: Absolute assurance provided that the decomposed software design appropriately captures the software safety requirements defined at the previous level of decomposition. There is no uncertainty in the correctness of the requirements decomposition.

Hazardous Behaviour: Absolute assurance provided that all potential hazardous behaviour of the executing software has been identified and addressed. There is complete certainty that all potential hazardous behaviour of the software has been considered.

Errors: Absolute assurance provided that the software does not contain development errors. There is no uncertainty regarding the presence of residual development errors.

High (2)

Satisfaction: All reasonably practicable steps have been taken to demonstrate that the software safety requirements will always be met as required by the executed software in a defined execution and operational context. There may be some remaining uncertainty but this is unlikely to lead to violation of the safety requirement.

Decomposition: All reasonably practicable steps have been taken to demonstrate that the decomposed software design appropriately captures the software safety requirements defined at the previous level of decomposition. There may be some remaining uncertainty in the adequacy of the requirements decomposition but this is will not impact the safe behaviour of the software.

Hazardous Behaviour: All reasonably practicable steps have been taken to demonstrate that all potential hazardous behaviour of the executing software has been identified and addressed. There may not be complete certainty that the identification is exhaustive.

Errors: All reasonably practicable steps have been taken to demonstrate that the software does not contain development errors. There may be some remaining uncertainty regarding the presence of residual development errors but this is not expected to impact the safe behaviour of the software.

Medium (3)

Satisfaction: Steps have been taken to demonstrate that the software safety requirements will always be met as required by the executed software in a defined execution and operational context. The remaining uncertainty would only lead to violation of the safety requirement under exceptional defined operational circumstances.

Decomposition: Steps have been taken to demonstrate that the decomposed software design appropriately captures the software safety requirements defined at the previous level of decomposition. The remaining uncertainty is unlikely to impact the safe behaviour of the software.

Hazardous Behaviour: Steps have been taken to demonstrate that all potential hazardous behaviour of the executing software has been identified and addressed. The identification is not exhaustive but is systematic and thorough.

Errors: Steps have been taken to demonstrate that the software does not contain development errors. The remaining uncertainty is unlikely to impact the safe behaviour of the software.

Low (4)

Satisfaction: Steps have been taken to demonstrate that the software safety requirements will be met as required by the executed software in a defined execution and operational context. The remaining uncertainty could lead to violation of the safety requirement, but this would not be expected under normal operating conditions.

Decomposition: Steps have been taken to demonstrate that the decomposed software design appropriately captures the software safety requirements defined at the previous level of

decomposition. The remaining uncertainty could potentially impact the safe behaviour of the software.

Hazardous Behaviour: Steps have been taken to demonstrate that potential hazardous behaviour of the software has been identified and addressed. The identification may not be systematic and thorough.

Errors: Steps have been taken to demonstrate that the software does not contain development errors. The remaining uncertainty could potentially impact the safe behaviour of the software.

2.3. Limitations of Evidence Types

It is expected that many different types of evidence will be used to provide assurance in the safety of the software. In previous work [8] we have described the limitations of different types of software assurance evidence with respect to different claims in a software safety argument. These limitations identify the factors that influence the confidence that can be gained from each type of evidence. These limitations should be considered when using the criteria in section 2.2 on the required level of confidence for each criticality level. The limitations of the different types of evidence mean that in order to achieve the required level of assurance against one of the principles, it may be necessary to use multiple forms of evidence.
The types of evidence that would typically be used to address each of the principles are indicated below. The limitations of each type of evidence are summarised.

Satisfaction:

Evidence Type	Limitations
Testing	Test cases may not have sufficient coverage to trigger all possible outputs.
Analysis	Reliant upon the accuracy of model and hardware assumptions. Non-formal analysis may not be repeatable.
Design / Code Reviews	Reviews cannot directly demonstrate achievement of the requirement. Reviews are subjective and not repeatable. Information reviewed is documentation dependant (can only reveal errors in what's documented).
Field Experience	Cannot guarantee that field experience has exposed all relevant errors. The environment and operational context of the field experience may not be exactly that of the target system.

Decomposition:

Evidence Type	Limitations
Formal Analysis	Reliant upon the accuracy of model assumptions.
Manual Design Review	Reviews are subjective and not repeatable.

Hazardous Behaviour:

Evidence Type	Limitations
Failure/Hazard Analysis	These techniques are varyingly subjective and are not necessarily exhaustive in consideration of erroneous behaviour.
Field Experience	Cannot guarantee that all hazardous functional failures have been exposed by the field experience. The environment and operational context of the field experience may not be exactly that of the target system.

Errors:

Evidence Type	Limitations
Formal Analysis / Model Checking	Reliant upon the accuracy of model assumptions.
Manual Design Review / Review of conformance to standards / guidelines	Reviews are subjective and not repeatable.

2.4. Trustworthiness Criteria

For each item of evidence generated the trustworthiness of that item must be appropriate for the level of criticality of the software contribution. The trustworthiness of the evidence item is determined by the rigour of the process used to generate that item. The main parameters that will affect trustworthiness are:

- Independence
- Personnel
- Methodology
- Level of Audit and Review
- Tools

We have provided guidance on how to assess the sufficiency of an evidence item against each of these parameters at different levels of criticality (see Table 1). This guidance is based upon best practice as defined in standards such as [9], [10], [11]. Note that the criteria used is the same for both High and Medium criticality levels (2 and 3).

3. Use of Previously Developed Software

On large complex project, particularly one with multiple external software suppliers, it is likely that at least some of the software used will have been previously developed (either commercially or on another project). In these situations, although it is not possible to constrain the development of the software, it is still necessary to show how the existing

software product and evidence set meets the criteria for the required criticality. This may, of course, identify some gaps in the existing evidence that then need to be addressed before the software can be used as part of the platform. It is not within the scope of this paper to discuss strategies for generating additional evidence relating to previously developed software (other work in this area exist such as [12] and [13]), however the explicitness of the criteria and guidance presented in this framework will facilitate filling identified gaps.

3.1. Use of Other Standards

The software suppliers may utilise a range of software assurance and development standards and processes. One of the aims of the framework was to provide the flexibility to allow suppliers to use other standards as part of an assurance strategy that ensures compliance with the framework criteria. Where software standards (such as DO-178B [10] or IEC 61508 [9]) have been used by a supplier the framework criteria remain, and the supplier must indicate how the principles are addressed in the use of the standard. It is anticipated that there may be gaps identified between the evidence generated by the supplier in complying with the standard, and the evidence requirements laid out in this framework. The supplier must define a strategy for generating the additional evidence required.

4. Conclusions

The framework we have described in this paper has been developed in order to address particular challenges of complex projects utilising a large and diverse range of both bespoke and previously developed software products. One of the biggest challenges identified on previous projects working within a Def Stan 00-56 regime is how to provide guidance to suppliers in enough detail that they understand clearly what is expected from them without unnecessarily constraining them by prescribing how this is to be achieved. We believe that the framework described here achieves this difficult balance, in particular it makes clear how other software standards (such as IEC 61508 or DO178B) may be used by the suppliers as part of a sufficient software safety approach.

Another advantage of the framework is that it makes it relatively easy to integrate the software safety justification produced by the supplier as part of the overall platform safety case. This is due to enabling the supplier to produce a software risk argument focussing on the explicit contributions of the software to platform hazards. Without this the supplier can only reason about the planned and unplanned behaviour of their software, but not the impact of this on the safety of the platform. This is important since it reduces the additional work required by the integrator in order to demonstrate that the supplier software is safe in the particular platform context. It also ensures that responsibility for developing safe software and demonstrating its safety remains, as far as practicable with the software suppliers.

We believe that although 'pockets' of useful guidance and best practice have existed for some time, this framework presents a complete, coherent, consistent and easy to use approach with benefits for both integrators and suppliers.

Acknowledgements

The authors would like to thank BAE Systems for their support in funding this work.

References

[1] MoD. "Defence Standard 00-56 Issue 4: Safety Management Requirements for Defence Systems," *HMSO*, (2007).

[2] R. Hawkins, T. Kelly, J. Knight, P. Graydon. "A New Approach to Creating Clear Safety Arguments", In Proceedings of the Nineteenth Safety-Critical Systems Symposium (SSS '11), Southampton (2011)

[3] US DoD. "MIL-STD-882C – System Safety Program Requirements", *US DoD,* (1993).

[4] R. Hawkins, T. Kelly. "A Systematic Approach for Developing Software Safety Arguments", *Journal of System Safety,* Volume 46, No. 4, pp 25-33, System Safety Society Inc. (2010).

[5] R. Hawkins, T. Kelly. "A Software Safety Argument Pattern Catalogue", Available at www.cs.york.ac.uk/~rhawkins/pubs.html, (2010).

[6] R. Hawkins, K. Clegg, R. Alexander, T. Kelly. "Using a Software Safety Argument Pattern Catalogue: Two Case Studies", 30th International Conference on Computer Safety, Reliability and Security (SAFECOMP '11), Naples, Italy, (2011).

[7] D. W. Reinhardt, J. A. McDermid. "Contracting for architectural, claims and evidence assurance for military aviation systems", In Proceeding of the Australian System Safety Conference, (2012).

[8] R. Hawkins, T. Kelly. "A Structured Approach to Selecting and Justifying Software Safety Evidence", In Proceedings of the IET International System Safety Conference, (2010).

[9] IEC, "61508 – Functional Safety of Electrical / Electronic / Programmable Electronic Safety-Related Systems", *IEC*, (1998).

[10] RTCA. "DO178B – Software Considerations in Airborne Systems and Equipment Certification", *RTCA,* (1992)

[11] IEE. "Safety, Competency and Commitment: Competency Guidelines for Safety-Related System Practitioners", *IEE,* (2000)

[12] F. Ye. "Justifying the Use of COTS Components within Safety Critical Applications", PhD Thesis, The University of York, (2006).

[13] C. Menon, J. McDermid, P. Hubbard. "Goal–based Safety Standards and COTS Software Selection", In Proceedings of the 4th IET International System Safety Conference, (2009).

Independence		
1	**Independent Organisation**	Organisation separate and distinct, by management and other resources, from the organisations responsible for the activities that take place
2/3	**Independent Department**	Department separate and distinct from the departments responsible for the activities that take place
4	**Independent Person**	Person separate and distinct from the activities that take place with no direct responsibility for those activities
Personnel		
1	**Expert**	Has sufficient understanding of why things are done in certain ways, and sufficient demonstrated management skills, to be able to undertake overall responsibility for the performance of a function. Familiar with the ways in which systems have failed in the past. Keeps abreast of technologies, architectures, application solutions, standards and regulatory requirements. Has sufficient breadth of experience, knowledge and deep understanding to be able to work in novel situations. Is able to deal with a multiplicity of problems under pressure without jeopardising safety issues.
2/3	**Practitioner**	Has sufficient knowledge and understanding of best practice, and sufficient demonstrated experience, to be able to work on the task without the need for detailed supervision. Maintains knowledge and aware of current developments.
4	**Supervised Practitioner**	Has sufficient knowledge and understanding of best practice to be able to work on the task without placing an excessive burden on the practitioner or expert which might compromise safety. Supervised practitioners may not have had experience of working on a safety-related project.
Methodology		
1	**Objective Reasoning of Achievement**	Method has objective, systematic reasoning that the required outcome is achieved
2/3	**Objective Acceptance Criteria**	Method has objective acceptance criteria that can give a high level of confidence that the required outcome is achieved (exceptions to be identified and justified).
4	**No Objective Acceptance Criteria**	Method has no or little objective acceptance criteria
Level of Audit and Review		
1	**Independent Audit**	Independent check that the activities undertaken are compliant with the defined process. Audit should be undertaken by person(s) from independent organisation.
2/3	**Independent Audit**	Independent check that the activities undertaken are compliant with the defined process. Audit should be undertaken by person(s) from independent department.
4	**Self-Check**	No independent check that the activities undertaken are compliant with the defined process.
Tools		
1	**Objective Reasoning of Correct Operation**	Objective, systematic reasoning that the (verification) tool functions according to its operational requirements under normal operational conditions
2/3	**Demonstrated Correct Operation**	Evidence provided that the (verification) tool functions according to its operational requirements under normal operational conditions
4	**Unverified**	Little or no evidence provided that the (verification) tool functions according to its operational requirements under normal operational conditions

Table 1 – Assessment of evidence trustworthiness criteria