A Structured Approach to Selecting and Justifying Software Safety Evidence

R. Hawkins and T. Kelly

{richard.hawkins | tim.kelly@cs.york.ac.uk}
Department of Computer Science,
The University of York, York, YO10 5DD
UK

Keywords: software, safety, evidence, safety argument.

Abstract

The safety assurance of software is ultimately demonstrated by the evidence that is put forward. There is a range of existing guidance on the types of evidence that may be used to demonstrate the safety of software, however questions remain as to the sufficiency of the evidence suggested by such guidance. We propose that the only way to determine the sufficiency of the evidence is to consider its capability to address specific explicit safety assurance claims in a software safety argument. In this paper we propose a lightweight approach to selecting and assessing software safety evidence.

1 Introduction

It is possible to demonstrate the safety of a software system through the presentation of a software safety argument as required by Defence Standard 00-56 [10]. The purpose of such an argument is to demonstrate that sufficient assurance in the safety of the software has been achieved. In previous work undertaken in the SSEI [4], [5] we have looked at how a compelling software safety argument may be constructed. This work provided guidance on the nature of the argument and safety assurance claims that must be supported for the software. We also provided guidance on how to identify and mitigate assurance deficits which may arise throughout the development of the software system and associated safety assurance argument.

The overall assurance that is achieved is ultimately determined by the evidence that is put forward to support the argument. There is a range of existing guidance on the types of evidence that may be used to demonstrate the safety of software, such as in [12] and [1]. Particularly detailed is that provided within Part 3 of the safety standard IEC 61508 [6]. Guidance such as this provides detailed lists of the types of evidence that may be appropriate to provide in support of software developed to different levels of integrity. Such processes are useful as they set out explicitly the various types of evidence that may be generated. They can also provide valuable guidance on how to implement a high-quality and repeatable software engineering process. However questions remain as to the sufficiency of the evidence suggested by such guidance. The main problem lies in the lack of rationale as to why those particular items of evidence are generated, and thus their sufficiency for demonstrating that the software is acceptably safe to operate in a particular context.

To solve this problem we propose that the software safety assurance argument be used to determine the requirement for, and sufficiency of, particular items of software assurance evidence. Software safety evidence should be selected principally upon its capability to address specific explicit safety assurance claims in a software safety argument. We propose a lightweight approach to selecting and assessing evidence based upon the consideration of three simple questions:

1. Is the *type* of evidence capable of supporting the safety claim?
2. Is the particular *instance* of that type of evidence capable of supporting the safety claim?
3. Can the instance of that type of evidence be *trusted* to deliver the expected capability?

Such questions could be asked of any item of evidence being used to support any argument and can identify weaknesses or limitations in the sufficiency of the evidence. We refer to such weaknesses as *assurance deficits* since they have potential to undermine our assurance in a safety claim. By identifying and mitigating assurance deficits throughout system development, it becomes possible to justify that the evidence provided is sufficient. In this paper we investigate how these questions can be applied to software safety. We use this to define a systematic software safety evidence selection process.

2 The Nature of Software Safety Assurance Claims

The starting point for selecting and assessing software safety assurance evidence is a safety argument for the software aspects of the system under consideration. In previous work, we have considered how to structure explicit hazard-focused arguments regarding the safety of the software. This included the generation of a software safety argument pattern catalogue. The full software safety argument pattern catalogue is documented in Appendix B of [9]. These software safety argument patterns provide guidance on the structure of the software safety argument, and the nature of the safety claims that may be expected to be made for any safety-related software system.

To ensure flexibility, the structure of the software safety argument patterns is based upon a generalized ‘tier’ model of development such as that proposed in [7]. Each tier corresponds to one level of decomposition of the design. Key to the arguments is establishing the satisfaction of software safety requirements (SSRs) and the absence of hazardous errors throughout the tiers of design decomposition of the software. More specifically, evidence is required to support safety assurance claims regarding:

Safety Requirement Satisfaction - Evidence that demonstrates SSRs have been met. An example of evidence that might be used to support this is evidence from testing, which can be used to demonstrate that the required behaviour occurs.

Safety Requirement Decomposition - Evidence that demonstrates the requirements and design are appropriately allocated, decomposed, apportioned and interpreted at each tier of decomposition of the software design. Formal proof of specification equivalence could be used to support a decomposition argument.

Absence of design errors - Evidence that demonstrates (hazardous) errors have not been introduced into the design. Such claims could, for example, be supported by evidence from a manual design review.

Assessment of hazardous failure behaviour - Evidence that demonstrates hazardous failure behaviour of software has been assessed e.g. field experience of similar systems.

Having established the nature of the software safety assurance claims that require support, it possible to consider how the three simple questions described earlier can be applied.

3 Question 1 – Capability of Evidence Types

All types of evidence have certain limitations, or fundamental weaknesses. In the same way that fault trees could never be used as evidence of hazard identification, different types of software evidence will also have certain inherent limitations. When Dijkstra famously said in his Turing Award Lecture in 1972, ``Program testing can be used to show the presence of bugs, but never to show their absence!", he was neatly capturing a fundamental limit on the capability of program testing as a type of evidence. These limitations can give rise to questions as to the capability of a particular type of evidence to support a particular software safety assurance claim.

It is possible, in considering question 1, to determine how different common types of software safety assurance evidence may support the different broad types of safety claim described in section 2. For this purpose we can use a simple categorisation of the types of evidence that might be expected to be used for software safety. These are:

Testing - Any evidence based upon the execution of the software
Analysis - Any evidence based upon repeatable and objective analysis of an artefact. This may include evidence based upon a formal approach.
Review - Any evidence based upon subjective manual assessment of an artefact.
Field experience - Any evidence based upon experience from operation of a system.

It is important here to note a clear distinction between reviews (which tend to be manual qualitative assessments) and analysis (which is repeatable and systematic evidence of correctness).

For each evidence type the *role* and *limitations* of that type of evidence in supporting a particular type of claim can be captured. It is possible to break the type of claim down further by considering the nature of the software safety requirement (SSR) to which that claim relates. SSRs can be categorised as functional requirements, timing requirements, or value (data) requirements. The types of evidence that could provide strong support to claims relating to a functional SSR, may be quite different from the types of evidence appropriate for a timing SSR. Table 1 shows an example of how information on the role and limitations of each type of evidence could be captured. This information can then be used to inform a decision on the most appropriate type of evidence to support a particular claim.

This approach is similar to the concept of `expressive power' used in the systems domain. In [8] the authors establish a hierarchy among the most commonly used types of dependability models, according to their modelling power. They describe how the choice of a suitable model type is determined by factors such as ease of use in a particular application, the kind of system behaviour to be modelled, and the conciseness and ease of model specification.

4 Question 2 – Capability of Evidence Instances

Because evidence of a particular type has the potential to support a particular claim in the software safety argument, this does not imply that every item of evidence of that type will be appropriate. The second question considers factors which affect the capability of individual evidence instances to support specific claims in the argument. There are a number of factors that can affect the capability of an item of software evidence to support a particular safety claim. These include:

Relevance – Is the item of evidence relevant to the claim within this particular argument? For example was a code review performed upon the correct version of the software.
Context and Assumptions – The suitability of an item of evidence can only be judged within the specific system and argument context and assumptions. For example failure analysis evidence may have been generated on the assumption that a robust partitioning mechanism was in place. The

Nature of Safety Claim		Evidence Type: Testing	Analysis	Review	Field Experience
Functional Requirement	**Satisfaction claim** – demonstrate that the required output will always be provided when required	**Role –** Demonstrate that each test case results in the required output **Limitations** – Test cases may not be sufficient to trigger all possible outputs	**Role –** Demonstrate that input will *always* result in expected output **Limitations** – Reliant upon the accuracy of model and hardware assumptions. Non-formal methods may not be repeatable.	**Role –** Check errors are not made in the design/code which affect the achievement of the requirement. **Limitations** – Reviews cannot directly demonstrate *achievement* of the requirement. Reviews are subjective and not repeatable.	**Role –** Identify any errors that occur relating to the requirement. **Limitations** – Cannot guarantee that all the relevant errors have been exposed by the experience. The environment and operational context of the experience may not be exactly that of the target system.
	Decomposition claim – demonstrate that the decomposed specification correctly captures the functional requirement of the previous tier of design	**Role –** N/A	**Role –** Demonstrate that the decomposed design is equivalent to previous specification. **Limitations** – Reliant upon the accuracy of model assumptions.	**Role –** Check that the decomposed design is equivalent to previous specification. **Limitations** – Reviews are subjective and not repeatable.	**Role –** N/A
	Absence of Design Errors claim – demonstrate that…	**Role –** N/A	**Role –** Demonstrate that…	**Role –** Check that…	**Role –** N/A

Table 1: Example table comparing types of evidence against types of safety claim

evidence item will therefore only be suitable if such a mechanism exists in the implemented system.

Coverage – Is the coverage provided by testing or analysis sufficient to demonstrate the required properties?

Depth[1] **–** Depth represents the level of design detail against which the evidence is provided. The depth of the evidence must be appropriate for the claim being supported. For example a claim relating to properties of the software code could not be supported by evidence from integration testing (which does not consider code structure).

Note that the factors discussed above are properties of a particular instance of evidence, rather than a limitation common to all evidence of that type.

5 Question 3 – Trustworthiness of Evidence

The trustworthiness of an item of evidence is the confidence that the item of evidence delivers its expected capability. This property could also be considered to be the integrity of the item of evidence, or its rigour. An item of evidence which is untrustworthy may not provide the support to the safety assurance claim that was expected when that item of evidence was selected.

We propose that the best way to justify that an item of evidence is sufficiently trustworthy is to provide an assurance argument relating specifically to that item of evidence, an approach suggested in [3]. For each item of evidence it is necessary to consider the reasons why that evidence may fail to deliver its expected capability. Each item of evidence is generated as the result of undertaking a process. It is when the process is being performed that weaknesses are introduced into the evidence. We propose that the factors that affect the trustworthiness of the evidence are best identified by considering potential failures in the evidence generation process. The assurance argument for the item of evidence is then used to demonstrate confidence in these process aspects.

It is described in [11] how a process model can be used to capture the important characteristics of a development process. It notes that it is important that such a model should not be subsumed in huge amounts of detail. It suggests the process model should:

- Capture inputs and outputs
- Capture resources
- Capture role of human beings
- Reveal the process dynamics, for example if a process contains a high degree of iteration or parallelism then the model should show this

Based on this we can provide simple models of the process used to generate items of evidence, as shown in figure 1a. Figure 1b shows how this simple model might apply in the case of system functional test results.

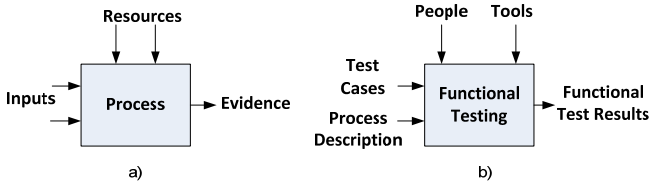

Figure 1: Simple process model for evidence generation

In order to reveal weaknesses in the process it is necessary to consider deviations in that process. We propose applying

[1] The Common Criteria [2] concept of depth considers that greater assurance is achieved at a finer level of design and implementation detail.

Input	Guideword	Interpretation	Effect on safety claim	Argument consideration
People	less	The personnel carrying out the tests are not sufficiently competent	The tests may not be sufficient to demonstrate the requirements	Personnel are sufficiently competent
	As well as	The developers also carry out the testing	Lack of independence can undermine the confidence gained from functional testing	Development team and testing team are independent
Tools	less	Testing tools used are of insufficient integrity	Unreliable tools may indicate a successful test when the required functionality is not implemented	Tools used are of sufficient integrity
Process description	No, less	Functional testing process is not defined or not defined clearly	Having no clearly defined process can undermine confidence in the outcome of the testing	Functional testing process is clearly defined
		Defined process is not followed	If the process is not followed, this can undermine confidence in the outcome of the testing	Defined process is followed correctly
Test Cases	other than	False positives - required output defined incorrectly	Although the tests may still pass, the required functionality is not implemented	The test cases have been correctly defined

Table 2: Example considering potential weaknesses in trustworthiness of functional test results

simple HAZOP-like guidewords to each input and resource identified in the process model. The effect these deviations may have on the output of the process (the item of evidence) are then considered. In doing this we are attempting to identify specifically those deviations which could result in the evidence not providing the required support to the safety claim. Table 2 shows how the guidewords may be applied and interpreted for the system functional test results. The argument considerations identified in the table can be used to structure the assurance argument relating to that item of evidence. This argument will itself require evidence relating to the evidence generation process (for example evidence from auditing activities), we shall refer to such evidence as *process evidence*.

6 Addressing Assurance Deficits

Through systematically considering the three questions described above we can identify potential weaknesses or limitations in the sufficiency of any evidence being considered to support the defined safety claim. As discussed earlier, we refer to such weaknesses as *assurance deficits*. If assurance deficits are identified for the selected evidence, it becomes necessary to try to mitigate those assurance deficits. There are essentially two strategies that can be adopted to mitigate assurance deficits, either through *primary* evidence and arguments, or through *backing* evidence and arguments. Primary evidence is that which is used to directly support the safety claim. Backing argument and evidence is that which is used to provide assurance for that support (i.e. it is arguing about *how* the primary argument supports the claim). In this section we describe the use of both primary and backing evidence and arguments.

6.1 Primary Evidence and Arguments

Primary evidence and argument is that which is used to mitigate an assurance deficit by directly addressing the safety claim. There are a number of ways in which this can be achieved:

Evidence Diversity - Evidence diversity involves using multiple items of evidence of *different types* in order to support a claimed position within an argument. To be effective in mitigating the assurance deficit, any diverse evidence provided must extend the capability beyond that provided by the other evidence used to support the claim. This requires understanding of the particular limitations of the different types of diverse evidence being used as discussed in section 3.

Changes to the Argument - It may be possible to address an assurance deficit by making changes to the existing argument. By changing the safety claim to be supported, the capability of an item of evidence to support the claim may then be sufficient. If a claim in the argument is changed, the knock-on effect of that change on the rest of the argument must be considered. In particular the higher level claims in the safety argument may no longer by sufficiently supported by the new claim. In practice, it is likely that such changes would be limited to redefining the basis on which the claim is stated through changes to context and assumptions.

6.2 Backing Evidence and Arguments

Sometimes the sufficiency of a particular item of evidence to support the safety claim can be unclear and hard to determine. In such cases, it may be possible to clarify the sufficiency of

the evidence by providing backing argument and evidence. There are a number of ways in which this can be achieved:

Develop a Backing Argument - An example of when a backing argument may be used is in order to demonstrate the relevance of an item of evidence to the claim being supported. The relevance of the evidence may for example be open to question because the data used for analysis was not actual system data. It may be possible in such a situation to provide a backing argument to explain why the data used was sufficiently representative of the real system data.
Another example would be to explain why an assumption is valid. For example, if an item of software hazard analysis evidence is only valid if an assumption about the partitioning properties of the system holds, a backing argument could be used to explain how the architecture of the system guarantees those properties.
Provide Additional Process Evidence – This evidence can be used to provide additional confidence in the trustworthiness of the item of evidence, or to further clarify the role of the evidence in supporting the safety claim.

7 Systematic Evidence Selection Process

Based on the guidance we have discussed in this report, we have been able to define a systematic process for selecting and justifying sufficient evidence to support a software safety argument. The process is represented in figure 2. The process is based around consideration of the three simple questions described in this report. The process starts by considering a claim, and the evidence being used to support that claim. If the response to any of the three main questions running down the centre of figure 2 is positive, then one proceeds to consider the next question. Where the response to a question is negative, this identifies that the evidence being selected is not sufficient, and could lead to an unacceptable assurance deficit. It is therefore then necessary to identify additional or alternative primary evidence or argument and return to the start. In many cases it may be unclear whether the selected evidence is sufficient, in which case the provision of additional backing evidence and argument may help to justify the support offered by the evidence to the claim.

The final stage in the process is to provide an explicit justification as to why the software assurance evidence provided is sufficient. We believe that by following the process described in this paper, it becomes easier to make a compelling justification. The process encourages a systematic consideration of the sufficiency of each item of evidence specifically with respect to the explicit safety claim in the argument.

8 Conclusions and Future Work

The safety assurance of software is ultimately demonstrated by the evidence that is put forward. There is a range of existing guidance on the types of evidence that may be used to demonstrate the safety of software and we have not attempted to reproduce guidance of this form. We assert the only way to determine the sufficiency of the evidence is considering its capability to address specific explicit safety assurance claims in a software safety argument. In this paper we have proposed a lightweight approach to selecting and assessing evidence. This approach has been applied as part of existing case study work on software safety case construction with encouraging results. Further validation of the approach is planned.

Acknowledgements

The authors would like to thank the U.K. Ministry of Defence for their support and funding. This work is undertaken as part of the research activity within the Software Systems Engineering Initiative (SSEI), www.ssei.org.uk.

References

[1] CAA Safety Regulation Group, “CAP 670 - Air Traffic Services Safety Requirements,” *Civil Aviation Authority*, (2007).

[2] Common Criteria Management Committee. “Common criteria for information technology security evaluation – Version 3.1”, (2007)

[3] I. Habli and T. Kelly. “Achieving integrated process and product safety arguments.”, *In Proceedings of the 15th safety Critical Systems Symposium*, (2007)

[4] R. Hawkins and T. Kelly, "A Systematic Approach for Developing Software Safety Arguments", *In Proceedings of the 27th International System Safety Conference*, Huntsville, AL (2009).

[5] R. Hawkins and T. Kelly. “Software safety assurance – what is sufficient”, *In Proceedings of the 4th IET International Conference on System Safety*, (2009).

[6] IEC. “61508 - Functional Safety of Electrical / Electronic / Programmable Electronic Safety-Related Systems,” *International Electrotechnical Commision*, (1998).

[7] M. Jaffe, R. Busser, D. Daniels, H. Delseny and G. Romanski. “Progress report on some proposed upgrades to the conceptual underpinnings of DO-178B/ED-12B.”, *In Proceeding of the 3rd IET International Conference on System Safety*, (2008)

[8] M. Malhotra and K. Trivedi, “Power-hierarchy of Dependability-model Types”, *IEEE Transactions on Reliability*, (1994)

[9] C. Menon, R. Hawkins and J. McDermid, “Interim Standard of Best Practice on Software in the Context of DS 00-56 Issue 4”, SSEI-BP-000001 issue 1, *Software Systems Engineering Initiative (SSEI)*, (2009), available at www.ssei.org.uk.

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

[11] MoD. “Defence Standard 00-55 Issue 2: Requirements for Safety Related Software in Defence Equipment”, *HMSO*, (2007).

[12] RTCA. “DO178B - Software Considerations in Airborne Systems and Equipment Certification,” *Radio and Technical Commission for Aeronautics*, (1992).

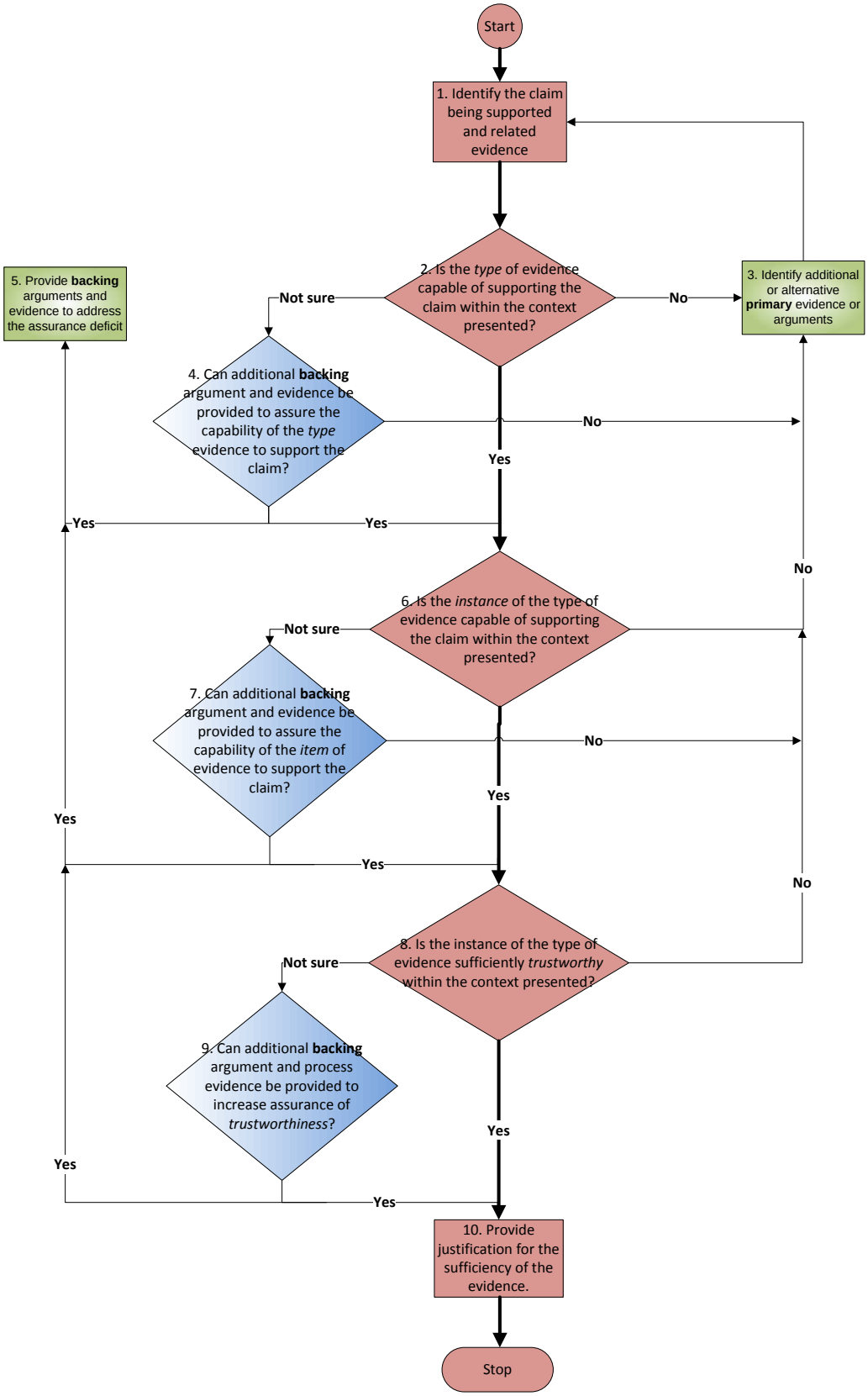

Figure 2: Evidence Selection Process