# Using Sensitivity Analysis to Facilitate The Maintenance of Safety Cases

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

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

**Abstract.** A safety case contains safety arguments together with supporting evidence that together should demonstrate that a system is acceptably safe. System changes pose a challenge to the soundness and cogency of the safety case argument. Maintaining safety arguments is a painstaking process because it requires performing a change impact analysis through interdependent elements. Changes are often performed years after the deployment of a system making it harder for safety case developers to know which parts of the argument are affected. Contracts have been proposed as a means for helping to manage changes. There has been significant work that discusses how to represent and to use them but there has been little on how to derive them. In this paper, we propose a sensitivity analysis approach to derive contracts from Fault Tree Analyses and use them to trace changes in the safety argument, thus facilitating easier maintenance of the safety argument.

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

## 1 Introduction

Building a safety case is an increasingly common practice in many safety critical domains [7]. A safety case comprises both safety evidence and a safety argument that explains that evidence. The safety evidence is collected throughout the development and operational phases, for example from analysis, test, inspection, and in-service monitoring activities. The safety argument shows how this evidence demonstrates that the system satisfies the applicable operational definition of acceptably safe to operate in its intended operating context.

A safety case should always justify the safety status of the associated system, therefore it is described as a living document that should be maintained as needed whenever some aspect of the system, its operation, its operating context, or its operational history changes. However, safety goals, evidence, argument, and assumptions about operating context are interdependent and thus,

seemingly-minor changes may have a major impact on the contents and structure of the safety argument. Any improper maintenance in a safety argument has a potential for a tremendous negative impact on the conveyed system safety status 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.

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. In the updated system, existing safety claims might not make any sense, no longer reflect operational intent, or be contradicted by new data. Analysing the impact of a change in a safety argument is not trivial: doing so requires awareness of the dependencies among the argument's contents and how changes to one part might invalidate others. In other words, if a change was applied to any element of a set of interdependent elements, then the associated effects on the rest of the elements might go unnoticed. Without this vital awareness, a developer performing impact analysis might not notice that a change has compromised system safety. Implicit dependencies thus pose a major challenge. Moreover, evidence is valid only in the operational and environmental contexts in which it was obtained or to which it applies. Operational or environmental changes might therefore affect the relevance of evidence and, indirectly, the validity and strength of the safety argument.

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 the maintenance efforts. Nevertheless, planning the maintenance of a safety case still faces two key issues: (1) system changes and their details 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. Part of what we aim for in this work is to provide 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.

Sensitivity analysis helps the experts to define the uncertainties involved with a particular system change so that those experts can judge on the potential change based on how reliable they feel the consequences are. The analysis can deal with what aim for since it allows us to define the problematic changes. More specifically, we exploit the Fault Tree Analyses (FTAs) which are supposed to have been done by developers through the safety analysis phase and apply the sensitivity analysis to those FTAs in order to identify the sensitive parts in them. We define a sensitive part as one or multiple events whose minimum changes have the maximal effect on the FTA, where effect means exceeding reliability targets due to a change.

In spite of the assumption we make that the safety arguments logic is based on the causal pathways described in the FTAs, tracking the changes from the FTAs of a system down to its safety argument still requires a traceability mechanism between the two. To this end, we use the concept of contract to highlight the sensitive parts in FTAs, and to establish a traceability between those parts and the corresponding safety argument. In our work, we assume that safety arguments are recorded in the Goal Structuring Notation (GSN) [6]. However, the approach we propose might (with suitable adaptations) be compatible for use with other graphical assurance argument notations.

Combining the sensitivity analysis together with the concept of contracts to identify the sensitive parts of a system and highlight these parts may help the experts to make an educated decision as to whether or not apply changes. This decision is in light of beforehand knowledge of the impact of these changes on the system and its safety case. Our hypothesis in this work is that it is possible to use the sensitivity analysis together with safety contracts to (1) bring to developers' attention the most sensitive parts of a system for a particular change, and (2) manage the change by guiding the developers to the parts in the safety argument that might be affected after applying a change. However, using contracts as a way of managing change is not a new notion since it has been discussed in some works, such as [2][5], but deriving the contracts and their contents have received little or even no support yet. The main contribution of this paper is to propose a safety case maintenance technique. However, we focus on the first phase of the technique and explain how to apply the sensitivity analysis to FTAs and derive the contracts and their contents. We also explain how to associate the derived arguments with safety argument goals. The paper illustrates the technique and its key concepts using the a hypothetical aircraft Wheel Braking System (WBS).

The paper is structured as follows: in Section 2 we present background information. In Section 3 we propose a technique for maintaining safety cases using sensitivity analysis. In Section 4 we use the WBS example to illustrate the technique. In Section 5 we present the related work. Finally, we conclude and derive future works in Section 6.

## 2 Background and Motivation

This section gives background information about (1) the GSN, (2) the concept of contract, (3) some of the current challenges that are facing safety case maintenance including a brief review of the state-of-the-art, and (4) the sensitivity analysis including some possible applications.

### 2.1 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 [6] provides a graphical means of communicating (1) safety argument elements, claims (goals),

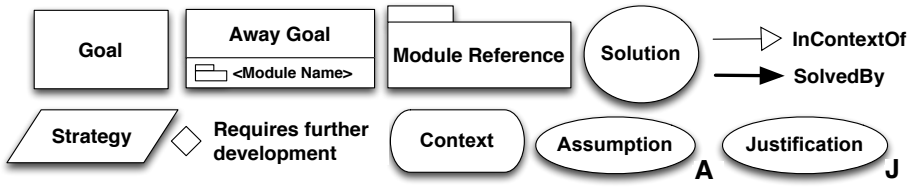


**Fig. 1.** Notation Keys of the Goal Structuring Notation (GSN)

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.

### 2.2 The Concept of Safety Contracts

The concept of contract is not uncommon in software development and it was first introduced in 1988 by Meyer [12] to constrain the interactions that occur between objects. Contract-based design [3] is defined as 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 behavior. Hence, contracts are intended to describe functional and behavioral properties for each design component in form of assumptions and guarantees. In this paper, a contract which describes properties that are only safety-related is referred to as a safety contract.

### 2.3 Safety Case Maintenance and Current Practices

A safety case is a living document that should be maintained as the system, its operation, or its operating context changes. In this paper, we refer to the process of updating the safety case after implementing a change as safety case maintenance. Developers are experiencing difficulties with safety case maintenance, including difficulty identifying the direct and indirect impact of change. Two main causes of this difficulty are a lack of traceability between a system and its safety case and a lack of documentation of dependencies among the safety cases contents. Systems tend to become more complex, this increasing complexity can exacerbate safety case maintenance difficulties. The GSN is meant to reduce these difficulties by providing a clear, explicit conceptual model of the safety cases elements and interdependencies [10].

Our discussion of documenting interdependencies within a safety case refers to two different forms of traceability. Firstly, we refer to the ability to relate safety

argument fragments to system design components as component traceability (through a safety argument). Secondly, we refer to evidence across system's artefacts as evidence traceability.

Current standards and analysis techniques assume a top-down development approach to system design. When systems that are built from components, mismatch with design structure makes monolithic safety arguments and evidence difficult to maintain. Safety is a system level property; assuring safety requires safety evidence to be consistent and traceable to system safety goals [10]. 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 [7]), and organisational structure (e.g., who is working on what). 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.

### 2.4 Sensitivity Analysis

Sensitivity analysis helps to establish reasonably acceptable confidence in the model by studying the uncertainties that are often associated with variables in models. There are different purposes for using sensitivity analysis, such as, providing insight into the robustness of model results when making decisions [4]. The analysis can be also used to enhance communication from modelers to decision makers, for example, by making recommendations more credible, understandable, compelling or persuasive [13]. The analysis can be performed by different methods, such as, mathematical, graphical, statistical, etc.

In this paper, we use sensitivity analysis to identify the sensitive parts of a system that might require unnecessary painstaking maintenance. More specifically, we apply the 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 rate in a component). The sensitivity is defined as $\Delta B/B$, where $\Delta B$ is the smallest change in $B$ that changes $A$ (e.g., the smallest increase in failure rate that makes safety requirement A false). Hence, a sensitive part is defined as one or multiple FTA events whose minimum changes have the maximal effect on the FTA, where effect means exceeding failure probabilities (reliability targets) to inadmissible levels due to the change. The failure probability values that are attached to the FTA events are considered input parameters to the sensitivity analysis. A sensitive event is the event whose failure probability value can significantly influence the validity of the FTA once it increases. A sensitive part of a FTA is assigned to a system design component that is referred to as sensitive component in this paper. Hence, changes to a sensitive component cause a great impact to system design.

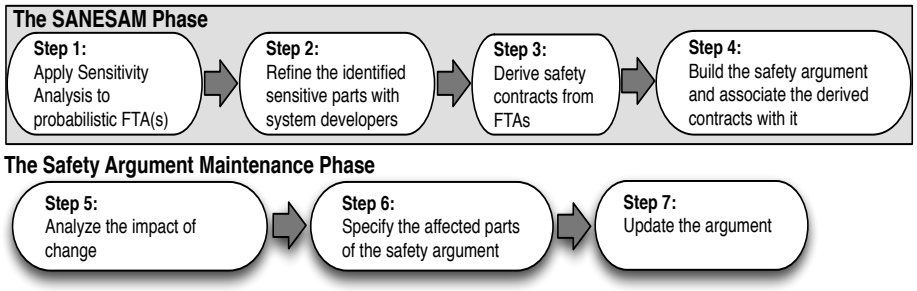


**Fig. 2.** Process diagram of the proposed technique

## 3 Using Sensitivity Analysis To Facilitate The Maintenance of A Safety Case

In this section, we build on the background information provided in Section 2 to propose a technique that aims to facilitate the maintenance of a safety case. The technique comprises 7 steps that are distributed between the Sensitivity ANalysis for Enabling Safety Argument Maintenance (SANESAM) phase and the safety argument maintenance phases as shown in Figure 2. The steps of the SANESAM phase are represented along the upper path, whilst the lower path represents the steps of the safety argument maintenance phase. The SANESAM phase, however, is what is being discussed in this paper.

A complete approach to managing safety case change would include both (a) mechanisms to structure the argument so as to contain the impact of predicted changes and (b) means of assessing the impact of change on all parts of the argument [8]. As discussed in Section 1, system changes and their details cannot be fully predicted and made available up front. Predicting potential changes to the software aspects of a safety case is even harder than other parts because software is highly changeable and harder to manage. Consequently, considering a complete list of anticipated changes is difficult. What can be easier though is to determine the flexibility (or compliance) of each component to changes. This means that regardless of the type of changes the latter will be seen as factors to increase or decrease a certain parameter value. Thus system developers can focus more on predicting those changes that might make the parameter value inadmissible.

The rationale of our technique is to determine, for each 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.). To this end, we use the sensitivity analysis as a method to determine the range of failure probability parameter for each component. Hence, the technique assumes the existence of a probabilistic FTA where each event in the tree is specified by an actual (i.e., current) failure probability $FP_{Actual|event(x)}$. In addition, the technique assumes the existence of the required failure probability for

the top event $FP_{Required(Topevent)}$, where the FTA is considered unreliable if: $FP_{Actual(Topevent)} > FP_{Required(Topevent)}$. The steps of the SANESAM phase are as follows:

- ***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 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"[14].*
  2. Calculate the maximum possible increment in the failure probability parameter of event $x$ before the top event $FP_{Actual(Topevent)}$ is no longer met, where $x \in MC$ and
  $$(FP_{Increased|event(x)} - FP_{Actual|event(x)}) \nRightarrow FP_{Actual(Topevent)} > FP_{Required(Topevent)}.$$
  3. Rank the sensitive events from the most sensitive to the less sensitive. The most sensitive event is the event for which the following equation is the minimum:
  $$(FP_{Increased|event(x)} - FP_{Actual|event(x)})/FP_{Actual|event(x)}$$

- ***Step 2. Refine the identified sensitive parts with system developers***: In this step, the generated list from Step 1 should be discussed with system developers (e.g., safety engineers) and ask them to 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 may be removed from the list or the rank of some of them change.

- ***Step 3. Derive safety contracts from FTAs***: In this step, the refined list from Step 2 is used as a guide to derive the safety contracts, where each event in the list should have at least one contract. The main objective 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. We create a template to document the derived safety contracts as shown in Figure 3a, where G

and A stand for Guarantee and Assumption, respectively. Furthermore, each safety contract should contain a version number (it is shown as **V** in Figure 3a). The version number of the contract should match the *artefact version number* (as described in the next step), otherwise it will be considered out of date. We also introduce a new notation to the FTA to annotate the contracted events where every created contract should have a unique identifier, see Figure 3b.

- ***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. In order to associate the derived safety contracts with GSN arguments, we reuse our previous work [8]. The essence of that work is storing additional information in the safety argument to facilitate identifying the evidence impacted by change. This is done by annotating each reference to a development artefact (e.g. an architecture specification) in a goal or context element with an *artefact version number*. Also by annotating each solution element with:
  1. An *evidence version* number
  2. An *input manifest* identifying the inputs (including version) from which the evidence was produced
  3. The *lifecycle phase* during which the evidence obtained (e.g. Software Architecture Design)
  4. A *safety standard reference* to the clause in the applicable standard (if any) requiring the evidence (and setting out safety integrity level requirements)

  However, the approach description, just as it is, does not support associating our derived safety contracts in ***Step 3*** with the safety argument without proper adjustments. Hence, a set of rules are introduced to guide the reuse of the approach in the work of this paper, as follows:

  1. GSN element names should be unique.
  2. At least one GSN goal should be created for each guarantee (i.e., for each safety contract). Moreover, the contract should be annotated in the goal which is made for it. The annotation should be done by using the contract ID and the notation in Figure 3b.

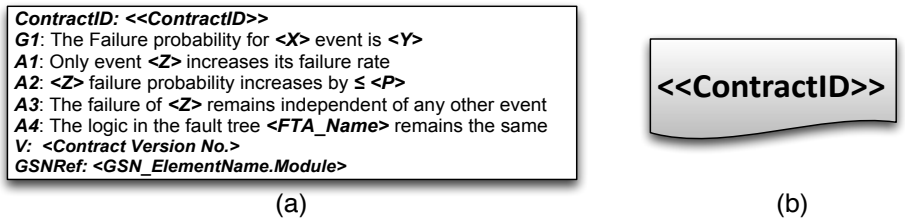


**Fig. 3.** (a) Safety Contract Template (b) Safety Contract Notation for FTA

3. Assumptions in each safety contract should be restricted to one event only. If the guarantee requires assumptions about another event, a new contract should be created to cover these assumptions.
4. An event in the assumptions list of a safety contract may be also used as a goal in the argument. In this case, the goal name should be similar to the event name.
5. Each safety contract should contain the GSN reference within it. The reference is the unique name of the GSN element followed by a dot and the name of the GSN module (if modular GSN is used). It is worth noting that while documenting the safety contracts, the GSN references might not be available as the safety argument itself might not be built yet. Hence, whenever GSN references are made available, system developers are required to revisit each contract and add the corresponding GSN reference to it. GSN reference parameter is shown as **GSNRef** in Figure 3a.

It is worth saying that the technique shall not affect the way GSN is being produced but it brings additional information for developers' attention.

## 4 An Illustrative Example: The Wheel Braking System (WBS)

In this section, we illustrate the proposed technique and its key concepts using the hypothetical aircraft braking system described in Appendix L of Aerospace Recommended Practice ARP-4761 [1]. Figure 4 shows a high-level architecture view of the WBS

### 4.1 Wheel Braking System (WBS): System Description

The WBS is installed on the two main landing gears. The main function of the system is to provide wheel braking as commanded by the pilot when the aircraft is on the ground. The system is composed of three main parts: Computer-based part which is called the Brake System Control Unit (*BSCU*), *Hydraulic* part, and *Mechanical* part.

The *BSCU* is internally redundant and consists of two channels, *BSCU System 1 and 2* (*BSCU* is the box in the gray background in Figure 4). Each channel consists of two components: *Monitor* and *Command*. *BSCU System 1 and 2* receive the same pedal position inputs, and both calculate the command value. The two command values are individually monitored by the *Monitor 1 and 2*. Subsequently, values are compared and if they do not agree, a failure is reported. The results of both *Monitors* and the compared values are provided to a the *Validity Monitor*. A failure reported by either system in the *BSCU* will cause that system to disable its outputs and set the *Validity Monitor* to invalid with no effect on the mode of operation of the whole system. However, if both monitors report failure the *BSCU* is deemed inoperable and is shut down [11].

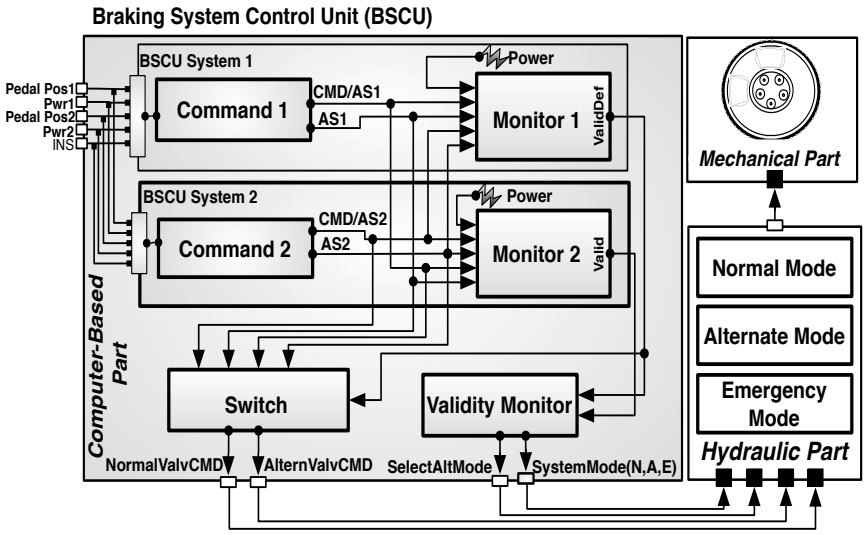


**Fig. 4.** A high-level view of the WBS

It worth noting that Figure 4 shows high-level view of the *BSCU* implementation and it omits many details. However, the figure is still sufficient to illustrate key elements of our technique. More details about the *BSCU* implementation can be found in ARP-4761 [1].

### 4.2 Applying the Technique

Before we can apply the technique, both the required and actual failure probabilities of the top event should be clearly defined, where $FP_{Required(Topevent)} > FP_{Actual(Topevent)}$. Appendix L of the ARP-4761 states, as a safety requirement on the *BSCU*, that: *"The probability of BSCU fault causes Loss of Braking Commands shall be less than 3.30E-5 per flight"*. This means that: $FP_{Required(Topevent)} < \mathbf{3.30E\text{-}5}$. In line with this, we assumed that the $FP_{Actual(Topevent)} \approx \mathbf{1.50E\text{-}6}$. Figure 5 shows the "Loss of Braking Commands" probabilistic FTA.

- ***Step 1. Apply the sensitivity analysis to the "Loss of Braking Commands" probabilistic FTA***: the following steps were performed to apply the sensitivity analysis:
  1. Find minimal cut set $MC$ in the FTA: there are several algorithms to find the $MC$. We apply Mocus cut set algorithm [14], as follows:

     $$\mathbf{MC} = \{\mathbf{BSVMIRFCSTA} + \mathbf{SWFSIIP} + (\mathbf{BSS1EF} * \mathbf{BSS2EF}) + (\mathbf{BSS1EF} * \mathbf{BSS2PSF}) + (\mathbf{BSS1EF} * \mathbf{SWFSIS1P}) + (\mathbf{BSS1PSF} * \mathbf{BSS2EF}) + (\mathbf{BSS1PSF} * \mathbf{BSS2PSF}) + (\mathbf{BSS1PSF} * \mathbf{SWFSIS1P}) + (\mathbf{BSS2EF} * \mathbf{SWFSIS2P}) + (\mathbf{BSS2PSF} * \mathbf{SWFSIS2P})\}.$$

2. A simple C program was coded to calculate the maximum possible failure probability $FP_{Increased|event(x)}$ for each event in the $MC$. Subsequently, the $FP_{Actual|event(x)}$ was subtracted from the $FP_{Increased|event(x)}$ to obtain $\Delta FP$ for each event. Table 1 shows the calculated $FP_{Increased|event(x)}$ and $\Delta FP$.
3. Applying the sensitivity equation: $(FP_{Increased|event(x)} - FP_{Actual|event(x)})/FP_{Actual|event(x)}$ determines the sensitivity for $x$ where $x \in MC$. Table 1 shows the sensitivity values and the ranking, where 1 indicates the most sensitive event.

- ***Step 2. Refine the identified sensitive parts with system developers***: the WBS is a hypothetical system and no discussions have been made with the system developers. For the sake of the example, however, a pessimistic decision was made to consider all the events in Table 1 as liable to change. It is worth noting that in more complex examples the volume of sensitive event lists will be quite big. Hence, discussing those lists with system developers can lead to more selective events and thus alleviating the number of safety contracts.

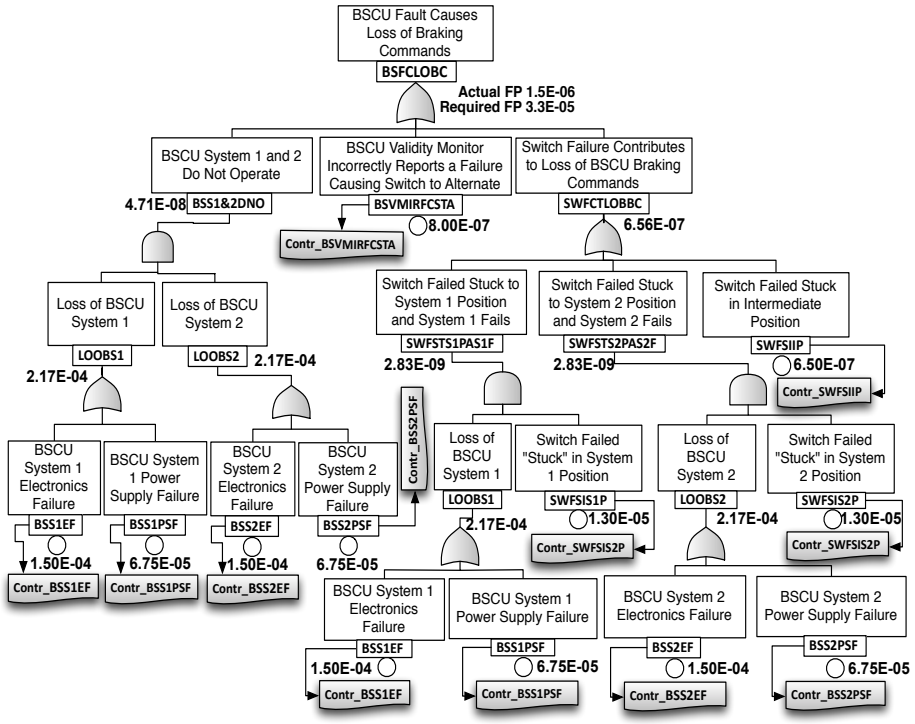


**Fig. 5.** Loss of Braking Commands FTA

**Table 1.** The results of the sensitivity analysis

| ***Event*** | $FP_{Actual\|event(x)}$ | $\approx \Delta FP$ | $FP_{Increased\|event(x)}$ | ***Sensitivity*** | ***Rank*** |
|---|---|---|---|---|---|
| BSVMIRFCSTA | 8.00E-07 | 3.150E-05 | 3.2304E-05 | 39 | 1 |
| SWFSIIP | 6.50E-07 | 3.150E-05 | 3.2154E-05 | 48 | 2 |
| SWFSIS1P | 1.30E-05 | 1.448E-01 | 1.4484E-01 | 51182 | 5 |
| SWFSIS2P | 1.30E-05 | 1.448E-01 | 1.4484E-01 | 51182 | 5 |
| BSS1EF | 1.50E-04 | 1.448E-01 | 1.4498E-01 | 965 | 3 |
| BSS1PSF | 6.75E-05 | 1.448E-01 | 1.4490E-01 | 2145 | 4 |
| BSS2EF | 1.50E-04 | 1.448E-01 | 1.4498E-01 | 965 | 3 |
| BSS2PSF | 6.75E-05 | 1.448E-01 | 1.4490E-01 | 2145 | 4 |

- ***Step 3. Derive safety contracts from "Loss of Braking Commands" FTA***: based on the list of the sensitive events from Step 2, a safety contract was derived for each event in the list. The introduced safety contract template in Figure 3a was used to demonstrate the derived safety contracts. For lack of space, we show only one example of the eight derived safety contracts (see Figure 6).

***ContractID: Contr_BSVMIRFCSTA***
***G1***: The Failure probability for the top event ***BSFCLOBC ≤ 3.30E-05***
***A1***: Only event ***BSVMIRFCSTA*** increases its failure rate
***A2***: ***BSVMIRFCSTA*** failure rate increases by ***≤ 3.2304E-05***
***A3***: The failure of ***BSVMIRFCSTA*** remains independent of any other event
***A4***: The logic in the fault tree "***Loss of Braking Commands***" remains the same
***V:*** V1.0
***GSNRef:*** BSCUAllFailures.WBSSafety

**Fig. 6.** A derived safety contract

- ***Step 4. Build the safety argument for the BSCU and associate the derived contracts with it***: a safety argument fragment was built as shown in Figure 7. The derived safety contracts are associated with the derived safety contracts according to ***Steps 4*** in Section 3. *BSCUAllFailures* claims that *BSCU* faults cause Loss of Braking commands are sufficiently managed. The possible faults of *BSCU*, based on "Loss of Braking Commands" FTA, are addressed by the subgoals below the *ArgAllCaus* strategy. Hence, *BSCUAllFailures* represents the top event of the FTA and thus the derived safety contracts are associated with it. The single black star on the left refers to the notation that is used to associate the contracts with *BSCUAllFailures*. It is important to note that goals in the gray color background represent assumptions in the safety contract. Each goal of those has the same name of the event in the assumptions list of the corresponding contract. For instance, *BSVMIRFCSTA* is a goal that represents an assumption within

*Contr_BSVMIRFCSTA* contract which, in turn, guarantees a property for another event.
The double black stars in the lower right corner refer to that annotation which is described in Section 3. It is important to make sure that contracts, related artefacts and items evidence have the same version number. The main idea of having the information within this notation is to pave the way to highlight the impact of changes. However, this idea will be described for the last three steps of the technique which is left for future work.

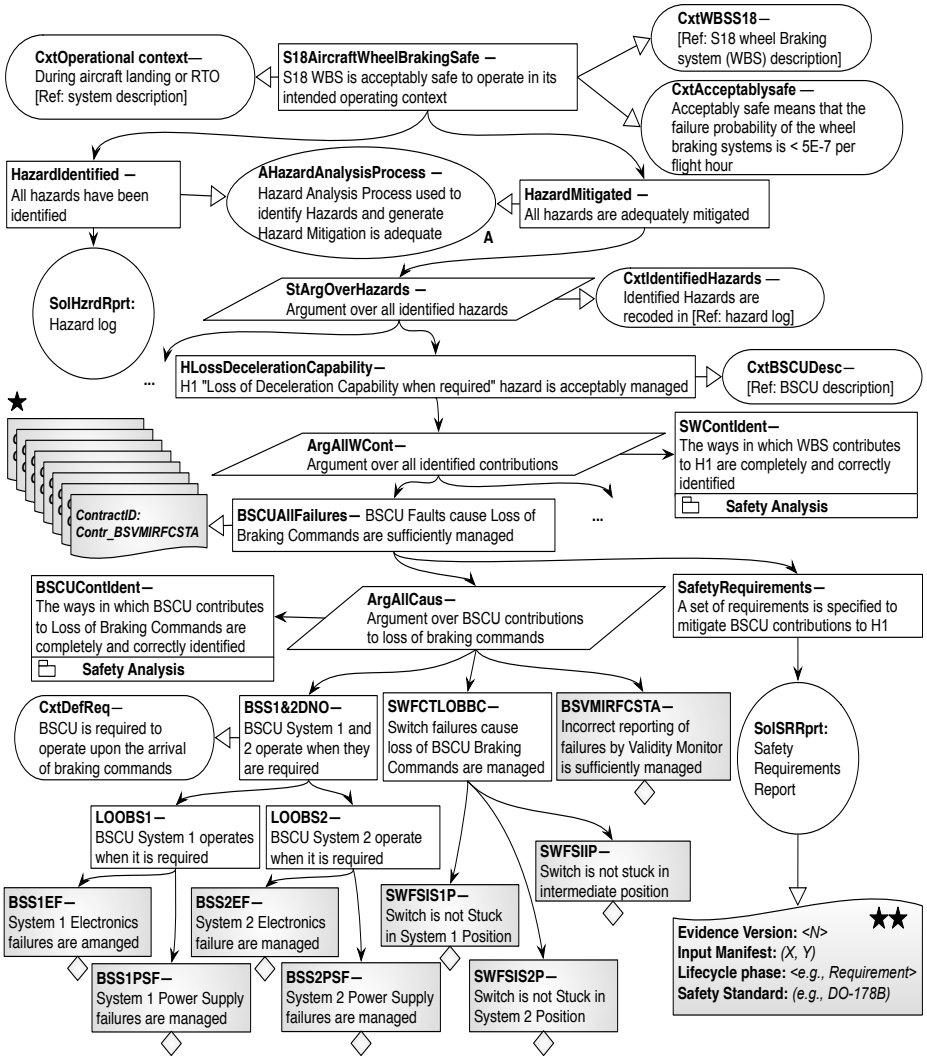


**Fig. 7.** Safety argument fragment for WBS

## 5 Related Work

A consortium of researchers and industrial practitioners called the Industrial Avionics Working Group (IAWG) has proposed using modular safety cases as a means of containing the cost of change. IAWGs Modular Software Safety Case (MSSC) process facilitates handling system changes as a series of relatively small increments rather than occasional major updates. The process proposes to divide the system into a set of blocks [2][5]. Each block may correspond to one or more software components but it is associated to exactly one dedicated safety case module. Engineers attempt to scope blocks so that anticipated changes will be contained within argument module boundaries. The process establishes component traceability between system blocks and their safety argument modules using Dependency-Guarantee Relationships (DGRs) and Dependency-Guarantee Contracts (DGCs). Part of the MSSC process is to understand the impact of change so that this can be used as part of producing an appropriate argument. The MSSC process, however, does not give details of how to do this. The work in this paper addresses this issue.

Kelly [9] suggests identifying preventative measures that can be taken when constructing the safety case to limit or reduce the propagation of changes through a safety case expressed in goal-structure terms. For instance, developers can use broad goals (goals that are expressed in terms of a safety margin) so that the these goals might act as barriers to the propagation of change as they permit a range of possible solutions. A safety case therefore, interspersed with such goals at strategic positions in the goal structure could effectively contain "firewalls" to change. Some of these initial ideas concerning change and maintenance of safety cases have been presented in [15]. However, no work was provided to show how these thoughts can facilitate the maintenance of safety cases.

## 6 Conclusion and Future Work

Changes are often only performed years after the initial design of the system making it hard for the designers performing the changes to know which parts of the argument are affected. Using contracts to manage system changes is not a novel idea any more since there has been significant work discusses how to represent contracts and how to use them. However, there has been little work on how to derive them. In this paper, we proposed a technique in which we showed a way to derive safety contracts using the sensitivity analysis. We also proposed a way to map the derived safety contracts to a safety argument to improve the change impact analysis on the safety argument and eventually facilitate its maintenance. Future work will focus on describing the last three steps of the technique. Also, creating a case study to validate both the feasibility and efficacy of the technique is part of our future work.

## Acknowledgment

We acknowledge the Swedish Foundation for Strategic Research (SSF) SYNOPSIS Project for supporting this work. We thank Patrick Graydon for his help and fruitful discussions of this paper.

## References

1. SAE ARP4761 Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment, Dec. 1996.
2. Modular software safety case (MSSC) — process description. https://www.amsderisc.com/related-programmes/, Nov 2012.
3. 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.
4. A. Cullen and H. Frey. *Probabilistic techniques in Exposure assessment.* Plenum Press, New York, 1999.
5. J. L. Fenn, R. D. Hawkins, P. Williams, T. P. Kelly, M. G. Banner, and 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.
6. GSN Community Standard: (http://www.goalstructuringnotation.info/) Version 1; (c) 2011 Origin Consulting (York) Limited.
7. ISO 26262:2011. Road Vehicles — Functional Safety, Part 1-9. International Organization for Standardization, Nov 2011.
8. O. Jaradat, P. J. 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)*, August 2014.
9. T. Kelly. Literature survey for work on evolvable safety cases. Department of Computer Sceince, University of York, 1st Year Qualifying Dissertation, 1995.
10. T. Kelly and J. McDermid. A systematic approach to safety case maintenance. In *Computer Safety, Reliability and Security*, volume 1698, pages 13–26. Springer Berlin Heidelberg, 1999.
11. O. Lisagor, M. Pretzer, C. Seguin, D. J. Pumfrey, F. Iwu, and T. Peikenkamp. Towards safety analysis of highly integrated technologically heterogeneous systems – a domain-based approach for modelling system failure logic. In *The 24th International System Safety Conference (ISSC)*, Albuquerque, USA, 2006.
12. B. Meyer. *Object-Oriented Software Construction.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1st edition, 1988.
13. D. J. Pannell. Sensitivity analysis of normative economic models: theoretical framework and practical strategies. *Agricultural Economics*, 16(2):139 – 152, 1997.
14. M. Rausand and A. Høyland. *System Reliability Theory: Models, Statistical Methods and Applications.* Wiley-Interscience, Hoboken, NJ, 2004.
15. S. P. Wilson, T. P. Kelly, and J. A. McDermid. Safety case development: Current practice, future prospects. In *proc. of Software Bases Systems - 12th Annual CSR Workshop.* Springer-Verlag, 1997.