Process and Product Certification Arguments – Getting the Balance Right

Ibrahim Habli Tim Kelly

Rolls-Royce University Technology Centre
Department of Computer Science
University of York, UK
{ibrahim.habli, tim.kelly}@cs.york.ac.uk

Abstract. Many current safety certification standards are process-based, i.e. they prescribe a set of development techniques and methods. This is perhaps best exemplified by the use of Safety Integrity Levels (SILs), e.g. as defined by IEC 61508 and UK Defence Standard 00-55. SILs are defined according to the level of the risk posed by a system, and hence prescribe the tools, techniques and methods that should be adopted by the development and assessment lifecycle. Product-based certification relies on the generation and assurance of product-specific evidence that meets safety requirements derived from hazard analysis. This evidence can be used as the argument basis in a safety case. However, uncertainty about the provenance of evidence in such a safety case can undermine confidence. To address this problem, we argue that process arguments remain an essential element of any safety case. However, unlike the sweeping process-based integrity arguments of the past, we suggest instead that highly directed process arguments should be linked to the items of evidence used in the product case. Such arguments can address issues of tool integrity, competency of personnel, and configuration management. Much as deductive software safety arguments are desirable, there will always be inductive elements. Process-based arguments of the type we suggest address partly this problem by tackling the otherwise implicit assumptions underlying certification evidence.

1 Introduction

Many software safety standards recommend or mandate a set of practices (tools, techniques and methods) that should be used in the development and assessment lifecycle. Such practices are often categorised according to Safety Integrity Levels (SILs) [1,2] or Development Assurance Levels (DALs) [3] that correspond to the degree of risk reduction expected from the system. However, the fundamental limitation of process-based certification lies in the observation that good tools, techniques and methods do not necessarily lead to the achievement of a specific level of integrity of a SIL (e.g. as defined by a target failure rate). It is infeasible to justify the correlation between the prescribed techniques and the failure rate considered by many to be defined by a SIL [4].

Recently, there has been a marked shift towards safety standards that recommend or mandate well structured and reasoned software safety arguments. Such arguments justify the acceptability of software safety based on product-specific and targeted evidence. Product-based arguments rely on the generation and assurance of product-specific evidence demonstrating the satisfaction of safety requirements derived from hazard analysis. This approach is typically referred to as evidence-based certification.

Nevertheless, evidence-based certification is based on the provision of both *product* and *process* evidence. Safety arguments, along with their evidence, are typically presented in a safety case. A safety case is defined in UK Defence Standard 00-56 as [5]:

> *"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 operating environment"*

The role of the process should not be underestimated even in product-based arguments. The process evidence should be focused and have well-defined relationship to product integrity (and the assurance of product integrity). Just as product evidence must be targeted towards specifications, so should process evidence [15]. Uncertainty about the provenance of evidence in product arguments can undermine confidence. Hence, the balance between process-based and product-based arguments should be managed carefully. In this paper, we propose that process arguments should be linked to the items of evidence used in the product-based safety case. Such arguments can address issues of tool integrity, competency of personnel, and configuration management. Process-based arguments of the type we suggest tackle the implicit process assumptions underlying the product-based argument and evidence.

The rest of this paper is as follows. Section 2 presents the underlying elements of a safety argument. Section 3 introduces the Goal Structuring Notation (GSN) – a structured technique for representing safety arguments. Then, a product-based software safety argument is presented and analysed in Section 4. In Section 5, this argument is then extended with process evidence that justifies the trustworthiness of the product evidence. The process is discussed and evaluated in Section 6. Finally, conclusions are presented in Section 7.

2 Safety Arguments

Underlying the descriptions of the safety case given in the previous section is a view of the safety case consisting of three principal elements: Requirements, Argument and Evidence. The relationship between these three elements is depicted in Figure 1.

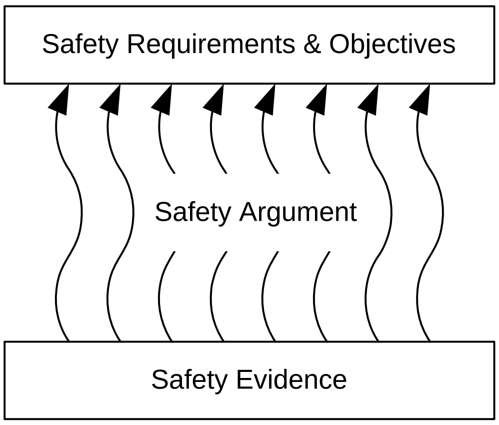


Figure 1. The Role of Safety Argumentation

The safety argument is that which communicates the relationship between the evidence and objectives. However, a commonly observed failing of safety cases is that the *role of the safety argument is often neglected*. In such safety cases, many pages of supporting evidence are often presented (e.g. hundreds of pages of fault trees or Failure Modes and Effects Analysis tables), but little is done to explain how this evidence relates to the safety objectives. The reader is often left to guess at an unwritten and implicit argument.

Both argument and evidence are crucial elements of the safety case that must go hand-in-hand. Argument without supporting evidence is unfounded, and therefore unconvincing. Evidence without argument is unexplained – it can be unclear that (or how) safety objectives have been satisfied.

A valid and sound deductive argument is the strongest form of argument. Validity indicates that the argument conclusion follows from its premises. In other words, true premises of a valid argument provide conclusive basis for the truth of the conclusion [6]. On the other hand, soundness implies that the argument premises are true. Although deductive arguments are desirable, software safety argument premises comprise, for the most part, subjective elements. Such subjective elements make it infeasible to declare that the premises are true. For example, even when formal proof principles are used to create deductive arguments, subjectivity may exist in the scoping and abstraction of the formal model. Therefore, software safety arguments are predominantly inductive. The premises of an inductive argument offer support for the conclusion. However, the support of these premises, even if they are true, is insufficient to attain absolute certainty. Therefore, the weakness or strength of an inductive argument depends on the level of confidence in the support of the premises for the conclusion. The stronger the evidence provided in the premises, the more confidence is gained in the argument. Hence, rather being either true or false, the evidence in software safety arguments is examined against the degree of confidence or assurance it provides.

This degree of confidence in a software safety argument can be defined with respect to the evidence's relevance, trustworthiness, and independence [7]. The assurance of the development process plays a key role weakening or strengthening the software safety argument. Uncertainty about the provenance of the software evidence, such as testing and analysis, can undermine confidence, and consequently weaken the evidence's relevance, trustworthiness, and independence. For example, a software safety argument, supported by direct evidence from formal methods, may be weakened by detecting flaws in the underlying proof attributable to a lack of training in discrete mathematics. Regardless of the type of argument (product or process), a clear and unambiguous representation of the argument is a prerequisite for communicating and analysing the strength and weakness of the argument. The next section presents a structured technique for representing safety arguments clearly and systematically.

3 Representing Safety Argument using the Goal Structuring Notation (GSN)

Safety arguments are most typically communicated in existing safety cases through free text. However not all engineers responsible for producing safety cases write clear and well-structured English. Consequently, the meaning of the text, and therefore the structure of the safety argument, can be ambiguous and unclear. Structured argumentation techniques can overcome the limitations of free text arguments. The Goal Structuring Notation (GSN) [8] - a graphical argumentation notation - explicitly represents the individual elements of any safety argument (requirements, claims, evidence and context) and (perhaps more significantly) the relationships that exist between these elements (i.e. how individual requirements are supported by specific claims, how claims are supported by evidence and the assumed context that is defined for the argument). The principal symbols of the notation are shown in Figure 2 (with example instances of each concept).

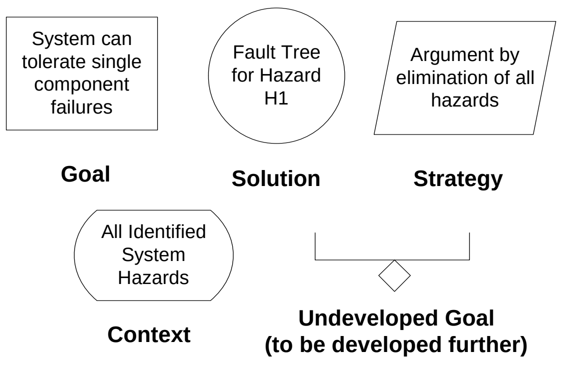

System can tolerate single component failures
Fault Tree for Hazard H1
Argument by elimination of all hazards
Goal
Solution
Strategy
All Identified System Hazards
Context
Undeveloped Goal (to be developed further)

Figure 2. Principal Elements of the Goal Structuring Notation

When the elements of the GSN are linked together in a network they are described as a 'goal structure'. The principal purpose of any goal structure is to show how goals (claims about the system) are successively broken down into sub-goals until a point is reached where claims can be supported by direct reference to available evidence (solutions). As part of this decomposition, using the GSN it is also possible to make clear the argument strategies adopted (e.g. adopting a quantitative or qualitative approach), the rationale for the approach and the context in which goals are stated (e.g. the system scope or the assumed operational role).

Within Europe, GSN has been adopted by a growing number of companies within safety-critical industries (such as aerospace, railways and defence) for the presentation of safety arguments within safety cases. The following list includes some of the applications of GSN to date:

- Eurofighter Aircraft Avionics Safety Justification
- Hawk Aircraft Safety Justification
- U.K. Ministry of Defence Site Safety Justifications
- U.K. Dorset Coast Railway Re-signalling Safety Justification
- Submarine Propulsion Safety Justifications
- Safety Justification of UK Military Air Traffic Management Systems
- London Underground Jubilee Line Extension Safety Justification
- Swedish Air Traffic Control Applications
- Rolls-Royce Trent Engine Control Systems Safety Arguments

The key benefit experienced by those companies adopting GSN is that it improves the comprehension of the safety argument amongst all of the key project stakeholders (i.e.

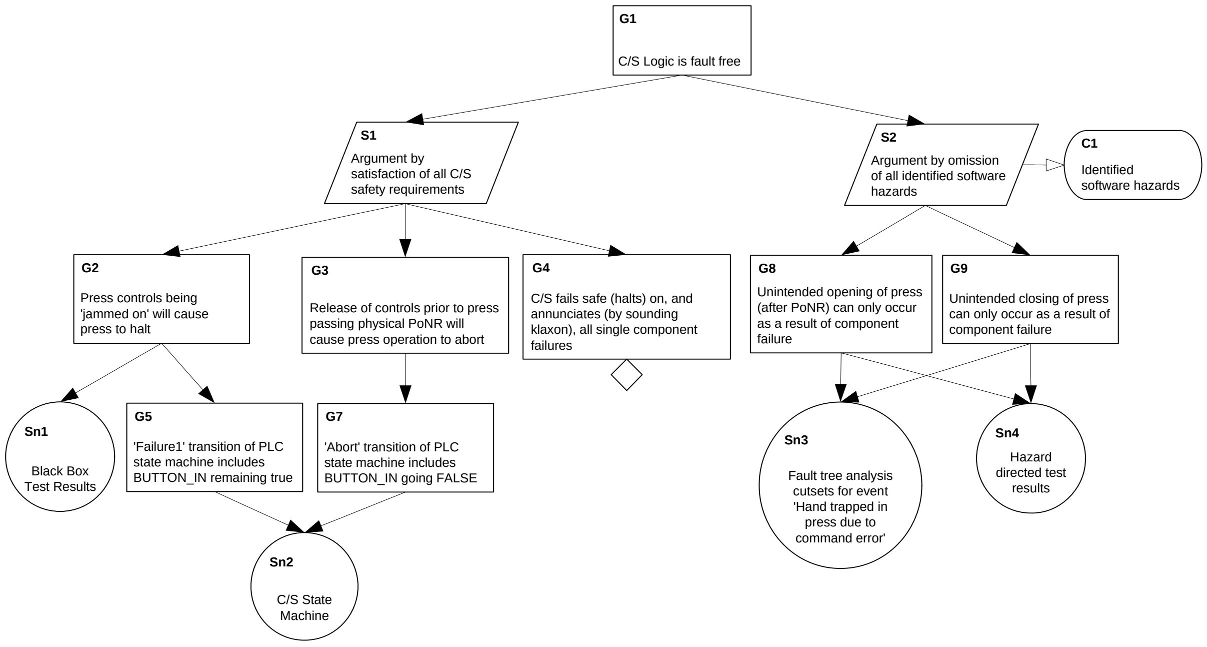
system developers, safety engineers, independent assessors and certification authorities). In turn, this has improved the quality of the debate and discussion amongst the stakeholders and has reduced the time taken to reach agreement on the argument approaches being adopted.

4 An Example Software Product Argument

Rather than arguing software safety based on the compliance with prescribed methods and techniques, software product-based arguments justify the acceptability of software safety based on product-specific evidence. Product-based arguments rely on the generation and assurance of product-specific evidence that meets safety requirements derived from hazard analysis. Product evidence plays a key role in evidence-based safety certification. McDermid outlines the concepts of evidence-based safety certification as follows [9]:

> *"First, identify the potential failure modes of software which can give rise to, or contribute to, hazards in the system context. Second, provide evidence that these failure modes:*
>
> – *Cannot occur, or*
> – *Are acceptably unlikely to occur, or*
> – *Are detected and mitigated so that their effects are acceptable."*

The evidence should be explicit and directly related to the software under analysis [10]. For example, Weaver identifies three categories of evidence that are needed for the support of a software safety argument [14]: requirements validation, requirements satisfaction and requirements traceability.



G1
C/S Logic is fault free
S1
Argument by satisfaction of all C/S safety requirements
S2
Argument by omission of all identified software hazards
C1
Identified software hazards
G2
Press controls being 'jammed on' will cause press to halt
G3
Release of controls prior to press passing physical PoNR will cause press operation to abort
G4
C/S fails safe (halts) on, and annunciates (by sounding klaxon), all single component failures
G8
Unintended opening of press (after PoNR) can only occur as a result of component failure
G9
Unintended closing of press can only occur as a result of component failure
Sn1
Black Box Test Results
G5
'Failure1' transition of PLC state machine includes BUTTON_IN remaining true
G7
'Abort' transition of PLC state machine includes BUTTON_IN going FALSE
Sn3
Fault tree analysis cutsets for event 'Hand trapped in press due to command error'
Sn4
Hazard directed test results
Sn2
C/S State Machine

Figure 3. An Example Goal Structure

Figure 3 shows an example goal structure of a product-based argument. The goal structure is for an argument for the safe operation of a sheet metal press. This operation is controlled by an operator via a simple control system based on a Programmable Logic Controller (PLC). In this structure, as in most, there exist 'top level' goals – statements that the goal structure is designed to support. In this case, "C/S (Control System) Logic is fault free", is the (singular) top level goal. Beneath the top level goal or goals, the structure is broken down into sub-goals, either directly or, as in this case, indirectly through a strategy. The two argument strategies put forward as a means of addressing the top level goal in Figure 3 are "Argument by satisfaction of all C/S (Control System) safety requirements", and, "Argument by omission of all identified software hazards". These strategies are then substantiated by five sub-goals. At some stage in a goal structure, a goal statement is put forward that need not be broken down and can be clearly supported by reference to some evidence. In this case, the goal "Press controls being 'jammed on' will cause press to halt" is supported by direct reference to the solutions, "Black Box Test Results" and "C/S State Machine".

The above argument makes it clear how the safety requirements are achieved by the software-specific evidence (solutions). Black box testing and state machine analysis provide explicit and independent evidence (solutions) directly related to the software artefact rather than appealing to the quality of the development process. The evidence provided in the argument relies on diverse solutions, hence avoiding common mode failures (e.g. required by safety standards such as the UK Defence Standard 00-55 [1]). Testing and analysis are dissimilar conceptually and mechanistically (providing the highest form of independence [14]). Conceptual diversity relies on two different underlying concepts while mechanistic diversity relies on two different applications of same underlying concepts. However, in the next section we show how the confidence in the argument gained by this seemingly independent and direct evidence can be undermined by a lack of process evidence.

5 Software Process Argument

Confidence in the evidence (i.e. the test and state machine based evidence) shown in the product argument in Figure 3 can be weakened by the uncertainty about its provenance. Firstly, black box testing (*Sn1*) is an effective verification technique for showing the achievement of safety requirement specifications. However, confidence in the black box testing depends on justifying the trustworthiness of testing process. For example, factors that need to be addressed by a process evidence include issues such as:

– the testing team is independent from the design team
– the process of generating, executing and analysing test cases is carried out systematically and thoroughly
– the traceability between safety requirements and test cases is well established and documented.

Similarly, a state machine (*Sn2*) is a powerful formal method for specification and verification. Nevertheless, process information is required to reveal the mathematical qualification of the verification engineers and their ability to demonstrate correspondence between the mathematical model and the software behaviour at runtime [11]. Mistakes can be made in formal proofs the same way that they can be made in coding. Therefore, the quality of the verification process by means of formal methods is as important as the deterministic results such methods produce.

To address the above limitation, we propose addressing process uncertainty through linking process arguments to the items of evidence used in the product safety argument. Such process arguments address issues of tool and method integrity, competency of personnel, and configuration management.

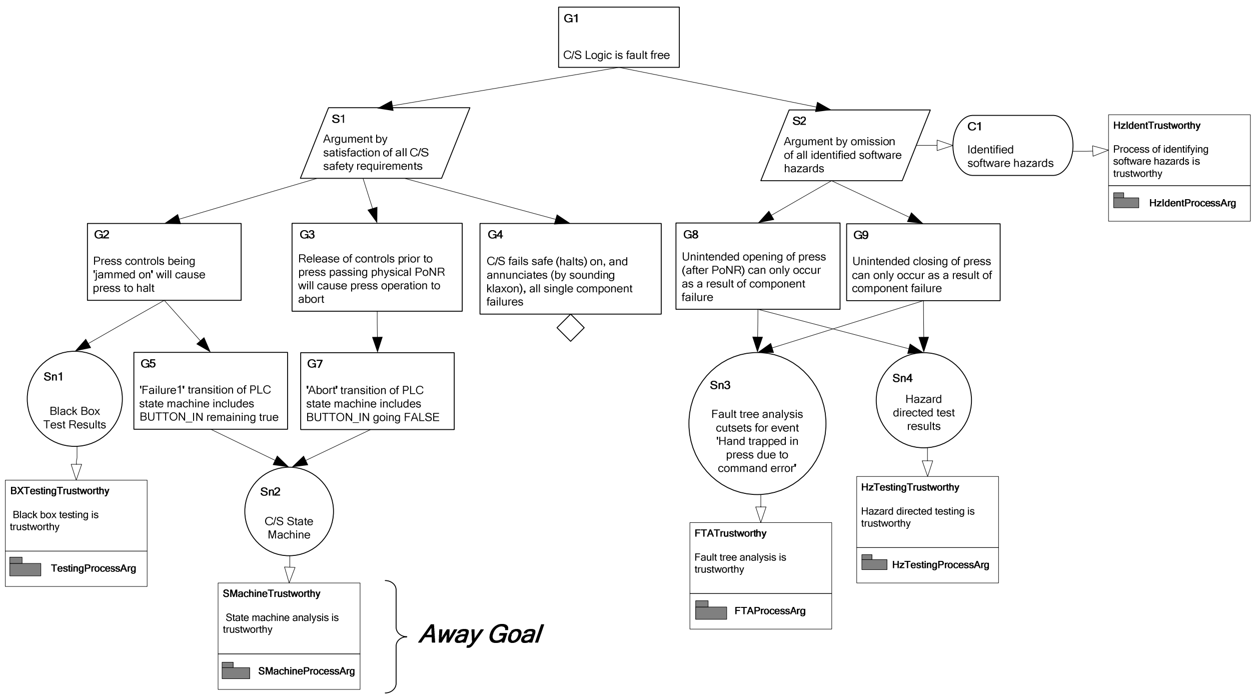

G1
C/S Logic is fault free
S1
Argument by satisfaction of all C/S safety requirements
S2
Argument by omission of all identified software hazards
C1
Identified software hazards
HzIdentTrustworthy
Process of identifying software hazards is trustworthy
HzIdentProcessArg
G2
Press controls being 'jammed on' will cause press to halt
G3
Release of controls prior to press passing physical PoNR will cause press operation to abort
G4
C/S fails safe (halts) on, and annunciates (by sounding klaxon), all single component failures
G8
Unintended opening of press (after PoNR) can only occur as a result of component failure
G9
Unintended closing of press can only occur as a result of component failure
Sn1
Black Box Test Results
G5
'Failure1' transition of PLC state machine includes BUTTON_IN remaining true
G7
'Abort' transition of PLC state machine includes BUTTON_IN going FALSE
Sn3
Fault tree analysis cutsets for event 'Hand trapped in press due to command error'
Sn4
Hazard directed test results
BXTestingTrustworthy
Black box testing is trustworthy
TestingProcessArg
Sn2
C/S State Machine
HzTestingTrustworthy
Hazard directed testing is trustworthy
HzTestingProcessArg
FTATrustworthy
Fault tree analysis is trustworthy
FTAProcessArg
SMachineTrustworthy
State machine analysis is trustworthy
SMachineProcessArg
Away Goal

Figure 4. Process Argument Extension

Figure 4 shows a modified version of the goal structure of the sheet metal press safety argument. This version uses a new extension to GSN [12] — the 'Away' Goal (*BXTestingTrustworthy, SMachineTrustworthy, FTATrustworthy, HzTestingTrustworthy, and HzIdentTrustworthy*) to attach process arguments to the solutions. Away Goals are used within the arguments to denote claims that must be supported but whose supporting arguments are located in another part of the safety case. Away Goals were developed to enable modular and compositional safety case construction, such that safety case elements may be safely composed, removed and replaced. In the rest of this section, the process arguments justifying the trustworthiness of the black box testing and the state machine analysis are presented.

Figure 5 shows the goal structure for the *BXTestingTrustworthy* away goal. Here, the argument stresses the importance of process information to justify the trustworthiness of the black box testing evidence. The process information addresses team competency, test case generation, execution and analysis, and testing traceability. Firstly, the competency of the testing team (goal: *BXTestTeam*) is supported by claims regarding the team's qualifications and independence from the design team (avoiding common mode failures with the design team). Secondly, the goal structure contains an argument that claims that the process of generating, executing, and analysing test cases is systematic (argument: *ArgBXTestCaseGen*). This argument is supported by items of evidence such as the fact that the test cases cover all defined safety requirements and executed on the final source code and target platform. Finally, the goal structure shows that the black box testing process is traceable. However, in order to avoid complicating the goal structure, the justification argument for traceability is documented elsewhere (module: *ConfigProcessArg*).

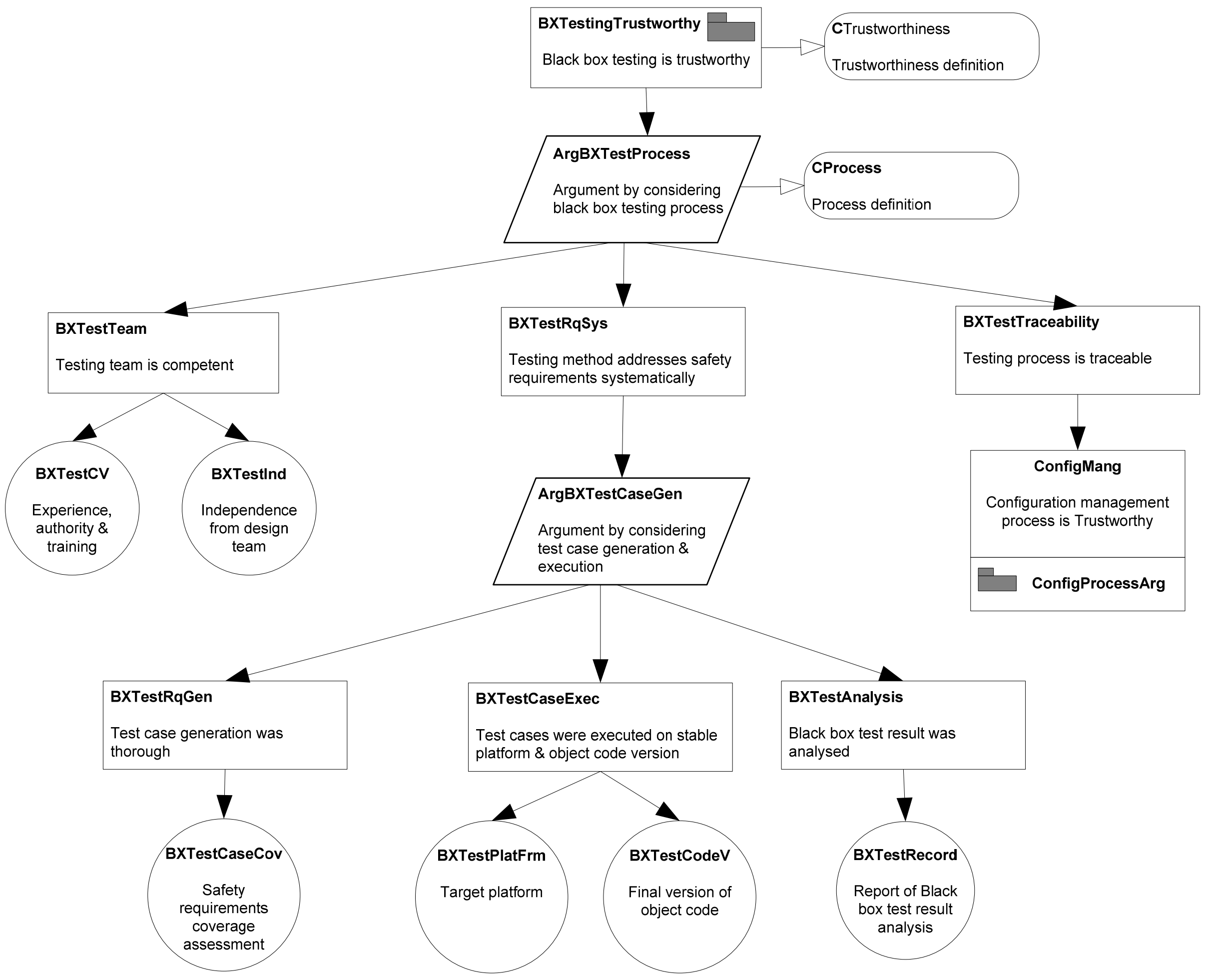

BXTestingTrustworthy
Black box testing is trustworthy
CTrustworthiness
Trustworthiness definition
ArgBXTestProcess
Argument by considering black box testing process
CProcess
Process definition
BXTestTeam
Testing team is competent
BXTestRqSys
Testing method addresses safety requirements systematically
BXTestTraceability
Testing process is traceable
BXTestCV
Experience, authority & training
BXTestInd
Independence from design team
ArgBXTestCaseGen
Argument by considering test case generation & execution
ConfigMang
Configuration management process is Trustworthy
ConfigProcessArg
BXTestRqGen
Test case generation was thorough
BXTestCaseExec
Test cases were executed on stable platform & object code version
BXTestAnalysis
Black box test result was analysed
BXTestCaseCov
Safety requirements coverage assessment
BXTestPlatFrm
Target platform
BXTestCodeV
Final version of object code
BXTestRecord
Report of Black box test result analysis

Figure 5. Black Box Process Argument

Similarly, the goal structure depicted in Figure 6 justifies the trustworthiness of the state machine analysis by referring to process information. In addition to the consideration of staff competency and process traceability, this goal structure depends on the state machine's tool and notations. The dependability of the state machine tool is verified against the tool's operational requirements (solution: *SMachineToolRq*). A formal approach such as state machine analysis requires a simple and unambiguous representation. This facilitates the definition of correct correspondence between the formal model and the actual software artefact. This claim about correct representation is supported by referring to the adequacy of the notations and the clarity of the accompanying natural language (solutions: *SMachineNotation* and *SMachineLang*).

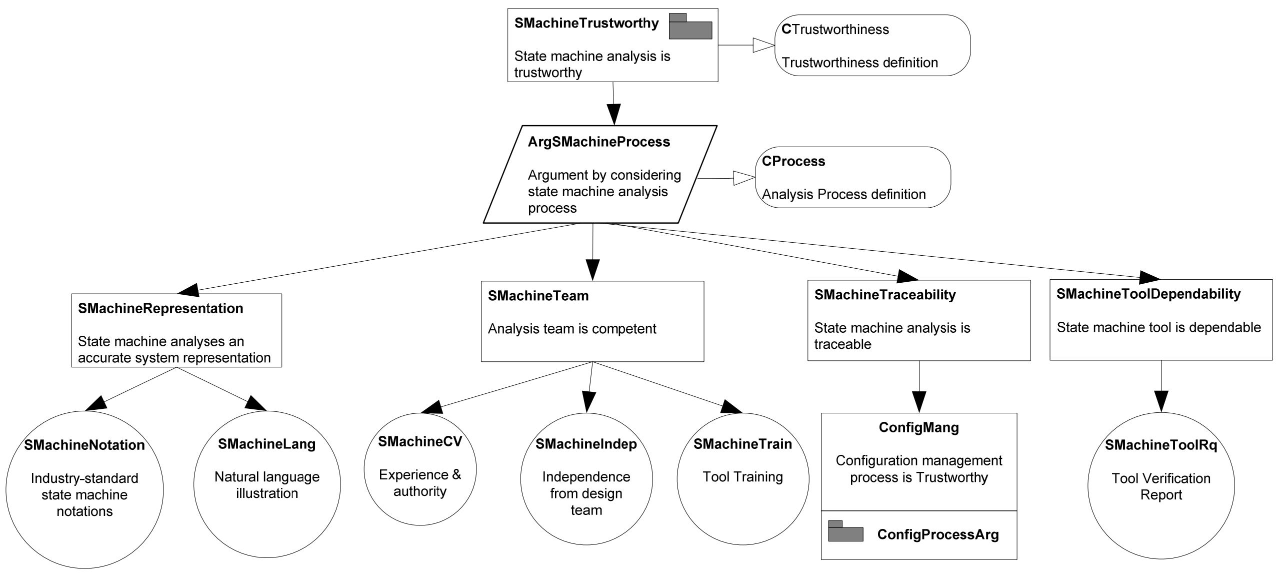

SMachineTrustworthy
State machine analysis is trustworthy
CTrustworthiness
Trustworthiness definition
ArgSMachineProcess
Argument by considering state machine analysis process
CProcess
Analysis Process definition
SMachineRepresentation
State machine analyses an accurate system representation
SMachineTeam
Analysis team is competent
SMachineTraceability
State machine analysis is traceable
SMachineToolDependability
State machine tool is dependable
SMachineNotation
Industry-standard state machine notations
SMachineLang
Natural language illustration
SMachineCV
Experience & authority
SMachineIndep
Independence from design team
SMachineTrain
Tool Training
ConfigMang
Configuration management process is Trustworthy
ConfigProcessArg
SMachineToolRq
Tool Verification Report

Figure 6. State Machine Process Argument

In short, in this section we have showed how to attach process-based arguments to the product evidence. In the next section, we evaluate the effectiveness of such an approach, particularly in uncovering hidden factors that can weaken or strengthen confidence in the safety argument.

6 Discussion

The process arguments presented in this paper aim, primarily, to support product-specific evidence. We believe that process-based evidence should not be correlated with the direct achievement of safety risk levels or failure rates. Instead, process arguments should only be used to strengthen product-based arguments. Arguing explicitly about the trustworthiness of the process protects the safety case against counter-arguments based on argument deconstruction and common mode failures.

Firstly, argument deconstruction attempts to undermine an argument by referring to doubts about hidden assumptions and conclusions [13]. However, by justifying the process behind the generation of the evidence, safety engineers can address and mitigate the impact of such hidden assumptions explicitly early on during the safety case development. For example, in the sheet metal press safety argument shown in Figure 4, the Context *C1 "Identified Software Hazards"* is supported by an argument that justifies the trustworthiness of the hazard identification process (*HzIdentTrustworthy*). By arguing explicitly about the trustworthiness of the hazard identification process, the safety argument is *defended against* counter-arguments that question the completeness of the list of defined hazards.

Secondly, arguing explicitly about the trustworthiness of the process can demonstrate independence between the evidence items provided in a safety argument. Evidence independence is particularly necessary to protect against common mode failures. The goal structure in Figure 7 depicts a safety case pattern for a diverse argument [8]. The diversity strategy (*S1*) is supported by one or more distinct statements (*Gn*). Although diversity might be proven by referring to the conceptual and mechanistic differences between evidence types, underestimating diversity at the process level can undermine the diversity of product evidence.

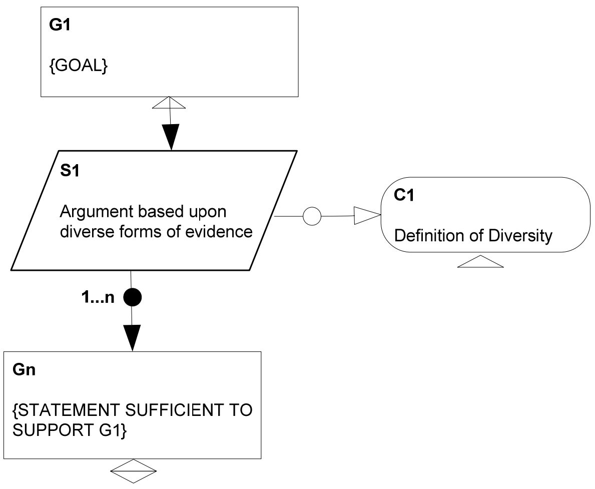

G1
{GOAL}
S1
Argument based upon diverse forms of evidence
C1
Definition of Diversity
1...n
Gn
{STATEMENT SUFFICIENT TO SUPPORT G1}

Figure 7. Diverse Argument Safety Case Pattern [8]

Figure 8 depicts an extended version of the above safety case pattern (diverse argument). An away goal (*GArgDiverse*) is attached to the argument strategy (*S1*). This away goal is used to justify diversity of the items of evidence (*Gn*) at both the product and process levels.

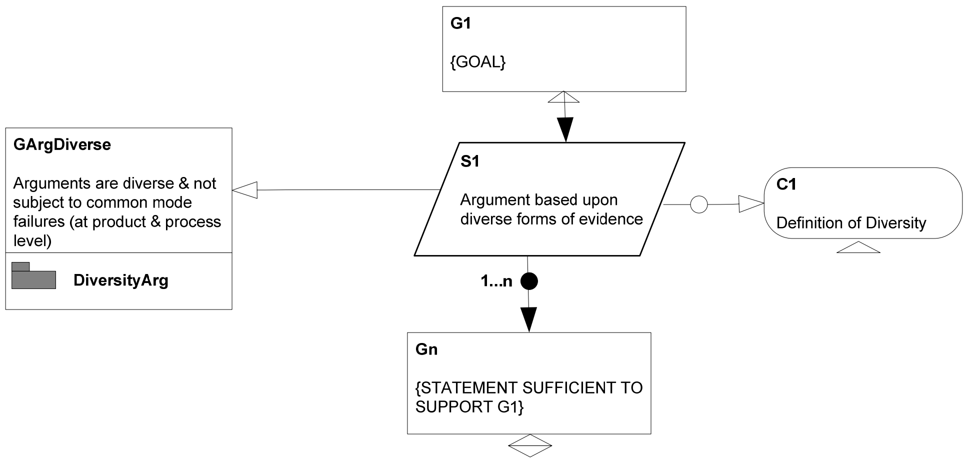

G1
{GOAL}
GArgDiverse
Arguments are diverse & not subject to common mode failures (at product & process level)
DiversityArg
S1
Argument based upon diverse forms of evidence
C1
Definition of Diversity
1...n
Gn
{STATEMENT SUFFICIENT TO SUPPORT G1}

Figure 8. Extended Diverse Argument Safety Case Pattern

It is important to address the approach presented in this paper from a practical perspective. It may be complicated to attach a process argument to each item of product evidence. However, this can be simplified by using GSN modular features, i.e. away goals. Away goals can support process claims by arguments located in another part of the safety case (modules). It may also be possible to present process justification in less detail. Instead of linking a process argument to each item of product evidence (i.e. solutions), it may be feasible to link the process argument to a high-level strategy, as shown in the safety case pattern in Figure 8. Additionally, not all safety arguments are of the same significance. Safety case developers may choose to elaborate only on high-priority safety arguments, where hidden process assumptions have direct impact on undermining confidence. Finally, in our example safety argument (sheet metal press), we have backed items of product evidence, such as black box testing and state machine analysis, by linking them to claims about the trustworthiness of the underlying process. However, in reality, there are different levels of trustworthiness in process evidence (similar to the different levels of confidence in product evidence [7]). The level of process trustworthiness varies with factors such as the degree of independence and rigor. For example, testing performed by an external and independent company will provide a higher level of trustworthiness than one performed by an internal testing team. Therefore, further work is required that defines levels of process trustworthiness based on process-specific measures and analysis.

7 Summary

In this paper, we have stressed the importance of process-based arguments in justifying the trustworthiness of the evidence that is presented in product-based arguments. The software safety arguments are predominantly inductive, and therefore process-based arguments play a key role in strengthening or weakening confidence in the software safety. The Goal Structuring Notation (GSN) can be used to represent highly integrated and directed process arguments that are linked to the evidence in the product-based arguments. This, in turn, protects the software safety argument against hidden process assumptions that make the argument vulnerable to criticism and counter-arguments.

References

1. U.K. Ministry of Defence, “00-55 Requirements of Safety Related Software in Defence Equipment”, Part 1: Requirements, Issue 2, MOD, Defence Standard, August 1997.
2. IEC (International Electrotechnical Commission), IEC 61508: “Functional safety of electrical/electronic/programmable electronic safety related systems”, Draft International Standard, International Electrotechnical Commission, 1997.
3. EUROCAE (European Organisation for Civil Aviation equipment) ED-12B/DO-178B, “Software considerations in airborne systems and equipment certification”, EUROCAE, 1994.
4. F Redmill, “Safety Integrity Levels – Theory and Problems, Lessons in System Safety”, Proceedings of the Eighth Safety-Critical Systems Symposium, edited by T. Anderson & F. Redmill, Springer-Verlag, 2000.
5. UK Ministry of Defence, “00-56 Safety Management Requirements for Defence Systems”, Part 1: Requirements, Issue 3, UK MOD, August 2004.
6. I. M. Copi, C. Cohen, “Introduction to Logic”, 10th Edition, Prentice Hall, 1998.
7. R. A. Weaver, “The Safety of Software – Constructing and Assuring Arguments”, DPhil Thesis, Department of Computer Science, University of York, UK, 2003.
8. T. P. Kelly, Arguing Safety – A Systematic Approach to Safety Case Management, DPhil Thesis, Department of Computer Science, University of York, UK, 1998.
9. J. McDermid, “Software Safety: Where's The Evidence?”, Proceedings of the Sixth Australian Workshop on Industrial Experience with Safety Critical Systems and Software, Australian Computer Society, 2001.
10. CAA (Civil Aviation Authority), “SW01 - Regulatory Objective for Software Safety Assurance in Air Traffic Service Equipment”, CAA, 1999.
11. A. Hall, “Seven Myths of Formal Methods”, IEEE Software archive, Volume 7, Issue 5, September 1990.

12. I. J. Bate, T. P. Kelly, “Architectural Considerations in the Certification of Modular Systems”, Reliability Engineering and System Safety, Elsevier, vol. 81, Issue 3, Pages 303-324, September 2003.
13. J Armstrong, S. Paynter, “The Deconstruction of Safety Arguments Through Adversarial Counter-Argument”, Proceedings of the Computer Safety, Reliability, and Security, 23rd International Conference, SAFECOMP 2004, Potsdam, Germany, September 21-24, 2004.
14. R. A. Weaver, J. A. McDermid, T. P. Kelly, “Software Safety Arguments: Towards a Systematic Categorisation of Evidence”, Proceedings of the 20th International System Safety Conference, Denver, USA, 2002.
15. UK Ministry of Defence, “00-56 Safety Management Requirements for Defence Systems”, Part 1: Requirements, Issue 3, UK Ministry of Defence, August 2004.