Systematic Maintenance of Safety Cases to Reduce Risk

Omar Jaradat[1] and Iain Bate[1,2]

[1] School of Innovation, Design, and Engineering
Mälardalen University, Västerås, Sweden
[2] Department of Computer Science
University of York, York, United Kingdom
omar.jaradat@mdh.se
iain.bate@york.ac.uk

Abstract. The development of safety cases has become common practice in many safety critical system domains. Safety cases are costly since they need a significant amount of time and efforts to be produced. Moreover, safety critical systems are expected to operate for a long period of time and constantly subject to changes during both development and operational phases. Hence, safety cases are built as living documents that should always be maintained to justify the safety status of the associated system and evolve as these system evolve. However, safety cases document highly interdependent elements (e.g., safety goals, evidence, assumptions, etc.) and even seemingly minor changes may have a major impact on them, and thus dramatically increase their cost. In this paper, we identify and discuss some challenges in the maintenance of safety cases. We also present two techniques that utilise safety contracts to facilitate the maintenance of safety cases, we discuss the roles of these techniques in coping with some of the identified maintenance challenges, and we finally discuss potential limitations and suggest some solutions.

Keywords: Safety Case, Safety Argument, Maintenance, FTA, Sensitivity Analysis, Safety Contracts, Impact Analysis.

1 Introduction

The size and complexity of safety critical systems are considerable. Without a clear demonstration for the safety performance of a system, it is difficult for inspector organisations or system engineers themselves to build a confidence in the safety performance of the system. System engineers of some safety critical systems are required to demonstrate the safety performance of their systems through a reasoned argument that justifies why the system in question is acceptably safe (or will be so) [10]. This argument is communicated via an artefact that is known as a *safety case*. The safety case is the whole safety justification that comprises every appropriate piece of evidence to make a convincing argument to support the safety performance claims [13].

Moreover, safety critical systems can be evolutionary as they are subject to perfective, corrective or adaptive maintenance or through technology obsolescence [24]. Changes to the system during or after development might invalidate safety evidence or argument. Evidence might no longer support the developers' claims because it reflects old development artefacts or old assumptions about operation or the operating environment. After a change, original safety claims might be nonsense, no longer reflect operational intent, or be contradicted by new data [17]. Eventually, the real system will have diverged so far from that represented by the safety case argument and the latter is no longer valid or useful [13]. Hence, it is almost inevitable that the safety case will require updating throughout the operational lifetime of the system. In addition, any change that might compromise system safety involves repeating the certification process (i.e., re-certification) and repeating the certification process necessitates an updated and valid safety case that considers the changes. For example, the UK Ministry of Defence Ship Safety Management System Handbook JSP 430 requires that *"the safety case will be updated ... to reflect changes in the design and/or operational usage which impact on safety, or to address newly identified hazards. The safety case will be a management tool for controlling safety through life including design and operation role changes"* [12,25]. Similarly, the UK Health and Safety Executive (HSE) — Railway safety case regulations 1994 — states in regulation 6(1) that *"a safety case to be revised whenever, appropriate that is whenever any of its contents would otherwise become inaccurate or incomplete"* [6,12].

However, a single change to a safety case may necessitate many other consequential changes — creating a ripple effect [24]. Any improper maintenance in a safety argument might cause unforeseen violations of the acceptable safety limits, which will negatively impact the system safety performance conveyed by the safety case. Hence, a step to assess the impact of this change on the safety argument is crucial and highly needed prior to updating a safety argument after a system change. Despite clear recommendations to adequately maintain and review safety cases by safety standards existing standards offer little advice on how such operations can be carried out [24].

The concept of contract has been around for a few decades in the system development domain. There have been significant works that discuss how to represent and to use contracts (e.g., [3,26]). Also, researchers have used assume-guarantee contracts to propose techniques to lower the cost of developing software for safety critical systems. Moreover, contracts have been exploited as a means for helping to manage system changes in the system domain or in its corresponding safety case [5,9,18]. However, using contracts as a way of managing change was discussed in some works [2,9], but deriving the contracts and their contents have received little support yet [16].

In this paper, we present and discuss techniques that utilises the concept of contracts to facilitate the accommodation of system changes in safety cases to ultimately support the maintainability of safety cases. Our work focuses on: 1. How and where to derive safety contracts and their contents, 2. using the derived contracts to support the decision as to whether or not apply changes,

and 3. using the derived contracts to guide developers to the parts in the safety case that might be affected after applying a change. This paper is composed of three further sections. In Section 2, we present background information and we also present some safety cases' challenges. In Section 3, we describe two techniques to facilitate the maintenance of safety cases. In Section 4 we discuss some limitations, draw a conclusion and propose potential future work.

2 Background and Motivation

2.1 Safety Contracts

In 1969, Hoare introduced the pre- and postcondition technique to describe the connection (dependency) between the execution results (R) of a program (Q) and the values taken by the variables (P) before that program is initiated [7]. Hoare introduced a new notation to describe this connection, such as: **P {Q} R**. This notation can be interpreted as: *"If the assertion P is true before initiation of a program Q, then the assertion R will be true on its completion"* [7].

Contracts are widely used in software development. For instance, Design by Contract (DbC) was introduced by Meyer [14, 15] to constrain the interactions that occur between objects. Moreover, contract-based design is an approach where the design process is seen as a successive assembly of components where a component behaviour is represented in terms of assumptions about its environment and guarantees about its behaviour [4].

The following is an example that depicts the most common used form of contracts [11]:

Guarantee: The WCET of task X is $\leq$ 10 milliseconds
Assumptions:
X is:
1. compiled using compiler $[C]$,
2. executed on microcontroller $[M]$ at 1000 MHz with caches disabled, and
3. not interrupted

A contract is said to be a *safety contract* if it guarantees a property that is traceable to a hazard. There have been significant works that discuss how to represent and to use contracts [3, 26]. In the safety critical systems domain, researchers have used, for example, assume-guarantee contracts to propose techniques to lower the cost of developing software for safety critical systems. Moreover, contracts have been exploited as a means for helping to manage system changes in a system domain or in its corresponding safety case [5, 9, 18].

2.2 The Goal Structuring Notation (GSN)

A safety argument organizes and communicates a safety case, showing how the items of safety evidence are related and collectively demonstrate that a system is acceptably safe to operate in a particular context. GSN [1] provides a

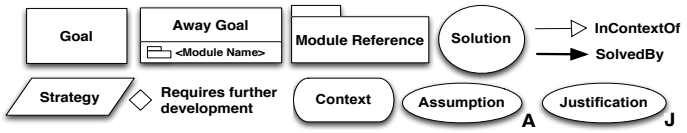

Fig. 1. Notation Keys of the Goal Structuring Notation (GSN)

graphical means of communicating (1) safety argument elements, claims (goals), argument logic (strategies), assumptions, context, evidence (solutions), and (2) the relationships between these elements. The principal symbols of the notation are shown in Figure 1 (with example instances of each concept).

A goal structure shows how goals are successively broken down into ('solved by') sub-goals until eventually supported by direct reference to evidence. Using the GSN, it is also possible to clarify the argument strategies adopted (i.e., how the premises imply the conclusion), the rationale for the approach (assumptions, justifications) and the context in which goals are stated.

2.3 Safety Cases Maintainability: What does it mean?

The goal of the work which is being discussed in this paper is to **facilitate the accommodation of system changes in safety cases to ultimately enhance safety case maintainability.** Hence, it is vital to explicitly define what do we mean by safety case maintainability. We refer to *"Safety Case Maintainability"* as the ability to repair or replace the impacted elements of a safety case argument, without having to replace still valid elements, to preserve the validity of the argument. The maintainability degree is said to be high whenever the following three activities are done efficiently:

1. Identifying the impacted elements and those that are not impacted.
2. Minimising the number of impacted elements.
3. Reducing the work needed to make the impacted elements valid again.

However, the work presented by this paper does not focus on how to measure the efficiency of achieving these three activities, but rather it strives to enable them and improve on them. In order to achieve this goal, we should resolve the problems that affect the accommodation of system changes in safety cases.

2.4 Safety Cases Maintainability: Why is it painstaking?

Safety assurance and certification are amongst the most expensive and time-consuming tasks in the development of safety-critical embedded systems [10]. A key reason behind this is because the increasing complexity and size of these systems combined with their growing market demands. The cost of system changes including the cost of the activities that will follow them (e.g., regression testing), are another key reason that exacerbates the problems of cost and time in

safety certification. Coherent strategies are required to reduce the cost and time of safety certification.

One of the biggest challenges that affects safety case revision and maintenance is that a safety case documents a complex reality that comprises a complex web of interdependent elements. That is, safety goals, evidence, argument, and assumptions about operating context are highly interdependent. Hence, seemingly minor changes may have a major impact on the contents and structure of the safety argument. Basically, operational or environmental changes may invalidate a safety case argument for two main reasons as follows:

1. Evidence is valid only in the operational and environmental context in which it is obtained, or to which it applies. During or after a system change, evidence might no longer support the developers' claims because it could reflect old development artefacts or old assumptions about operation or the operating environment.
2. Safety claims, after introducing a change, might be nonsense, no longer reflect operational intent, or be contradicted by new data. Changing safety claims might change the argument structure.

In order to deal with problems that impede safety cases maintenance, we start by identifying and describing these problems.

Main Problem: *Maintaining safety cases after implementing a system change is a painstaking process.* This main problem is caused by three sub-problems.
Sub-problem (1): *The lack of documentation of dependencies among the safety cases contents.*

Developers of safety cases are experiencing difficulties in identifying the direct and indirect impact of change due to high level of dependency among safety case elements. If developers do not understand the impact of change then they have to be conservative and do wider verification (i.e., check more elements than strictly necessary) and this increases the maintenance cost. The use of GSN might help to produce well-structured arguments that clearly demonstrate the relationships between the argument claims and evidence. However, GSN has not solved the problem of documenting dependencies among the safety cases contents [22]. In other words, a well-structured GSN argument helps the developers to mechanically propagate the change through the goal structure. However, it does not evaluate whether the suspect elements of the argument are still valid or not (or it does not show why the element is impacted), but rather it can bring these elements to the developers' attention [22].

Safety is a system level property; assuring safety requires safety evidence to be consistent and traceable to system safety goals [24]. Moreover, current standards and analysis techniques assume a top-down development approach to system design. One might suppose that a safety argument structure aligned with the system design structure would make traceability clearer. It might, but safety argument structures are influenced by four factors: (1) modularity of evidence, (2) modularity of the system, (3) process demarcation (e.g., the scope of ISO 26262 items [8]), and (4) organisational structure (e.g., who is working on

what) [2]. These factors often make argument structures aligned with the system design structure impractical. However, the need to track changes across the whole safety argument is still significant for maintaining the argument regardless of its structure.

As explained in Section 2.1, a contract is conceived as an extension to the specification of software component interfaces that specifies preconditions and postconditions to describe what properties a component can offer once the surrounded environment satisfies one or more related assumption(s). Based on this description, safety contracts can be used as a means to record the dependencies among system components. If we assume a one-to-one mapping between a system component and all the claims that are articulated about it, dependencies among safety argument elements can be conceived through the dependencies between components of the corresponding system that are recorded in contracts. In practice, this notion is far from straightforward because it is infeasible to be achieved and impossible to prove the completeness of the generated contracts, and the expected number of contracts will be too large to easily manage.

Sub-problem (2): *The lack of traceability between a system and its safety case.*

We refer to the ability to relate safety argument fragments to system design components as component traceability (through a safety argument). We refer to evidence across a system's artefacts as evidence traceability.

System developers need both top-down and bottom-up impact analysis approaches to maintain safety cases. A top-down approach is dedicated for analysing the impacted artefacts from the system domain down to the safety argument. In contrast, a bottom-up approach is dedicated for analysing impacted elements from the argument to the corresponding artefacts such as a safety analysis report, test results or requirements specification, etc. The lack of systematic and methodical approaches to analysing impact of change is a key reason behind the maintenance difficulties. However, conducting any style of impact analysis requires a traceability mechanism between the system and safety arguments.

There has been significant work on how to use safety contracts as a means to establish the required traceability [2]. The guaranteed properties in the contracts can be mapped to safety argument goals. If the derived safety contracts are associated with the corresponding argument elements, any broken contracts will reveal (i.e., highlight) the associated argument elements and thus enabling easier identification for the impacted parts in the argument due to a system change. However, this is not as simple as it first appears because we still do not know which contracts were affected by the change. In other words, how does a change lead to broken contracts?

Predicting system changes before building a safety argument can be useful because it allows the safety argument to be structured to contain the impact of these changes. Hence, anticipated changes may have predictable and traceable consequences that will eventually reduce maintenance effort. Nevertheless, planning the maintenance of a safety case still faces a key problem.

Sub-problem (3): *System changes and their details cannot be fully predicted and made available up front.*

Modularity has been proposed as the key element of the 'way forward' in developing systems [19, 21]. Although the most influential approach for using modularity effectively in software design is information hiding, modularity can also be beneficial for systems maintenance. For modular systems, it is claimed that the required maintenance efforts to accommodate predicted changes can be less than the required efforts to accommodate arbitrary changes. This is because having a list of predicted changes during the system design phase allows system engineers to contain the impact of each of those changes in a minimal number of system's modules. Predicting system changes before building a safety argument can be useful because it allows the safety argument to be structured to contain the impact of these changes. Hence, predicted changes may also have predictable and traceable consequences that will eventually reduce the maintenance efforts. Nevertheless, planning the maintenance of a safety case still faces two key issues: (1) system changes cannot be fully predicted and made available up front, especially, the software aspects of the safety case as software is highly changeable and harder to manage as they are hard to contain and (2) those changes can be implemented years after the development of a safety case [16].

3 Sensitivity Analysis for Enabling Safety Argument Maintenance (SANESAM)

Sensitivity analysis can be defined as: *"The study of how uncertainty in the output of a model (numerical or otherwise) can be apportioned to different sources of uncertainty in the model input"* [23]. The analysis helps to establish reasonably acceptable confidence in the model by studying the uncertainties that are often associated with variables in models.

In our previous work [16], we introduced a Sensitivity ANalysis for Enabling Safety Argument Maintenance (SANESAM) technique, in which we apply sensitivity analysis on FTAs to measure the sensitivity of outcome A (e.g., a safety requirement being true) to a change in a parameter B (e.g., the failure probability in a component). The sensitivity is defined as $\Delta B/B$, where ΔB is the smallest change in B that changes A (e.g., the smallest increase in failure probability that makes safety requirement A false). The failure probability values that are attached to FTA's events are considered input parameters to the sensitivity analysis. A sensitive part of a FTA is defined as one or multiple FTA events whose minimum changes (i.e., the smallest increase in its failure probability due to a system change) have the maximal effect on the FTA, where effect means exceeding failure probabilities (reliability targets) to inadmissible levels. SANESAM was extended by SANESAM+ [11] to consider the change's impact on: (1) intermediate events of FTAs, (2) multiple events, and (3) duplicated events. The key principle of both techniques is to determine, for each system component, the allowed range for a certain parameter within which a component may change before it compromises a certain system property (e.g., safety, reliability, etc.). Sensitivity analysis is used in the techniques as a method to determine the range of failure probability parameter for each component. The techniques

assume the existence of a probabilistic FTA where each event in the tree is specified by a current estimate of failure probability $FP_{Current|event(x)}$. In addition, they assume the existence of the required failure probability for the top event $FP_{Required(Topevent)}$, where the FTA is considered unreliable if:

$$FP_{Currentl(Topevent)} > FP_{Required(Topevent)} \text{ [16].}$$

The steps of SANESAM are shown in Figure 2 and described as follows [16]: ***Step 1. Apply the sensitivity analysis to a probabilistic FTA***: In this step the sensitivity analysis is applied to a FTA to identify the sensitive events whose minimum changes have the maximal effect on the $FP_{Topevent}$. Identifying those sensitive events requires the following steps to be performed:

1. Find the Minimal Cut Set (MC) in the FTA. The minimal cut set definition is: *"A cut set in a fault tree is a set of basic events whose (simultaneous) occurrence ensures that the top event occurs. A cut set is said to be minimal if the set cannot be reduced without losing its status as a cut set"* [20].
2. Calculate the maximum possible increment to the failure probability parameter of event x before the top event $FP_{Required(Topevent)}$ is no longer met, where $x \in MC$, and

$$(FP_{Increased|event(x)} - FP_{Current|event(x)}) \nRightarrow \\ FP_{Increased(Topevent)} > FP_{Required(Topevent).}$$

3. Rank the sensitive events from the most sensitive to the less sensitive. The most sensitive event is the event for which the following formula is the minimum:

$$\frac{FP_{Increased|event(x)} - FP_{Current|event(x)}}{FP_{Current|event(x).}}$$

Step 2. Refine the identified sensitive parts with system developers: In this step, the generated list of sensitive events from Step 1 should be discussed by system developers (e.g., safety engineers) as they should choose the sensitive events that are most likely to change. The list can be extended to add any additional events by the developers. Moreover, it is envisaged that some events might be removed from the list or the rank of some of them might change.

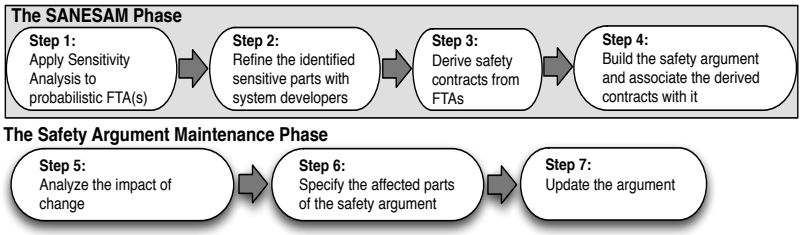

Fig. 2. Process diagram of SANESAM and SANESAM+ [16]

Step 3. Derive safety contracts from FTAs: In this step, a safety contract or contracts should be derived for each event in the list from Step 2. The main objectives of the contracts are to: (1) highlight the sensitive events to make them visible up front for developers attention, and (2) to record the dependencies between the sensitive events and the other events in the FTA. Hence, if the system is later changed in a way that increases the failure probability of a contracted event where the increased failure probability is still within the defined threshold in the contract, then it can be said that the contract(s) in question still hold (intact) and the change is containable with no further maintenance. The contract(s), however, should be updated to the latest failure probability value. On the other hand, if the change causes a bigger increment to the failure probability value than the contract can hold, then the contract is said to be broken and the guaranteed event will no longer meet its reliability target. It is worth noting that the role of safety contracts in SANESAM is to highlight sensitive events, and not to enter new event failure probabilities. Figure 3 an example of a derived safety contract from FTA.

Step 4. Build the safety argument and associate the derived contracts with it: In this step, a safety argument should be built and the derived safety contracts should be associated with the argument elements. Essentially, SANESAM calculates the maximum possible increment to the failure probability parameter of only one event at a time before the top event $FP_{Required(Topevent)}$ is no longer met. It considers the events within the MC only. SANESAM+ was introduced to provide more freedom by considering multiple events at a time

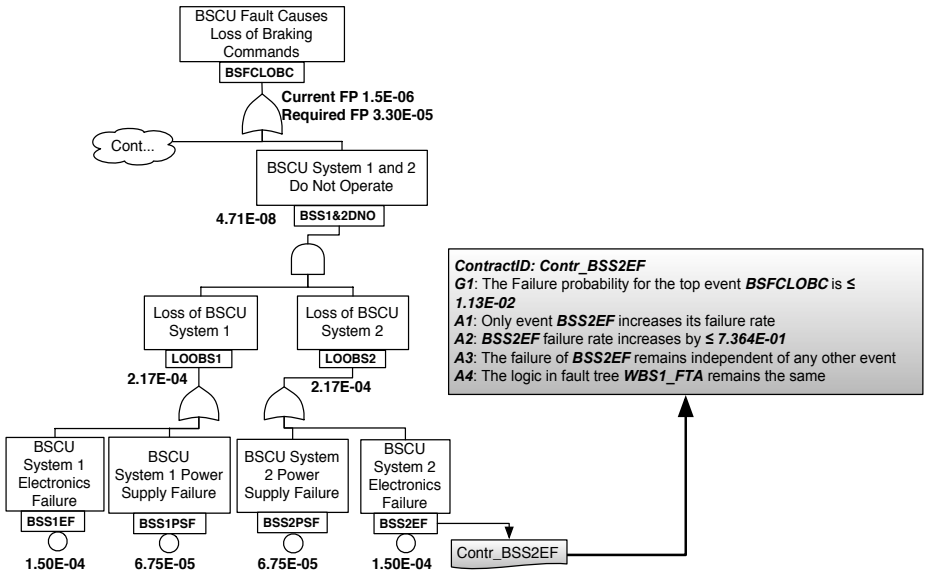

Fig. 3. Example of a derived safety contract

and not only the events in the MC. The key principle of SANESAM+ is to distribute $\Delta FP_{(Topevent)}$ over all of the events in FTA, or over the events that are relevant to a particular change. Hence, the difference between SANESAM and SANESAM+ is observed only in Step 1, all other steps are identical.

3.1 The roles of safety contracts in SANESAM and SANESAM+

SANESAM and SANESAM+ derive safety contracts for the identified sensitive parts. The main objectives of the contracts is to (1) highlight the sensitive events to make them visible up front for developers attention and (2) to record the dependencies between the sensitive events and the other events in the FTA. Hence, if any contracted event has received a change that necessitates increasing its failure probability where the increment is still within the defined threshold in the contract, then it can be said that the contract(s) in question still holds (intact) and the change is containable with no further maintenance. The contract(s), however, should be updated to the latest failure probability value. On the contrary, if the change causes a bigger increment in the failure probability value than the contract can hold, then the contract is said to be broken and the guaranteed event will no longer meet its reliability target. Hence, SANESAM and SANESAM+ may address the first and the second identified sub-problems in Section 2.4.

3.2 Support the prediction of potential system changes

Expectedly, if we ask system engineers to anticipate the potential future changes for a system they might brainstorm and come up with a list of changes. However, the list can be incomplete or contain unlikely changes that might influence the system design to little or no avail. Instead, we propose providing system developers a list of system parts that may be more problematic to change than other parts and ask them to choose the parts that are most likely to change. Of course our list can be augmented by additional changeable parts that may be provided by the system developers. Hence, SAMESAM and SANESAM+ may address the third identified sub-problem in Section 2.4.

4 Conclusion and Future Work

System developers should understand the change and the potential risks that it might carry before they identify the impacted parts. For example, a change might turn some implicit assumptions about the context in which a system should operate to be wrong. Misunderstanding the change might lead to skip those parts of the system which are dependent on that assumptions. Also, the developers need to understand the dependencies between the system parts to identify the affected parts correctly. Hence, there is a pressing need for acceptable methods and techniques to enable easier change accommodation in safety critical systems without incurring disproportionate cost compared to the size of the change.

In this paper, SANESAM and SANESAM+ were discussed as techniques to facilitate the maintenance of safety cases. The techniques were introduced and illustrated in our previous work. More specifically, we proposed SANESAM [16] through which we: (1) measure the sensitivity of FTA events to system changes using the events' failure probabilities, (2) derive safety contracts based on the results of the analysis, and (3) map the derived safety contracts to a safety argument to improve the change impact analysis on the safety argument. We used an aircraft Wheel Braking System (WBS) to illustrate the application of SANESAM. We also developed SANESAM+ [11] as another version of SANESAM to cover wider variety of change scenarios, where we also used the WBS to illustrate it. This paper also identifies some challenges in the maintenance of safety cases, and shows how the techniques might help to address them.

A foreseen limitation in the techniques is that they can be less useful while dealing with software changes as it is recognised as being difficult to quantify the failure probabilities of the system software components.

To further develop the approach, SANESAM+ is being migrated to timing. More specifically, the problem of the Worst Case Execution Time (WCET) is considered as the property where sensitivity is judged in terms of its impact on the ability to meet the system's timing requirements. We also plan to create several case studies to validate both the feasibility and efficacy of the techniques.

Acknowledgment

This work has been partially supported by the Swedish Foundation for Strategic Research (SSF) through SYNOPSIS and FiC Projects. The work is also partially supported by SafeCOP project.

References

1. GSN community standard version 1. Technical report, Origin Consulting (York) Limited, Nov. 2011.
2. Modular Software Safety Case (MSSC) — Process Description. [online]. available: https://www.amsderisc.com/related-programmes, Nov 2012.
3. A. Benveniste, B. Caillaud, A. Ferrari, L. Mangeruca, R. Passerone, and C. Sofronis. Multiple viewpoint contract-based specification and design. In *Proceedings of the 6th International Symposium on Formal Methods for Components and Objects*, pages 200–225, Springer Berlin Heidelberg, 2007.
4. L. Benvenuti, A. Ferrari, E. Mazzi, and A. L. Vincentelli. Contract-based design for computation and verification of a closed-loop hybrid system. In *Proceedings of the 11th International Workshop on Hybrid Systems: Computation and Control*, HSCC '08, pages 58–71, Berlin, Heidelberg, 2008. Springer-Verlag.
5. P. Graydon and I. Bate. The nature and content of safety contracts: Challenges and suggestions for a way forward. In *Proceedings of the 20th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC)*, November 2014.
6. Health and Safety Executive (HSE). *Railway Safety Cases - Railway (Safety Case) Regulations - Guidance on Regulations*, 1994.

7. C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, Oct. 1969.
8. ISO 26262:2011. Road Vehicles — Functional Safety, Part 1-9. International Organization for Standardization, Nov 2011.
9. J. L. Fenn, R. Hawkins, P. J. Williams, T. Kelly, M. G. Banner, Y. Oakshott. The who, where, how, why and when of modular and incremental certification. In *Proceedings of the 2nd IET International Conference on System Safety*, pages 135–140. IET, 2007.
10. O. Jaradat. Enhancing the maintainability of safety cases using safety contracts, Mälardalen University, Västerås, Sweden. http://www.es.mdh.se/publications/4082-, November 2015.
11. O. Jaradat and I. Bate. Deriving hierarchical safety contracts. In *The 21st IEEE Pacific Rim International Symposium on Dependable Computing*, November 2015.
12. T. Kelly. *Arguing Safety – A Systematic Approach to Managing Safety Cases*. PhD thesis, Department of Computer Science, University of York, 1998.
13. R. Maguire. *Safety Cases and Safety Reports: Meaning, Motivation and Management*. Ashgate Publishing, Ltd., 2012.
14. B. Meyer. Design by contract. Technical Report TR-EI-12/CO, Interactive Software Engineering Inc., 1986.
15. B. Meyer. *Object-Oriented Software Construction*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1st edition, 1988.
16. O. Jaradat, I. Bate, and S. Punnekkat. Using sensitivity analysis to facilitate the maintenance of safety cases. In *Proceedings of the 20th International Conference on Reliable Software Technologies (Ada-Europe)*, pages 162–176, June 2015.
17. O. Jaradat, P. Graydon and I. Bate. An approach to maintaining safety case evidence after a system change. In *Proceedings of the 10th European Dependable Computing Conference (EDCC)*, Newcastle, UK, August 2014.
18. P. Conmy, J. Carlson, R. Land, S. Björnander, O. Bridal, I. Bate. Extension of techniques for modular safety arguments. Deliverable d2.3.1, technical report, Safety certification of software-intensive systems with reusable components (SafeCer), 2012.
19. D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Commun. ACM*, 15(12):1053–1058, Dec. 1972.
20. M. Rausand and A. Høyland. *System Reliability Theory: Models, Statistical Methods and Applications*. Wiley-Interscience, Hoboken, NJ, 2004.
21. S. Bates, I. Bate, R. Hawkins, T. Kelly, J. McDermid, and R. Fletcher. Safety case architectures to complement a contract-based approach to designing safe systems. In *Proceedings of the 21st International System Safety Conference (ISSC)*, 2003.
22. S. Wilson, T. Kelly, and J. McDermid. Safety case development: Current practice, future prospects. In *Proceedings of the 12th Annual CSR Workshop - Software Bases Systems*. Springer-Verlag, 1997.
23. A. Saltelli. *Global sensitivity analysis: the primer*. John Wiley, 2008.
24. T. Kelly and J. McDermid. A systematic approach to safety case maintenance. In *Proceedings of the Computer Safety, Reliability and Security*, volume 1698 of *Lecture Notes in Computer Science*, pages 13–26. Springer Berlin Heidelberg, 1999.
25. U.K. Ministry of Defence, "JSP 430 - Ship Safety Management System Handbook", Ministry of Defence January 1996.
26. W. Damm, H. Hungar, J. Bernhard, T. Peikenkamp, and I. Stierand. Using contract-based component specifications for virtual integration testing and architecture design. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition*, pages 1–6, 2011.