Accounting for Evidence: Managing Evidence for Goal Based Software Safety Standards

Vivien Hamilton

Viv Hamilton Associates Ltd

Wethersfield, Essex, UK

Abstract Goal-based safety standards require an evidence-based approach from suppliers and the large volume of evidence for safety assurance that is generated by a software project needs to be effectively assessed and managed. A structured safety argument needs to be created and agreed with regulators and other stakeholders early in the project lifecycle so that project processes can be designed to produce the required evidence. This safety argument needs to be abstracted in that it should define the requirements for evidence without attempting to explicitly identify the concrete evidence generated. A means of traceability between abstract requirements for evidence and concrete realization needs to be provided: an SQL database which can be hyperlinked to the argument is an efficient means of managing both the status of evidence and the traceability to the argument. The safety case is completed once the evidence has been successfully generated and assessed by an evidence report in which the assessment of limitations in evidence and counter-evidence can be effectively managed.

1 Introduction

This paper is concerned with the practical issues of managing the evidence that is created during the development of large software products and is required as part of a safety case before the software can be put into service. A number of research papers have discussed safety arguments but very few discuss the evidence specifically. This paper is principally applicable to projects working to goal-based standards such as Defence Standard 00-56 (Ministry of Defence 2007) and EC 482/2008 (European Commission 2008) for air traffic control that is implemented in SW01 in the UK (Civil Aviation Authority 2010). These standards are not prescriptive about how safety of software is achieved, but require instead a structured safety argument, supported by evidence for why the software product is safe and generated within the context of an overall safety management system. Of course large volumes of evidence are also generated when complying with other standards so some of the issues discussed in this paper may be relevant to projects using more prescriptive process based standards such as DO-178B (RTCA 1992) or IEC 61508 (IEC 2000).

Goal based standards require an evidence based approach from the supplier (Hamilton 2006) for, although the argument structures and makes sense of the evidence, it is the evidence that delivers the assurance. Ultimately the safety case must argue that there is adequate evidence to demonstrate that the product is acceptably safe.

On a small project, the volume of evidence produced is more tractable, but when the effort for development and verification of the software can be in the hundreds of man-years, the evidence generated by these processes will be correspondingly large, and the effort needed to ensure that evidence is properly assessed could be excessive.

In the context of a regulatory system that requires that the risk must be ALARP (As Low As Reasonably Practicable), there is an implied corollary, that the effort to achieve assurance needs to be applied as efficiently as practicable, since if it is inefficient, or includes nugatory effort, that wasted effort could potentially have been applied elsewhere to achieve further reduction in risk. Effective management of the evidence is crucial to achieving this, as well as potentially delivering savings to the project and ensuring that regulatory approval can be achieved in a timely manner.

2 Managing the argument

A safety argument is a complex piece of analysis which has to be scrutinized and checked for validity through manual scrutiny. Although the introduction of structuring notations, such as *Goal Structuring Notation* (GSN) (Kelly 1999) and *Claims Argument Evidence* (CAE) (Adelard 1998), enables a more rigorous analysis of the argument, verification of the correctness of the argument still requires human review. The argument therefore needs to be presented in a way that promotes readability. An argument that is cluttered with detail is difficult to comprehend and incurs the risk that it is considered in a piecemeal way such that interactions across different parts of the argument are not noted. In managing arguments and evidence, safety engineering needs to learn lessons from software engineering and adopt the principles of abstraction and of separating the requirements from the implementation: the argument needs to define the requirements for different types of evidence abstracting away from the details of the realization of the evidence. This approach enables the safety argument to be produced and agreed at an early point in the lifecycle, but in order to complete the safety case, an evidence report that traces back to the argument will need to be produced at the end of the assurance process.

An additional problem for gaining agreement on the safety argument is that typically consensus is required amongst a large number of stakeholders, including regulatory authority, operating authority, system integrator, software producer and independent safety assessor. Moreover the safety argument is a critical document for both safety and project risk, so it is likely, and desirable, especially given the focus in this paper on large software projects, that it is approved at a senior level in each of the organizations involved. It is highly desirable for the argument to be agreed at an early point in the lifecycle and preferably as part of the planning stage alongside the production of the safety management plan and technical plans, such as software development and verification plans: any later and it will be more difficult for the needs of safety assurance to influence the development and verification activities. By making a concise argument that is restricted to the requirements of the evidence to be produced and avoids implementation detail it should be easier to achieve this consensus at an early stage. Moreover, the requirements from the evidence can then be stated as objectives in the software development and verification processes and the concrete detail of how the evidence will be generated and presented can then emerge from the design of the software processes. This is far more efficient than having a safety team dictate the detailed form of the evidence to the software engineers which may result in unnecessary changes being made to the established and mature processes of the software supplier.

The large number of stakeholders who are generally required to agree the safety argument means that it is likely to be costly and time-consuming to update it. Changes to the safety argument may be unavoidable, for example if new safety requirements or safety constraints emerge in the course of system development, if significant counter-evidence is uncovered or if the intended design and implementation processes undergo significant change. A good safety argument however needs to be written in a way which means that minor changes can be absorbed within the detail of the delivery of evidence, so that only such significant changes require corresponding changes to the argument. The safety argument will need to make reference to the safety case evidence report and since, as described later in this paper, the safety case evidence report addresses limitations in the evidence and counter-evidence, this reference to the evidence report provides a further example of abstraction of unnecessary detail, making the safety argument robust to minor inconsistencies in the processes followed and evidence generated.

Given that the argument should only address the abstract requirements for evidence, a means of tracing from those requirements to the concrete realization of evidence will be needed. A simple method is for the argument to identify placeholders in the configuration management system: these placeholders can later be filled either with a single document or a hierarchy of folders and evidence. However, a more satisfactory approach is to use a relational database to link nodes of the argument to items of evidence. The advantage of a database is that it can hold additional metadata: e.g. the person responsible for producing the evidence, date created, its status in terms of completeness and also in terms of assessment against the argument. The Object Management Group (OMG) has recently generated a standardized schema for safety arguments as the Software Assurance Evidence Metamodel (SAEM) (Object Management Group 2010). A database that is accessible through hyperlinks, e.g. implemented in SQL, can also enable hyperlinks to the database entries for the evidence to be embedded in the argument.

3 Managing the processes that create evidence

As already discussed, the safety argument needs to be agreed at the earliest practicable stage of the project. In order to understand what needs to be argued, significant systems engineering and safety assessment will need to be undertaken before the safety argument can be created, but it is desirable for the argument to be in place at the planning stage before software development. This ensures that the processes that generate the largest volumes of evidence (detailed software specification; software development; software verification; system verification and system validation) can be designed to produce evidence that is suitable to provide assurance.

The abstract requirements for evidence as stated in the safety argument should be explicitly identified in the documents that define the processes that will generate that evidence: e.g. references in the safety argument to the identification and assessment of hazards will presumably be addressed by safety engineering processes; reference to demonstration of software behavior will presumably be addressed by software testing or analysis processes. By stating the requirements for evidence as objectives of the process, the process can be designed to generate evidence of the required form and verification of the acceptability of the evidence can be made a normal part of the verification of the process. This also has the added benefit that every member of the project can see how their work contributes to producing a safe (and assured) product.

A previous paper (Hamilton 2006) discussed how there are two types of properties of evidence: those that can be objectively assessed and those that are subject to (expert) judgment. Processes can often be defined such that many of the objective properties of the evidence are automatically achieved. For example, in software engineering, the development environment can be configured to ensure that correctly named objects are stored in the configuration management system with the appropriate meta-data fields completed. An example from safety engineering could be a tool to support HAZOP that ensures that entries are recorded for all of a defined set of guidewords. The properties that are subject to judgment cannot be automated in this way, for example the software engineering object will need to be reviewed to ensure that it is appropriately designed and the HAZOP will need to be reviewed to ensure that all the entries are correct and meaningful; however it may still be possible to provide tool support to provide an audit trail, for example to enforce that a person of a defined role completes a review field. From the perspective of good safety engineering, evidence, and the assessment of evidence, should be as objective as possible. It is not possible for expert judgment to be removed entirely but it should be relied on only for those properties of evidence that are necessarily subject to judgment. An argument that uses subjective judgment for objective properties that could have been objectively assessed may appear 'handwaving' and unconvincing. To ensure that these objective properties can be objectively assessed, processes need to be designed to systematically produce the evidence and verify its properties.

In the CAE notation for safety cases, evidence is linked to claims through arguments that explicitly justify why the evidence is adequate to discharge the claim. GSN does not have such an explicit argument: it does have an optional justification notation but in general the goals should be decomposed sufficiently that the relationship between the evidence and goal is obvious. In either case thought needs to be given as to why the evidence is adequate to fulfil the safety claims, why this approach has been selected and how possible weaknesses have been addressed. These are also the questions that need to be considered when planning the processes to be adopted in the project, so the document that provides the process definition is also the obvious place to record the justification for the approach, bearing in mind from the previous discussion that the process document should explicitly state, as objectives of the process, the requirements for evidence as defined in the safety argument. So in the planning stages of the project, the safety engineer should be actively in discussion with each of the other technical experts in the project, covering software design, coding, testing etc. to ensure that each process definition meets the needs of safety assurance in addition to the other technical goals of the project.

There is one further area in which the project processes can be optimized to deliver evidence in a form suitable for the safety case and that is in the final form of documentation. Each project process on completion should produce some form of summary document that confirms successful completion of the process, achievement of the objectives of the process (including the objectives to generate the safety assurance evidence) and also makes explicit reference to the body of evidence generated by that process. Of course most projects already do this for testing, but it is not necessarily the case that such a summary report exists for specification, design and coding processes. On a small project this may not be such an issue: if, for example, the entire specification can be encompassed in a single document, then the final issue of the document implicitly provides such a document. On a large project however the specification is likely to be spread over multiple documents and whichever is the latest such document to be issued or updated is relevant only to one area of the software and says nothing about the specification process as a whole. Imposing an additional requirement for such a summary document provides a convenient means of simplifying traceability from the argument to the body of evidence. It also has the benefit that formal documents normally have a defined process of document review and approval with signatories so a formal document provides a means of generating evidence that the appropriate persons, who were both competent and accountable for the delivery of a part of the evidence, have confirmed that the evidence is correct.

Figure 1 illustrates how the reference to evidence, using summary documents and process definitions that justify the process, might appear in a safety argument constructed using GSN.

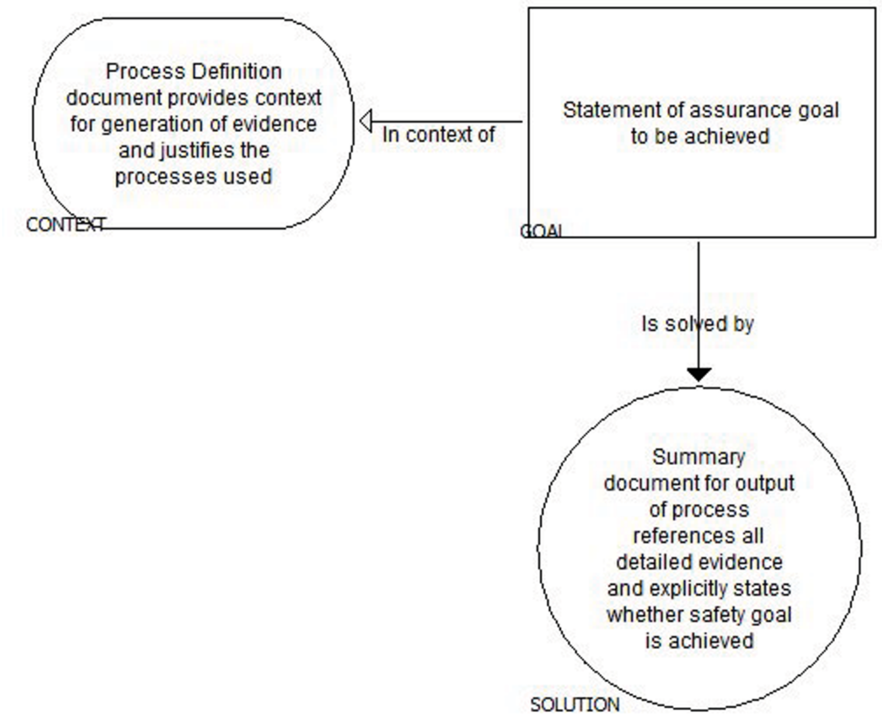
Process Definition document provides context for generation of evidence and justifies the processes used
CONTEXT
In context of
Statement of assurance goal to be achieved
GOAL
Is solved by
Summary document for output of process references all detailed evidence and explicitly states whether safety goal is achieved
SOLUTION

Fig. 1. Example of argument using GSN relating safety goals to process definitions and summary evidence documents

4 Assessing the evidence

4.1 Overall process of assessment

The process for assessment should ensure that the assessment is as objective as possible and that any anomalies, limitations, assurance deficits and counter-evidence are dealt with transparently and at an appropriate stage of the assessment. Software engineering uses a V model to explore the relationship between requirements and implementation and between test evidence produced at progressive stages of integration, and a similar model can illustrate how evidence should be progressively assessed.

Figure 2 illustrates, on the left hand side, how the requirements for evidence are refined from the safety argument through the definitions of the processes that generate the evidence. The right hand side illustrates the progressive assessment from individual items of evidence in their raw state, to an overall judgment of the acceptability (or otherwise) of the safety case. Each individual item of evidence is first verified against the exit criteria of the process to produce pass/fail results, e.g. was a test result within the defined tolerances, did a code file pass static analysis checks, did a design document successfully pass its review process? For most of these the assessments are being made against objective criteria defined in the process definition; the verification is a normal part of the process and the involvement of safety experts should not be necessary.

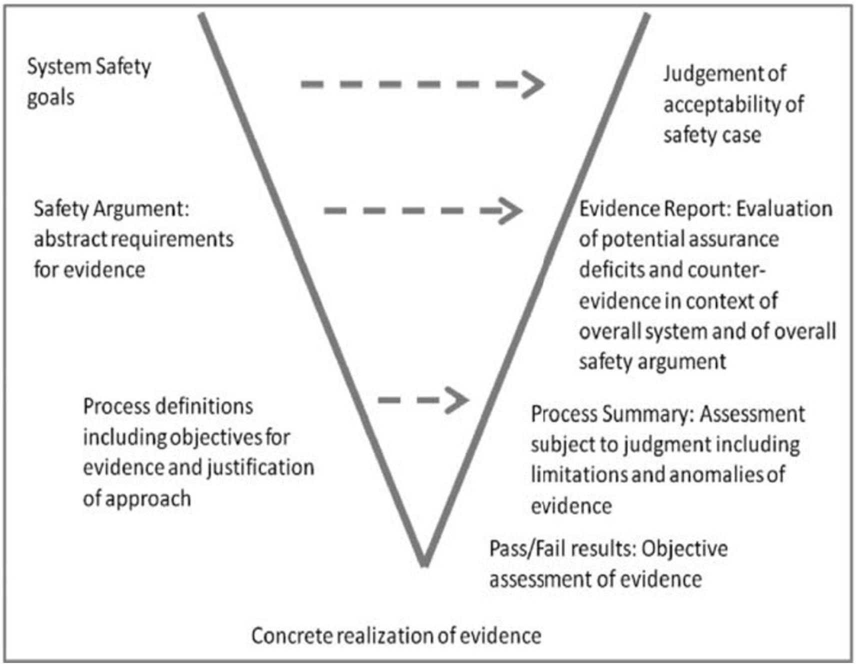
System Safety goals
Judgement of acceptability of safety case
Safety Argument: abstract requirements for evidence
Evidence Report: Evaluation of potential assurance deficits and counter-evidence in context of overall system and of overall safety argument
Process definitions including objectives for evidence and justification of approach
Process Summary: Assessment subject to judgment including limitations and anomalies of evidence
Pass/Fail results: Objective assessment of evidence
Concrete realization of evidence

Fig. 2. V diagram showing relationship between objectives for evidence and assessment stages

The process summary document then identifies the extent to which the outputs of that process have met the safety objectives. Did all of the tests pass? If not, what was the pass rate and where are the failures recorded and analyzed? Were all of the tests carried out exactly as specified? If not what were the deviations and what are the consequences for the safety objectives? Were all of the tests completed? If not where is it stated which parts of the software are not tested? Since this part of the assessment of evidence is concerned with process compliance, additional backing evidence for the safety case can be provided through quality auditing to check that processes are being properly carried out.

Both Defence Standard 00-56 and EC 482/2008 require the safety argument to be produced within the context of a safety management system. Objective evidence of the correct operation of the safety management system and also project quality management system should also be recorded and summarized and audits carried out to confirm the effectiveness of these management systems. Assuming that the quality management system contains methods of managing deviations to processes and that the safety management system contains systems for managing potential safety issues in the product or the evidence, then these too are processes that need to be summarized, with any observed limitations recorded.

The title of this paper is *accounting for evidence* since this reflects that what should occur is systematic recording and objective assessment. Moreover all of the limitations with evidence should be recorded so that the overall presentation of the evidence shows transparently exactly what has been achieved, and what still remains unproven. As the processes continue up the right hand side of the V model, more safety judgment is required in assessing the impact on the safety case of the individual faults and limitations uncovered in the detailed evidence.

4.2 Limitations, counter-evidence and assurance deficits

Life is not perfect and neither is evidence. Software will contain faults; tests will fail; documentation will contain errors and audits will find deviations in processes. The problem for assessment of the safety case is how much of this actually matters to safety.

In this paper the word *limitation* has been used as a general term to describe any difference from the ideal state of the evidence. *Counter-evidence* is defined in Defence Standard 00-56 as evidence with the *potential* to undermine safety claims. As Defence Standard 00-56 requires a pro-active search for counter-evidence, a limitation needs to be considered as possible counter-evidence unless or until it can be shown that the safety claims are not undermined by that limitation. Suppose for example that a test has failed, and as a result a fault has been found in the software, this limitation (in the correctness of the software) might be counter-evidence. On the other hand if the fault is in some functionality that is not safety related, then it is likely that, from a safety perspective, the existence of the fault is acceptable, and so this limitation is not counter-evidence. Any member of the project who is competent in a particular process area could record limitations and assess their impact in relation to the scope of that process. However, counter-evidence is wholly related to the safety of the product and must be assessed by a competent safety professional. Hence it is important that limitations are accurately and transparently recorded in the evidence generated by all of the project processes and identified in the summary process documents, so that they can be assessed by a safety engineer.

What if a problem with a test meant that instead of identifying a fault in the software, the test simply could not be executed? Here the problem is not that the software is known to be incorrect: the problem is that because the test could not be executed there is no evidence that the software is correct. The term *assurance deficit* (Menon et al. 2010) is used where such uncertainty exists. However, as with counter-evidence, a limitation in evidence is not automatically an assurance deficit when considered in the overall context of the safety case. Suppose the software verification was being done in an automated test environment in which some test could not be executed, so that there is a limitation in the extent of the software verification, but it is possible to execute a test that achieves the same test objectives in the overall system environment: in this case a limitation in software testing is addressed through system testing so there is no assurance deficit. Assurance deficits are concerned with the overall uncertainty in the safety case, so as with counter-evidence, assessment of whether or not limitations constitute assurance deficits should be determined by a competent safety professional.

By freeing the safety engineers from having to scrutinize a large amount of low level raw evidence, they should be able to better concentrate on the issue of safety on the overall context. In particular, as required by Defence Standard 00-56, the search for counter-evidence should be pro-active and look beyond the immediate project. Additionally, by seeing the whole picture presented by the evidence, the safety engineers may be in a better position to identify the 'counter-intuitive' possibilities identified by Littlewood and Wright (Littlewood and Wright 2007) in which being more successful in generating evidence can occasionally reduce confidence overall, an example of which is that if the entire test programme fails to find any faults, it may be suspected that the test system is ineffective.

The safety management system should have specific objectives to manage the identification, assessment and mitigation of assurance deficits and counter-evidence. The output of that activity will be recorded in the safety case evidence report.

4.3 Safety case evidence report

The safety case evidence report is the final deliverable of the assurance process. Since the safety argument should be designed to be abstract, it is possible for a single version of a safety argument to cover multiple software installations, or multiple deliveries of a phased-delivery project. However, the safety case evidence report will need to address a specific software version and a specific installation. The safety case evidence report will identify the unique set of relevant evidence for the safety case. It will need to refer to the baseline of the software (which may be addressed by standard documentation such as a release note) and it will need to relate this to the appropriate set of safety evidence, including system level evidence. Each installation may have unique evidence: specific site constraints; a unique history of field experience perhaps using previous versions of the software, and different experiences of commissioning.

The safety case evidence report is the output of the process of assessment of limitations, assurance deficits and counter-evidence. It provides the summary of the assessment process and the resultant identification of assurance deficits and counter-evidence in the context of the overall safety case. It will also confirm the successful operation of the safety management system.

Since it assesses the complete set of evidence it is able to evaluate which parts of the argument are supported by evidence that is compelling and which parts are less convincing and therefore where the residual risk lies. Although an extension to the GSN notation exists to identify the relative contribution of different parts of the argument, it does not appear to be used in practice. This is possibly because it adds notational complexity to what are already complex pieces of analysis. The safety case evidence report would appear to be a more suitable place in which to note the relative weights of different parts of the argument, since this can be stated in the context of the actual evidence delivered. As the final output of the safety assurance process, the safety case evidence report is issued to those who need to make the decision to accept or reject the safety case.

5 Conclusion

This paper is not a theoretical study but instead attempts to give pragmatic guidance on the issues of managing a large body of safety assurance evidence, since for large software projects any inefficiencies in this area will be at best costly and at worst lead to a weak or incorrect safety case or severe delays to delivery into service of the product.

It is the responsibility of the project manager to deliver a product that is both safe and seen to be safe through the body of safety assurance evidence. One of the goals embedded in the approach in this paper is to establish a culture such that all members of the project team are accountable for delivery of safety assurance evidence in their area of responsibility. In such a culture, the safety engineer works not in isolation retrospectively reviewing and judging project artifacts, but as a partner working in dialogue with other technical experts to establish processes that enable the project as a whole to deliver the safety assurance in an efficient manner.

The approach has drawn on three principles learned from software engineering: information hiding; separating requirements from concrete realization, and progressive integration. The safety argument should address the abstract requirements for evidence, deferring unnecessary detail on the realization of evidence to a safety case evidence report. A means of traceability is needed between the nodes of the argument and the eventual evidence, perhaps using a relational database which can also record the status of evidence. Finally the evidence needs to be assessed as it is generated, working progressively from detailed anomalies specific to the type of evidence, to understand how this might contribute potential assurance deficits or counter-evidence, and then sentencing this information in the overall system context to evaluate whether or not the evidence delivers a satisfactory safety case.

The process of accounting for evidence includes the systematic recording and objective assessment of evidence that generates, through the exercise of appropriate judgment, an accurate overall picture in which any problems with evidence are transparently stated in order that the safety case shows exactly what has been achieved, and what still remains unproven.

Acknowledgments The author gratefully acknowledges the discussions on safety arguments and insight into the SAEM provided by Dr Tim Kelly, University of York.

References

Adelard (1998) ASCAD – Adelard safety case development manual

Civil Aviation Authority (2010) CAP 670 ATS safety requirements

European Commission (2008) Commission Regulation (EC) No 482/2008 Establishing a software safety assurance system to be implemented by air navigation service providers and amending annex II to regulation (EC) No 2096/2005 http://www.caa.co.uk/docs/952/SESESARR%28482-2008%29.pdf. Accessed 12 September 2010

Hamilton V (2006) Criteria for safety evidence – goal-based standards require evidence based approaches. Safety Systems 16:1 September 2006. http://www.vivhamilton.co.uk/Papers/SCEvCriteria.pdf. Accessed 12 September 2010

IEC (2000) ISO/IEC 61508 Functional safety of electrical/electronic/programmable electronic safety related systems, Parts 1 to 7. International Electrotechnical Commission

Kelly T (1999) Arguing safety – a systematic approach to safety case management. PhD thesis, University of York YCST99/05

Littlewood B, Wright D (2007) The use of multi-legged arguments to increase confidence in safety claims for software-based systems: a study based on a BBN of an idealized example. IEEE Trans Softw Eng 33:347-365

Menon C, Hawkins R, McDermid J, Kelly T (2010) An overview of the SOBP for software in the context of DS 00-56 issue 4. In: Dale C, Anderson T (eds) Making systems safer. Springer-Verlag, London

Ministry of Defence (2007) Defence Standard 00-56 Issue 4: Safety management requirements for defence systems

Object Management Group (2010) Software assurance evidence metamodel (SAEM) Sysa/10-02-01 http://www.omg.org/cgi-bin/doc?sysa/10-02-01. Accessed 23 March 2010

RTCA (1992) RTCA/DO-178B: Software considerations in airborne systems an equipment certification. RTCA