Mälardalen University Press Dissertations
No. 280

CONTRACTS-BASED MAINTENANCE OF SAFETY CASES

Omar Jaradat

2018

School of Innovation, Design and Engineering

ISBN 978-91-7485-417-6
ISSN 1651-4238
Printed by E-Print AB, Stockholm, Sweden

Mälardalen University Press Dissertations
No. 280

CONTRACTS-BASED MAINTENANCE OF SAFETY CASES

Omar Jaradat

Akademisk avhandling

som för avläggande av teknologie doktorsexamen i datavetenskap vid
Akademin för innovation, design och teknik kommer att offentligen försvaras
måndagen den 3 december 2018, 09.30 i Kappa, Mälardalens högskola, Västerås.

Fakultetsopponent: Senior Lecturer Mark Nicholson, University of York

Akademin för innovation, design och teknik

Abstract
Safety critical systems are those systems whose failure could result in loss of life, significant property damage, or damage to the environment. System safety is a major property that shall be adequately assured to avoid any severe outcomes in safety critical systems. Safety assurance should provide justified confidence that all potential risks due to system failures are either eliminated or acceptably mitigated. System developers in many domains (e.g., automotive, avionics, railways) should provide convincing arguments regarding the safe performance of their systems to a national or international regulatory authority and obtain approvals before putting the system into service. Building 'Safety cases' is a proven technique to argue about and communicate systems' safety and it has become a common practice in many safety critical system domains. System developers use safety cases to articulate claims about how systems meet their safety requirements and objectives, collect and document items of evidence, and construct a safety argument to show how the available items of evidence support the claims.

Safety critical systems are evolutionary and constantly subject to preventive, perfective, corrective or adaptive changes during both the development and operational phases. Changes to any part of those systems can undermine the confidence in safety since changes can refute articulated claims about safety or challenge the supporting evidence on which this confidence relies. Hence, safety cases need to be built as living documents that should always be maintained to justify the safety status of the associated system and evolve as these systems evolve. However, building safety cases are costly since they require a significant amount of time and efforts to define the safety objectives, generate the required evidence and conclude the underlying logic behind the safety case arguments. Safety cases document highly dependent elements such as safety goals, assumptions and evidence. Seemingly minor changes may have a major impact. Changes to a system or its environment can necessitate a costly and painstaking impact analysis for systems and their safety cases. In addition, changes may require system developers to generate completely new items of evidence by repeating the verification activities. Therefore, changes can exacerbate the cost of producing and maintaining safety cases.

Safety contracts have been proposed as a means for helping to manage changes. There have been works that discuss the usefulness of contracts for reusability and maintainability. However, there has been little attention on how to derive them and how exactly they can be utilised for system or safety case maintenance.

The main goal of this thesis is to support the change impact analysis as a key factor to enhance the maintainability of safety cases. We focus on utilising safety contracts to achieve this goal. To address this, we study how safety contracts can support essential factors for any useful change management process, such as (1) identifying the impacted elements and those that are not impacted, (2) minimising the number of impacted safety case elements, and (3) reducing the work needed to make the impacted safety case elements valid again. The preliminary finding of our study reveals that using safety contracts can be promising to develop techniques and processes to facilitate safety case maintenance. The absence of safety case maintenance guidelines from safety standards and the lack of systematic and methodical maintenance techniques have motivated the work of this thesis. Our work is presented through a set of developed and assessed techniques, where these techniques utilise safety contracts to achieve the overall goal by various contributions. We begin by a framework for evaluation of the impact of change on safety critical systems and safety cases. Through this, we identify and highlight the most sensitive system components to a particular change. We propose new ways to associate system design elements with safety case arguments to enable traceability. How to identify and reduce the propagation of change impact is addressed subsequently. Our research also uses safety contracts to enable through-life safety assurance by monitoring and detecting any potential mismatch between the design safety assumptions and the actual behaviour of the system during its operational phase. More specifically, we use safety contracts to capture thresholds of selected safety requirements and compare them with the runtime related data (i.e., operational data) to continuously assess and evolve the safety arguments.

In summary, our proposed techniques pave the way for cost-effective maintenance of safety cases upon preventive, perfective, corrective or adaptive changes in safety critical systems thus helping better decision support for change impact analysis.

ISBN 978-91-7485-417-6
ISSN 1651-4238

Abstract

Safety critical systems are those systems whose failure could result in loss of life, significant property damage, or damage to the environment. System safety is a major property that shall be adequately assured to avoid any severe outcomes in safety critical systems. Safety assurance should provide justified confidence that all potential risks due to system failures are either eliminated or acceptably mitigated. System developers in many domains (e.g., automotive, avionics, railways) should provide convincing arguments regarding the safe performance of their systems to a national or international regulatory authority and obtain approvals before putting the system into service. Building 'Safety cases' is a proven technique to argue about and communicate systems' safety and it has become a common practice in many safety critical system domains. System developers use safety cases to articulate claims about how systems meet their safety requirements and objectives, collect and document items of evidence, and construct a safety argument to show how the available items of evidence support the claims.

Safety critical systems are evolutionary and constantly subject to preventive, perfective, corrective or adaptive changes during both the development and operational phases. Changes to any part of those systems can undermine the confidence in safety since changes can refute articulated claims about safety or challenge the supporting evidence on which this confidence relies. Hence, safety cases need to be built as living documents that should always be maintained to justify the safety status of the associated system and evolve as these systems evolve. However, building safety cases are costly since they require a significant amount of time and efforts to define the safety objectives, generate the required evidence and conclude the underlying logic behind the safety case arguments. Safety cases document highly dependent elements such as safety goals, assumptions and evidence. Seemingly minor changes may have a major impact. Changes to a system or its environment can necessitate a costly

and painstaking impact analysis for systems and their safety cases. In addition, changes may require system developers to generate completely new items of evidence by repeating the verification activities. Therefore, changes can exacerbate the cost of producing and maintaining safety cases.

Safety contracts have been proposed as a means for helping to manage changes. There have been works that discuss the usefulness of contracts for reusability and maintainability. However, there has been little attention on how to derive them and how exactly they can be utilised for system or safety case maintenance.

The main goal of this thesis is to support the change impact analysis as a key factor to enhance the maintainability of safety cases. We focus on utilising safety contracts to achieve this goal. To address this, we study how safety contracts can support essential factors for any useful change management process, such as (1) identifying the impacted elements and those that are not impacted, (2) minimising the number of impacted safety case elements, and (3) reducing the work needed to make the impacted safety case elements valid again. The preliminary finding of our study reveals that using safety contracts can be promising to develop techniques and processes to facilitate safety case maintenance. The absence of safety case maintenance guidelines from safety standards and the lack of systematic and methodical maintenance techniques have motivated the work of this thesis. Our work is presented through a set of developed and assessed techniques, where these techniques utilise safety contracts to achieve the overall goal by various contributions. We begin by a framework for evaluation of the impact of change on safety critical systems and safety cases. Through this, we identify and highlight the most sensitive system components to a particular change. We propose new ways to associate system design elements with safety case arguments to enable traceability. How to identify and reduce the propagation of change impact is addressed subsequently. Our research also uses safety contracts to enable through-life safety assurance by monitoring and detecting any potential mismatch between the design safety assumptions and the actual behaviour of the system during its operational phase. More specifically, we use safety contracts to capture thresholds of selected safety requirements and compare them with the runtime related data (i.e., operational data) to continuously assess and evolve the safety arguments.

In summary, our proposed techniques pave the way for cost-effective maintenance of safety cases upon preventive, perfective, corrective or adaptive changes in safety critical systems thus helping better decision support for change impact analysis.

Swedish Summary

Säkerhetskritiska system är system där fel kan resultera i förlust av människoliv, betydande skada på egendom, eller skador på miljön. Systemsäkerhet är en viktig egenskap som måste säkerställas för att minimera riskerna för allvarliga fel i säkerhetskritiska system. Säkerställande av systemsäkerheten bör resultera i väl underbyggda argument för att alla potentiella risker som orsakas av systemfel eliminerats eller reducerats till en acceptabel nivå. Systemutvecklare inom många områden (t.ex. bilar, flyg, järnvägar) behöver tillhandahålla information om systemens säkerhetsstatus till en nationell eller internationell tillsynsmyndighet för att denna ska kunna bedöma systemet kan sättas i drift eller inte. Mer specifikt bör systemutvecklare skapa ”säkerhetsfall” bestående av (1) krav som om de är uppfyllda leder till att systemen är tillräckligt säkra och (2) bevis för att dessa krav är uppfyllda, i form av väl dokumenterade bevismaterial och säkerhetsargument som tydligt visar att detta bevismaterial innebär att kraven är uppfyllda. Att på detta sätt konstruera säkerhetsfall är en beprövad teknik för att argumentera för och kommunicera systemens säkerhet och är praxis inom många säkerhetskritiska områden.

Många säkerhetskritiska system är under ständig utveckling och föremål för förebyggande, förbättrande, och korrigerande förändringar under såväl utvecklings- som driftsfasen. ändringar av någon del av dessa system kan undergräva förtroendet för säkerheten, eftersom de kan ändra förutsättningarna för påståenden om säkerhet eller utmana de stödjande bevis som detta förtroende bygger på. Därför är säkerhetsfall byggda som levande dokument som ständigt behöver hållas uppdaterade för att motivera säkerhetsstatus för systemet. Konstruktion och underhåll av säkerhetsfall är dock kostsamt eftersom det krävs betydande tid och ansträngningar för att definiera säkerhetskraven, generera de nödvändiga bevisen och utforma den underliggande logiken bakom säkerhetsargumenten. Säkerhetsfall dokumenterar starkt ömsesidiga beroenden (t.ex. mellan säkerhetskrav, bevis och antaganden) och även

mindre förändringar kan ha stor inverkan. Förändringar i ett system eller dess miljö kan kräva en dyr och noggrann säkerhetsanalys för system och dess säkerhetsfall. Dessutom kan ändringar kräva att systemutvecklare genererar helt nya bevismaterial. Därför kan förändringar väsentligt öka kostnaden för att producera och bibehålla säkerhetsfall.

Säkerhetskontrakt har föreslagits som ett medel för att hjälpa till att hantera förändringar. Det finns forskning som diskuterar användbarheten av kontrakt för återanvändning och underhåll, men hur de ska härledas och exakt hur de kan användas vid systemunderhåll har fått mindre uppmärksamhet.

Huvudsyftet med denna avhandling är att ge stöd för analys av de effekter som systemförändringar har på systemsäkerheten. Vi använder säkerhetskontrakt för att uppnå detta mål. Specifikt studerar vi hur säkerhetskontrakt kan stödja analys av väsentliga faktorer i förändringshanteringen, såsom (1) identifiering av vilka delar som påverkas, respektive inte påverkas, (2) hur antalet påverkade delar kan minimeras och (3) hur det arbete som behövs för att göra säkerhetsfallet giltiga igen kan minimeras. Våra resultat indikerar att användandet av säkerhetskontrakt är en lovande metod för att utveckla tekniker och processer som underlättar underhåll av säkerhetsfall. Frånvaron av stöd och riktlinjer för detta i säkerhetsstandarder och brist på systematiska och metodiska underhållstekniker har motiverat denna avhandling. Vårt arbete presenteras i form av en uppsättning nyutvecklade och utvärderade metoder som använder säkerhetskontrakt för att uppnå det övergripande målet.

Den första metoden utgörs av ett ramverk för utvärdering av förändringars inverkan på säkerhetskritiska system och deras säkerhetsfall, vilket låter oss identifierar de systemkomponenter som är mest känsliga för en viss förändring. För att öka spårbarheten föreslår vi även nya sätt att associera systemkomponenter till specifika delar av motsvarande säkerhetsfall. Vårt nästa bidrag fokuserar på hur spridningen av effekterna av en förändring kan minskas. Vi använder säkerhetskontrakt för att säkerställa säkerheten under systemets hela livscykel. Genom övervakning kan vi upptäcka brister i överensstämmelsen mellan säkerhetsantaganden och systemets faktiska beteende under drift. Mer specifikt använder vi säkerhetskontrakt för att identifiera kritiska trösklar för utvalda säkerhetskrav och jämför dessa med motsvarande data (dvs operativa data) under drift för att kontinuerligt utvärdera och skapa förutsättningar för utveckling av säkerhetsfallen.

Sammanfattningsvis visar våra föreslagna metoder på en väg mot kostnadseffektivt underhåll av säkerhetsfall vid förebyggande, korrigerande eller adaptiv förändring i säkerhetskritiska system, vilket bidrar till bättre stöd för beslut i förändringsarbetet.

"O' Lord! Increase me in knowledge"
Holy Quran (20:114)

Acknowledgments

First and foremost, I am deeply grateful to my supervisors, Sasikumar Punnekkat, Iain Bate and Hans Hansson. Without your continuous help and support this thesis would not be possible. Sasikumar, you are a big source of hope and talking to you is always a successful way for me to think positively and make more educated decisions. Iain, you always help me to build a stronger self confidence and never underestimate what I can do, I owe you a debt of gratitude for all you have done for me. I want to express my gratitude to Kristina Lundqvist for her encouragement, recommendations and support during my master and PhD studies. Next, I want to thank Patrick Graydon[1], your patience, discussions and opinions are truly constructive and appreciated. Thank you all for supporting me in taking this PhD and for believing in me.

This thesis is the culmination of a long journey which was just like climbing a high peak step by step. This journey was accompanied with hardship, stress and frustration, and without the endless love, support and continuous encouragement of my parents, the peak was never reachable. Many thanks to my strong father and to my wonderful mother, I love you and always will do. Rawan, you are a great wife who is always, without hesitation, ready to motivate me whenever I am down, thanks for having my back! My sons Rayyan and Ibrahim, you are the secret of my patience to keep moving forward. I am sorry guys for ruining many of your weekends and school breaks. I promised you before to try not to do it again and I failed but I am asking for one more chance now. Special thanks to my lovely sister Arwa and my dear brothers Mohammad, Ahmad, Abdallah and Ali you are always there when I need you. I will probably be in trouble if I forget to thank my parents-in-law, thanks a lot for your continuous support, encouragement and delicious food.

In the same day when I set off on my PhD journey, Irfan Šljivo was another

[1] Patrick was my advisor since I started my PhD studies in Sep 2012 until Nov 2014

candidate who was setting off on his PhD journey at the same office. Since then Irfan and I became journey companions, officemates, project mates and friends or even brothers. We shared unforgettable good and tough times, stress, frustration, project trips, conferences, etc. A tremendous thank you goes to you Irfan for the memorable companionship. You have always been there to lend a helping hand when I stumble (I hope I did the same for you). Of course, I cannot forget one of my best friends Gabriel Campeanu who joined two journeys with me, a colleague during our MSc studies and an officemate during the PhD work. A very big thank you goes to you Gabriel, you never hesitate to help your friends whenever they need you.

I further thank all my co-authors and colleagues with whom I had the pleasure to work with during this time: Sasikumar Punnekkat, Iain Bate, Irfan Šljivo, Ibrahim Habli, Richard Hawkins, Abdallah Salameh, Svetlana Girs, Elena Lisova, Mohammad Ashjaei, Kester Clegg , Lorenzo Corneo, Vincenzo Gulisano and Yiannis Nikolakopoulos.

Next, I would like to thank the head of our division Radu Dobrin for his tips and support. I also want to thank the administrative staff, Malin Rosqvist, Carola Ryttersson, Sofia Jäderén, Susanne Fronnå, et al., for facilitating all paperworks and routines. I would like to thank all researchers at Mälardalen University for the wonderful moments we have shared in lectures, meetings and fika time (coffee breaks). I also owe a great debt of gratitude to my project mates (members of SYNOPSIS, SafeCOP and FiC) for fruitful meetings, discussions, disputes and support. I cannot leave out my office mates and friends, Husni Khanfar, Irfan Šljivo, Gabriel Campeanu, Filip Markovic and Julieth Patricia Castellanos Ardila. I want to thank the football gang who was warming up the cold and lazy weekends. Special thank you goes to Radu Dobrin for organising the games and for my brother-in-law Zaid Darwish for motivating me every week to join.

The work in this thesis has been supported by the Swedish Foundation for Strategic Research (SSF) via the projects SYNOPSIS[2] and FIC[3] as well as EU and VINNOVA via SafeCOP[4] project.

Omar T. Jaradat
October, 2018
Västerås, Sweden

[2]http://www.es.mdh.se/SYNOPSIS/
[3]http://www.es.mdh.se/fic
[4]http://www.safecop.eu/

List of Publications

Papers Included in the PhD Thesis

Paper A *Using Sensitivity Analysis to Facilitate The Maintenance of Safety Cases*, Omar Jaradat, Iain Bate, Sasikumar Punnekkat, In Proceedings of the 20th International Conference on Reliable Software Technologies (Ada-Europe), June 2015.

Paper B *Deriving Hierarchical Safety Contracts*, Omar Jaradat, Iain Bate, In Proceedings of the 21st IEEE Pacific Rim International Symposium on Dependable Computing (PRDC), Nov 2015.

Paper C *Using Safety Contracts to Guide the Maintenance of Systems and Safety Cases*, Omar Jaradat, Iain Bate, In Proceedings of the 13rd European Dependable Computing Conference (EDCC), Sep 2017.

Paper D *Using Safety Contracts to Verify Design Assumptions During Runtime*, Omar Jaradat, Sasikumar Punnekkat, In Proceedings of the 23rd International Conference on Reliable Software Technologies (Ada-Europe), June 2018.

Paper E *A Safety-Centric Change Management Framework by Tailoring Agile and V-Model Processes*, Abdallah Salameh and Omar Jaradat, In Proceedings of the 36th International System Safety Conference (ISSC), Aug 2018.

Related Papers Not Included in the PhD Thesis

1. *Automated Verification of AADL-Specifications Using UP-PAAL*, Andreas Johnsen, Kristina Lundqvist, Paul Pettersson, Omar Jaradat, In Proceedings of the 14th IEEE International Symposium on High Assurance Systems Engineering (HASE 2012).

2. *Towards a Safety-oriented Process Line for Enabling Reuse in Safety Critical Systems Development and Certification*, Barbara Gallina, Irfan Sljivo, Omar Jaradat, In Proceedings of the 35th Annual IEEE Software Engineering Workshop (FedCSIS Conference) (SEW-36 2012).

3. *The Role of Architectural Model Checking in Conducting Preliminary Safety Assessment*, Omar Jaradat, Patrick Graydon, Iain Bate, In Proceedings of the 31st International System Safety Conference (ISSC 2013).

4. *An Approach to Maintaining Safety Case Evidence After A System Change*, Omar Jaradat, Patrick Graydon, Iain Bate, In Proceedings of the 10th European Dependable Computing Conference (EDCC 2014)).

5. *Deriving Safety Contracts to Support Architecture Design of Safety Critical Systems*, Irfan Sljivo, Omar Jaradat, Iain Bate, Patrick Graydon, In Proceedings of the 16th IEEE International Symposium on High Assurance Systems Engineering (HASE 2015).

6. *Facilitating the Maintenance of Safety Cases*, Omar Jaradat, Iain Bate, Sasikumar Punnekkat, In Proceedings of the 3rd International Conference on Reliability, Safety and Hazard - Advances in Reliability, Maintenance and Safety (ICRESARMS 2015).

7. *Systematic Maintenance of Safety Cases to Reduce Risk*, Omar Jaradat, Iain Bate, In Proceedings of the 4th International Workshop on Assurance Cases for Software-intensive Systems (ASSURE 2016).

8. *Challenges of Safety Assurance for Industry 4.0*, Omar Jaradat, Irfan Sljivo, Ibrahim Habli, Richard Hawkins, In Proceedings of the 13rd European Dependable Computing Conference (EDCC 2017).

9. *Contract-Based Assurance for Wireless Cooperative Functions of Vehicular Systems*, Svetlana Girs, Irfan Sljivo, Omar Jaradat, In Proceedings of the 43rd Annual Conference of the IEEE Industrial Electronics Society (IECON 2017).

10. *Service Level Agreements for Safe and Configurable Production Environments*, Mohammad Ashjaei, Kester Clegg , Lorenzo Corneo , Richard Hawkins , Omar Jaradat, Vincenzo Gulisano , Yiannis Nikolakopoulos, In Proceedings of the 23rd International Conference on Emerging Technologies and Factory Automation (ETFA 2018).

11. *Using Safety Contracts to Guide the Maintenance of Systems and Safety Cases: An Example*, Omar Jaradat, Iain Bate, MRTC technical report, Mälardalen University, April 2017.

Contents

I

Thesis

Chapter 1

Introduction

Safety critical systems are those systems whose failure could result in loss of life, significant property damage or damage to the environment [1]. Assuring safety for such systems should provide justified confidence that all potential risks due to system failures are either eliminated or acceptably mitigated. Hence, all failures which might expose the manufacturing processes to hazards shall be analysed and controlled as part of pre-deployment safety assurance and monitored and controlled as part of operational phase.

The size and complexity of safety critical systems are considerable. Without adequate evidence to support the safe performance and clear demonstration of that performance, it is difficult for safety assessors or system developers themselves to build sufficient confidence in their safety critical systems. Therefore, developers of safety critical systems in several domains are required to demonstrate the safe performance of their systems through a reasoned argument that justifies why the system in question is acceptably safe (or will be so) [2], in the light of the available evidence. This argument is communicated via an artefact that is known as a *safety case*. Typically, a safety case comprises both safety evidence (e.g. safety analyses, software and hardware inspection reports, or functional test results) and a safety argument (i.e., reasoning) explaining that evidence. The safety argument shows how developers use available evidence to support safety claims and how those claims, in turn, support broader claims about system behaviour, hazards addressed, and, ultimately, acceptable safety [3].

An organisation building a safety case should be accountable for the ownership of the risks to be controlled by adopting an appropriate safety manage-

ment system, performing a hazard assessment, selecting appropriate controls, and implementing them [4]. In order to help building a sufficient and credible (i.e., on a scientific basis) confidence in the safe performance of a system, its safety case shall always communicate the safe performance of the system, and shall always contain only acceptable items of evidence that the system meets its safety requirements.

Moreover, safety critical systems can be evolutionary as they are subject to changes due to perfective, corrective or adaptive maintenance or through technology obsolescence [5]. An item of evidence is valid only in the operational and environmental context in which it is obtained or to which it applies. Since changes to a system during or after its development may add, remove or modify its operational or environmental assumptions, changes may undermine the collected items of evidence and thus defeat articulated safety claims. 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 [3]. 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 [6]. 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"* [7, 8]. 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"* [9, 8].

Furthermore, change to a safety case may necessitate many other consequential changes — creating a ripple effect [5]. 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. 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 the significance of how to deal with changes to the

system or its operational environment, as well as the 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 [5]. More clearly, some safety standards provide generic guidance of the change management and impact analysis on the system level, and briefly describe the expected outputs.

Using contracts has been around for a few decades in the system development domain [10]. Generally speaking, contracts are intended to describe functional and behavioural properties for each design component in the form of *assumptions* and *guarantees*. Contracts have been discussed as a means for helping to manage system changes in the system domain or in its corresponding safety case [11, 12, 13]. The cost of maintaining, reusing and changing software components is lessened when using contracts as developers may rework components with knowledge of the constraints placed upon them [14]. However, deriving safety contracts and their contents have received little support yet [15]. Also, most of the works that discuss contracts for system maintenance are limited to component-to-component contracts with not much focus on the dependencies between a component and its operational environment, related safety requirements and relevant safety argument.

In this thesis, we consider a safety case as a living document that should always be maintained to correctly portray the safety of a system [5], and evolves as the system evolves. The overall goal of this thesis is *to introduce new change management techniques in order to facilitate the maintenance of safety cases due to system or environment changes.* Our work focuses on: (1) How and where to derive safety contracts and their contents as a supportive means to our suggested change management techniques, (2) using the derived contracts to support the decision as to whether or not apply changes, (3) using the derived contracts to guide developers to the parts in the safety case that might be affected after applying a change, and (4) using the derived contracts to enable through-life safety assurance.

The rationale of our suggested techniques 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.). We use sensitivity and importance analyses to define safety thresholds (i.e., the maximum allowed change). Subsequently, we derive safety contract where the guarantees represent these thresholds. We also include in each contract whatever assumption that can violate a guaranteed threshold. We use the derived safety contracts to facilitate the accommodation of system changes in safety cases to ultimately support the maintainability of safety cases.

1.1 Thesis Outline

The thesis report is organised into two main parts. **Part I** includes five chapters. Chapter 1 provides an introduction to the thesis where an overview of the research problem, motivation and the thesis contributions are presented. In Chapter 2, we present background information and overview of the most prominent terms that appear frequently in this thesis. In Chapter 3, we describe the research scope, state-of-the-art, research problem, and derive the research goal and research questions. We also describe the overall methodology that is adopted to perform the research. In Chapter 4, we present the contributions of the research and reflect how they address the research questions. In Chapter 5, we draw the conclusion and describe possible directions for future work. **Part II** contains the research papers included in the thesis.

Paper A (Chapter 6): Using Sensitivity Analysis to Facilitate The Maintenance of Safety Cases, Omar Jaradat, Iain Bate, Sasikumar Punnekkat.
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."* [15]

Status: Published in Proceedings of the 20th International Conference on Reliable Software Technologies, Ada-Europe 2015.

My contribution: I was the main contributor of the work under supervision of the co-authors. My contributions include combining the results of sensitivity analysis together with the concept of contracts to identify the sensitive parts of a system and highlight these parts to help the experts to make an educated decision as to whether or not apply changes.

Paper B (Chapter 7): Deriving Hierarchical Safety Contracts, Omar Jaradat, Iain Bate, Sasikumar Punnekkat.
Abstract: *"Safety cases are costly since they need significant amount of time and efforts to produce. This cost can be dramatically increased (even for already certified systems) due to system changes as they require maintaining the safety case before it can be submitted for certification. Anticipating potential changes is useful since it reveals traceable consequences that will eventually reduce the maintenance efforts. However, considering a complete list of anticipated changes is difficult. What can be easier though is to determine the flexibility of system components to changes. Using sensitivity analysis is useful to measure the flexibility of the different system properties to changes. Furthermore, contracts have been proposed as a means for facilitating the change management process due to their ability to record the dependencies among system's components. In this paper, we extend a technique that uses a sensitivity analysis to derive safety contracts from Fault Tree Analyses and uses these contracts to trace changes in the safety argument. The extension aims to enabling the derivation of hierarchical and correlated safety contracts. We motivate the extension through an illustrative example within which we identify limitations of the technique and discuss potential solutions to these limitations."*

Status: Published in Proceedings of the 21st IEEE Pacific Rim International Symposium on Dependable Computing, PRDC 2015.

Main contribution: I was the main contributor of the work under Bate's supervision. My contribution comprises: (1) identifying possible limitations for the proposed technique in Paper A and (2) suggesting an extension to the technique to resolve the identified limitations.

Paper C (Chapter 8): Using Safety Contracts to Guide the Maintenance of Systems and Safety Cases, Omar Jaradat and Iain Bate.
Abstract: *"Changes to safety critical systems are inevitable and can impact the safety confidence about a system as their effects can refute articulated claims about safety or challenge the supporting evidence on which this confidence relies. In order to maintain the safety confidence under changes, system developers need to re-analyse and re-verify the system to generate new valid items of evidence. Identifying the effects of a particular change is a crucial step in any change management process as it enables system*

developers to estimate the required maintenance effort and reduce the cost by avoiding wider analyses and verification than strictly necessary. This paper presents a sensitivity analysis-based technique which aims at measuring the ability of a system to contain a change (i.e., robustness) without the need to make a major re-design. The proposed technique exploits the safety margins in the budgeted failure probabilities of events in a probabilistic fault-tree analysis to compensate for unaccounted deficits or changes due to maintenance. The technique utilises safety contracts to provide prescriptive data for what is needed to be revisited and verified to maintain system safety when changes happen. We demonstrate the technique on an aircraft wheel braking system."

Status: Published In Proceedings of the 13rd IEEE European Dependable Computing Conference (EDCC), Geneva, Switzerland, December 2017.

Main contribution: I was the main contributor of the work under Bate's supervision. My contributions include introducing a new technique besides SANESAM and SANESAM+, which focuses on containing (i.e., localising) the potential changes in the smallest possible part of a system. More clearly, the technique in this paper makes use of the margins in the failure probability of some FTA events to backup the violated MAFP (Maximum Allowed Failure Probability) of other affected events due to a change.

Paper D (Chapter 9): Using Safety Contracts to Verify Design Assumptions During Runtime, Omar Jaradat and Sasikumar Punnekkat.
Abstract: *"A safety case comprises evidence and argument justifying how each item of evidence supports claims about safety assurance. Supporting claims by untrustworthy or inappropriate evidence can lead to a false assurance regarding the safe performance of a system. Having sufficient confidence in safety evidence is essential to avoid any unanticipated surprise during operational phase. Sometimes, however, it is impractical to wait for high quality evidence from a system's operational life, where developers have no choice but to rely on evidence with some uncertainty (e.g., using a generic failure rate measure from a handbook to support a claim about the reliability of a component). Runtime monitoring can reveal insightful information, which can help to verify whether the preliminary confidence was over- or underestimated. In this paper, we propose a technique which uses runtime monitoring in a novel way to detect the divergence between the failure rates (which were used in the safety analyses) and the observed failure rates in the operational life. The technique utilises safety contracts to provide prescriptive*

data for what should be monitored, and what parts of the safety argument should be revisited to maintain system safety when a divergence is detected. We demonstrate the technique in the context of Automated Guided Vehicles (AGVs)."

Status: Published In Proceedings of the 23rd International Conference on Reliable Software Technologies (Ada-Europe), Lisbon, Portugal, June 2018.

Main contribution: I was the main contributor of the work under Punnekkat's supervision. My main contributions includes: (1) A novel technique to continuously reassess the failure rates and use the results to suggest system changes or maintenance, (2) a new way to derive safety contracts to facilitate the traceability between the system design, safety analysis and the safety case, (3) an example of how to argue more compelling over the failure rate in the light of the derived evidence from the operational phase, and (4) an example of how to carry out a through-life safety assurance.

Paper E (Chapter 10): A Safety-Centric Change Management Framework by Tailoring Agile and V-Model Processes, Abdallah Salameh and Omar Jaradat.
Abstract: *"Safety critical systems are evolutionary and subject to preventive, perfective, corrective or adaptive changes during their lifecycle. Changes to any part of those systems can undermine the confidence in safety since changes can refute articulated claims about safety or challenge the supporting evidence on which this confidence relies. Changes to the software components are no exception. In order to maintain the confidence in the safety performance, developers must update their system and its safety case. Agile methodologies are known to embrace changes to software where agilists strive to manage changes, not to prevent them. In this paper, we introduce a novel framework in which we tailor a hybrid process of agile software development and the traditional V-model. The tailored process aims to facilitate the accommodation of non-structural changes to the software parts of safety critical systems. We illustrate our framework in the context of ISO 26262 safety standard."*

Status: Published In Proceedings of the 36th International System Safety Conference (ISSC), Arizona, USA, August 2018.

Main contribution: Myself and Salameh were the main drivers. The

main contribution in the paper is the introduction of *XP-Kan-Safe* as a novel maintenance framework to facilitate the accommodation process of software non-structural changes in safety critical systems by utilising the strengths of agile methods and the V-model. My contribution focused on deriving safety contracts from safety analyses and associate them with test cases and safety cases to enable a tri-directional change management process.

Chapter 2

Background

In this chapter, we provide background and overview of the most prominent terms that appear frequently in this thesis.

2.1 Safety Critical Systems

The word 'safety' means: *"The condition of being protected from or unlikely to cause danger, risk, or injury"* [16]. *Safety critical "is a term applied to a condition, event, operation, process or item that is essential to safe system operation or use, e.g., safety critical function, safety critical path, and safety critical component"* [17]. ***Safety critical systems*** *are those systems whose failure might endanger human life, lead to substantial economic loss, or cause extensive environmental damage* [1]. The operation of safety critical systems should be safe and, ideally, never cause severe consequences. However, developing an absolutely safe system is unattainable even if the project has an open budget. This is because severe consequences are typically linked to system faults and we cannot be 100% certain that a system is fault free. However, this shall not discourage the efforts that aim at assuring systems' safety.

The key to assuring safety is to eliminate hazards or to ensure that the consequences of these hazards are minimal. The word ***hazard*** in English is defined as: *"a potential source of danger"* [16]. In the context of safety critical systems, there are different suggestions to explain what the word *hazard* means. Some definitions suggest that a hazard is simply a system state that could lead to accidents. For example, Knight [18] indicates that the word *hazard* is an

abbreviation of ***hazard state*** and it means: *"a system state that could lead to an unplanned loss of life, loss of property, release of energy, or release of dangerous materials"*. Some other definitions suggest to consider the potential environmental conditions in the definition to clarify the relationship between hazards and accidents. For example, Leveson [19] define *hazard* as: *"a state or condition of a system that, combined with certain environmental conditions, could lead to accidents"*. Anyway, all definitions agree that a hazard is a system state in which an accident might occur. An ***accident*** can be defined as: *"An unplanned event or sequence of events which results in human death or injury, damage to property, or to the environment"* [20]. The measure of the probability that a system will cause an accident is referred to as ***risk***. The risk is assessed by considering the probability that someone/something will be harmed if exposed to a hazard (also known as ***exposure***) and the severity of that hazard.

Hazards are caused by malfunctioning behaviours (i.e., failures) [21]. A ***Failure*** is defined as: *"an event that occurs when the delivered service deviates from correct service"* [22]. Failures are caused by errors. An ***error*** is defined as: *a part of the total state of the system that may lead to its subsequent service failure* [22]. Finally, errors are caused by faults. A ***fault*** is defined as: *an adjudged or hypothesized cause of an error* [22].

Figure 2.1 illustrates some of the system safety concepts and how they relate to each other. The figure uses a scenario from an adaptive cruise control system[1] to exemplify these concepts.

Any process or activity that aims at assuring or improving systems' safety should identify and eliminate potential hazards of those systems. If the elimination is not possible then they should be mitigated to an acceptable level. *Preliminary Hazard Analysis (PHA)* can be done with only a description of the system's concept and functions. That is, PHA is typically used in the early stages of the system's lifecycle where not enough design details are available. PHA consists of four main tasks as follows [23]:

- Identify system hazards
- Translate system hazards into high-level system safety design constraints
- Assess hazards if required to do so
- Establish the hazard log

[1] Adaptive Cruise Control (ACC) system is an optional system for road vehicles that automatically adjusts the vehicle speed to maintain a safe distance from vehicles ahead.

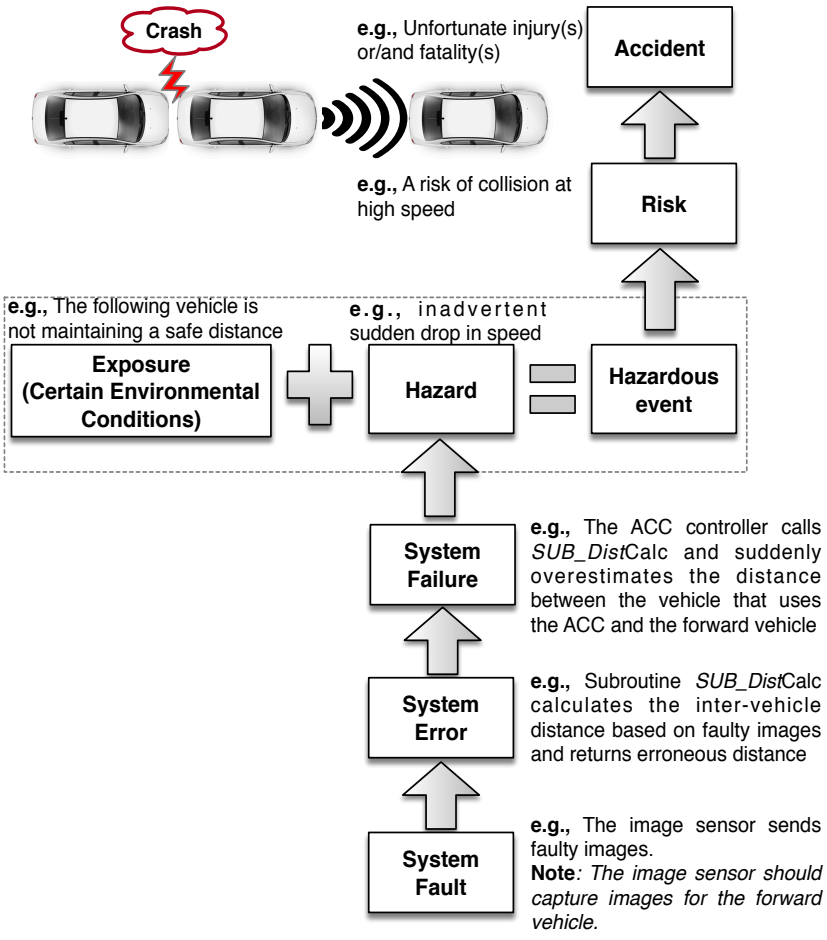

Figure 2.1: Overview of basic system safety concepts

Safety functions (also know as *safety barriers*) shall be identified, implemented and verified to achieve or maintain the safe state of a system (with respect to the identified hazards). These functions can be safeguards, countermeasures, or protection layers (e.g., car collision avoidance system, fire and gas detection system, pressure relief system, emergency shutdown system). The reliability of such functions are crucial to achieve safety. Reliability here

means *"the ability of the item to perform a required function, under given environmental and operational conditions and for a stated period of time"* [24]. However, safety should not be confused with reliability. A reliable system can be unsafe and vice versa. The software of a reliable system may still behave in such a way that the resultant system behavior leads to an accident.

The Lufthansa flight 2904 accident can be an example of how a reliable system may be unsafe. The plane was landing at Warsaw airport in Poland when the computer-controlled braking system did not work. While landing, the braking system did not recognise that the plane touched the ground and assumed that the aircraft was still airborne. A safety feature on the aircraft had stopped the deployment of the reverse thrust system, which slows down the aircraft. The plane ran off the end of the runway, hit an earth bank, and caught fire [20]. The investigations revealed that the braking system software was reliable and had operated according to its specification, but this did not lead to a safe system [20].

2.2 Safety Analysis

In order to assure that their systems perform as needed and provide acceptable levels of safety, safety engineers need to find the causal dependencies between system level hazards and failures of individual components. The automotive safety standard ISO 26262, for instance, states that the objectives of safety analyses are to 1) examine the consequences of faults and failures on the functions, behaviour and design of items and elements, and 2) provide information on conditions and causes that could lead to the violation of a safety goal or safety requirement [21]. Hence, all potential failures that can contribute to identified hazards shall be identified, analysed and managed. To this end, a wide variety of safety analysis techniques with different methodologies, are available. Moreover, there are different criteria used to distinguish these techniques (e.g., qualitative vs. quantitative, formal vs. informal, deductive vs. inductive). The work in this thesis, employs, to a large extent, the fault tree analysis technique. The work also employs the failure mode and effects analysis technique but to a smaller extent.

2.2.1 Failure Mode and Effects Analysis (FMEA)

The first formal FMEAs were conducted in the aerospace industry in the mid-1960s and were specifically focused on safety issues [25]. The goal with safety

FMEAs was, and remains today, to prevent safety accidents and incidents from occurring [25]. FMEA is a bottom-up analytical technique, and it usually depends on architectural system design or a functional block diagram to identify failure modes for each function in a system. The effects of the identified failure modes are described and, in some cases, assigned a probability based on the failure rate and failure mode ratio of the function or component. It is worth mentioning that performing FMEA is recommended by many safety standards from different domains. However, FMEA is not the best technique to investigate the effects of multiple failures. Hence, safety analysts usually combine it with other techniques (e.g., FTA) to perform more complete safety analysis.

2.2.2 Fault Tree Analysis (FTA)

In 1962, Bell Telephone Laboratories introduced the fault tree technique as a means to evaluate safety in the launching system of the intercontinental *Minuteman* missile [26]. The Boeing Company improved the technique and introduced computer programs for both qualitative and quantitative fault tree analysis. Today FTA is the most commonly used technique for safety and reliability studies.

FTA is a top-down failure analysis method which focuses on one particular undesired event and provides a method for determining causes of this event [27]. In other words, FTA is used to specify the occurrence of critical states (from a safety or reliability standpoint). These states might be associated with component hardware failures, human errors, software errors, or any other pertinent events. FTA helps safety engineers to identify plausible causes (i.e., faults) of undesired events [28].

A fault tree illustrates the logical interrelationships of the system's components (***Basic Events***) that lead to the undesired event or the system's state (***Top Event***) [28, 26]. These logical interrelationships are called ***Logical Gates***. Figure 2.2 shows the most commonly used FTA symbols.

Similar to FMEA, Performing FTA is recommended by many safety standards from different domains. FTA is also used as a method to achieve Probabilistic Safety Analysis (PSA) 2.2.3. More specifically, it is used to quantify system failure probability. Quantitative FT evaluation techniques produce three types of results: (1) numerical probabilities, (2) quantitative importance, and (3) sensitivity evaluations [27]. In this thesis, we exploit the results obtained by sensitivity evaluations to measure how sensitive a system design is to a particular aspect of individual event. Section 2.2.4 provides more details about sensitivity analysis.

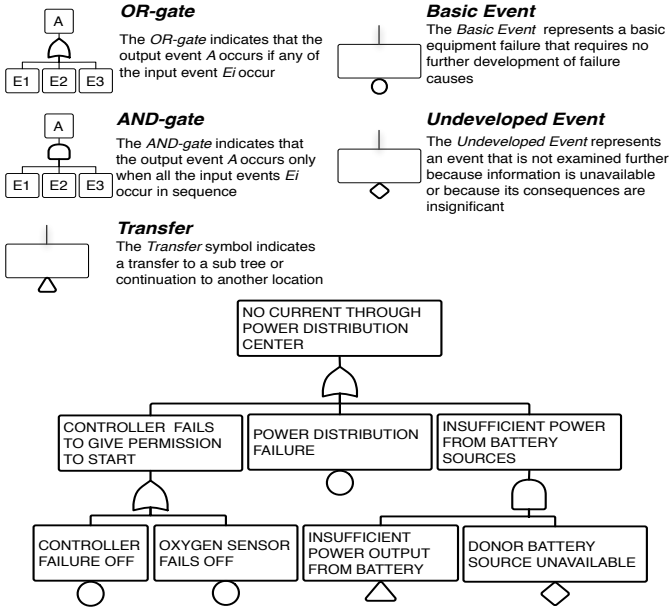

Figure 2.2: The principal FTA symbols and an instantiation [26, 29]

2.2.3 Probabilistic Safety Assessments (PSA)

PSA is a qualitative safety analysis to evaluate the probability that an accident might occur. More clearly, PSA is a technique used to numerically quantify risk measures by determining undesired scenarios as well as their likelihoods and consequences. Safety engineers typically starts PSAs by identifying undesired risk or scenario as a top event and investigate the causes that may lead to it. These causes include system failures that need to be identified and quantified using some models like FTA. Failure rates are typically used to quantify failures and they can be assigned to the basic events of the FTAs.

Failure rate is defined as: "*The total number of failures within an item population, divided by the total number of life units expended by that population, during a particular measurement period under stated conditions*" [30].

$$\lambda(t) = \frac{R(t) - R(t+\Delta t)}{\Delta t R(t)} \tag{2.1}$$

where $R(t)$ is the probability of a device not failing prior to some time t, t = t_1 and t_2 = t + Δt. FR is also the inverse of the Mean Time Between Failure (MTBF = $1/\lambda$) , which is generally measured as follows:

$$MTBF = \frac{C_n * T}{F_n} \tag{2.2}$$

where C_n = number of components, T = time period, F_n = number of failures.

The assigned failure rates to the basic events propagate up through the FTA logic to reach the failure rate of the top event.

The result of a PSA is important because it determines the minimum required levels of fault tolerance (e.g., redundancy). However, since the quality of PSA's results is dependent on the quality of the λ estimates, safety analysts should have clear confidence in the used λs in PSAs. For example, the IEC 61508 standard [31] requires failure rates with a 90% confidence level (λ 90%) in order to minimise the requirements for hardware fault tolerance. The standard accepts failure rates with only a 70% confidence level (λ 70%) but a fault tolerance mechanism should be considered.

It is not realistic to assume that all components' failure rates used in a PSA come from the operating experience of a specific system in a statistically meaningful way [32]. Typically, λs come from generic data sources (OREDA, SINTEF, SERH), where a generic failure rate (λ_G) of a particular device might vary from a source to another. The variation depends largely on the service, operating environment and maintenance practices [33]. Imprecise λ can contribute to cognitive impairment of safety. Nevertheless, generic λs are indispensable in any PSA and they are used in practice but the degree in which they are used varies from case to case [32]. More clearly, some PSAs are totally based on λ_G estimates while others use generic estimates as preliminary data that should be replaced later with more specialised data (system specific estimates). This implies that the quality of λ estimates should be reconsidered for a specific system in a specific environment.

Moreover, the failures of components in safety critical systems are typically divided into four modes, namely, Safe Detected (SD), Safe Undetected (SU), Dangerous Detected (DD), and Dangerous Undetected (DU) [34]. DD and DU failures can cause a loss of a safety function while we believe that we are protected and this might happen in fraction of diagnostic interval in case of DD failures or during the unknown downtime in case of DU failures [35]. DU failures are typically due to either random or systematic failures. However, we are after dangerous failures DD and DU. For SD and SU, on the other hand, we

will be aware that the safety function is unavailable to whatever reason (e.g., testing, repair, planned preventive maintenance) and we can take precautions to avoid hazardous situations [35]. Whenever FTAs are constructed to evaluate hazards, the basic event failure data must describe only failures that contribute to that hazard and thus only dangerous failure rates (λ_D) should be included for the basic events, where $\lambda_D = \lambda_{DD} + \lambda_{DU}$.

However, the reliability measure of any hazard that is being investigated by a FTA is not determined by simply calculating its λ_D. In fact, the Average Probability of Dangerous Failure on Demand (PFD_{Avg}) is used to determine the reliability measure of the top event (i.e., hazard). The PFD_{Avg} is basically a number from 0 to 1 which indicates the likelihood of a safety function to fail to operate on demand, where λ_D is just a variable within the PFD_{Avg} equation. Typically, safety standards specify the minimum allowable PFD_{Avg} of a safety function in order to meet its safety requirement based on SIL, ASIL, DAL, etc. More clearly, the failure measure is defined by PFD_{Avg} of a safety function and the result is compared to predefined target PFD_{Avg} for determined SIL of that function in safety standards. There are different formulae used to calculate PFD_{Avg} depending on different factors, such as system's structure (K-out-of-N structures), Common Cause Factor (CCF), operational maintenance, etc.

2.2.4 Sensitivity Analysis

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"* [36]. The analysis helps to establish reasonably acceptable confidence in the model by studying the uncertainties that are often associated with variables in models. Many variables in system analysis or design models represent quantities that are very difficult, or even impossible to measure to a great deal of accuracy. In practice, system, developers are usually uncertain about variables in the different system models and they estimate those variables. Sensitivity analysis allows system developers to determine what level of accuracy is necessary for a parameter (variable) to make the model sufficiently useful and valid [37].

There are different purposes for using sensitivity analysis. The analysis can be used to provide insight into the robustness of model results when making decisions [38]. Also, the analysis can be used to enhancing communication from modelers to decision makers, for example, by making recommendations more credible, understandable, compelling or persuasive [39]. In safety domains, sensitivity analysis can be used in risk analysis models to determine the

most significant exposure or risk factors so to speak, and thus, it can support the prioritisation of the risk mitigation. Sensitivity analysis methods can be classified in different ways such as mathematical, graphical, statistical, etc. In this paper we use the sensitivity analysis to identify the safety argument parts (i.e., sensitive parts) that might require unneeded painstaking work to update with respect to the benefit of a given change. The results of the analysis should be presented in the safety argument so that it is always available up front to get developers' attention.

In this thesis, 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 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. A sensitive event is an event whose failure probability value can significantly influence the validity of the FTA once it increases. In this this, system components whose failure rates correspond to FTA's events whose likelihoods are sensitive are referred to as sensitive components. Hence, changes to a sensitive component cause a great impact to system design [15].

We use the sensitivity analysis as a method to determine the range of failure probability parameter for each event. Hence, our work 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, our work assumes the existence of the required failure probability for the top event $FP_{Required(Topevent)}$, where the FTA is considered unreliable if:
$FP_{Actual(Topevent)} > FP_{Required(Topevent)}$.

2.3 Safety Assurance and Certification

It is common to adopt a highly prescriptive approach to assure system safety, where safety assurance is demonstrated by showing compliance with the requirements set out as a prescribed process in a safety standard [40]. One purpose of safety assurance is to ensure that a rigorous process has been fol-

lowed to build a system and an adequate evidence from the audit, inspection and assessment activities, is available. Safety assurance should also continuously assess the adequacy of the identified risk controls and identify additional or modify existing controls to ensure the safety performance and, ultimately, maintain the acceptable safety levels.

2.3.1 Safety Case

In 1965, section 14 of the UK Nuclear Installations Act states:

"Without prejudice to any other requirements of the conditions attached to this license the licensee shall make and implement adequate arrangements for the production and assessment of safety cases consisting of documentation to justify safety during the design, construction, manufacture, commissioning, operation and decommissioning phases of the installation." [41]

Hence, the notion of building safety cases to justify safety is not new and it has been around for almost fifty years. In 1989, the British chemical industry requested from nuclear sites (according to the Control of the Industrial Major Accident Hazards (CIMAH) regulations) to generate a written report that should contain (1) facts about the site, and (2) reasoned arguments about the hazards and risks from the site [42]. This report was also known as a safety case. The objective of the report was to demonstrate to the UK Health and Safety Executive (HSE) that the site is satisfactory by listing the major hazards and risks and shows how they are adequately mitigated.

The development of the safety case as a means of demonstrating acceptable risk began in the nuclear industry but the application of this means was uncommon in other industries. For example, in the Clapham junction accident in Chapter 1, there was no safety case and although the transport system was allegedly mature, regulated and safe, British Rail could not demonstrate why their system was acceptably safe to operate [43]. From 1990s onwards the development of safety cases spread across many other major safety critical industries, such as the railways, offshore oil and gas facilities. [44].

2.3.2 Safety Case Definition

It is worth mentioning that in addition to the term 'safety case', there are different other terms such as '*Assurance Case*' and '*Safety Assurance Case*' that are, sometimes, used interchangeably. An assurance case is defined as: "*A*

reasoned and compelling argument, supported by a body of evidence, that a system, service or organisation will operate as intended for a defined application in a defined environment" [45]. It is also defined as: *A collection of auditable claims, arguments, and evidence created to support the contention that a defined system/service will satisfy the particular requirements* [46]. As observed from the former or latter definitions, the term 'assurance case' is generic and does not necessarily indicate safety as the property to be assured. Hence, the term 'assurance case' by its own has no particular focus, but if safety is the intended property to be assured, then using terms such as 'safety case' or 'safety assurance case' is more precise where both can be thought of as an instance of an assurance case.

Although the term 'safety case' has become popular today in many safety critical system domains, but its precise meaning is dependent on the purpose that the safety case is intended to satisfy [6]. This raises the question of: *Why do industries need a safety case?* During this work, different purposes that safety cases can satisfy are observed. A safety case is built as a tool:

- To manage residual risks [47]
- To record engineering practices [6]
- In a court of law to address and reduce legal liability [48, 6]
- For marketing

However, before any safety case is attempted, the rationale and purpose of it must be clearly understood. This is vitally important, because if the specific requirements for compiling a safety case are not clear, then the following safety case will also be not clear [6].

There are different definitions of safety case [49, 6]. Most of the available definitions indicate the consensus that a safety case is oriented to demonstrate how a system reduces risk of specific losses to an acceptable level and thus enable a regulator to assess whether the system is acceptably safe to operate. It is worth pointing out that the definition of safety case by the UK Defence Standard 00-56 [50] is the most common. The standard defines the safety case as: "*A structured argument, supported by evidence, intended to justify that a system is acceptably safe for a specific application in a specific operating environment*".

$$\text{Safety Argument} \in \text{Safety Case}$$

The work presented in the two parts of this thesis assumes that the main purpose of a safety case is to justify safety and it refers to the safety case definition by the UK Defence Standard 00-56 wherever the term 'safety case' appears.
A safety case comprises elements as follows:

- *Safety requirements or objectives* that are mainly derived to eliminate or mitigate hazards (also known as goals)
- *Lifecycle artefacts* (also know as work products) which are basically the results of each development phase (e.g. safety analyses, software inspections, or functional tests)
- a *Safety argument* explaining how safety goals (in form of safety claims) are supported by available artefacts (in form of safety evidence)
- *Context* and *Assumptions* about the operating environment and usage

Figure 2.3 shows an overview of the safety case elements and the relationships between them.

2.3.3 Safety Argument

The main purpose of a safety case is to communicate an argument. The argument demonstrates how someone can reasonably conclude that a system is acceptably safe from the evidence available [51]. In English, the word 'argument' is defined as: *"A reason or set of reasons that somebody uses to show that something is true or correct"* [16]. A more technical definition for the word 'argument' is: *A body of information presented with the intention to establish one or more claims through the presentation of related supporting claims, evidence, and contextual information.* [46]. An argument in the safety case definition is called a 'safety argument' or 'safety case argument' and it can be defined as a hierarchically connected series of claims supported by evidence. Safety arguments are intended to demonstrate to the reader that a system is acceptably safe as an overall claim. The claim is defined as: *A proposition being asserted by the author or utterer that is a true or false statement* [46]. The evidence is defined as: *Information or objective artifacts being offered in support of one or more claims* [46].

In order for safety cases to be developed, discussed, challenged, presented and reviewed amongst stakeholders, as well as maintained throughout the product lifecycle, it is necessary for the (1) argument to be clearly structured

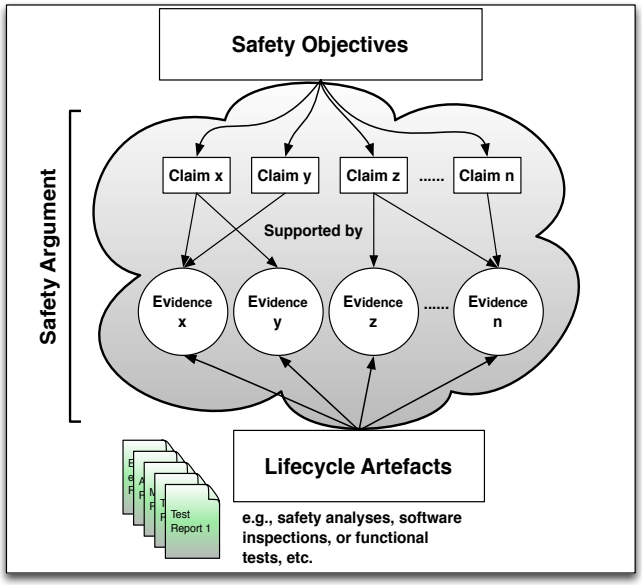

Figure 2.3: Overview of a safety case and its elements

and (2) items of evidence to be clearly asserted to support the argument [45]. There are several ways to represent safety arguments. Safety arguments might be represented by:

- **Prose:** Safety arguments are typically communicated in existing safety cases through free text [51]. Perhaps this is the easiest way to represent safety arguments but not necessarily the most efficient. There are several problems observed while reviewing real safety cases written in prose. We describe some of them. Noticeably, not all engineers who are involved in writing a safety case can write clear and well-structured English [51]. An instance of this problem is, probably, the unconscious use of the ellipsis process in natural languages when authors, unconsciously, leave some non-described crucial details in some statements because they assume that the readers are aware of the context of these statements. For example, the following text describes an evidence item used to support some claims about a bug tracking system of software failures:

"Here we provide evidence of bug tracking for the software. 'XXXXX' is the database that is used to track all issues regarding this system. It has full visibility and is extremely detailed." What does 'all issues' mean? Is it the safety, software or bug issues? How about 'visibility'? Does the writer mean the visibility of the software or the visibility of the bug information? [6]

There are more problems in the description above but the idea is to give an example of the text quality problem.

Another problem observed in safety cases written in prose is the cross-references among texts. Multiple cross-references in texts can be awkward and disrupt the flow of the main argument [51].

- **Tabular notations:** This way to demonstrate safety arguments is not common. The idea, however, is to arrange an argument claim together with its supportive items of evidence in rows and columns.
- **Graphical notations:** This way represents the individual elements of safety arguments (e.g., safety goals, items of evidence and assumptions) using graphical symbols (e.g., squares, circles, parallelograms, etc.). The Goal Structuring Notation (GSN), as well as, the Claims Argument Evidence (CAE) notation are two examples of this way.

Discussing the advantages and disadvantages of the three ways listed above is not an objective of this thesis. We do not claim that a problem in one way can not apply to other ways. However, we use the graphical notation since it can clearly represent the elements of safety arguments and their relationships. Moreover, almost all of the related works to this thesis use GSN thus adopting GSN can make the discussions, comparisons and explanations of our work more clear with respect to other works.

2.3.4 The Goal Structuring Notation (GSN)

GSN is a graphical argument notation which can be used to document explicitly the elements and structure of an argument and the argument's relationship to evidence [45]. GSN's notations are used as a means for communicating (1) safety argument elements, claims, argument logic, assumptions, context, evidence and (2) the relationships between these elements [15].

A goal structure shows how goals are successively ***solved by*** sub-goals until a point is reached where claims can be supported by direct reference to evidence. Using 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 [15].

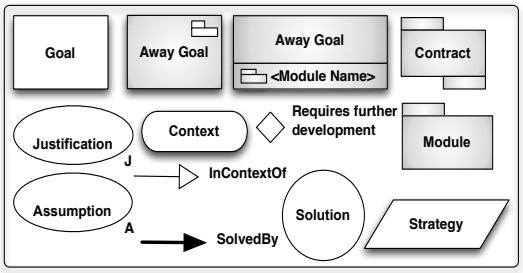

Figure 2.4: Overview of the GSN principal elements

In GSN, rectangles are used to present the argument's claims (***Goals*** in GSN). Parallelograms is used to present the argument's logic (***Strategies*** in GSN). Circles are used to present items of evidence (***Solutions*** in GSN). Ovals with the letter 'J' at the bottom-right are used to present a statement of rationale (***Justifications*** in GSN). Ovals with the letter 'A' at the bottom-right are used to present an intentionally unsubstantiated statement (***Assumptions*** in GSN) [45]. Squashed rectangles are used to present a reference to contextual information or a statement (***Context*** in GSN). Hollow diamonds are applied to the centre of an element (e.g., goal, assumptions, context, etc.) to indicate that a line of argument has not been developed (***Undeveloped*** $<$***element name***$>$ in GSN) [45]. ***SupportedBy*** is an evidential relationship which declares the link between a goal and the evidence used to substantiate it [45]. Permitted supported by connections are: goal-to-goal, goal-to-strategy, goal-to-solution, strategy-to-goal. ***InContextOf*** is a link that declares a contextual relationship [45]. Permitted connections are: goal-to-context, goal-to-assumption, goal-to-justification, strategy-to-context, strategy-to-assumption and strategy-to-justification [45].

Figure 2.4 shows the principal GSN elements, and Figure 2.5 shows an example of a safety argument represented by those elements.

GSN has been extended to enable modularity in a safety case (i.e., module-based development of safety cases). Hence, modular GSN enables the partitioning of a safety case into an interconnected set of modules [52].

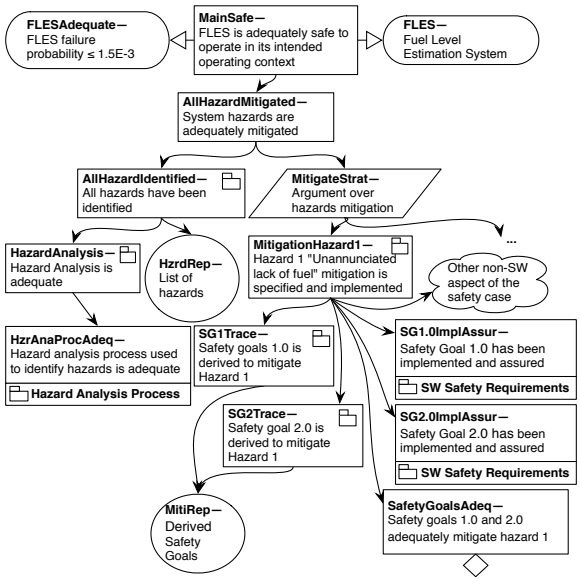

Figure 2.5: A safety argument example represented by GSN [53]

Figure 2.4 presents the principal notations of GSN after the extension in gray. An ***Away Goal*** with a bisecting line in the lower half of it the repeats a claim presented in another argument module which is used to support the argument in the local module [45]. The Module Identifier provides a reference to the module that presents the original claim. A ***Module*** reference presents a reference to a module containing an argument. A ***Contract*** module reference presents a reference to a contract module containing definition of the relationships between two modules, defining how a claim in one supports the argument in the other [45].

2.3.5 Confidence in Safety

The safety case (according to our adopted definition) should provide information about what does it mean for a system to be safe and how the safety is measured and assured. Reviewers (or assessors) of the safety case use that information to assess the safety of the whole system by looking into what the system is made of and whether the failures of the different system components are managed in a way that the safety requirements are still respected. Based on

the confidence in the adequacy of the articulated claims, in the soundness of the structured argument, and in the trustworthiness, suitability and completeness of the gathered evidence, the reviewers build an overall confidence in the safety performance of a system. That is, the overall confidence in safety is a superset that is composed of and influenced by different subset confidences. The trustworthiness of evidence is one subset that affect the reviewers' overall confidence in safety. This raises two questions:

- What do we mean by evidence?
- What might affect the confidence in trustworthiness of an item of evidence?

Although the common definition of the safety case by many safety standards in different domains requires the existence of a body of *evidence* to support the safety argument, there is little agreement about what evidence is [54]. However, some standards explicitly explain what the evidence should look like or the forms of evidence. For example, the UK defence standard 00-56 describes that the generated item of evidence should consist of one or more forms of four types, namely, **direct, quantitative, process, and qualitative evidence** [50].

To narrow down the focus of this thesis, we point out that we are particularly interested of any item of evidence that supports articulated claims about the failure rates of hardware components. Hence, we define an item of evidence, in our particular context, as any available body of fact or information (source of knowledge) which indicates whether our belief in failure rate measures of hardware components is true or false. Figure 2.6 shows potential sources from which we can support a claimed failure rate.

As for the question: What might affect our confidence in trustworthiness of the items of evidence? In fact, for evidence to be relied on as honest or truthful depends on how complete and accurate is our knowledge about it. Judgment on the trustworthiness of an item of evidence depends on our conception and epistemology (e.g., what we know about the evidence, the context it was obtained for, and the environment it was tested within, etc.). Claiming the completeness of our epistemology is perfect but it is impossible to be proven. However, we shall not make the perfect as the enemy of the good by asking either for everything (including what is beyond our knowledge) or nothing. Alternatively, we should accept practical epistemology by verifying that the evidence exists and adequately supports the claim [54], given our knowledge. Even claims that are supported by expert judgements, they should not be dealt with as axioms but they should be subject to verification whenever possible.

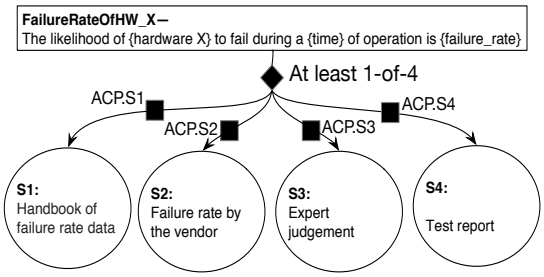

Figure 2.6: Possible items of evidence to support failure rate measures

Since the realisation of the uncertainty and confidence in safety cases is a key to assess their sufficiency and trustworthiness, it is important to clearly identify, explicitly represent and qualitatively or quantitively assess the confidence and the uncertainty. Some researchers suggested quantitative models to assess the confidence in safety arguments (e.g., [55, 56, 57, 58]). However, adopting the current proposed techniques to quantitatively assess the confidence in safety cases is debatable. This is because quantitative confidence techniques require further validation to prove their efficacy [4]. Other researchers suggested to increase the clarity of the contained confidence in safety cases by developing explicit and separate confidence argument [59]. Assessing the confidence quantitively is beyond the scope of our thesis. However, we are particularly interested of developing explicit and separate confidence arguments and then describe how obtaining information about the system behaviour from its operational life might impact these arguments.

2.3.6 Assured Safety Argument

Hawkins et al., [59] introduced "An assured safety argument" as a structure for arguing safety in which the safety argument is accompanied by a confidence argument that documents the confidence in the structure and bases of the safety argument. More specifically, the suggested structure explicitly separates the safety case argument into two parts [59]:

1. A safety argument: the safety argument is constructed to argue over risk reduction —anything related to the identification and mitigation of hazards associated with a system— and should not cope with confidence.

2. An accompanying confidence argument: the confidence argument is

structured according to the assertions of the safety argument and it is not allowed to contain general 'confidence raising' arguments that cannot be clearly related to the structures of the core safety argument.

Assertions in a safety argument relate to the sufficiency and appropriateness of the inferences declared in the argument, the context and assumptions used and the evidence cited [59]. For example, when an item of evidence is used to support a claim, it is asserted that this evidence is sufficient to support the claim. However, a simple 'SolvedBy' relation between the evidence and the claim will not satisfy a reviewer's concerns to reach a certain level of confidence, such as, why the reviewer should believe that the evidence is appropriate for the claim? Or whether or not it is trustworthy. Instead of decomposing the arguments further to argue over the appropriateness and trustworthiness of the supporting evidence, an Assurance Claim Points (ACP) can be created to indicate an assertion in the safety argument. An ACP is indicated in GSN with a named black rectangle on the relevant link and a confidence argument should be developed for each ACP [59]. Three types of assertions were defined, where an ACP can be created, as follows (Figure 9.3 instantiates an example of each type):

1. Asserted inference: the ACP for an asserted inference is the link between the parent claim and its strategy or sub-claims (e.g., ACP.I1).

2. Asserted context: the ACP for asserted context is the link to the contextual element (e.g., ACP.C2).

3. Asserted solution: the ACP for asserted solutions is the link to the solution element (e.g., ACP.S3).

Other examples are found in Figure 2.6 where *ACP.S1-S4* are examples of asserted solutions.

Incomplete knowledge that can preclude a perfect confidence is referred to as an *Assurance Deficit (AD)* [59]. Highlighting and managing the ADs in safety cases are essential to build an overall all confidence in the safety case. The main motivation of proposing the "assured safety argument" structure is to identify and manage the ADs in safety cases through the ACPs and apart from the safety argument. Having a separate confidence argument eliminates or reduces the difficulties that can emerge when merging confidence and safety arguments in one argument. Including unnecessary material, excluding necessary material and increasing the size and complexity of safety arguments, are examples of these difficulties [59].

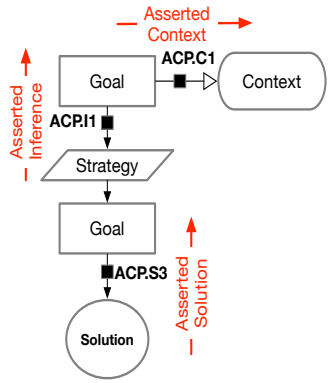

Figure 2.7: Types of ACPs with an example of each usage [59]

2.3.7 Dynamic Safety Case (DSC)

Denney et al., [60] introduced the term "Dynamic Safety Cases (DSCs)" as a novel operationalisation of the concept of through-life safety assurance. The main motivation for introducing DSCs is that the appreciable degree of certainty about the expected runtime behaviour of a system might not be precise or it perhaps over- or underestimate the actual behaviour, which can create deficiencies in the reasoning about the safety performance of that system. Hence, there is a need for a new class of safety assurance techniques that exploit the runtime related data (operational data) to continuously assess and evolve the safety reasoning to, ultimately, provide through-life safety assurance [60].

The suggested lifecycle of DSCs comprises four main activities as follows [60]:

1. *Identify* the sources of uncertainty in a safety case (i.e., the ADs — as described in Subsection 2.3.6).

2. *Monitor* the runtime operation of the related system to collect data about system and environment variables, events, and the ADs in the safety argument(s).

3. *Analyse* the collected operational data from the former activity to examine whether the threshold defined for ADs are met, and to update the confidence in the associated claims

4. *Respond* to operational events that affect safety assurance. Deciding on the appropriate response depends on a combination of factors including the impact of confidence in new data, the available response options already planned, the level of automation provided, and the urgency with which certain stakeholders have to be alerted.

Figure 2.8 shows the four activities of the DSCs' lifecycle.

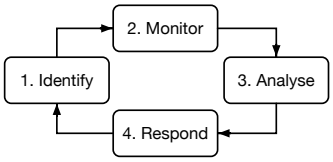

Figure 2.8: An overview of DSCs's lifecycle [60]

2.4 Safety Contracts

The term 'contract' is defined in English as: *"A written or spoken agreement, especially one concerning employment, sales, or tenancy, that is intended to be enforceable by law"* [16]. A contract is intended to (1) establish a binding relationship between one party's offer and the acceptance of that offer by one or more parties, and (2) set out the terms and conditions that constrain this relationship. Using the contracts is familiar in software development. For instance, Design by Contract (DbC) was introduced by Meyer [61, 62] 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 behavior [10].

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 [63]. Hoare introduced a new notation to describe this connection, as follows:

$$P\ \{Q\}\ R.$$

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"* [63].

In the context of contract-based design, 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 surrounding environment satisfies one or more related assumption(s).

Safety and non-safety contracts might describe similar properties; the distinction is whether the guaranteed property is traceable to a hazard [13]. In this thesis, 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 [64, 65, 66]. 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 [11, 12, 13].

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

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

In this thesis, we distinguish between safety contracts within the system domain and safety argument contracts in the safety case. The former type of contracts captures the dependencies among the system's components, whereas a safety argument contract captures the dependencies among the safety case modules. More specifically, a safety argument contract describes the connection between a *consumer* goal in one safety case module and a *provider* goal in another module [45].

Chapter 3

Research Overview

This chapter comprises five main sections. The first section describes the research scope. The second section presents literature review of safety maintenance and change management. The third section describes the problem context and provides a motivation of our research goal. The fourth section introduces the research goal and the derived research questions addressed by the thesis. The last section explains the research methodology.

3.1 Research Scope

Safety critical systems are evolutionary and they are always exposed to both predicted and unpredicted changes during the different stages in their lifecycle. System changes are typically introduced to [67]:

1. correct discovered problems (i.e., corrective changes),

2. accommodate changes in the environment in which a system must operate (i.e., adaptive changes),

3. detect and correct latent faults in a system before they are manifested as failures (i.e., perfective changes), or

4. detect and correct latent faults in a system before they become operational faults (i.e., preventive changes).

Changes to a system can negatively affect the gained confidence in the safe performance because they have the potential to compromise the safety evidence which has been already collected. In fact, operational or environmental changes may invalidate a safety case argument for two main reasons [3]:

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 maintain the confidence in the safe performance of a system, safety engineers must maintain the safety case so that it reflects the reality of the current operating status. In practice, this requires (1) identifying, reanalysing, and re-checking the impacted parts of the system, (2) reviewing the safety case and the logic of its argument to evaluate the impacted elements, and (3) correcting the logic of the argument and generating a new valid set of evidence to support any refuted claims.

Maintenance (or maintainability) is not defined in terms of safety cases by safety standards nor the literature. However, an understanding of the term 'safety case maintenance' can be deduced by looking at the definition of 'Maintenance' by relevant standards in computer systems and software engineering domains, and also by considering the objectives of safety cases by the safety standards. For instance, the systems and software engineering standard ISO/IEC 25010 [68] defines the word 'Maintainability' as: *"The degree of effectiveness and efficiency with which a product or system can be modified by the intended maintainers"*. The standard adds a note to the definition:

NOTE 1: Modifications can include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications. Modifications include those carried out by specialized support staff, and those carried out by business or operational staff, or end users.

On the other hand, the safety standard ISO 26262 [21] states that: *"Throughout the operational life of any system, the corresponding safety case*

might be challenged by additional safety evidence arising from operation, changes and updates to a design, and a shifting regulatory context. In order to maintain an accurate account of the safety of the system, such challenges are assessed for their impact on the original safety argument".

Moreover, the regulations of the offshore installations [69] by the Health and Safety Executive (HSE) in the UK state that: *"The safety case is intended to be a living document that reflects the reality of the current operating status on the installation. Changes are likely to occur in the environment, in the activities carried out or in other factors that may affect risks to people. It is therefore important that the safety case is reviewed in the light of any such changes and revised as often as may be necessary to ensure it reflects reality"*. The same regulations also state that *" Safety cases are intended to be living documents, kept up to date and revised as necessary during the operational life of the installation"*.

Also, the UK Ministry of Defence Ship Safety Management System Handbook JSP 430 [7] 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"*.

From the forgoing, one can deduce that the term 'safety case maintenance' indicates the act of updating a safety case to keep it as a living document that always reflects the reality of the current status of the safe performance of the corresponding system.

Moreover, the maintainability definition by ISO/IEC 25010 [68] can be adapted to define the safety case maintainability so that it can be interpreted as *the degree of effectiveness and efficiency with which a safety case can be modified by the intended maintainers*. The modifications can include corrections in the safety case to respond to system and environment changes, and improvements to address new or higher quality evidence than exist.

Before changing (i.e., modifying) any part of a safety case, maintainers should assess which areas and elements of the safety case are impacted (or will be so) by the change. That is, any approach to maintain a safety case should employ a change management strategy which includes an impact analysis activity to make accurate estimates of the area of the safety case that is affected by a change. The change management strategy should also control the propagation of ripple effects. The accuracy of impact analysis results is dependent on:

1. If the recorded dependencies among the parts of a safety case truly reflect the actual dependencies among them.

2. If the recorded dependencies between the parts of a safety case and the system design truly reflect their actual dependencies.

3. The reliance upon correspondence between safety argument and safety case [5].

There are two techniques used to determine the dependencies in 1 and 2, namely, traceability analysis and dependency analysis.

3.2 Literature Review

This section presents a literature review of safety case maintenance. However, since change management and impact analysis are key principles to achieve maintenance, the review also considers their related contributions from the state-of-the-art and the state-of-the-practice. We divide this section into two subsection. The first subsection discusses the related contributions to safety case maintenance and the second discusses the related contributions to change management.

3.2.1 Safety Case Maintenance

Kelly et al., [5] emphasise that safety engineers have difficulties with safety case maintenance because they lack a systematic and methodical approach to examine the impact of change on safety argument. The authors, therefore, define and describe a tool-supported process to facilitate a systematic impact assessment of safety cases. The process is couched in terms of a safety argument recorded as a goal structure and it comprises two phases, namely, the *Damage* phase and the *Recovery* phase. In the damage phase, safety engineers should assess the impact of change on the safety argument of the safety case. This phase contains three steps as follows:

1. Recognise challenges to safety case

2. Expressing challenge in goal structure terms

3. Using the goal structure to identify impact of challenge

In the recovery phase, the engineers should identify a recovery action and check if the action has further impact in the argument (i.e., propagation). This phase contains two steps as follows:

4. Deciding upon action to recover damaged argument

5. Recover identified damaged argument

The two phases are correlated since a decided action may cause further damages in the safety case.

According to the authors, one limitation of the process is that its ability to express accurately and fully the impact of changes on the safety case depends on the degree to which the goal structured safety argument corresponds to the documented safety case. Hence, preforming an impact analysis in case of a poor match between what is contained in the safety case and what is presented by the safety argument will most likely mislead the engineers' attention. The second limitation of the process is that it considers the dependencies among the elements of the safety argument only and neglects any external dependencies.

Kelly in [70] 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 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 [71]. However, no work was provided to show how these thoughts can facilitate the maintenance of safety cases.

A consortium of researchers and industrial practitioners called the Industrial Avionics Working Group (IAWG) has proposed modular safety cases as a means of containing the cost of change. IAWG's Modular Software Safety Case (MSSC) process [72] requires the development of a modular safety case, where the safety case is composed of a set of independent modules within a safety case architecture to form a coherent safety case for the system. The MSSC process facilitates handling system changes as a series of relatively small increments rather than occasional major updates as shown in Figure 3.1. The process proposes to divide the system into a set of blocks [11]. 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

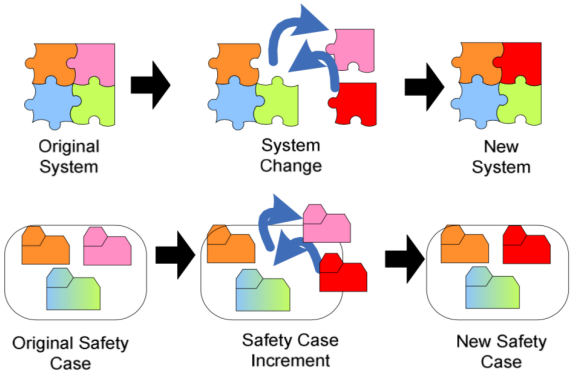

Figure 3.1: An incremental safety case updates [72]

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). DGRs and DGCs serve as a means to record the dependencies among the system blocks to allow efficient impact analysis.

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. Moreover, the MSSC process is dependent on a list of predicted change scenarios and it is almost useless for arbitrary changes. The lack of systematic ways to enable better changes prediction might lead to a big limitation to the process.

Nicholson et al., [73] propose a maintenance approach which is limited to Integrated Modular Systems (IMS). The underling logic of the approach is to contain the impact of changes, initiated as necessary during the operational life of the system, within a system' partition without requiring partition boundaries to be moved. More clearly, the presented approach assumes that IMS are built to be extended and modified after they are released. This means that there are safety margins (i.e., slack in the system resources) not used until a critical limit is reached. Hence, as long as the impact of a change to a partition in the system will not exceed the safety margin in the resources of this partition, the change can be accommodated without further changes to the system. However, evidence should be provided to prove that the safety margin is sufficient to cover

the needs of the change. Also, the impact on the safety basis of the system can then be assessed via an incremental certification process. The authors identify eight potential categories of changes and explain that IMS can enable an incremental maintenance so that some, or all, of identified categories of change can be implemented without the need to re-certify the entire system.

Björnander et al., [74] use introduce the GSN and AADL (Architecture Analysis and Design Language (AADL) Graph Evaluation (GAGE) method which parses and maps safety argumentation structure against system architecture. The underlying logic behind the method is that bridging the gap between the safety case and the system architecture allows system developers to evaluate the safety assurance case against the properties of the components of the system. Björnander motivates the method as a means for tracing potential changes in the system onto the safety argumentation. The method comprises three main steps: 1) organise the system architecture as a directed acyclic graph, 2) organise the safety case as a directed acyclic graph, where each formal argument is connected to a Boolean expression, and finally 3) for each argument, identify the goals supporting the argument in the structure. One limitation of GAGE is that it does not consider the traceability analysis between the safety case and the model in order to detect the parts of the safety cases affected by a change in a component property, and to detect the components affected by a change in a safety case.

Kokaly et al., [75] propose a technique which uses a model-based approach to perform impact assessment on GSN-based compliance argument with ISO 26262 safety standard due to system changes. Kokaly defines and describes an impact assessment algorithm called GSN-IA (GSN Impact Assessment) and a supporting model transformations. The objective of GSN-IA is to highlight "mark" the safety case elements which are affected due to a change. GSN-IA depends on a model slicer used to determine how change impact propagates within a system model so that the slicer itself is a main input to GSN-IA. Kokaly considers two ways that a change to the system can impact the elements of a safety case: 1) *Revise* where the content of an element of a safety case may have to be revised because it referred to a system element that has changed and the semantics of the content may have changed. 2) *Recheck* where the state of the element must be rechecked because it may have changed. If an element is not impacted by a system change, it is marked as *reuse*. Hence, the output of GSN-IA is design model in which the elements are marked for revise, recheck or reuse.The technique addresses the effect of adding components in the system on the existing parts of the safety case. However, it does not consider how adding a component can require additions to the safety case. Also, it is unclear

how the safety analyses are associated with the impact assessment process and how they can get impacted by a change.

3.2.2 Change Management and Impact Analysis

Generally, the change management process in systems engineering is the process of requesting, determining attainability, planning, implementing, and evaluating of changes to a system. Its main goals are to support the processing and traceability of changes to an interconnected set of factors [76]. In computer systems, a change to specific part of a system often impact other parts of the same system. Change impact analysis should be performed prior to the change implementation in oder to identify what might get impacted by the change and also gives an estimate of the adaptation costs.

Many researchers proposed change management approaches and techniques. For example, Dick [77] suggests a five steps change management process to conduct changes in a system. Dick believes that the investment in traceability is the key driver of the change management, therefore, his suggested change management process is dependent on building traceability matrices. According to Dick, traceability is the activity of documenting the relationships between layers of information — for instance between system requirements and software design. The required steps to perform Dick's change management process is as follows:

1. Determine which artefacts a change affects most directly
2. Calculate the potential impact tree by processing the traceability relationships
3. Prune and elaborate the impact tree using engineering judgements, where traceability rationale helps to determine the precise nature of change propagation
4. Define change by traversing the impact tree, working out the precise details of the changes at each point
5. Apply the change.

Other researchers propose investing in traceability to eventually enable similar change impact analyses like the one proposed by Dick (e.g., [78, 79, 80, 81]).

Almost all the existing impact analysis techniques in the literature focus on either identifying or enhancing the identification of the affected parts of

the systems due to changes. The main downside of most of these techniques is that they neglect the amount of potential suspect parts due to changes. In fact, the inability of highlighting the impacted parts or highlighting a big set of impacted parts are two different drawbacks that are equally important. The unawareness of what might get impacted will push the developers to carry out a conservative and wider verification than strictly necessary, whereas detecting a big set of impacted parts will decrease the accuracy of the impact analysis and also increases the re-verification efforts [82].

Bohner [83], however, suggests to calculate the accuracy of detecting the affected parts of a system under a change within his proposed impact analysis approach. Bohner suggests different types of sets while performing the change impact analysis, as follows:

1. The Starting Impact Set (SIS): is an initial set of system elements that are thought to be impacted by a change. The SIS is usually thought of once the change specifications are available.

2. The Candidate Impact Set (CIS): is a set of system elements that are estimated to be affected. The CIS is produced while conducting the impact analysis.

3. The Actual Impact Set (AIS): is a set of system elements actually modified which is obviously produced after modifying system elements.

4. The Discovered Impact Set (DIS): is a set represents an under-estimate of impacts. The idea of producing this set is to identify other impacted system elements that were never thought of before implementing a change.

5. The False-Positive Impact Set (FPIS): is a set represents the over-estimate of impacts in the analysis.

The objective of the impact analysis, according to Bohner, is to conclude the CIS produced from tracing potential impacts as close to the AIS as possible by:

$$AIS = CIS + DIS - FBIS \tag{3.1}$$

Moreover, the accuracy is determined by:

$$Accuracy = \frac{DIS + FPIS}{CIS} \tag{3.2}$$

Bohner suggests to including more semantic information in the impact analysis process to reduce the number of false positives. Semantic information is

actually the type of relationship between Software Life-Cycle Objects (SLOs), such as decomposition relationships among requirements. However, he does not provide any clues as how to use this information [82].

Oertel [82] proposes an impact analysis technique which provides a linear relation between the re-verification efforts and the size of the change by still guaranteeing safety. The proposed technique supports decisions about how to compensate a change by modifying a set of requirements instead of needing to change an implementation. The technique is based on a formal safety model using contracts to express fault containment properties and safety mechanisms. Although the author claims that the confidence in the safety of the system after the change is incorporated is identical or higher compared with the current practiced approach, there is no reflection of this claim on the safety case.

As for the state-of-the-practice, Nair et al., in [84] conducted a survey to determine practitioners' perspectives and practices on safety evidence management. The survey consists of six questions, where one of the questions is: How is evidence change managed? The aim of this question is to identify industrial practices for managing evidence evolution and performing evidence change impact analysis. A total of 52 practitioners from 15 countries and 11 application domains responded. The authors further analyse practices for safety evidence change management and give insights into the current challenges that practitioners face in terms of safety evidence provision. When asked about how they analyse the effect of the change of a piece of evidence on other pieces, 46% of the respondents noted manual checks according to some predefined process. Approximately the same percentage of respondents replied that the effect is checked manually without following any predefined process. The survey suggests that evidence change management is mainly performed manually and highlights the need for further analyses.

3.3 Problem Description

Safety assurance and certification are amongst the most expensive and time-consuming tasks in the development of safety-critical embedded systems [85]. Changes to those systems may require safety engineers to review and maintain the safe performance of their systems and repeat the certification process. Thereby, the cost of system changes including the cost of the activities that will follow them, such as regression testing, exacerbate the problems of cost and time for safety certification.

Safety case maintenance is more complicated task than it might first appear.

This is because change requests should be assessed before decision makers decide whether or not to accept them. The assessment should reveal if the change can cause unreasonable risks, and the required cost to implement the change. Hence, 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. For example, the effect of a change can propagate to other parts of the system — creating a ripple effect — and cause unforeseen violations of the acceptable safety limits. If the impact of change is not clear, developers might be conservative and do wider analyses and verification (i.e., check more elements than strictly necessary), and this will exacerbate the cost problem of safety cases. It is also necessary for the developers to describe how the change affects the system parts — that are listed as affected — in order to correctly estimate the cost of the response to that change. Otherwise, the response to a change might generate unplanned further changes to which the system must again respond [86], and this requires more cost than originally calculated.

One of the biggest challenges that affects safety case maintenance is that safety cases feature highly dependent 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. The lack of documentation of dependencies among the contents of safety cases is deemed as a challenge for safety case maintenance.

Moreover, the lack of traceability between a system and its safety case is another challenge that can impede safety case maintenance. More specifically, 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 domain and its safety case.

Safety case maintenance is inevitable task for many safety critical systems. The result of this task can communicate false status of the actual safe performance of a system if it is done incorrectly (intentionally or unintentionally). This false status, in the worst case, might not be detected by safety assessors, which can lead to certifying unsafe system. Many safety engineers are experiencing difficulties with safety case maintenance [5]. One reason for that is because they lack a systematic and methodical approach to identify the impact of change on safety cases.

Furthermore, using safety standards is one of the ways to control the risks in safety critical systems. Nowadays, safety standards heavily guide the development of these systems and form the basis for their approval and certification [60]. Despite clear recommendations to adequately maintain the systems and their safety cases by safety standards, existing standards offer little or no advice on how such operations can be carried out [5].

Hence, there is an increasing need for globally acceptable systematic and methodical approaches to facilitate the safety case maintenance without incurring disproportionate cost compared to the size of the change.

3.4 Research Goal

In this section, we derive the research questions that should address the identified problems as described in Section 3.3. We first identify the goal of our research and further break it down to pertinent research questions.

Based on the lack of standardised methods and techniques to enable change accommodation in safety critical systems and safety cases, the overall research goal of this thesis is to:

Design new techniques to facilitate the maintainability of safety cases due to system changes.

We refer to *changes* as modifications in the system due to anomalies, removals, additions, enhancements of components or subsystems.

We also use the adapted definition in Section 3.1 (i.e., *the degree of effectiveness and efficiency with which a safety case can be modified by the intended maintainers*) to refer to *maintainability of safety cases*. The maintainability degree is said to be high whenever the following three activities are

done efficiently and effectively:

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

The work in this thesis does not aim to measure the efficiency of achieving the three activities, but rather it strives to enable them and improve on them.

We refine our broad research goal by breaking it into four research questions (RQs), as follows:

RQ1: ***How can safety contracts be used to enable maintainability of safety cases?***

A component contract is a pair of properties, called the assumption, which must be satisfied by the component environment, and the guarantee, which must be satisfied by the component implementation when the assumption holds [87]. It is claimed that the cost of maintaining, reusing and changing software components is lessened while using contracts as developers may rework software components with knowledge of the constraints placed upon them [14]. Contract-based maintenance (or contract-based change management) can be a promising approach for safety critical systems [11, 12, 13]. In the literature, however, there is no consensus on what role the contracts should play in the overall change management process, how and where they should be derived, what they should contain, and how they should be presented. Also, there are different suggestions regarding the association of the contracts with the different properties of safety critical systems.

RQ2: ***How can the number of the impacted safety case elements be minimised so that the re-verification effort needed to make the impacted elements valid again be reduced?***

Changes to a system or its environment often necessitate tremendous re-verification activities. To reduce the re-verification efforts, the change propagation (aka ripple effect [88] or snowball effect [89]) should be as low as practicably possible. Hence, it is important for any proposal aims at facilitating system changes to contain (i.e., localise) the impact of changes. More clearly, to alleviate the cost of updating both a system and its safety case due to a

change, it is crucial to minimise the effects of that change and prevent these effects from propagating into other parts of the system as long as it is practically possible.

Fricke et al. [90] claim, according to their case studies, that accommodating changes can take 30% of the development efforts. If systems' 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). This wider verification can include massive re-engineering efforts, which will increase the maintenance cost. Any effective change management process should help systems' developers to understand how the change affects a particular element (i.e., component, claim, work product) so that they know what is needed to make it valid again.

RQ3: ***How can traceability between a system design and its safety case be established to highlight the impacted parts upon changes?***

Any approach intends to maintain safety cases due to system changes should investigate 1) how a given change impacts system components and 2) how the impacted system components are presented in the safety case. Hence, performing safety impact analysis is a significant step to understand the impact of changes, the underlying reason for change or root cause, and the proposed solution in terms of the existing system and its constraints and requirements.

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. Enabling component and evidence traceability is very useful to analyse the impact of change on a safety argument, and eventually, facilitates the overall maintenance of the safety case. Current standards and analysis techniques assume a top-down development approach to system design. The structure of safety arguments does not necessarily follow the same structure of the system design. Hence, it is unreasonable to assume a structured mapping between the elements of a system design and the elements of the safety argument related to that design. GSN can assist system developers to mechanically propagate the change through the goal structure and scope areas affected by a particular change. Using GSN makes capturing the underlying rationale of the argument easier since it clearly demonstrates the argument elements and their relationships. However, GSN is not enough since it does not tell if the suspect elements of an argument are still valid and expert judgements are still needed.

RQ4: ***How to incorporate safety case maintenance and through-life assurance into development and operational processes?***

Safety cases can be more readily maintained when using process models that encourage evolutionary development of the safety case in parallel with system development, and through explicitly recording applied engineering judgment and experience [71]. To increase the degree of effectiveness and efficiency with which a safety case can be modified, system development processes should consider and support safety case maintenance during the lifecycle of safety critical systems. For instance, some software corrective changes (e.g., bug fixes) can change several items of evidence (i.e., work products) that support safety claims in the safety case. If the development process does not mandate adequate activities to establish and maintain the integrity of the impacted evidence and update the safety case, there will be a gap between the documented safety reasoning in the safety case and the actual safety of the system. However, safety case is provided not only during initial development and deployment, but also at runtime based on operational data [60]. Hence, the operational processes should also mandate activities to continuously assessing and evolving the safety case according to the collected operational data. Such activities can reinforce a safety case for through-life safety assurance.

3.5 Research Methodology

Research is an investigation to find solutions to scientific and social problems through objective and systematic analysis. Research methods are basically all the methods (e.g., theoretical procedures, experimental studies, numerical schemes, statistical approaches) that are used by a researcher during a research study [91]. In this section, we demonstrate the process of our research and the followed research method. Figure 3.2 describes the process step by step.

Our research work started with an initial literature review, where we looked into both the state-of-the-art and the state-of-the-practice regarding the maintainability of safety critical systems and safety cases under system changes. Our literature review revealed that the maintenance of safety cases is significant and required by many safety standards but it has received no or little support yet. This lead us to formulate the problem statement, which we used later to derive a generic research goal (i.e., maintain safety cases after introducing a system change). Since more clarity of ideas, assumptions and practices can be acquired through study of literature, we started digging deeper and chasing the challenges of safety cases maintenance. After many iterations of the

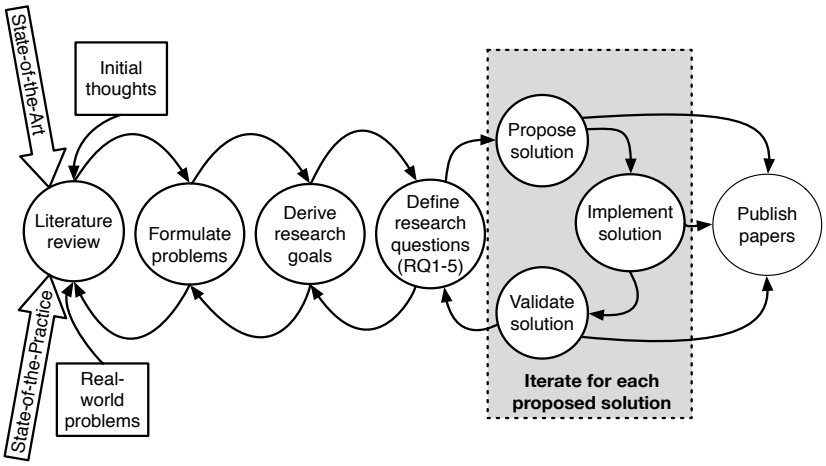

Figure 3.2: Overview of our research process

literature survey and problem formulation, we derived the main research goal, which is *"Design new change management techniques in order to facilitate the maintainability of safety cases due to system changes"*.

Since changes to safety critical systems are of different kinds and they can cause different effects, there is no one technique or methodology that can respond to all kinds of changes. However, we focused on the tools and mechanisms that can facilitate any change management process. To this end, we refined our overall research goal by defining five research questions, which investigated how safety contracts can be exploited to facilitate the maintenance of safety cases. To the best of our knowledge neither the state-of-the-art nor the state-of-the-practice contain supporting processes or methods that provide detailed steps of how to analyse the impact of change on safety cases using safety contracts and sensitivity analysis.

For our research questions, we used the iterative process presented in the grey dotted box in Figure 3.2. We started every iteration by proposing a solution that might answer one or multiple research questions. We implemented each solution by providing a motivation of the solution as well as a detailed description, which usually takes the form of a technique, method or algorithm. Each solution was assessed and validated by a case study, where we listed the limitations and the possible improvements. We evaluated the limitations, after each proposed solution, with respect to the research questions and proposed a

new solution that should address these limitations. The new proposed solution is not designed to address the limitations of the previous solutions only, but it can also answer different research questions.

The direct result of our proposed approaches and techniques is a series of published research papers that communicate our contributions to the research community.

Chapter 4

Research Contributions

This chapter provides an overview of the contributions included in this thesis. First, an overview is provided to describe the different contributions by each paper (included ***Part II***). Second, we group the technical contributions presented in this thesis into five main contributions. Figure 4.1 shows how the main contributions answer the research questions (in Section 3.4), and ultimately, achieve the research goal.

4.1 Contributions of the Included Papers

- ***Paper A:*** *Using Sensitivity Analysis to Facilitate The Maintenance of Safety Cases*

  One contribution of the paper is to introduce a maintenance technique named Sensitivity ANalysis for Enabling Safety Argument Maintenance (SANESAM), which derives safety contracts from FTA using sensitivity analysis. SANESAM combines sensitivity analysis together with safety contracts to identify the sensitive parts of a system and highlight these parts to help the experts to make an educated decision as to whether or not apply changes. Also, since considering a complete list of anticipated changes is difficult, the paper shows how to determine the flexibility (or compliance) of each component to changes. This means that regardless of the type of changes, the change will be seen as a factor to increase or decrease a certain parameter value. Thus system developers can focus more on predicting those changes that might make the parameter value

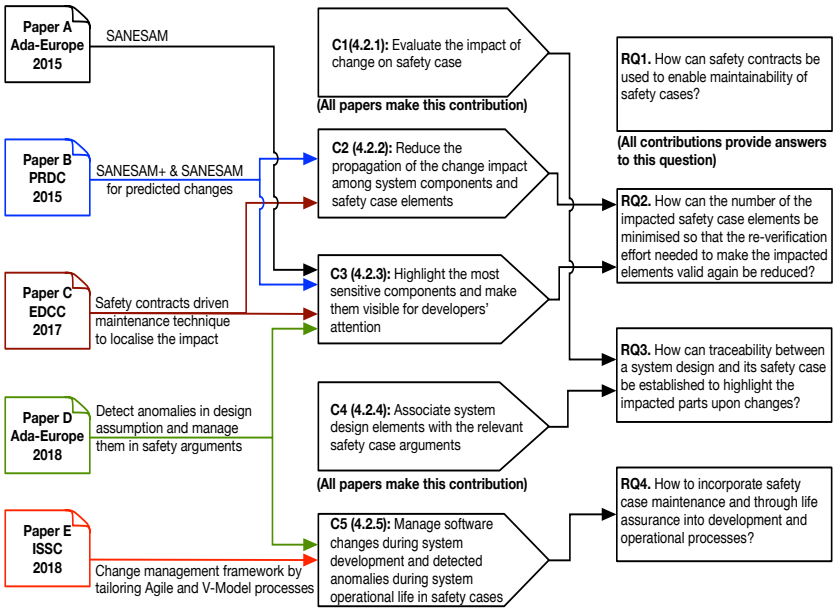

Figure 4.1: The connections between the published papers, research questions and contributions

inadmissible. Another contribution by the paper is introducing a traceability mechanism between a structural design of a system and its safety case to improve the change impact analysis on the safety case. More clearly, the paper provides a set of steps which guides system developers to associate the derived contracts with particular elements of the safety case argument. The association includes a versioning management approach, which stores additional information (e.g., artefact version number) into the safety contracts that are associated with the safety argument.

- ***Paper B:*** *Deriving Hierarchical Safety Contracts*

  The main contribution of the paper is to identify some limitations to SANESAM technique (Paper A), and suggest two options as extensions to resolve these limitations. The first option is SANESAM+, which is useful in the case of arbitrary changes because it calculates the Failure Probability (FP) for all events in the FTA regardless of any change

scenario. SANESAM+ assumes that all events in FTA may change at a time. Hence, the technique requires measuring the sensitivity of all events in the FTA, which means that the Maximum Allowed Failure Probability (MAFP) should be calculated for each event. The second option is SANESAM+ *For Predicted Changes*, this option excludes the events that are unlikely to change and it increases the FP for only the events that are associated to a predicted change. A derived safety contract by SANESAM+ by *Predicted Changes* can guarantee higher FP than the guaranteed FP (for the same event and using the same set of assumptions) in a derived safety contract by SANESAM+. Hence, the derived safety contracts by SANESAM+ *For Predicted Changes* are more tolerant and robust than those derived by SANESAM+ but it requires in-advance prediction of changes.

- ***Paper C:*** *Using Safety Contracts to Guide the Maintenance of Systems and Safety Cases*

  The main contribution of the paper is to introduce a new technique that can save huge efforts in re-verification or re-certification due to some design changes. The technique uses the key principle of SANESAM and SANESAM+ in Paper A and B, respectively, to contain (i.e., localise) the potential changes in the smallest (in terms of number of parts affected) possible segment in the system architecture and its safety case. More clearly, the new technique compares the calculated MAFP of the events with new estimated failure probability of those events due to a change. If a new estimate failure probability of an event is $\leq$ MAFP, then the change will not, necessarily, require a considerable system modification. However, if the estimate of the new failure probability is $>$ MAFP, then the technique investigates whether or not the deficits in the failure probability is containable by the budgeted failure rates of events in higher levels in the fault tree. The ripple effects of the change will stop at the event which will have enough margin in its failure probability to contain the deficit (i.e., change). The technique can serve as a first impact analysis layer that helps system's developers to estimate the required effort to accommodate a design change. It also guides the developers to avoid massive re-engineering efforts when it is not really needed.

- ***Paper D:*** *Using Safety Contracts to Verify Design Assumptions During Runtime*

  The main contribution of this paper is to introduce a novel runtime mon-

itoring technique to detect the discrepancies between the failure rates of system's components during their operational life and their generic failure rates used for analysis and assurance during the design time. Since it is infeasible to monitor the failure rates of all components of a system, the technique utilises importance and sensitivity analyses to evaluate the criticality of the system components, and selects the most critical ones for monitoring. The technique derives safety contracts for the selected components and associate them with the relevant events in the FTA and the relevant parts in the safety case. If a discrepancy is detected between an observed failure rate (λ_O) and a generic failure rate (λ_G) of the same component, where $\lambda_O > \lambda_G$, then the relevant contract should be flagged and the referred parts of both the FTA and the safety case should be revisited. Detecting more precise measure of failure rates than the predicted ones will 1) improve the efficacy of the system design to reduce the risk (mitigate by design), 2) define stronger evidence (e.g., refine or rectify the test results) and 3) highlight the required preventive, corrective, perfective or adaptive maintenance for safer operation

The second contribution of the paper is to provide GSN argument patterns to help system's developers to argue more compelling over the failure rates of the monitored components in the light of the derived evidence from the operational phase (confidence from use). The suggested patterns take into consideration the association of the derived contracts from the FTA and recommend to associate these contracts with the included ACPs.

As a support for the concept of "Through-Life Safety Assurance", the paper shows how to utilise safety contracts to provide prescriptive data for what should be monitored, and what parts of the safety argument should be revisited to maintain system safety whenever a divergence between the design assumptions of system and its actual operation, is detected.

- ***Paper E:*** *A Safety-Centric Change Management Framework by Tailoring Agile and V-Model Processes*

  Following the V-model to accommodate system changes might be very strict, which might be justifiable for structural system changes since many parts get impacted and no accurate estimation for the size of work needed to maintain the system. For software non-structural changes, this might not be justifiable. The main contribution of this paper is to propose XP-Kan-Safe as a novel maintenance framework to facilitate the accom-

modation process of software changes in safety critical systems by utilising the strengths of agile methods and the V-model. More clearly, we reconcile the known effective validation and verification process of the V-model to the known effective practices and the TDD process of agile methods. XP-Kan-Safe comprises two main processes. The first process is called the preliminary process and its main objective is to derive safety contracts and enrich them with additional information to enhance the traceability between the safety requirements (i.e., guarantees) and different related artefacts (i.e., test suites). The preliminary process is applicable to any development process that decomposes safety requirements from the very high level the lowest possible level. The second process is called the change management and it contains ten activities to guide system developers to accommodate software changes by using the derived safety contracts during the preliminary process.

4.2 Main Contributions

The contributions presented in this thesis can be grouped into five main contributions.

4.2.1 Evaluate the impact of change on safety case

Changes to safety critical systems can require a tremendous effort to accommodate. The size of a change's effect varies based on the nature of the change. Ideally, the amount of work needed to accommodate a change should commensurate with the size of this change. Impact analysts shall determine the robustness of their system against a give change request. They should provide the decision makers sufficient information about the size of impact in order to decide wether or not accept the change request.

We propose different techniques which serve as a first impact analysis layer that helps system's developers to estimate the size of effort needed to accommodate a design change. In Paper A, we introduce a Sensitivity ANalysis for Enabling Safety Argument Maintenance technique (*SANESAM*) that supports system engineers to accommodate changes to the reliability figures of system components. We also propose *SANESAM+* and *SANESAM+ for Predicted Changes* in paper B as a modified version of *SANESAM* that covers wider variety of changes. The main objective of the techniques is to 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. Using the techniques helps to bring to developers' attention the most sensitive parts of a system for a particular change.

The key principle of *SANESAM*, *SANESAM+* and *SANESAM+ for Predicted Changes* is to determine the flexibility (or robustness) of a system to changes using sensitivity analysis. The techniques determine, for each component, the allowed range (i.e., thresholds) 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 utilised as a means to determine the range of failure probability parameter for each component. Hence, the techniques assume 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 techniques assume the existence of the required failure probability for the top event $FP_{Required(Topevent)}$, where the FTA is considered unacceptable if: $FP_{Actual(Topevent)} > FP_{Required(Topevent)}$.

As part of *SANESAM*, *SANESAM+* and *SANESAM+ for Predicted Changes*, safety contracts are derived 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.

One difference between the three techniques is based on the number of changes addressed by each one. For example, *SANESAM* calculates the MAFP for all events in FTA but it does not allow multiple changes at a time, *SANESAM+* calculates the MAFP for all events in FTA but it allows multiple changes at a time. Finally, *SANESAM+ for Predicted Changes* calculates the MAFP for particular events based on a predicted change and it allows multiple changes at a time but only to those events.

In paper C, we propose another technique which relies on *SANESAM* and *SANESAM+* but the objective of the technique is not limited to determine the robustness of a system against the potential changes (see Section 4.2.2). Moreover, the proposed technique in paper D also utilises *SANESAM* to determine the robustness of system models against changes (see Section 4.2.5).

In paper E, we propose a new technique, different from the sensitivity-based techniques discussed above, to evaluate the impact of potential software changes in a system. The main objective of this technique is to enable tri-directional impact analysis process between safety requirements (i.e., guaran-

tees), system design elements and the related test suites. The technique derives safety contracts to capture the dependencies among the safety requirements, and enriches the derived contracts with additional traceability information to map the guaranteed requirements with the related system design elements, test cases and safety case elements. Using this additional information helps impact analysts to evaluate the impact of changes on the system design once they know the entry points of the changes in safety requirements or test cases and vice versa.

4.2.2 Reduce the propagation of the change impact among system components and safety case elements

If system developers underestimate the actual impact of a change, there will be a considerable probability to overlook arising failures which can compromise system safety. If system's developers, on the other hand, overestimate the impact of change, there maybe no impact on safety but the cost will be higher than required.

SANESAM, *SANESAM+* and *SANESAM+ for Predicted Changes* can identify the propagation of the change impact through the guaranteed thresholds in the derived safety contracts. However, if a contract is broken due to a greater impact than what is actually assumed by this contract, these techniques do not investigate how to brace this contract to make it more robust.

After approving a change request, a big amount of time is spent to ensure that the system is still safe and all potential work products, i.e., items of evidence which are impacted, are maintained. Minimising the impact of change to the smallest possible part within a system will reduce the work required to verify the system and speed up the change management process. In paper C, we propose another technique to contain (i.e., localise) the potential changes in the smallest possible part of a system. This technique utilises the same rules by which *SANESAM+* and *SANESAM+* calculate the sensitivities and associate them with a safety argument via safety contracts. However, the technique adds additional steps to enable effective usage of the safety margins in a probabilistic FTA. More clearly, the technique compares the calculated MAFP of the events with new estimated FP after introducing a change. If the change's effect (i.e., difference between the new estimated FP and the MAFP) is not containable in the safety contract of the impacted event, then the safety contract of the ancestor event should be investigated as whether or not it can contain it. If the change's effect still cannot be contained by the ancestor, safety contracts in one more level up should be investigated and so on and so forth until

Figure 4.2: Safety contract notation

a safety contract contains it. Once the contract which contains the change's effect is identified, all associated claims with this contract together with their supporting arguments and evidence should be highlighted as suspect.

4.2.3 Highlight the most sensitive components and make them visible for developers' attention

Changes are often only performed years after the initial design of the system making it hard for system's developers performing the changes to know which parts of the system or the safety case are affected. We propose using safety contract to document (i.e., highlight) the most critical and sensitive components to changes to make them visible up front for developers' attention. In other words, we use safety contracts to highlight prescriptive data for what is needed to be revisited and verified to maintain system safety when changes happen. In papers A, B and C we propose to derive safety contracts from FTA to highlight the most sensitive FTA events to changes. To this end, we introduce a contract notation to annotate the contracted events in FTA, where every derived contract should have a unique identifier as shown in Figure 4.2.

In paper D, we propose to derive safety contracts from FTA to highlight the most important components to make them visible up front for developers' attention. Furthermore, the derived contracts is also used to record the thresholds of $\lambda_D(i)$ to continuously compare them with the monitoring results (λ_{D_O}). Hence, if λ_{D_O} of component i exceeds the guaranteed $\lambda_{D\_Max}(i)$ in the contract of that component, then we can infer that the contract in question is broken and the related FTA should be re-assessed in the light of the λ_{D_O}.

in Paper E, we propose to derive safety contracts from FTA and FMEA to highlight the generated safety requirements. These safety contracts should also highlight additional traceability information and make visible up front for developers' attention. Figure 4.3 presents an example of a derived safety contract and the suggested traceability information.

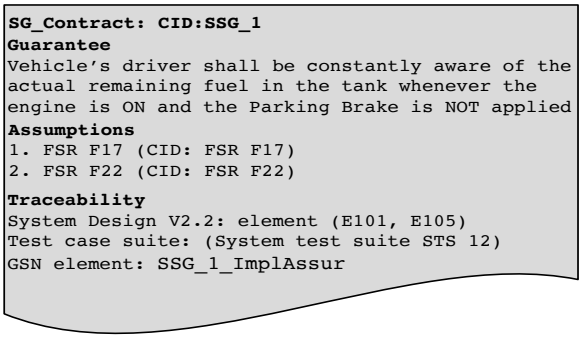

Figure 4.3: Traceability information in a safety contract [92]

4.2.4 Associate system design elements with the relevant safety case arguments

A traceability between the system domain and its safety case should be established to highlight the impacted parts in one side whenever the other side changes. To this end, we use the same contract notation, which we suggested to use in FTAs (in Figures 4.2 and 4.3), to associate the system elements with the relevant elements in the safety case arguments. Figure 4.4 show an example of how the related FTA events to a specific system components are associated with the relevant safety case argument. The mapping between safety contracts from the system domain (e.g., safety contracts in FTAs) with the related safety contracts in the safety case enables a bi-directional safety impact analysis. This means that once the entry point of a change is pinpointed in the safety case, the logic of the safety case argument can be used to evaluate the impact of this change in the system model and vice versa. It is worth noting that the safety contract notation shall not affect the way GSN is being produced but it brings additional information for developers' attention.

4.2.5 Manage software changes during system development and detected anomalies during system operational life in safety cases

Observing anomalies in design assumptions during testing or runtime is deemed as a change that might need to be considered based on its criticality.

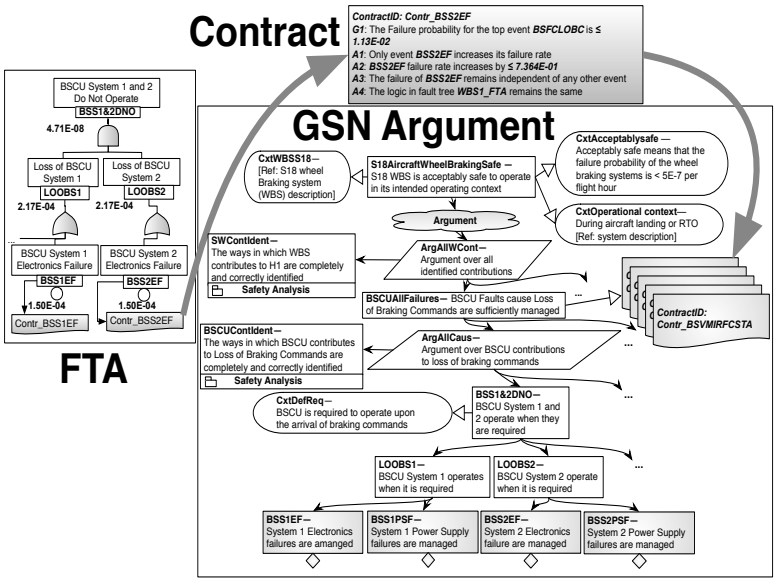

Figure 4.4: An example of an association of an FTA event with a safety case argument

As the system evolves after deployment, there could be a mismatch between our communicated understanding of the system safety by the safety case and the safety performance of the system in actual operation, which might invalidate many of the prior assumptions made, undermine the collected items of evidence and thus defeat safety claims [60]. In paper E, we propose a novel technique to detect the discrepancies between the failure rates of system's components during their operational life and their generic failure rates used for analysis and assurance during the design time. Since it is infeasible to monitor the failure rates of all components of a system, the technique utilises probabilistic FTA to evaluate the criticality of the system components, and selects the most critical ones for monitoring. The technique derives safety contracts for the selected components and associate them with the relevant events in the FTA and the relevant parts in the safety case. If a discrepancy is detected between an observed failure rate (λ_O) and a generic failure rate (λ_G) of the same component, where $\lambda_O > \lambda_G$, then the relevant contract should be flagged and the referred parts of both the FTA and the safety case should be revisited. The technique

also checks wether the changes to the assumed failure rates during runtime is containable without the need to make further changes.

In paper E, we propose and describe XP-Kan-Safe as a safety-centric change management framework. The main idea of XP-Kan-Safe is to incorporate an agile based change management in the development process of safety critical systems to systematically assess and implement software changes. XP-Kan-Safe utilises safety contracts as (1) stitches that connect the V-model, Extreme Programming (XP) and Kanban into our tailored process, and (2) means to enable a tri-directional impact analysis process. More specifically, the framework derives safety contracts from safety analyses (FTA and FMEA) to capture different levels of safety requirements decompositions. A guaranteed requirement at one level (e.g., functional safety requirement) is realised by assumptions that represent lower level requirements (e.g., technical safety requirements). The derived safety contracts record the dependencies among safety requirements. Additionally, the contracts comprise traceability related information which associates the safety requirements with the relevant design elements, safety argument elements and test suites. Hence, the safety contracts can efficiently support the impact analysis upon changes to safety requirements, safety argument or test suites (i.e., code level).

Chapter 5

Conclusions and Future Work

5.1 Conclusions

The main objectives of safety cases are to justify and communicate that a system is acceptably safe to operate in a specific context. The validity of a safety case relies on assumptions about the safe performance of a system as well as the assumptions made about the context in which that system should operate. Changes to these assumptions will lead to a mismatch between the real safe performance of a system and the communicated safe performance in its safety case. Thus, safety case should be maintained as a living document that should always justify and communicate the real safe performance of the system. Moreover, changes to safety critical systems lead to a re-certification process in which system developers review an existing safety case and maintain it by identifying its impacted parts, checking the validity of the safety arguments and generating a new set of evidence. If system developers are unable to identify the actual impacted parts in the safety case, they might re-execute more verification and validation activities than strictly necessary.

Safety case maintenance is not an easy task since changes are often performed years after the initial design of the system making it hard for system developers to know which parts of the safety case are affected. Also, system developers need to understand the dependencies between the different elements of safety case and identify the affected parts correctly. Hence, there is a press-

ing need for effective methods and techniques to enable efficient safety case maintenance without incurring disproportionate cost compared to the size of the change.

In this thesis we have designed new techniques to facilitate the maintainability of safety cases due to system changes. These techniques utilise safety contracts to provide feasible solutions to safety case maintenance such as helping to understand the indirect impact of a change on the safety case and providing a means to sufficiently record the dependencies between safety case contents and

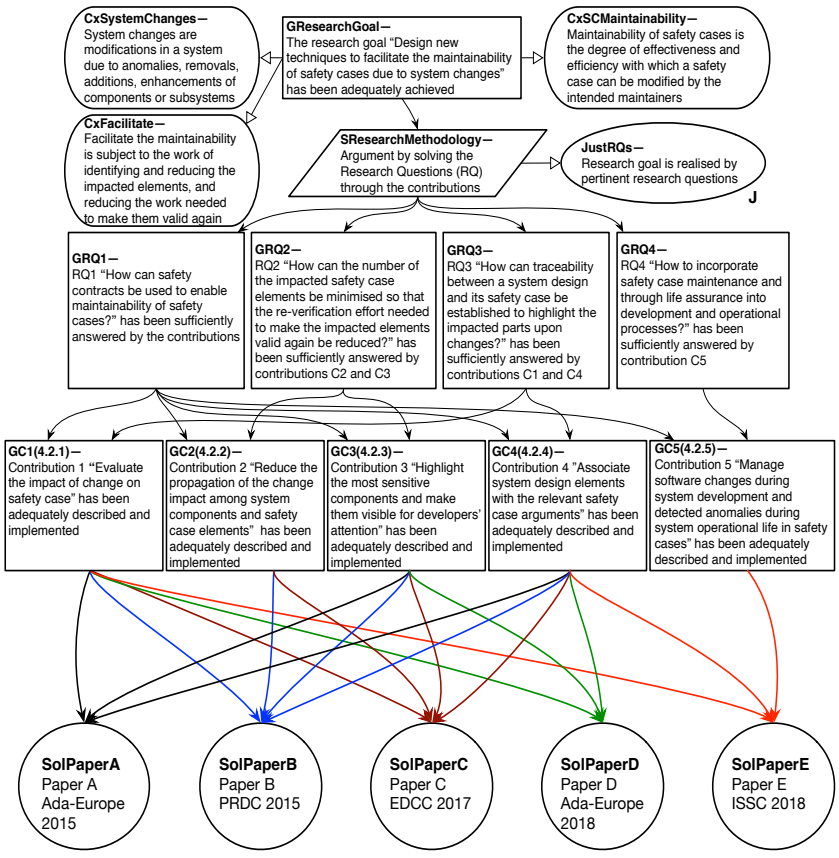

Figure 5.1: The logic of achieving the research goal

system elements. Additionally, the techniques offer new concepts of detecting change propagation, establish bidirectional traceability between a system and its safety case, and highlight the most sensitive system parts to changes. Figure 5.1 presents a GSN argument which explains how the research goal of the thesis is achieved.

SANESAM was designed 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.). The allowed ranges for different components result in safety contracts and if a contract is not violated by a change introduced to the system, the change can be easily contained. The contracts are mapped to a safety case argument and used in justification of the safe performance of the system. SANESAM+ and SANESAM for predicted changes are extensions to SANESAM and they provide more freedom by considering multiple events at a time and more robustness against predicted changes, respectively. We also designed a safety contracts driven maintenance technique to contain change impact in the smallest possible set of components to ultimately prevent (or minimise) the ripple of these effects from propagation. A Wheel Braking System (WBS) was used as a case study to evaluate the feasibility of the application of the described techniques so far. The techniques under discussion contributed to evaluate the impact of change on safety case, reduce the propagation of the change impact among system components and safety case elements, and highlight the most sensitive components and make them visible for developers‘ attention.

We designed another technique which also utilises safety contracts to continuously reassess the failure rates of the most critical components to safety and use the results to suggest system changes or maintenance. The main objective of the technique is to monitor the runtime of a system and detect the divergence between the failure rates (which were used in the safety analyses) and the observed failure rates in the operational life. The technique uses sensitivity analysis to define the maximum allowed deviation in the failure rate of a critical component before it compromises the safety requirements. Subsequently, a safety contract is derived to guarantee the maximum allowed deviation in the failure rate, and a monitoring algorithm continuously checks if the failure rate exceeds its acceptable threshold (i.e., the guaranteed failure rate). Any violated safety contract will indicate the affected part in the safety case. This technique enables through-life safety assurance by providing prescriptive data for what should be monitored, and what parts of the safety case arguments should be updated to maintain system safety when a divergence (i.e., anomaly) is detected in safety cases and during system operational life. We used an Automated

Guided Vehicle (AGV) system as a case study to evaluate the feasibility of the technique.

Finally, we designed a safety-centric change management framework called *XP-Kan-Safe* by tailoring agile and V-model processes. *XP-Kan-Safe* is a novel maintenance framework to facilitate the software change management process in safety critical systems. More clearly, we reconciled the known effective validation & verification process of the V-model to the known effective practices and the TDD process of agile methods. We used safety contracts as 1) stitches that connect the V-model, Extreme Programming (XP) and Kanban into our tailored process, and 2) means to enable a tri-directional impact analysis process (i.e., traceability between safety requirements, test suites and safety case). XP-Kan-Safe incorporates safety case maintenance into the software development process to manage software changes during system development in safety cases.

As a summary, the work in this thesis showed how safety contracts can be utilised to support the maintainability of safety cases. The work showed how the contracts can be derived, how they can be presented, what they might contain, and where to place them.

5.2 Future Research Directions

The designed techniques in this thesis provided different solutions to facilitate the accommodation of changes to system reliability as well as changes to the software of safety critical systems. However, several research directions remain for the future work. We envision three categories of research directions in the future, namely, validation, automation and extension:

- **Validation**: One direction for the future work is to carry out an industrial validation for the designed techniques. Such a validation should focus on measuring the feasibility and efficacy of using safety contracts for safety case maintenance as described in SANESAM and SANESAM+. Also, since failure rates are not constant across different applications, where conservative developers might assume less reliable failure rates than what is really reported, the future work should also validate the efficacy of our monitoring technique for reassessing more precise failure rates.

  An industrial validation of XP-Kan-Safe is also a direction for the future work. The designed framework can be validated in the context of AUTOSAR (AUTomotive Open System ARchitecture) based systems [93].

One of the purposes of AUTOSAR is to satisfy the need for modularity of the architectural components and their implementations, which facilitates the exchange of these components between different parties. Choosing to develop automotive systems based on the AUTOSAR standard requires continuous adoption of new AUTOSAR releases in the development projects in order to enable new innovative solutions in cars [94]. This means that there is a need for performing change impact analysis on AUTOSAR systems after each release of AUTOSAR standard and maintain safety cases to consider the new changes. XP-Kan-Safe can help to document the dependencies among AUTOSAR components in different layers (i.e., Basic Software, Runtime Environment and Software Applications), establish bidirectional traceability between safety requirements, system model, validation and verification artefacts, and the safety case arguments.

- **Automation**: Another direction for the future work is to automate the application of the techniques. The techniques can be implemented within a software tool to automatically identify the sensitive components of a system, derive the safety contracts and create traceable links between system elements, safety analyses and safety case arguments. Obviously, formalising the safety contracts is not only useful to support effective automation of the techniques but it also makes the safety contracts more checkable for completeness, consistency and correctness.

- **Extension**: Another direction for the future work is to extend the techniques to other properties. To this point, only failure rates are considered by SANESAM and SANESAM+. This can be less useful while dealing with software changes as it is recognised as being difficult to quantify the reliability (e.g., failure probabilities) of software components. Another future work can be stemmed from this limitation, which is extending the techniques to cover wider variety of changes. In other words, the underlying logic of the techniques can be utilised to consider software properties to ultimately facilitate the maintainability of safety cases upon the changes to these properties. For instance, SANESAM and SANESAM+ can address the problem of the Worst Case Execution Time (WCET) as a property where sensitivity is judged in terms of its impact on the ability to meet the system's timing requirements and the required level of confidence. For example, sensitivity analysis can be used to derive safety contracts that guarantee the reliability R of Task T to meet its deadline assuming that the WCET of T is within τ (time unit). Such contracts

can be associated with the parts of the safety case arguments to support the impact analysis in case if changes to the timing behaviour are either planned or detected.

Likewise, our designed monitoring technique can be extended to monitor other critical properties during the runtime to continuously maintain the documented confidence in safety cases [95]. Software temporal properties are good candidates for the technique (e.g., WCET), however some communication properties are also interesting to consider for monitoring. In [96], for example, we propose a methodology for assuring wireless cooperative functions of vehicular systems, where we suggest to identify and monitor all system parameters that may increase the communication failures (i.e., packet loss rate). This suggestion might be impracticable since it is costly and infeasible to monitor almost all parameters in cooperative functions. Applying importance and sensitivity analyses helps to identify a reasonable set of parameters for monitoring. Moreover, almost all proposed techniques for monitoring communication failures in the literature do not show how safety cases should be maintained according to the newly detected behaviours. Extending our monitoring technique and its algorithm to 1) monitor packet loss rate during runtime, 2) derive level of confidence, and 3) support a through-life safety assurance, is a reasonable direction for the future work.

Bibliography

[1] J.C. Knight. Safety critical systems: Challenges and directions. In *Proceedings of the 24rd International Conference on Software Engineering (ICSE).*, pages 547–550, May 2002.

[2] O. Jaradat. Enhancing the maintainability of safety cases using safety contracts, Mälardalen University, Västerås, Sweden. http://www.es.mdh.se/publications/4082-, November 2015.

[3] 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)*, UK, 2014.

[4] P. J. Graydon and C. M. Holloway. An investigation of proposed techniques for quantifying confidence in assurance arguments. *Safety Science*, 92(Supplement C):53 – 65, 2017.

[5] 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.

[6] R. Maguire. *Safety Cases and Safety Reports: Meaning, Motivation and Management*. Ashgate Publishing, Ltd., 2012.

[7] U.K. Ministry of Defence, "JSP 430 - Ship Safety Management System Handbook", Ministry of Defence January 1996.

[8] T. Kelly. *Arguing Safety – A Systematic Approach to Managing Safety Cases*. PhD thesis, Department of Computer Science, University of York, 1998.

[9] Health and Safety Executive (HSE). *Railway Safety Cases - Railway (Safety Case) Regulations - Guidance on Regulations*, 1994.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] *Oxford Dictionary of English (3 ed.)*. Oxford University Press, 2010.

[17] Industrial Avionics Working Group (IAWG). Modular Software Safety Case (MSSC): Glossary, November 2012.

[18] J. Knight. *Fundamentals of Dependable Computing for Software Engineers*. Chapman & Hall/CRC, 1st edition, 2012.

[19] M. Dorfman and C. Anderson. *Aerospace software engineering: a collection of concepts*. American Institute of Aeronautics and Astronautics, Washington, DC, 1991.

[20] I. Sommerville. *Software Engineering*. Addison-Wesley, Harlow, England, 9 edition, 2010.

[21] ISO 26262:2011. Road Vehicles — Functional Safety, Part 1-9. International Organization for Standardization, Nov 2011.

[22] A. Avižienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, Jan 2004.

[23] N. Leveson. Software system safety, Lecture Notes, Massachusetts Institute of Technology (MIT), Aero/Astro Department, 2002.

[24] M. Rausand. *Reliability of Safety-Critical Systems: Theory and Applications*. John Wiley and Sons, Inc., Hoboken, New Jersey, 2013.

[25] R.J. Mikulak, R. McDermott, and M. Beauregard. *The Basics of FMEA, 2nd Edition*. CRC Press, 2008.

[26] M. Rausand and A. Høyland. *System Reliability Theory: Models, Statistical Methods and Applications*. Wiley-Interscience, NJ, 2004.

[27] SAE ARP4761 Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment, December 1996.

[28] M. Stamatelatos, W. Vesely, J. Dugan, J. Fragola, J Minarick, and J. Railsback. Fault Tree Handbook with Aerospace Applications. Handbook, National Aeronautics and Space Administration, 2002.

[29] H. E. Lambert. Use of fault tree analysis for automotive reliability and safety analysis. SAE technical paper, Lawrence Livermore National Lab., 2004.

[30] US Department of Defense. *Military Handbook: Electronic Reliability Design Handbook (MIL-HDBK-338B)*, October 1998.

[31] IEC 61508:2010. Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems, Part 1-7, International Electrotechnical Commission, 2010.

[32] *Component Reliability Data for Use in Probabilistic Safety Assessment (IAEA-TECDOC-478)*. International Atomic Energy Agency, Vienna, 1988.

[33] M. Generowicz and A. Hertel. Reassessing failure rates. Technical report, I&E Systems Pty Ltd, 2017.

[34] International Society of Automation. *ISA-TR84.02-2002: Safety Instrumented Functions — Safety Integrity Level Evaluation Techniques*, North Carolina, 2002.

[35] M. Rausand. *Reliability of safety-critical systems: theory and applications*. John Wiley & Sons, 2014.

[36] A. Saltelli. *Global sensitivity analysis: the primer*. John Wiley, 2008.

[37] L. Breierova and M. Choudhari. An introduction to sensitivity analysis. Technical report, Massachusetts Institute of Technology (MIT), 1996.

[38] A.C. Cullen and H.C. Frey. *Probabilistic techniques in Exposure assessment*. Plenum Press, New York, 1999.

[39] D. J. Pannell. Sensitivity analysis of normative economic models: theoretical framework and practical strategies. *Agricultural Economics*, 16(2):139 – 152, 1997.

[40] R. D. Hawkins and T. P. Kelly. Software safety assurance - what is sufficient? In *4th IET International Conference on Systems Safety 2009. Incorporating the SaRS Annual Conference*, pages 1–6, Oct 2009.

[41] *Nuclear Installations Act 1965, section 14*. Her Majesty's Stationary Office, London, UK, 1965 (reprinted 1993).

[42] K. Cassidy. CIMAH Safety Cases — the HSE Approach. IChemE Symposium series no. 110, 1988.

[43] Anthony Hidden (QC). *Investigation into the Clapham Junction Railway Accident*. HMSO, 1989.

[44] R B. Whittingham. *Preventing corporate accidents : An Ethical Approach*. Elsevier Ltd, 2008.

[45] Goal Structuring Notation working group. GSN Community Standard Version 1. Origin Consulting Limited, York, UK, November 2011.

[46] Object Management Group (OMG). *Structured Assurance Case Metamodel (SACM)*, Technical report, Version 1.0, 2013. [Online]. Available: `http://www.omg.org/spec/SACM/1.0/PDF/`.

[47] P. Graydon. (personal communication), August 2015.

[48] Jacobs Sverdrup Australia Pty, Ltd. The development of safety cases for complex safety critical systems. lecture notes. April 2005. [online]. avaialble: `https://msquair.files.wordpress.com/2012/06/md12_safety_cases_r5.pdf`.

[49] T. Kelly. Introduction to safety cases, lecture notes, 2007. [online]. available: `http://www.omg.org/news/meetings/workshops/SWA_2007_Presentations/00-T3_Kelly.pdf`.

[50] U.K. Ministry of Defence. *00-56 Defence Standard — Issue 6, 2015. Safety Management Requirements for Defence Systems — Part 1: Requirements and Guidance. (UK) Ministry of Defence*, June 2015.

[51] T. Kelly. A systematic approach to safety case management. In *Proceedings of SAE 2004 World Congress, Detroit*. The Society for Automotive Engineers, March 2004.

[52] O. Jaradat, I. Bate and S. Punnekkat. Facilitating the maintenance of safety cases. In *Proceedings of the 3rd International Conference on Reliability, Safety and Hazard - Advances in Reliability, Maintenance and Safety (ICRESH-ARMS)*, Luleå, Sweden, June 2015.

[53] O. Jaradat, P. Graydon and I. Bate. The role of architectural model checking in conducting preliminary safety assessment. In *Proceedings of the 31st International System Safety Conference (ISSC)*, Boston, USA, August 2013.

[54] P. J. Graydon and C. M. Holloway. "Evidence" under a magnifying glass: Thoughts on safety argument epistemology. In *10th IET System Safety and Cyber-Security Conference 2015*, pages 1–6, Oct 2015.

[55] J. Guiochet, Q. A. D. Hoang and M. Kaaniche. A model for safety case confidence assessment. In *Proceedings of the 34th International Conference on Computer Safety, Reliability, and Security - Volume 9337*, SAFECOMP 2015, pages 313–327. Springer-Verlag New York, Inc., 2015.

[56] B. Littlewood and D. Wright. The use of multilegged arguments to increase confidence in safety claims for software-based systems: A study based on a bbn analysis of an idealized example. *IEEE Transactions on Software Engineering*, 33(5):347–365, May 2007.

[57] X. Zhao, D. Zhang, M. Lu and F. Zeng. *A New Approach to Assessment of Confidence in Assurance Cases*, pages 79–91. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[58] E. Denney, G. Pai, and I. Habli. Towards measurement of confidence in safety cases. In *2011 International Symposium on Empirical Software Engineering and Measurement*, pages 380–383, Sept 2011.

[59] R. Hawkins, T. Kelly, J. Knight and P. J. Graydon. *A New Approach to creating Clear Safety Arguments*, pages 3–23. Springer London, UK, 2011.

[60] E. Denney, G. Pai, and I. Habli. Dynamic safety cases for through-life safety assurance. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 2, pages 587–590, May 2015.

[61] B. Meyer. Design by contract. Technical Report TR-EI-12/CO, Interactive Software Engineering Inc., 1986.

[62] B. Meyer. *Object-Oriented Software Construction*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1st edition, 1988.

[63] C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, October 1969.

[64] 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, FMCO*, pages 200–225, Amsterdam, The Netherlands, October 2007. Springer.

[65] 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.

[66] S. Bauer, A. David, R. Hennicker, K. G. Larsen, A. Legay, U. Nyman, and A. Wasowski. Moving from specifications to contracts in component-based design. In *Proceedings of the 15th International Conference on*

Fundamental Approaches to Software Engineering, FASE'12, pages 43–58, Berlin, Heidelberg, 2012. Springer-Verlag.

[67] ISO/IEC/IEEE International Standard for Software Engineering - Software Life Cycle Processes - Maintenance, Sept 2006.

[68] ISO/IEC 25010:2011 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models, 2011.

[69] The Offshore Installations (Offshore Safety Directive)(Safety Case etc) Regulations. Health and Safety Executive (HSE), UK, 2015.

[70] T. Kelly. Literature survey for work on evolvable safety cases. Department of Computer Sceince, University of York, 1st Year Qualifying Dissertation, 1995.

[71] 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.

[72] Industrial Avionics Working Group (IAWG). Modular Software Safety Case (MSSC): Process Overview, November 2012.

[73] M. Nicholson, I. Bate and J. Mcdermid. Generating and maintaining a safety argument for integrated modular systems. In *Preceedings of the 5th Australian Workshop on Safety Critical Systems and Software, Adelard for the Health and Safety Executive, HSE Books, ISBN 0-7176-2010-7, and Contract Research*, pages 0–7176. Institution of Engineers Australia, 2000.

[74] S. Björnander, R. Land, P. Graydon, K. Lundqvist, and P. Conmy. A method to formally evaluate safety case evidences against a system architecture model. In *Preceedings of the 23rd IEEE International Symposium on Software Reliability Engineering Workshops*, pages 337–342, Nov 2012.

[75] S. Kokaly, R. Salay, M. Chechik, M. Lawford and T. Maibaum. Safety case impact assessment in automotive software systems: An improved model-based approach. In *Preceedings of the 36th Computer Safety, Reliability, and Security (SAFECOMP)*, pages 69–85. Springer International Publishing, 2017.

[76] I. Crnkovic, U. Asklund and A P. Dahlqvist. *Implementing and Integrating Product Data Management and Software Configuration Management*. Artech House, Inc., Norwood, MA, USA, 2003.

[77] J. Dick. Design traceability. *IEEE Software*, 22(6):14–16, November 2005.

[78] I. Sommerville and P. Sawyer. *Requirements Engineering: A Good Practice Guide*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1997.

[79] S. Robertson and J. Robertson. *Mastering the Requirements Process (2Nd Edition)*. Addison-Wesley Professional, 2006.

[80] S A. Bohner and R S. Arnold. *Software change impact analysis*. Los Alamitos, Calif. : IEEE Computer Society Press, 1996. Includes bibliographical references (p. 361-374).

[81] A. De Lucia, F. Fasano, and R. Oliveto. Traceability management for impact analysis. In *Preceedings of the 24th IEEE International Conference on Software Maintenance*, pages 21–30, Beijing, China, 2008.

[82] M. Oertel. *A linear scaling change impact analysis based on a formal safety model for automotive embedded systems*. PhD thesis, University of Oldenburg, 2016.

[83] S. A. Bohner. Software change impacts: an evolving perspective. In *Preceedings of the 18th International Conference on Software Maintenance*, pages 263–272. IEEE Computer Society, Canada, 2002.

[84] S. Nair, J. Luis de la Vara, M. Sabetzadeh and D. Falessi. Evidence management for compliance of critical systems with safety standards: A survey on the state of practice. *Information and Software Technology*, 60:1 – 15, 2015.

[85] H. Espinoza, A. Ruiz, M. Sabetzadeh, and P. Panaroni. Challenges for an open and evolutionary approach to safety assurance and certification of safety-critical systems. In *Proceedings of the 1st International Workshop on Software Certification (WoSoCER)*, pages 1–6, Nov 2011.

[86] N. Tracey, A. Stephenson, J. Clark and J. McDermid. A safe change oriented process for safety-critical systems. In *Proceedings of the International Workshop on Software Change Evolution*, 1999.

[87] M. Bozzano, A. Cimatti, C. Mattarei and S. Tonetta. Formal safety assessment via contract-based design. In Franck Cassez and Jean-François Raskin, editors, *Automated Technology for Verification and Analysis*, pages 81–97. Springer International Publishing, 2014.

[88] W. P. Stevens, G. J. Myers, and L. L. Constantine. Structured design. *IBM Syst. J.*, 13(2):115–139, June 1974.

[89] C. Terwiesch and C H. Loch. Managing the process of engineering change orders: the case of the climate control system in automobile development. *Journal of Product Innovation Management*, 16(2):160 – 172, 1999.

[90] E. Fricke, B. Gebhard, H. Negele and E. Igenbergs. Coping with changes: Causes, findings, and strategies. *Systems Engineering*, 3(4):169–179, 2000.

[91] S. Rajasekar, P. Philominathan, V. Chinnathambi. Research methodology. October 2013. [Online]. Available: http://arxiv.org/pdf/physics/0601009v3.pdf [Lastchecked]: October 2015.

[92] A. Salameh and O. Jaradat. A safety-centric change management framework by tailoring agile and V-model processes. In *Proceedings of the 36th International System Safety Conference (ISSC), Arizona, USA*, August 2018.

[93] AUTomotive Open System ARchitecture (AUTOSAR). www.autosar.org/ [Last checked] November 2018.

[94] D. Durisic, M. Staron and M. Tichy. ARCA – Automated Analysis of AUTOSAR Meta-model Changes. In *Preceedings of the 7th International Workshop on Modeling in Software Engineering*, pages 30–35, May 2015.

[95] O. Jaradat, I. Sljivo, I. Habli and R. Hawkins. Challenges of safety assurance for industry 4.0. In *European Dependable Computing Conference (EDCC)*. IEEE Computer Society, September 2017.

[96] S. Girs, I. Sljivo, and O. Jaradat. Contract-based assurance for wireless cooperative functions of vehicular systems. In *Preceedings of the 43rd Annual Conference of the IEEE Industrial Electronics Society IECON*, pages 8391–8396, Oct 2017.

II

Included Papers

Chapter 6

Paper A: Using Sensitivity Analysis to Facilitate The Maintenance of Safety Cases

Omar Jaradat, Iain Bate, Sasikumar Punnekkat.
In Proceedings of the 20th International Conference on Reliable Software Technologies (Ada-Europe 2015), Madrid, Spain, June 2015.

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.

6.1 Introduction

Building a safety case is an increasingly common practice in many safety critical domains [1]. 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 argument's 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) [2]. 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 [3][4], 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 6.2 we present background information. In Section 6.3 we propose a technique for maintaining safety cases using sensitivity analysis. In Section 6.4 we use the WBS example to illustrate the technique. In Section 6.5 we present the related work. Finally, we conclude and derive future works in Section 6.6.

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.

6.2.1 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 [2] provides a 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 10.1 (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.

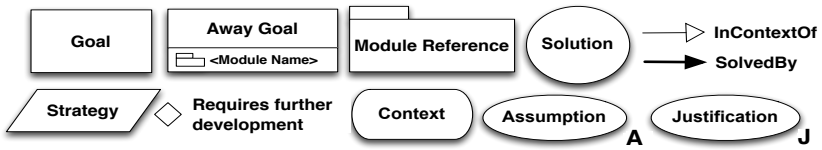

Figure 6.1: Notation Keys of the Goal Structuring Notation (GSN)

6.2.2 The Concept of Safety Contracts

The concept of contract is not uncommon in software development and it was first introduced in 1988 by Meyer [5] to constrain the interactions that occur between objects. Contract-based design [6] 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 that describes properties that are only safety-related is referred to as a safety contract.

6.2.3 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 case's 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 case's elements and interdependencies [7].

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 [7]. 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 [1]), 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.

6.2.4 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 [8]. 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 [9]. 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 ΔB 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.

6.3 Using Sensitivity Analysis To Facilitate The Maintenance of A Safety Case

In this section, we build on the background information provided in Section 6.2 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 6.2. 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.

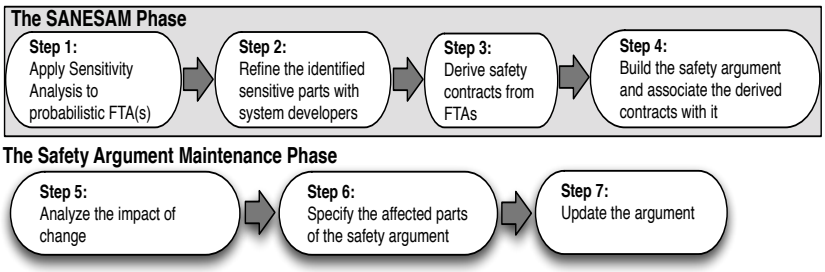

Figure 6.2: Process diagram of the proposed technique

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 [10]. As discussed in Section 9.1, 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"[11].*
  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
  $$(FP_{Increased|event(x)} - FP_{Actual|event(x)}) \not\Rightarrow FP_{Actual(Topevent)} > FP_{Required(Topevent)}.$$
  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 6.3a, 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 6.3a). 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 6.3b.

- ***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.

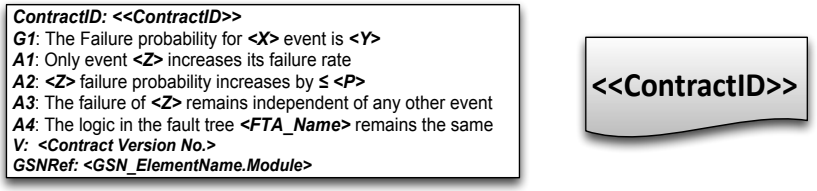

Figure 6.3: (a) Safety Contract Template (b) Safety Contract Notation for FTA

In order to associate the derived safety contracts with GSN arguments, we reuse our previous work [10]. 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 6.3b.
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

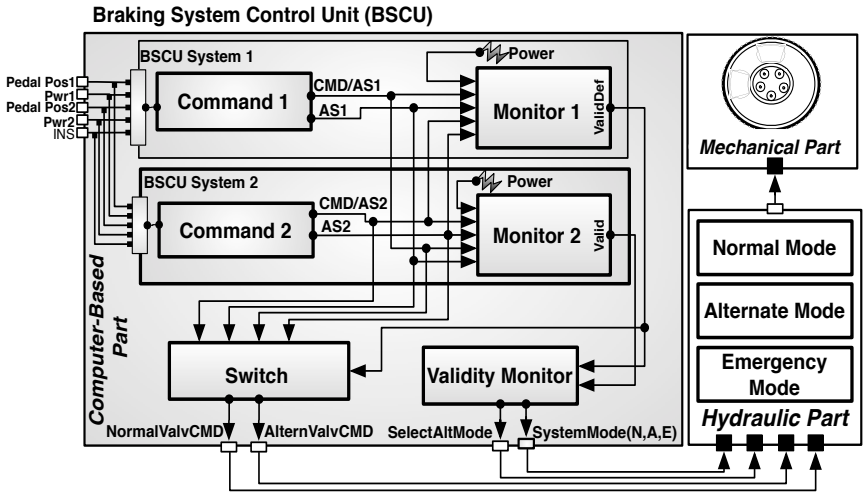

Figure 6.4: A high-level view of the WBS

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 6.3a.

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

6.4 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 [12]. Figure 6.4 shows a high-level architecture view of the WBS

6.4.1 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 6.4). 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 [13].

It worth noting that Figure 6.4 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 [12].

6.4.2 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: $FP_{Required(Topevent)} <$ **3.30E-5**. In line with this, we assumed that the $FP_{Actual(Topevent)} \approx$ **1.50E-6**. Figure 6.5 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:

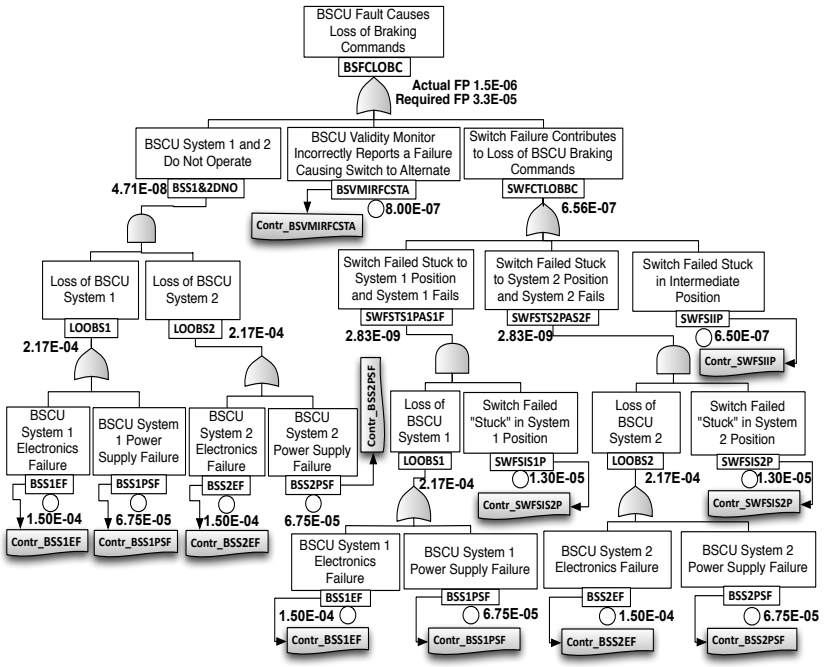

Figure 6.5: Loss of Braking Commands FTA

1. Find minimal cut set *MC* in the FTA: there are several algorithms to find the *MC*. We apply Mocus cut set algorithm [11], 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 ΔFP for each event. Table 6.1 shows the calculated $FP_{Increased|event(x)}$ and ΔFP.
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 6.1 shows the sensitivity values and the

Table 6.1: 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

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 6.1 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.

- ***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 6.3a 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 6.6).

- ***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 6.7. The derived safety contracts are associated with the derived safety contracts according to ***Steps 4*** in Section 6.3. *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

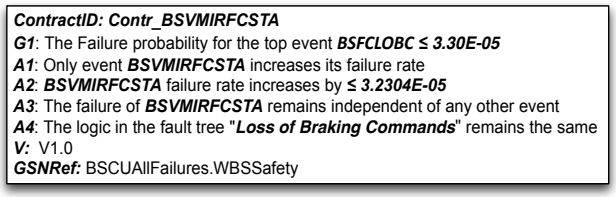

Figure 6.6: A derived safety contract

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 6.3. 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.

6.5 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. IAWG's 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 [3][4]. 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)

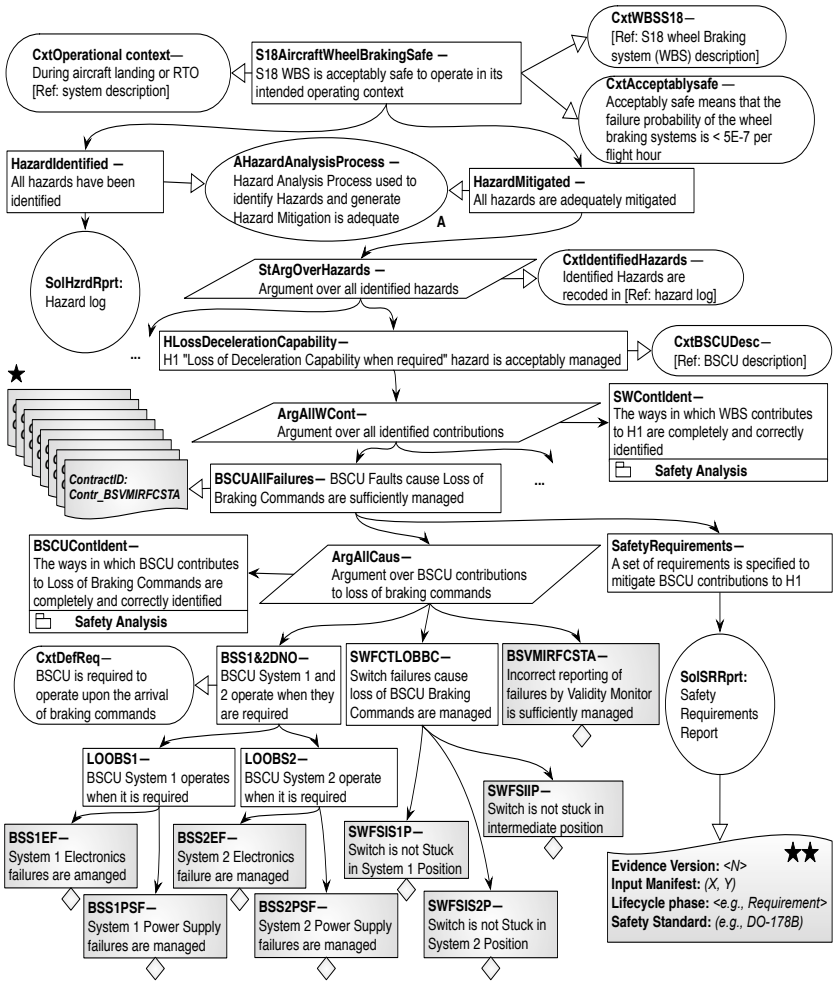

Figure 6.7: Safety argument fragment for WBS

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 [14] 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.

6.6 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.

Bibliography

[1] ISO 26262:2011. Road Vehicles — Functional Safety, Part 1-9. International Organization for Standardization, Nov 2011.

[2] GSN Community Standard: (http://www.goalstructuringnotation.info/) Version 1; (c) 2011 Origin Consulting (York) Limited.

[3] Modular software safety case (MSSC) — process description. https://www.amsderisc.com/related-programmes/, Nov 2012.

[4] Jane L Fenn, Richard D Hawkins, PJ Williams, Tim P Kelly, Michael 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.

[5] Bertrand Meyer. *Object-Oriented Software Construction*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1st edition, 1988.

[6] 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.

[7] T.P. Kelly and J.A. McDermid. A systematic approach to safety case maintenance. In *Computer Safety, Reliability and Security*, volume 1698, pages 13–26. Springer Berlin Heidelberg, 1999.

[8] A.C. Cullen and H.C. Frey. *Probabilistic techniques in Exposure assessment*. Plenum Press, New York, 1999.

[9] David J. Pannell. Sensitivity analysis of normative economic models: theoretical framework and practical strategies. *Agricultural Economics*, 16(2):139 – 152, 1997.

[10] Omar Jaradat, Patrick J. Graydon, and Iain Bate. An approach to maintaining safety case evidence after a system change. In *Proceedings of the 10th European Dependable Computing Conference (EDCC)*, August 2014.

[11] Marvin Rausand and Arnljot Høyland. *System Reliability Theory: Models, Statistical Methods and Applications*. Wiley-Interscience, Hoboken, NJ, 2004.

[12] SAE ARP4761 Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment, December 1996.

[13] 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.

[14] T. Kelly. Literature survey for work on evolvable safety cases. Department of Computer Sceince, University of York, 1st Year Qualifying Dissertation, 1995.

[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.

Chapter 7

Paper B: Deriving Hierarchical Safety Contracts

Omar Jaradat and Iain Bate.
In Proceedings of the 21st IEEE Pacific Rim International Symposium on Dependable Computing (PRDC 2015), IEEE, Zhangjiajie, China, November 2015.

Abstract

Safety cases need significant amount of time and effort to produce. The required amount of time and effort can be dramatically increased due to system changes as safety cases should be maintained before they can be submitted for certification or re-certification. Sensitivity analysis is useful to measure the flexibility of the different system properties to changes. Furthermore, contracts have been proposed as a means for facilitating the change management process due to their ability to record the dependencies among system's components. In this paper, we extend a technique that uses a sensitivity analysis to derive safety contracts from Fault Tree Analyses (FTA) and uses these contracts to trace changes in the safety argument. The extension aims to enabling the derivation of hierarchical and correlated safety contracts. We motivate the extension through an illustrative example within which we identify limitations of the technique and discuss potential solutions to these limitations.

7.1 Introduction

The concept of a safety case originated in 1989 when the UK Health and Safety Executive (HSE) requested from the British nuclear sites to generate a written report — according to the Control of the Industrial Major Accident Hazards (CIMAH) regulations — that should contain: (1) facts about the site, and (2) reasoned arguments about the hazards and risks from the site [1]. This report is known as a safety case, which should systematically demonstrate a reasoned argument that a nuclear site is acceptably safe to operate. More specifically, a safety case should identify the major hazards and risks in a nuclear site and demonstrate that the site is satisfactory since all of these hazards and risks are adequately mitigated. The increasing size and complexity of safety critical systems motivate the application of the safety case concept in different domains (e.g., oil, avionics, railway, automotive, etc.) to demonstrate a reasoned argument that those systems are acceptably safe to operate.

Safety certification is typically imposed by authorities as a censorship procedure to control the development of safety critical systems. The certification processes are based on an evaluation of whether the hazards associated with a system are mitigated to an acceptable level. Developers must provide evidence of safety to their regulators. Safety cases can explain how this evidence shows that the system is acceptably safe to operate. Hence, the development of safety cases has become common practice.

The certification process is amongst the most expensive and time-consuming tasks in the development of safety critical systems. A key reason behind that is the increasing complexity and size of these systems combined with their growing market demands. Moreover, safety critical systems are expected to operate for a long period of time and frequently subject to changes during both development and operational phases. Any change that might compromise system safety involves repeating the certification process (i.e., re-certification) and thus, ultimately, necessitates maintaining the corresponding safety case. 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" [2, 3]. Similarly, the UK HSE *Railway safety case regulations 1994* states in regulation *6(1)* that "A safety case is to be revised whenever, appropriate that is whenever any of its contents would otherwise become inaccurate or incomplete" [4, 3].

One of the biggest challenges that affects safety case revision and maintenance is that a safety case comprises a complex web of interdependent elements. That is, safety goals, evidence, argument, and assumptions about operating context are highly interdependent and thus, seemingly minor changes may have a major impact on the contents and structure of the safety argument. As such, a single change to a safety case may necessitate many other consequential changes — creating a ripple effect [5]. For example, if a new system component was integrated into a system, old items of evidence might no longer support the developers' claims about components' consistency because these claims reflect old assumptions in the development artefacts that do not take into consideration the new component integration.

In order to maintain a safety case after implementing a system change, system developers need to assess the impact of changes on the original safety argument. The assessment shall include reviewing the relevant assumptions made in the argument, and examining the adequacy of the collected body of evidence. For example, the UK Defence *Standard DS 00-56* states that: "Any amendments to the deployment of the system should be examined against the assumptions and objectives contained in the safety case" [6], [3]. 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 [5]. 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). This increases the maintenance cost.

Modularity has been proposed as the key element of the 'way forward' in developing systems [7]. For modular systems, 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. Hence, predicted changes can have traceable consequences as engineers will be aware of how a change in one module can result in a change in another module. In practice, it is hard to align the safety case structure with the system's modules [8]. However, a well-established traceability between system's modules and its safety case can provide the same traceable consequences of changes in the safety case. The problem though is that system changes and their details cannot be fully predicted and made available up front. In particular, software aspects of the safety case is hard to be predicted as software is highly change-

able and harder to contain. In this paper, we use sensitivity analysis based approach to assist system's engineers to predict changes.

In our previous work [8], we introduced a technique that contains Sensitivity ANalysis for Enabling Safety Argument Maintenance (SANESAM) phase that supports system engineers to anticipate potential changes. The key principle of SANESAM phase is to determine the flexibility (or compliance) of a system to changes using sensitivity analysis. The output is a ranked list of FTA events that system engineers can refine. The result after the refinement is a list of contracts that can be used as part of later change impact analysis. We also use safety contracts to record the information of changes that will ultimately advise the engineers what to consider and check when changes actually happen. The main contribution of this paper comprises (1) identifying possible limitations for SANESAM, and (2) suggest an a SANESAM extension to resolve the identified limitations. The paper uses the hypothetical aircraft Wheel Braking System (WBS) to illustrate SANESAM extension.

This paper is composed of four further sections. In Section 7.2 we present background information about sensitivity analysis, safety contracts, Goal Structuring Notations (GSN), incremental certification and WBS description. In Section 7.3, we give an overview of a technique to facilitate the maintenance of safety cases and identify limitations. In Section 7.4, we suggest extending the technique to resolve the identified limitations, we also use the WBS system to illustrate the extensions. Finally, we conclude and propose future works in Section 7.5.

7.2 Background

7.2.1 Sensitivity Analysis

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" [9]. The analysis helps to establish reasonably acceptable confidence in the model by studying the uncertainties that are often associated with variables in models [10]. There are different purposes for using sensitivity analysis, such as, providing insight into the robustness of model results when making decisions [11]. For instance, sensitivity analysis can be used to determine what level of accuracy is necessary for a parameter (variable) to make the model sufficiently useful and valid [12].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.

Emberson et al. [14] use sensitivity analysis to improve the flexibility of task allocation in real-time system design. More specifically, the analysis is used to evaluate the impact on task allocation solution after applying possible change scenarios to task allocation framework.

In this paper, 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 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. A sensitive event is an 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 a sensitive component in this paper. Changes to a sensitive component cause a great impact to system design. [8]

7.2.2 Safety Contracts

The concept of contract is familiar in software development. For instance, Design by Contract (DbC) was introduced in 1986 [15, 16] to constrain the interactions that occur between objects. 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 behavior [17]. In the context of contract-based design, 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 surrounding environment satisfies one or more related assumption(s). 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 [18, 19, 20]. 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 [21, 22, 23].

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

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

7.2.3 Safety Argumentation and Goal Structuring Notations (GSN)

GSN was introduced to provide a graphical means of communicating (1) safety argument elements, claims (goals), argument logic (strategies), assumptions, context, evidence (solutions), and (2) the relationships between these elements [24]. The principal symbols of the notations (with example instances of each concept) are shown in Figure 7.1. A goal structure shows how goals are successively broken down into ('solved by') sub-goals until a point is reached where claims can be supported by direct reference to evidence. Using GSN, the writer can clarify the adopted argument strategies (i.e., how the premises imply the conclusion), the rationale for the approach (assumptions, justifications) and the context in which goals are stated [8]. GSN has been extended to enable modularity in safety cases (i.e., module-based development of the safety case) so that it enables the partitioning of a safety case into an interconnected set of modules.

Well-structured argument may help the developers to mechanically propagate the change through the goal structure. However, it does not tell if the suspect elements of the argument in question are still valid. For example, having made a change to a model we must ask whether goals articulated over that model are still valid. Expert judgment is still required in order to answer such

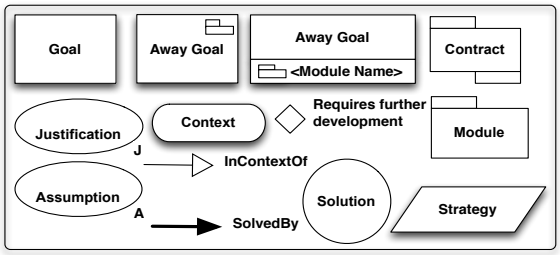

Figure 7.1: Overview of GSN Notations

questions [25]. Hence, merely having well-structured arguments does not directly help to preserve the soundness of the argument after a change, but it can more easily determine the questions to be asked to do so.

7.2.4 Incremental Certification

The Industrial Avionics Working Group (IAWG) — a consortium of researchers and practitioners — has proposed Modular Safety Cases as a means of containing the cost of change by dividing the safety case into a set of argument modules. IAWG's Modular Software Safety Case (MSSC) process [26] facilitates handling system changes as a series of relatively small increments rather than occasional major updates. The key principle of the state-of-the-art process is to modularise a safety case so as to contain changes within a minimal area of the safety case [26]. More specifically, the process starts by anticipating potential changes over the lifetime of a system. System developers modularise the argument so as to contain the impact of the anticipated changes. Hence, MSSC process ensures that the maximum amount of safety case material that was previously certified is not impacted, and thus it is available for re-use in the re-certification process without a need to be revisited [26].

7.2.5 Wheel Braking System (WBS): System Description

The WBS is a hypothetical aircraft braking system described in Appendix L of a popular standard for safety assessment processes, ARP4761 [27]. Figure 7.2 shows a high-level architecture view of the WBS. The system is installed on the two main landing gears of a civil air transport. 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: 1. Computer-based part which is called the Brake System Control Unit *(BSCU)*, 2. *Hydraulic* part, and 3. *Mechanical* part.

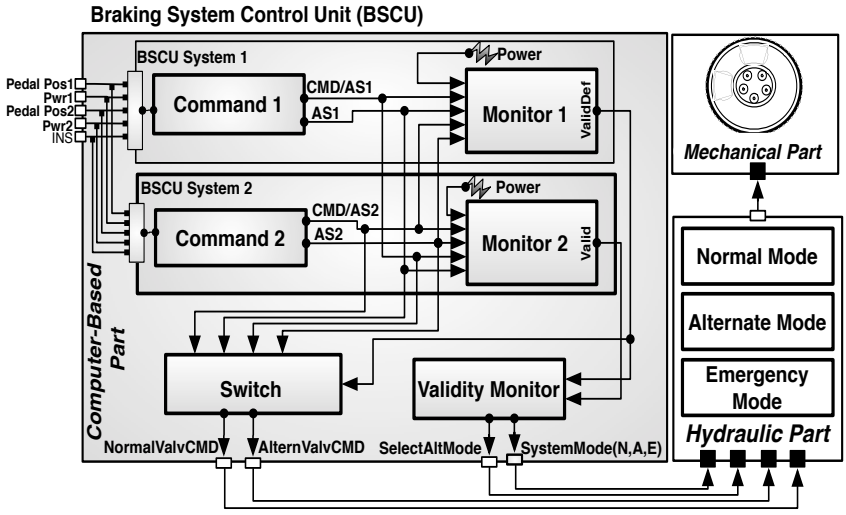

Figure 7.2: A high-level view of the WBS [8]

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 8.4). 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 [8, 27, 28]. Figure 7.2 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 [27]. Figure 7.3 shows the "Loss of Braking Commands" probabilistic FTA.

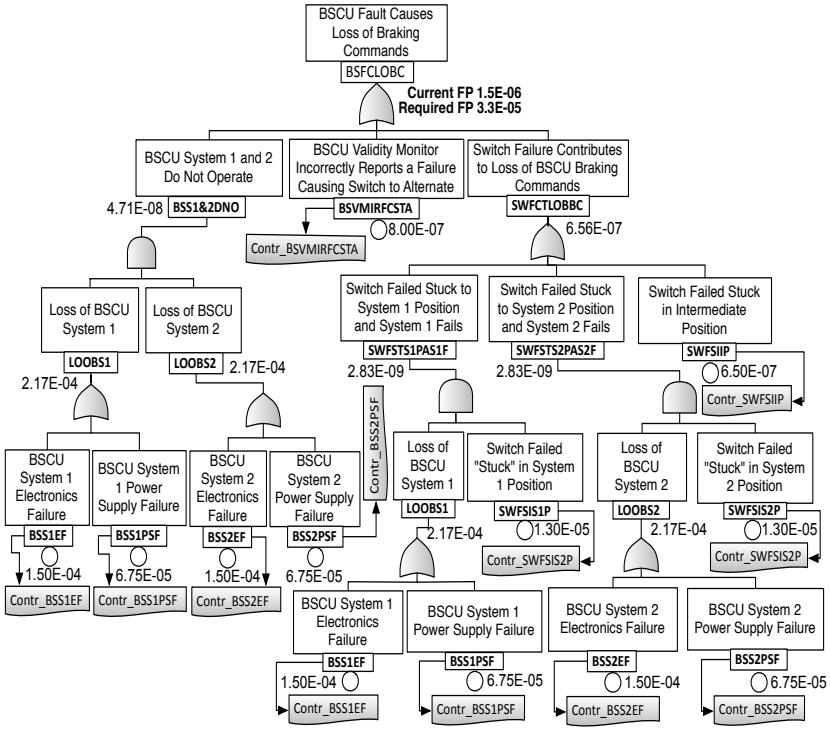

Figure 7.3: Loss of Braking Commands FTA [8]

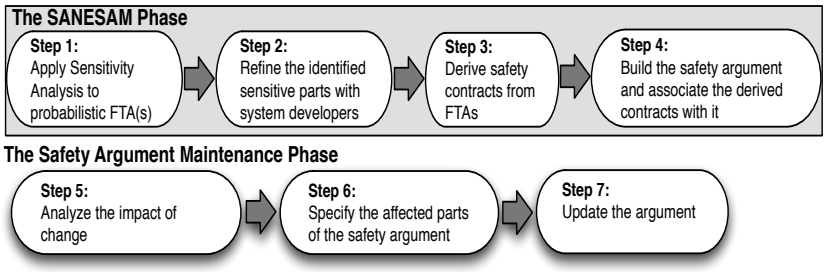

Figure 7.4: Process diagram of safety cases maintenance technique

7.3 A Technique to Facilitate the Maintenance of Safety Cases

In this section we give an overview of 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 7.4. 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. Considering a complete list of anticipated changes is difficult. This technique uses sensitivity analysis to measure the flexibility of system components to changes. That is, regardless of the type of changes it 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 [8]. Furthermore, the technique utilises the concept of contracts to record the information of changes that will ultimately advise the engineers what to consider and check when changes actually happen.

7.3.1 SANESAM Phase

The rationale of this phase 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.). Sensitivity analysis is used in this phase as a method to determine the range of failure probability parameter for each component. The technique assumes 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, the technique assumes the existence of the required failure probability for the top event $FP_{Required(Topevent)}$, where the FTA is considered unreliable if: $FP_{Current(Topevent)} > FP_{Required(Topevent)}$. [8]

The steps of SANESAM phase are as follows: [8]

- ***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"* [29].
2. Calculate the maximum possible increment in 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.

- ***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 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. It is worth noting that the role of safety contracts in SANESAM is

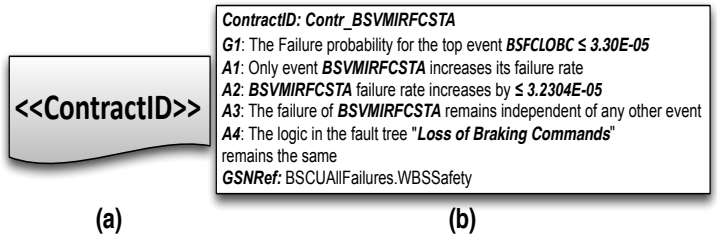

Figure 7.5: (a) FTA Safety contract notation, (b) Derived safety contract

to highlight sensitive events, and not to enter new event failure probabilities. We introduce a new notation to FTAs to annotate the contracted events, where every created contract should have a unique identifier, see Figure 7.5-a. Figure 7.3 shows the derived safety contracts as a result of SANESAM application to the Loss of Braking Command FTA. We also create a template to document the derived safety contracts. Figure 7.5-b shows an instantiation of the contents of one of the derived safety contracts for WBS.

- ***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. Associating the contracts with GSN goals is done by using the introduced notation in Figure 7.5-a.

7.3.2 SANESAM Limitations

The essence of SANESAM is to calculate the maximum possible increment in the failure probability parameter of only one event at a time before the top event $FP_{Required(Topevent)}$ is no longer met. In this section, we identify three limitations of the current version of SANESAM and we give an example for each limitation.

No Support for Intermediate Events

The followed method for applying sensitivity analysis relies on the calculated cut set for the full FTA. Hence, only basic events are considered during the

application of sensitivity analysis and no contracts are derived for the intermediate events. Figure 7.6-a shows an example of this limitation, where LOOBS1 is an intermediate event and based on SANESAM it cannot be provided as a sensitive event thus no contract can be derived for it. Having said that, system developers may need to contain the impact of changes in intermediate levels to prevent them from rippling through the top-level event. Additionally, some events might look trivial for system engineers but if those events were packed in events from higher levels, then they could look nontrivial. For example, providing system engineers with "Jam of speedbrake lever" as a sensitive event to a particular change might look less serious than the parent event "Mechanical failures of speedbrake lever". Another motivation for deriving contracts on intermediate levels comes from the fact that some intermediate events may represent top goals in the safety case modules which will be more supportive for incremental certification as introduced in Section 7.2.4. In other words, pinpointing the entire safety case module as affected is easier than starting from intermediate goals in that module. From the forgoing reasons, SANESAM should be able to provide system engineers with sensitive events from different levels of the FTA's hierarchy.

No Support for Multiple Events Impact

SANESAM calculates the highest possible boundary of failure probabilities for certain events. SANESAM also assumes independence of events and does not address the problem of event interdependencies that is typical for any realistic system. This means that only one event failure probability is allowed to increase per change. However, a change might impact multiple events in the same time. For instance, adding distinct functional redundancy of a critical software component might decrease the failure probability of multiple events in the FTA. Likewise, removing a redundant component to make the system simpler and cheaper might increase the failure probability of multiple events. Since the failure probabilities of multiple components often change at once, a SAMESAM extension to handle such changes is highly desirable. A clear example can be given by assuming a change to BSCU System 1 power supply in Figure 7.6-a. A change to BSCU System 1 power supply will necessitate a correlated change to BSCU System 2 power supply. Hence, when we need to calculate the possible increment to the failure probability of BSS1PSF (for this specific change), we must take into account the correlation between BSS1PSF and BSS2PSF.

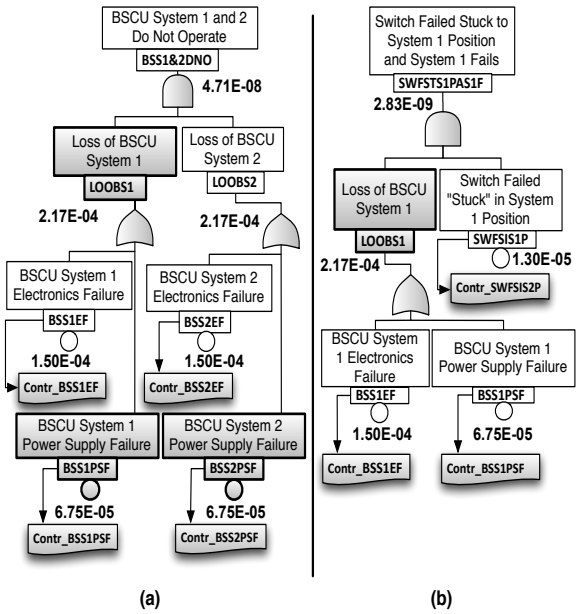

Figure 7.6: Limitation examples

Neglecting Duplication

It is possible for an event to be represented in more than one location in the same FTA. For example, LOOBS1 event is represented twice in this FTA. In the first representation (Figure 7.6-a) it is combined with LOOBS2 by AND gate, where both events have the same failure probability. In its second representation (Figure 7.6-b), LOOBS1 is combined with SWFSIS1P by AND gate, both events have different failure probabilities. Hence, the possible increment on LOOBS1 failure probability will vary in the two locations. SANESAM neglects events duplication, and this is considered a limitation because the calculated possible failure probability increment of the same event may vary in the same FTA if the event is duplicated. Calculating the possible increment on the failure probability of a duplicated event is based on the failure probability(s) of the combined events. More clearly, in each duplication of an event, the event may be combined with different event(s), different failure probability values, and by different gates (e.g., AND, OR, XOR, etc.).

7.4 SANESAM Extension

In this section, we suggest extending SANESAM to resolve the limitations that were identified in Section 7.3.2. The extended SANESAM is referred to as SANESAM+ in this paper. SANESAM and SANESAM+ are mutually exclusive and selecting among them is very dependent on the refined list by system engineers in Step 2 of the technique (described in Section 7.3.1). More clearly, if at least one of the events in the refined list is duplicated, or if its change necessitates a correlated event to change, then SANESAM+ is the one to go. Otherwise, developers are free to choose SANESAM or SANESAM+. However, choosing SANESAM means that the developers accept the assumption that only one event is allowed to change at a time.

SANESAM relies on the calculated *MC* (minimum cut set) for the full FTA which means that only the basic events are considered for sensitivity analysis. However, SANESAM+ requires measuring the sensitivity of all events in the FTA. This means that we need to calculate the Maximum Allowed Failure Probability (MAFP) for each event in the FTA taking into account that all events may change at a time. That is, $\Delta FP_{(Topevent)}$ = ($FP_{Required(Topevent)} - FP_{Current(Topevent)}$.) will be distributed over all FTA's events, where $\Delta FP_{(Topevent)} > 0$.
In order to apply SANESAM+ and calculate the MAFP for FTA events, we replace the procedure of Step 1 in Section 7.3.1 with the following procedure:

1. Find the difference between new and current FPs of the ancestor events, as follows:

   $$\Delta FP_{(Ancestor)} = FP_{New} - FP_{Current}$$

   The first run of this step should start with $\Delta FP_{(Topevent)}$, where the new FP in this specific case is the required FP. The second run should be for each event in the very next level and so on and so forth until the basic events are reached.

2. This sub-step is very dependent on the type of the gate between the ancestor and descendant events. In case of OR gate, sub-steps 2-A and 2-B should be followed. In case of AND gate, sub-step 2-C should be followed.

   (a) Find the ratio of the descendant events to the ancestor event. The first run of this step should start with the top event and the events beneath it. The second run of this step should consider one more

level down. In other words, descendant events in the first run will become ancestors in the second one. The ratio of a descendant event to its ancestor is calculated by Equation 7.1, as follows:

$$Ratio_{Desc(x)} = \frac{FP_{Current(Desc(x))}}{FP_{Current(Ancestor)}} \tag{7.1}$$

(b) Increase the FP for each of the descendant events by $\Delta FP_{(Ancestor)}$ which is calculated in step 1. Increasing events' FP is done by Equation 7.2, as follows:

$$\begin{aligned} FP_{Increased|Desc(x)} &= FP_{Current(Desc(x))} \\ &+ (Ratio_{(Desc(x))} * \Delta FP_{(Ancestor)}) \end{aligned} \tag{7.2}$$

(c) In this sub-step, we need to distribute the increment to the FP of an ancestor event over its descendent events in the presence of an AND gate. The increment to each descendant event is calculated in two different ways based on the number of descendent events and if their FPs vary.

Case 1. if the events share the same FP value, we can use: $\sqrt[n]{FP_{(Increased|Ancestor)}}$, where n is the number of the descendent events.

LOOBS1 and LOOBS2 in Figure 7.6-a represents an example of this case.

Case 2. if the descendent events do not share the same FP, then $FP_{(Topevent)}$ is distributed over them unevenly, but rather based on the FP ratio of every descendent event to $\Delta FP_{(Ancestor)}$ as described by Equation 7.3:

$$FP_{Current(Desc(x))} + (\frac{FP_{Current(Desc(x))}}{\sum FP_{Current(AllDesc)}} * I) \tag{7.3}$$

In order to determine I we need to consider all sibling events as

described in Equation 7.4:

$$\begin{aligned}
&(FP_{Current(Desc(x_1))} + (\frac{FP_{Current(Desc(x_1))}}{\sum FP_{Current(AllDesc)}} * I)) \\
&*(FP_{Current(Desc(x_2))} + (\frac{FP_{Current(Desc(x_2))}}{\sum FP_{Current(AllDesc)}} * I)) \\
&*(FP_{Current(Desc(x_n)} + (\frac{FP_{Current(Desc(x_n))}}{\sum FP_{Current(AllDesc)}} * I)) \\
&= FP_{Increased(Ancestor)}
\end{aligned} \tag{7.4}$$

LOOBS1 and SWFSIS1P in Figure 7.6-b represent an example of this case.

3. Repeat steps 1 and 2 until FP of the basic events get increased. Unlike SANESAM, SANESAM+ distinguish between duplicated events. That is, if an event shows up in multiple locations in the FTA, we still need to calculate its FP wherever we encounter it. Later on when finish calculating the FP for all duplicates of an event we unify the its FP by considering the minimum calculated FP of them.

4. Finally, rank the sensitivity of events from the most sensitive to the less sensitive. The most sensitive event is the event for which Equation 7.5 is the minimum, as follows:

$$Sensitivity_{(x)} = \frac{FP_{Increased(x)} - FP_{Current(x)}}{FP_{Current(x)}} \tag{7.5}$$

It is worth noting that the difference between the steps of SANESAM and SANESAM+ is observed only in Step 1, all other later steps are identical.

7.4.1 SANESAM+ Application: An Example

In this section, we use the Loss of Braking Commands FTA (in Figure 7.3) to show an application example of SANESAM+.

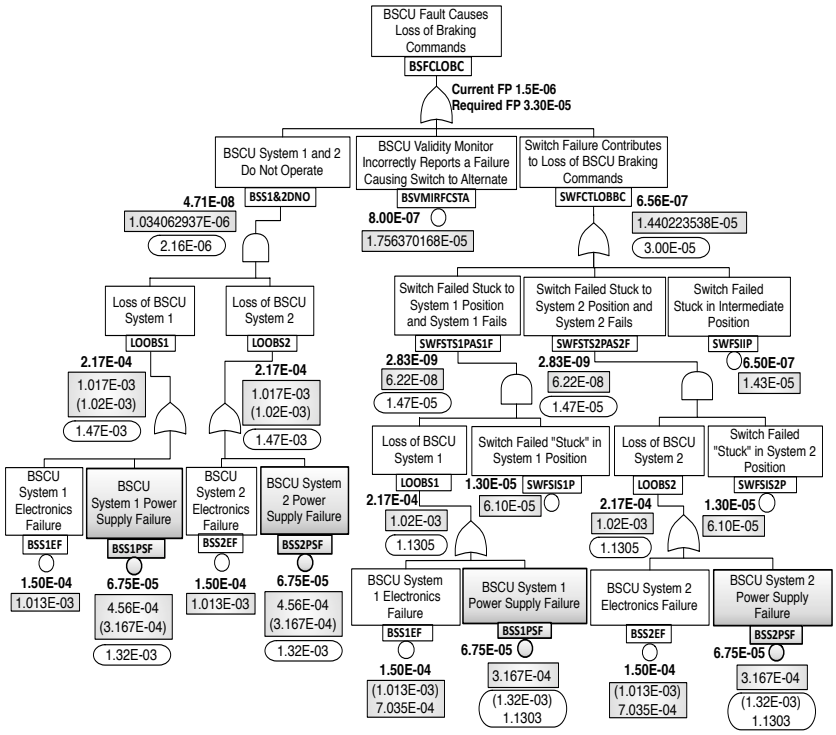

Figure 7.7: Loss of Braking Commands FTA [8]

1. Find $\Delta FP_{(Ancestor)}$ (which is the top event in the FTA for the first of this sub-step): $\Delta\, FP_{(BSFCLOBC)}$ = 3.30E-05 - 1.5031E-06
   $\Delta\, FP_{(BSFCLOBC)}$ = 3.14969E-05

2. Since BSFCLOBC is correlated with its descendants (i.e., BSS1&2DNO, BSVMIRFCSTA and SWFCTLOBBC) via OR gate, then sub-steps 2-A and 2-B should be followed.

   (a) We need to find FP ratio for each of BSS1&2DNO, BSVMIRFC-STA and SWFCTLOBBC to BSFCLOBC using Equation 7.1.

   Example: BSS1&2DNO

   $$Ratio_{BSS1\&2DNO} = \frac{4.71E{-}08}{1.5031E{-}06}$$

$Ratio_{BSS1\&2DNO}$ = 3.133524050E-02.

(b) In this sub-step, $\Delta FP_{(BSFCLOBC)}$ should be distributed over each of BSS1&2DNO, BSVMIRFCSTA and SWFCTLOBBC based on their ratios to $FP_{Actaul(Ancestor)}$ using Equation 7.2.

Example: BSS1&2DNO
$FP_{Increased(BSS1\&2DNO)}$ =
4.71E-08 + (3.133524050E-02 * 3.14969E-05)
$FP_{Increased(BSS1\&2DNO)}$ = 1.034062937E-06

(c) Now, let us take other examples where AND gate correlates an ancestor event with its descendent events. The example covers Case 1 and 2 as described in sub-step 2-C in Section 7.4.

Example of Case 1: LOOBS1 and LOOB2.
$FP_{(Increased(x)} = \sqrt[2]{FP_{(Increased|BSS1\&2DNO)}}$
$FP_{(Increased|LOOBS1)}$ = 1.016889E-03
$FP_{(Increased|LOOBS2)}$ = 1.016889E-03

Example of Case 2: LOOBS1 and SWFSIS1P
In this example, the FP of LOOBS1 and SWFSIS1P are increased using Equations 7.3 and 7.4.

$$= (2.17E\text{–}04 + (\frac{2.17E\text{–}04}{2.17E\text{–}04 + 1.30E\text{–}05} * I)) * (1.30E\text{–}05 + (\frac{1.30E\text{–}05}{1.30E\text{–}05 + 2.17E\text{–}04} * I)) = 6.216381375E\text{–}08$$

$FP_{(Increased|LOOBS1)}$ = 1.02E-03
$FP_{(Increased|SWFSIS1P)}$ = 6.10E-05

3. In this step, we repeat the 1 and 2 steps until FP of the basic events get increased. Figure 7.7 shows the calculated FPs (in boxes) using SANESAM+. Looking at the figure, It should be observed that the events, BSS1EF, BSS2EF, BSS1PSF and BSS2PSF are duplicated.

Table 7.1: The results of SANESAM+ sensitivity analysis

Event Name	Current FP	Increased FP	Sensitivity	Rank
SWFSIS1P SWFSIS2P	1.30E-05	6.10E-05	3.692307692	2
LOOBS1 LOOBS2	2.17E-04	1.02E-03	3.700460829	3
BSS1EF BSS2EF	1.50E-04	1.013E-03	5.753333333	4
BSS1PSF BSS2PSF	6.75E-05	3.167E-04	3.69185185	1
SWFCTLOBBC	6.56E-07	1.44E-05	20.95121951	5
SWFSTS1PAS1F SWFSTS2PAS2F	2.83E-09	6.22E-08	20.97879859	6
BSVMIRFCSTA	8.00E-07	1.76E-05	21	7
SWFSIIP	6.50E-07	1.43E-05	21	7
BSS1&2DNO	4.71E-08	1.04E-06	21.08067941	8

These events were assumed independent from each other while calculating their FPs. However, this assumption was vanished after the calculation and the minimum FP (values between brackets in Figure 7.7) was considered the maximum possible FP for each duplicate events. For example, two FP values were obtained for BSS1EF in different locations (i.e., 4.56E-04 and 3.167E-04) but since 3.167E-04 is the minimum FP value of the two duplicates, it is, therefore, the maximum possible FP for all BSS1EF's duplicates.

4. In this step, we use Equation 7.5 in Section 7.4. Table 7.1 shows the results of the sensitivity analysis and the ranking of the events' sensitivity where 1 is the most sensitive event.

7.4.2 SANESAM+ For Predicted Changes

SANESAM+ can be useful even for arbitrary changes. That is, even if the system engineers are not sure of the potential future changes, SANESAM+ enable the derivation of safety contracts for all events in different levels in the FTA. Hence, when a change request shows up, system engineers, and by returning to the sensitivity results, can decide whether the effect of the change is tolerable or not. However, SANESAM+ can be more useful in the presence of a predicted change as it can increase the effect tolerance of that change. More clearly, distributing $\Delta FP_{(Topevent)}$ over all FTA's events might increase the change impact tolerance of some events that are unlikely to change. On the other hand, the change impact tolerance might be slightly increased for events that are more likely to change. Consequently, having a change scenario in advance will motivate increasing the change impact tolerance for only the events that fall in the scope of that change. Since, however, SANESAM+ (for predicted changes) will exclude the events that are unlikely to change, we will slightly modify the steps by which we calculate the FP of events. The following steps give guidance on how to calculate the FP SANESAM+ for predicted change scenarios:

1. Find the difference between the current and required FP of the top event $\Delta FP_{(Topevent)}$.
2. Find the highest event that contains the effect. If the highest event does not fall directly under the top event, the effect should be traced up the fault tree further until we reach the affected event that falls directly under the top event.
3. Distribute the calculated $\Delta FP_{(Topevent)}$ in sub-step 1 to the identified events in sub-step 2 based on the determined ratio in sub-step 3. The first run of this sub-step should start with the top event and the events beneath it, and the second runshould consider one more level down.This sub-step is very dependent on the type of the gate between the ancestor and descendant events. In the case of an OR gate sub-step 4-A should be followed. In the case of an AND gate sub-step 4-B should be followed.
    (a) In this sub-step, we need to distribute the increment to the FP of an ancestor event over its descendent affected events in the presence of an OR gate. We first need to find the ratio of the affected event to its ancestor event. Afterwards, we need to use the calculated ratio to determine the amount of the increment to the affected event. The first run of this step should start with the affected events that

fall directly under the top event. The second run of this step should consider one more level down. In other words, descendant events in the first run will become ancestors in the second one. The simplest FP calculation is when to have two descendent events and only one of them is affected. This is because all what we need to do is to subtract the unaffected FP from the increased ancestor event to get the the increased FP of the affected event as presented in Equation 7.6:

$$FP_{Increased(Ancestor)} - FP_{Current(Unaffect|Desc(x))} = FP_{Increased(Desc(x))} \tag{7.6}$$

Otherwise, the ratio of a descendant event to its ancestor and the granted increment to an affected event is calculated by Equation 7.7 as follows:

$$FP_{Increased(Desc(x))} = \left(\frac{FP_{Current(Desc(x))}}{FP_{Current(Ancestor)} - \sum FP_{Current(Unaffect)}} * FP_{Increased(Ancestor)}\right) + FP_{Current(x)} \tag{7.7}$$

(b) In this sub-step, we distribute the increment to the FP of an ancestor event over its affected descendent events in the presence of an AND gate. The increment calculation is dependent on five cases, as follows:

Case 1. Two descendent events and only one of them is affected. This is the simplest case because all what we need to do is to divide the increased FP of the ancestor event on the current FP of the unaffected descendent event as presented in Equation 7.8:

$$FP_{Increased(Desc(x))} = \frac{FP_{Increased(Ancestor)}}{FP_{Current(Unaffect|Desc(x))}} \tag{7.8}$$

Case 2. All descendent events are affected and share the same FP value. In this case, we apply:
$\sqrt[n]{FP_{(Increased|Ancestor)}}$, where n is the number of the descendent events.

Case 3. All descendent events are affected and do **NOT** share the same FP value. In this case, we apply equations (3) and (4) as described in Section 7.4.

Case 4. NOT all descendent events are affected where the affected ones share the same FP value. In this case, we apply Equation 7.9 as follows:

$$FP_{Increased(Desc(x))} = \sqrt[n]{\left(\frac{FP_{Increased(Ancestor(x))}}{\sum FP_{Current|Unaffect(x_1)*(x_2)*...*(x_n))}}\right)} \tag{7.9}$$

where n is the number of the affected events.

Case 5. NOT all descendent events are affected where the affected ones do **NOT** share the same FP value. In this case, we use Equation 7.10, as follows:

$$\frac{FP_{Increased(Ancestor(x))}}{\sum FP_{Current|Unaffected(x_1)*(x_2)*...*(x_n))}} \tag{7.10}$$

4. Repeat step 3 until the FP of all affected events get increased.

7.4.3 SANESAM+ For Predicted Changes: An Example

In this example we again use the WBS FTA. We consider a predicted change that will be applied to the power supplies within both *BSCU* 1 and 2. However, it is still unknown how this change will increase the FPs of the two power supplies. We apply "SANESAM+ For Predicted Changes" to dedicate the maximum allowed FP to the affected events by the change, as follows:

1. $\Delta FP_{(BSFCLOBC)}$ = 3.30E-05 - 1.5031E-06
   $\Delta FP_{(BSFCLOBC)}$ = 3.14969E-05

2. Find the highest event that contains the effect. Changes to System 1 and 2 power supplies will directly affect the events BSS1EF and BSS2EF as highlighted in Figure 7.7 . These two events, however, are duplicated elsewhere in the FTA and thus there are multiple high events that contain the change.

   (a) **BSS1EF on the left-hand side of the FTA** falls under LOOBS1 but the latter does not fall directly under the top event thus BSS1&2DNO is the highest event that contains the effect on BSS1EF.

(b) **BSS2EF on the left-hand side of the FTA** falls under LOOBS2 but the latter does not fall directly under the top event thus BSS1&2DNO is the highest event that contains the effect on BSS2EF.

(c) **BSS1EF on the right-hand side of the FTA** falls under LOOBS1 but the latter does not fall directly under the top event thus it is not the required highest event and we need to take one more level up to find the highest event. Having done that will lead us to SWF-STS1PAS1F which is also not the highest event that falls directly under the top event thus we need to go up again which will result SWFCTLOBBC as the required highest event.

(d) **BSS2EF on the right-hand side of the FTA** falls under LOOBS2 but the latter does not fall directly under the top event thus it is not the required highest event and we need to take one more level up to find the highest event. Having done that will lead us to SWF-STS1PAS2F which is also not the highest event that falls directly under the top event thus we need to go up again which will result SWFCTLOBBC as the required highest event.

3. Distribute the increment to the FP of an ancestor event over its descendent affected events. Since BSS1&2DNO and SWFCTLOBBC are the events that contain the change, no further calculations will be applied to BSVMIRFCSTA.

$$\left(\frac{4.71E\text{–}08}{1.5031E\text{–}06 - 8.00E\text{–}07} * 3.30E\text{–}05\right) + 4.71E\text{–}08 = 2.16E\text{–}06$$

$$\left(\frac{6.56E\text{–}07}{1.5031E\text{–}06 - 8.00E\text{–}07} * 3.30E\text{–}05\right) + 6.56E\text{–}07 = 3.00E\text{–}05$$

4. In this step, we repeat the previous step until all FPs of the affected events get increased. Figure 7.7 shows the calculated FPs (in squashed boxes).

5. In this step, we use 7.5 to calculate events' sensitivity.

It is worth noting that the sensitivity of BSS1PSF and BSS2PSF using SANESAM+ is 3.69185185 as shown in Table 7.1. However, the sensitivity of these events is 18.55555556 when SANESAM+ For Predicted Changes is used.

7.5 Conclusions and Future Work

Sensitivity analysis is useful to measure the flexibility of different system properties to changes. In our previous work, we proposed a technique comprises of two phases to facilitate the maintenance of safety cases. SANESAM is the first phase of the technique in which we (1) measure the sensitivity of FTA events to system changes using the events' failure probabilities, and (2) derive safety contracts based on the results of the analysis. In the second phase, we map the derived safety contracts to a safety argument to improve the change impact analysis on the safety argument. In this paper, we identified some limitations to SANESAM and we suggested two options as extensions to resolve these limitations. The first option is SANESAM+, which is useful in the case of arbitrary changes because it calculates the FP for all events in the FTA regardless of any change scenario. The second option is SANESAM+ For Predicted Changes, this option increases the FP for only the events that are associated to a predicted change. A derived safety contract by SANESAM+ For Predicted Changes can guarantee higher FP than the guaranteed FP (for the same event and using the same set of assumptions) in a derived safety contract by SANESAM+. Hence, the derived safety contracts by SANESAM+ For Predicted Changes are more tolerant and robust than those derived by SANESAM+. Future work will focus on describing the second phase of the technique. Creating a case study to validate both the feasibility and efficacy of the technique is also 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.

Bibliography

[1] CASSIDY K. CIMAH safety cases - the HSE approach. IChemE Symposium series no. 110, 1988.

[2] U.K. Ministry of Defence, “JSP 430 - Ship Safety Management System Handbook,” Ministry of Defence January 1996.

[3] Timothy Patrick Kelly. Arguing safety – a systematic approach to managing safety cases, 1998.

[4] HSE, ”Railway Safety Cases - Railway (Safety Case) Regulations 1994 - Guidance on Regulations,” Health and Safety Executive, HSE Books 1994.

[5] T.P. Kelly and J.A. 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. 1999.

[6] U.K. Ministry of Defence, 00-56 Safety Management Requirements for Defence Systems, Ministry of Defence, Defence Standard December 1996.

[7] S Bates, I Bate, R Hawkins, T P Kelly, J A 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.

[8] Omar Jaradat, Iain Bate, and Sasikumar Punnekkat. Using sensitivity analysis to facilitate the maintenance of safety cases. In *Proceedings of the 20th International Conference on Reliable Software Technologies*, June 2015.

[9] A. Saltelli. *Global sensitivity analysis: the primer*. John Wiley, 2008.

[10] Omar Jaradat, Iain Bate, and Sasikumar Punnekkat. Facilitating the maintenance of safety cases. In *Proceedings of the 3rd International Conference on Reliability, Safety and Hazard - Advances in Reliability, Maintenance and Safety (ICRESH-ARMS)*, June 2015.

[11] A.C. Cullen and H.C. Frey. *Probabilistic techniques in Exposure assessment*. Plenum Press, New York, 1999.

[12] Mark Choudhari Lucia Breierova. An introduction to sensitivity analysis. Technical report, Massachusetts Institute of Technology, September 1996.

[13] David J. Pannell. Sensitivity analysis of normative economic models: theoretical framework and practical strategies. *Agricultural Economics*, 16(2):139 – 152, 1997.

[14] P. Emberson and I. Bate. Stressing search with scenarios for flexible solutions to real-time task allocation problems. *Software Engineering, IEEE Transactions on*, 36(5):704–718, Sept 2010.

[15] B. Meyer. Design by contract. Technical Report TR-EI-12/CO, Interactive Software Engineering Inc., 1986.

[16] Bertrand Meyer. *Object-Oriented Software Construction*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1st edition, 1988.

[17] 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*, pages 58–71, Berlin, Heidelberg, 2008. Springer-Verlag.

[18] Albert Benveniste, Benoit Caillaud, Alberto Ferrari, Leonardo Mangeruca, Roberto Passerone, and Christos Sofronis. Multiple viewpoint contract-based specification and design. In *Proceedings of the 6th International Symposium*, pages 200–225, October 2007.

[19] Werner Damm, Hardi Hungar, Bernhard Josko, Thomas Peikenkamp, and Ingo Stierand. Using contract-based component specifications for virtual integration testing and architecture design. In *Proceedings of the Design,*

Automation & Test in Europe Conference & Exhibition (DATE), pages 1–6, 2011.

[20] Sebastian S. Bauer, Alexandre David, Rolf Hennicker, Kim Guldstrand Larsen, Axel Legay, Ulrik Nyman, and Andrzej Wasowski. Moving from specifications to contracts in component-based design. In *Proceedings of the 15th International Conference on Fundamental Approaches to Software Engineering*, FASE'12, pages 43–58, Berlin, Heidelberg, 2012. Springer-Verlag.

[21] Jane L Fenn, Richard D Hawkins, PJ Williams, Tim P Kelly, Michael 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.

[22] Safecer. (2013, June) Safety certification of software-intensive systems with reusable components. [Online]. Available: http://www.safecer.eu.

[23] Patrick Graydon and Iain 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.

[24] GSN Community Standard: (http://www.goalstructuringnotation.info/) Version 1; (c) 2011 Origin Consulting (York) Limited.

[25] S P Wilson, T P Kelly, and J A McDermid. Safety case development: Current practice, future prospects. In *Proceedings of the 12th Annual CSR Workshop - Software Based Systems*. Springer-Verlag, 1997.

[26] Modular Software Safety Case (MSSC) — Process Description. [online]. available: https://www.amsderisc.com/related-programmes, Nov 2012.

[27] SAE ARP4761 Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment, December 1996.

[28] 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 *Proceedings of the 24th International System Safety Conference (ISSC)*, Albuquerque, USA, 2006.

[29] Marvin Rausand and Arnljot Høyland. *System Reliability Theory: Models, Statistical Methods and Applications*. Wiley-Interscience, Hoboken, NJ, 2004.

Chapter 8

Paper C:
Using Safety Contracts to Guide the Maintenance of Systems and Safety Cases

Omar Jaradat and Iain Bate
In Proceedings of the 13rd IEEE European Dependable Computing Conference (EDCC), Geneva, Switzerland, December 2017.

Abstract

Changes to safety critical systems are inevitable and can impact the safety confidence about a system as their effects can refute articulated claims about safety or challenge the supporting evidence on which this confidence relies. In order to maintain the safety confidence under changes, system developers need to re-analyse and re-verify the system to generate new valid items of evidence. Identifying the effects of a particular change is a crucial step in any change management process as it enables system developers to estimate the required maintenance effort and reduce the cost by avoiding wider analyses and verification than strictly necessary. This paper presents a sensitivity analysis-based technique which aims at measuring the ability of a system to contain a change (i.e., robustness) without the need to make a major re-design. The proposed technique exploits the safety margins in the budgeted failure probabilities of events in a probabilistic fault-tree analysis to compensate for unaccounted deficits or changes due to maintenance. The technique utilises safety contracts to provide prescriptive data for what is needed to be revisited and verified to maintain system safety when changes happen. We demonstrate the technique on an aircraft wheel braking system.

8.1 Introduction

System safety is a major property that should be adequately assured during the development process, the deployment and the operation life of safety critical systems. System safety is not assured by chance but rather it must be engineered and evaluated in a systematic manner that might be mandated by safety standards, best practices and experts' recommendations. Hence, safety critical systems are often subject to a compulsory or advisory certification process which often necessitates building the systems in compliance with domain-specific safety standards.

Following the standards' prescriptions leads system developers to generate a lot of artefacts during and after the development of their systems. These artefacts are used as safety evidence to prove that the standards obligations and recommendations were carried out. However, if the generated artefacts are not demonstrated and explained properly, there will be less certainty about their importance which may lead the overall confidence being undermined. Therefore, developers of some safety critical systems construct a *safety case* (also known as *"assurance case"*) to demonstrate the safety aspect of a system by identifying all potential risks and describing, in the light of the available evidence, how these risks have been eliminated or duly mitigated.

Typically, safety critical systems are evolutionary and they are always exposed to both predicted and unpredicted changes during the different stages in their lifecycle. Changes to a system can negatively affect the gained confidence because these changes have the potential to compromise the safety evidence which has been already collected. More clearly, evidence after a change might no longer support the developers' claims because it reflects old development artefacts or old assumptions about operation or the operating environment. In addition, the cost of obtaining certification is significant, with estimates such as 30% of lifecycle costs [1] and 25-75% of development costs [2] are spent on certification [3]. Hence, improper handling of system changes in the safety cases can reflect untrue safety status of the systems and it can also waste significant amount of the certification cost.

Despite clear recommendations to adequately maintain and review the systems and their safety cases by safety standards, existing standards offer little or no advice on how such operations can be carried out [4]. Hence, there is an increasing need for globally acceptable methods and techniques to enable easier change accommodation in safety critical systems without incurring disproportionate cost compared to the size of the change. However, since broader re-verification and re-validation require more effort and time, it is important

for any proposal that aims at facilitating system changes to localise the impact of the changes. More specifically, to alleviate the cost of updating both a system and its safety case due to a change, it is crucial to minimise the effects of that change and prevent these effects from propagating into other parts of the system as far as it is practically possible.

In our previous work [5], we introduced a Sensitivity ANalysis for Enabling Safety Argument Maintenance (SANESAM) technique that supports system engineers to accommodate some types of potential changes. We also developed SANESAM+ [6] as a modified version of SANESAM that covers wider variety of changes. The key principle of SANESAM and SANESAM+ is to determine the flexibility (or robustness) of a system to changes using sensitivity analysis. The output is a ranked list of Fault Tree Analysis (FTA) events that system engineers can refine. The result after the refinement is a list of events that will be, most likely, related to the future changes. We use safety contracts to record the information of the maximum allowed changes to those events without violating the minimum acceptable safety limits. Those contracts can be used as part of later change impact analysis to advise the engineers what to consider and check when changes actually happen. The main contribution of this paper is to propose a new technique through which SANESAM is used to contain (i.e., localise) the potential changes in the smallest possible part of a system. More clearly, we compare the calculated MAFP (Maximum Allowed Failure Probability) of the events with new estimated FP of those events due to a change. If a new estimate FP of an event is $\leq$ MAFP, then the change will not, necessarily, require a considerable system modification, otherwise, it means that there will be a deficit in that FP and more effort should be considered. There could be several ways to respond to the latter case, but some responses might require large planning and massive re-engineering effort. Alternatively, we suggest, in this paper, to use the FP margins of other events to compensate the resultant deficit. The paper uses the aircraft Wheel Braking System (WBS) [7] to illustrate different examples of changes containment.

The rest of the paper is organised as follows: In Section 8.2, we present necessary background information. In Section 8.3, we describe two techniques to facilitate the maintenance of safety cases. We use this description as a basis to introduce a new technique to facilitate the maintenance of safety critical systems and safety cases in Section 8.4. In Section 8.5, we use the WBS system as an illustrative example. Finally, we conclude and propose potential future works in Section 8.6.

8.2 Background and Motivation

8.2.1 Safety Case

A safety case is defined as: "*A structured argument, supported by evidence, intended to justify that a system is acceptably safe for a specific application in a specific operating environment*" [8]. Hence, a safety case comprises both safety evidence (e.g. safety analyses, software inspections, or functional tests) and a safety argument explaining that evidence [9]. In order for safety cases to be developed, discussed, challenged, presented and reviewed amongst stakeholders, as well as maintained throughout the product lifecycle, it is necessary that (1) the argument to be clearly structured and (2) items of evidence to be clearly asserted to support the argument [10]. There are several ways to represent safety arguments (e.g., textual, tabular, graphical, etc.). In this paper, we use the Goal Structuring Notation (GSN) [10], which provides a 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 10.1 (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 can 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.

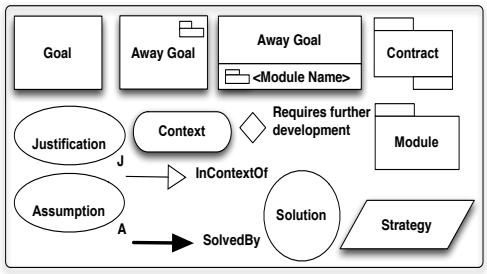

Figure 8.1: Notation Keys of the GSN

8.2.2 Fault Tree Analysis (FTA)

FTA is a failure analysis method which focuses on one particular undesired event and provides a method for determining causes of this event [7]. In other words, FTA uses abductive reasoning to identify different causes to critical states (from a safety or reliability standpoint). These states might be associated with component hardware failures, human errors, software errors, or any other pertinent events. FTA helps safety engineers to identify plausible causes (i.e., faults) of undesired events [11]. Moreover, FTA is used as a method to achieve Probabilistic Safety Analysis (PSA). More specifically, probability of failure is assigned to each of the failure events based on historical data, and the failure probability of the top event is determined [12].

8.2.3 Sensitivity Analysis

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"* [13]. The analysis helps to establish reasonably acceptable confidence in the model by studying the uncertainties that are often associated with variables in models. Many variables in system analysis or design models represent quantities that are very difficult, or even impossible to measure to a great deal of accuracy [14]. In practice, system developers are usually uncertain about variables in the different system models and they estimate those variables. Sensitivity analysis allows system developers to determine what level of accuracy is necessary for a parameter (variable) to make the model sufficiently useful and valid [15]. In this paper we use the sensitivity analysis to identify the safety argument parts (i.e., sensitive parts) that might require unneeded painstaking work to update with respect to the benefit of a given change. The results of the analysis should be presented in the safety argument so that it is always available up front to get developers' attention.

8.2.4 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 [16]. Hoare introduced a new notation to describe this connection, as follows:

$$P\ \{Q\}\ R$$

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"* [16].

In the context of contract-based design, 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 surrounding environment satisfies one or more related assumption(s).

A contract is said to be a *safety contract* if it guarantees a property that is traceable to a hazard. Contracts have been exploited as a means for helping to manage system changes in a system domain or in its corresponding safety case [17, 18, 19]. In this paper, we use safety contracts to record the dependencies among failure probabilities of FTA's events.

8.3 SANESAM and SANESAM+

In this section, we give an overview of SANESAM [5] and SANESAM+ [6]. SANESAM and SANEMSAM+ exploit 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. A sensitive event is an event whose failure probability value can significantly influence the validity of the FTA once it increases [5, 6].

The key principle of both techniques 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.). More clearly, 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:

$FP_{Current(Topevent)} > FP_{Required(Topevent)}$ [5].

SANESAM devotes $\Delta FP_{(Topevent)}$ for each event at a time, whereas

SANESAM+ distributes it over all of the events. The two techniques use sensitivity analysis to determine the range of failure probability parameter for each event. The steps of SANESAM phase are shown in Figure 8.2 and described as follows:

Step 1. Apply sensitivity analysis to a probabilistic FTA: In this step the sensitivity analysis is applied to a probabilistic FTA to identify the sensitive events whose minimum changes have the maximal effect on the $FP_{Topevent}$. However, applying this step is not identical for both SANESAM and SANESAM+. 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. On the other hand, SANESAM+ was introduced to provide more freedom by considering multiple events at a time. That is, if multiple events in FTA are expected to change, then SANESAM+ is the one to go. Choosing SANESAM means that the developers accept the assumption that only one event is allowed to change at a time. The difference between the process of SANESAM and SANESAM+ is observed in the way we apply Step 1. One more difference is that Step 2 should be completely neglected while applying SANESAM+ process, the rest of the steps are identical.

Applying Step 1 for SANESAM is done as follows [5]:

i) Find the Minimal Cut Set (MC) in the FTA [20].

ii) 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 $(FP_{Increased|event(x)} - FP_{Current|event(x)}) \not\Rightarrow$

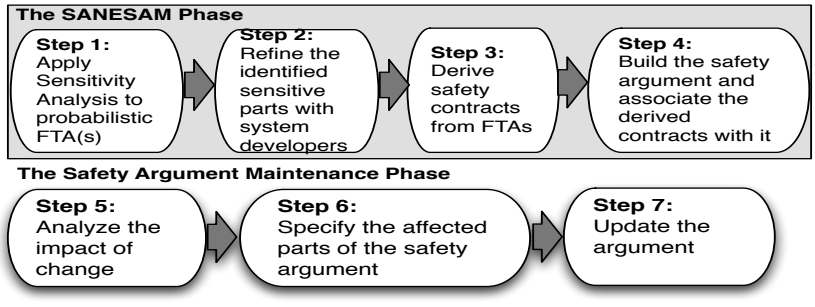

Figure 8.2: The process diagram of SANESAM [5]

$FP_{Increased(Topevent)} > FP_{Required(Topevent)}$.

iii) 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:

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

Applying Step 1 for SANESAM+ can be summarised as follows [6]:

i) Find $\Delta FP_{(Topevent)}$, where

$$\Delta FP_{(Topevent)} = FP_{Required} - FP_{Current}$$

ii) Distribute $\Delta FP_{(Topevent)}$ over all events in FTA. The distribution can be performed using different equations based on the logic gates in FTA. SANESAM+ steps from 1 to 4 in [6] describe those equations and give examples of how to perform the distribution.

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. This step shall not be applied for SANESAM+.

Step 3. Derive safety contracts from FTAs: At least one safety contract 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. In contrast, 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

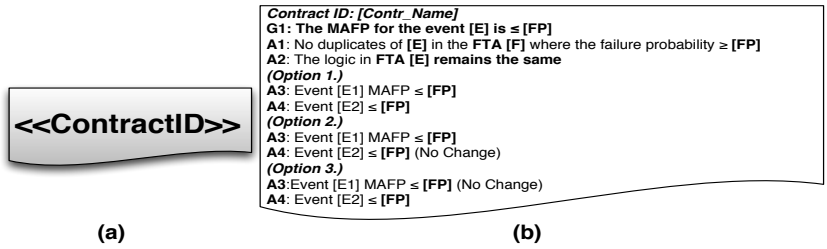

Figure 8.3: (a) FTA Safety contract notation, (b) Derived safety contract

enter new event failure probabilities. We introduce a new notation to FTAs to annotate the contracted events, where every created contract should have a unique identifier, as shown in Figure 8.3-a. We also create a template to document the derived safety contracts. Figure 8.3-b shows an instantiation of the contents of one of the derived safety contracts for WBS.

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. Further instructions of how to associate the derived safety contracts with a safety argument are described in [5].

8.4 Safety Contracts Driven Maintenance

The way we suggest to cope with some types of changes is to contain their effects in the smallest possible set of events to prevent (or minimise) the ripple of these effects from propagation. In this section, we describe a new technique that enables the containment of certain class of changes in safety critical systems and safety cases. It is worth noting that this technique utilises the same rules by which SANESAM and SANESAM+ calculate the sensitivities and associate them with a safety argument via safety contracts. However, the technique adds additional steps to enable effective usage of the safety margins in a probabilistic FTA. The new technique provides solutions to accommodate a change even if the change breaks one or more safety contracts. The only needed input for the process of the technique is a probabilistic FTA. The process comprises 6 steps that can fall into two main phases, before and after introducing a change. The three steps before performing a change are similar to the first phase of SANESAM and SANSAM+ as shown in Figure 8.2. However, we

have made some non-substantial changes to some of these steps, where we describe the change to each step when we describe the step itself. Steps 4-6 are novel and they were designed and specified for the new technique.

Steps before performing a change:

Step 1. Apply sensitivity analysis to a probabilistic FTA: This step is performed exactly as instructed in the process of either SANESAM or SANESAM+.

Step 2. Derive safety contracts: In this step, we need to derive safety contracts from FTAs as described in Section 8.3. However, there are two main differences in the derivation of safety contracts in this work. First, the guaranteed MAFPs in the safety contracts are basically the results of either multiplication or summation of multiple children events. Hence, there is no point to derive contracts for basic events in FTA because they simply do not have children events. The second main difference is that the contracts should provide multiple options for developers to measure the tolerance of a change's impact. More clearly, each derived safety contract should assume that only one child event is affected, multiple children events are affected or all of them are affected.

Step 3. Associate the derived contracts with safety arguments: Unlike the same step in SANESAM (in Section 8.3) and to enable more freedom, the proposed technique in this work considers that the construction of safety arguments is not necessarily a part of the process. Hence, we assume the existence of a safety argument no matter how it is represented (e.g., textual, tabular, graphical, etc.). The most important for us is the association itself because this association highlights the suspect elements in the argument to bring them to developers' attention. Typically, there is n-to-m mapping between the events of a FTA and different parts of a safety argument. Hence, a derived contract that guarantees a property, value, range, etc. should be associated with every part that is related to that guarantee in the argument.

Steps after performing a change:

Step 4. Check the ability of FTA to contain greater FP(s) than those already exist: The key principle of this step is to compare the new estimated FP of an affected event with the guaranteed MAFP in the safety contract of that event, or probably in the safety contracts of higher events.

As a quick check, we can determine the *MC* and calculate the expected FP of the top event taking into consideration the new increased FPs of the affected events and the current FPs (not the MAFP) of the unaffected ones. If the new calculated FP of the top event is $\geq$ MAFP, then the change is not containable and this means that there will be a deficit in the overall FP budget where the

response to the change might require re-engineering effort. Dealing with such a situation is beyond the scope of this paper. In contrast, if the new calculated FP of the top event is $\leq$ MAFP, this means that the change's impact is containable somewhere in the FTA, but we need to know which safety contract contains it. To do that, the new estimated FP of an impacted event should be checked against the guaranteed MAFP in the safety contract of that event, where it is containable iff it is $\leq$ MAFP. If the change's effect (i.e., difference between the new estimated FP and the MAFP) is not containable in the safety contract of the impacted event, then the safety contract of the ancestor event should be investigated as whether or not it can contain it. If the change's effect still cannot be contained by the ancestor, safety contracts in one more level up should be investigated and so on and so forth until a safety contract contains it. Once the contract which contains the change's effect is identified, all associated claims with this contract together with their supporting arguments and evidence should be highlighted as suspect.

Step 5. Re-balance the FPs of the FTA's events as a preparation for future changes. If any event has received a change that necessitates increasing its failure probability where the increment is still within the MAFP threshold in its safety contract, then it can be said that the safety contract in question still holds (intact) and the change is containable with no further significant maintenance. However, we need to re-balance the FPs of the FTA's events after accommodating a change to prepare for further accommodation(s) of future potential changes. Hence, we need to find $\Delta FP_{(Topevent)}$ which is the difference between the required FP and the new FP of the top event after containing a change. The current FPs of all unaffected events together with the new FPs of the affected events are used to calculate $FP_{New(Topevent)}$ based on the determined MC from

Table 8.1: Rating the impact of change

Impact Level	Highlight Colour	Description	Impact on Safety case	
			Argument	Evidence
Low		Change to an event is contained within its safety contract	No change necessary	Might reuse the same evidence
Medium		Change to an event is not contained within its safety contract, however is contained by another higher level safety contract with sufficient margin	No change necessary	Might need new evidence
High		The change is not contained within any of the derived safety contracts and the overall failure target of the system cannot be met	Major impact on the safety argument structure	Need new evidence

Step 4. The resultant $FP_{New(Topevent)}$ is subtracted from $FP_{Required(Topevent)}$.

The calculated $\Delta FP_{(Topevent)}$ might be equal to 0 or it can be an insignificant fraction which is not worth further effort. In this case, Step 1-ii should be omitted (i.e., no need for distribution) and we only need to update the safety contracts as described in the next step.

Step 6. Update the affected safety contracts with the new FPs: The contracts should be updated by the latest failure probability value(s) after containing a change. This step can be seen as Step 2, the difference is that we do not derive new contracts, but we rather update the ones we derived earlier.

In order to enhance the visibility of a change's impact on the system design and safety case, we highlight the parts of the system design and the elements of the safety case that are related to the affected events in FTA. Three levels of impact based on the impact propagation within FTA were defined, namely, *Low, Medium* and *High*. Table 8.1 categories the changes and suggest highlighting/representing them with different colours based on the rating of changes' impact.

8.5 Illustrative Example

We apply our proposed technique described in Section 8.4 to the Wheel Braking System (WBS) in which we assume three different change request scenarios and evaluate their impacts on the safety case. For the sake of both simplicity and space, we, in this section, summarise the key points of the application and provide its results. The detailed application of the technique is available in a separate technical report [21].

8.5.1 Wheel Braking System (WBS): System Description

The WBS is described in Appendix L of Aerospace Recommended Practice ARP-4761 [7] for safety assessment processes. 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: 1. Computer-based part which is called the Brake System Control Unit *(BSCU)*, 2. *Hydraulic* part, and 3. *Mechanical* part. The *BSCU* is internally redundant and consists of two channels, *BSCU System 1 and 2* (*BSCU* is the box in the grey background in Figure 8.4). 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 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 [22]. Figure 8.4 shows high-level view of the *BSCU* implementation. More details about the *BSCU* implementation can be found in ARP-4761 [7]. Figure 8.5 shows the "Loss of Braking Commands" probabilistic FTA (the original FTA is without the grey shapes) whilst Figure 8.6 shows GSN fragment of the WBS safety argument.

8.5.2 Safety Contracts Driven Maintenance: An Example

Since we have now all the required inputs for our technique (i.e., probabilistic FTA in addition to top event MAFP and current FP), we can start applying the *Steps before performing a change* (Steps 1-3 in Section 8.4), as follows:

Step.1: Apply sensitivity analysis: In this example we apply SANESAM+:

i) Find $\Delta FP_{(Topevent)}$. $\Delta FP_{(Topevent)}$ = **3.15E-05**.

ii) Distribute $\Delta FP_{(Topevent)}$ over all events. Figure 8.5 shows the distribution result for each event in the grey boxes.

Step 2. Derive safety contracts: After calculating the MAFPs for all of the

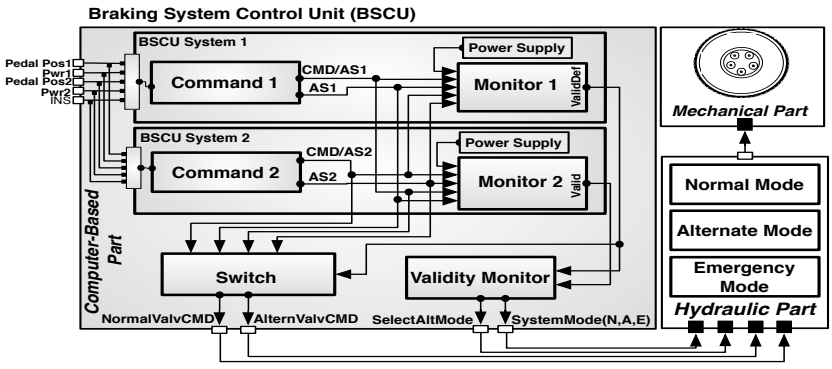

Figure 8.4: A high-level view of the WBS [5]

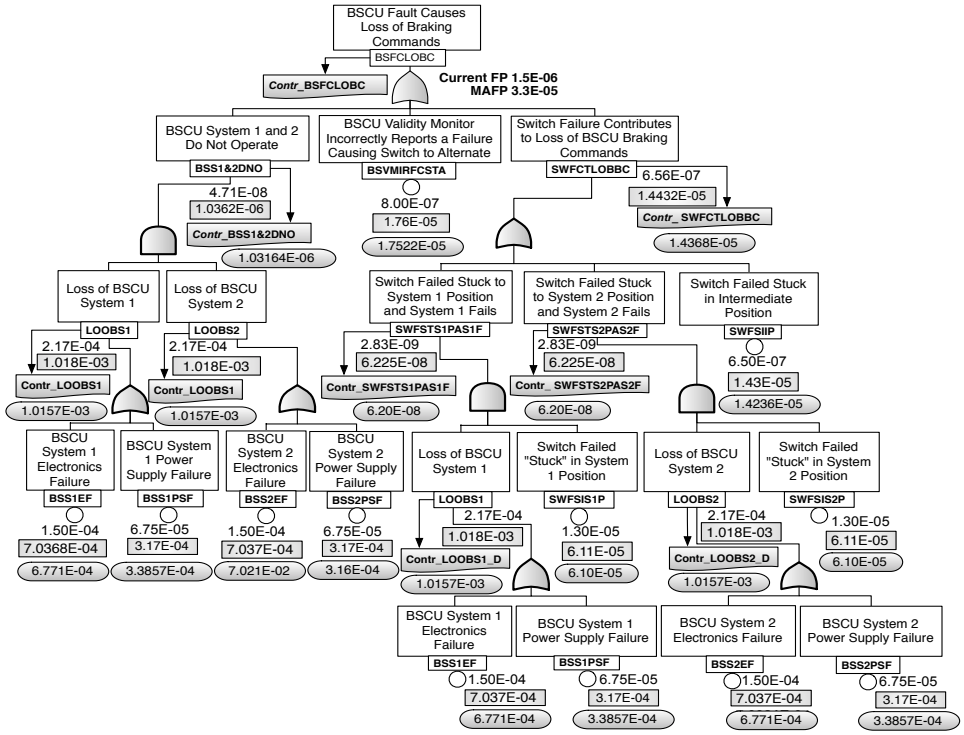

Figure 8.5: Loss of Braking Commands FTA [7]

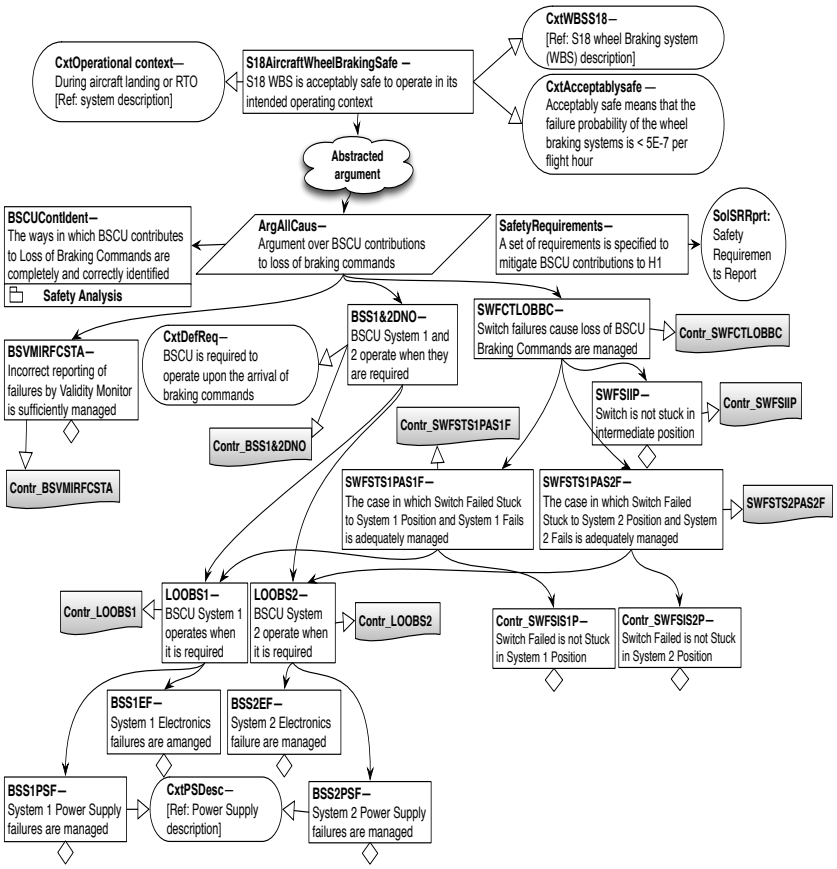

Figure 8.6: Safety argument fragment for WBS

events, a safety contract was derived for each non basic event. The template of the safety contract in Figure 8.3-a is used to represent the derived safety contracts. Also the contract notation in Figure 8.3-b is used to annotate the contracted events. Each contract considers multiple assumptions options based on the number of the children events. Figure 8.5 shows the derived contracts using the contract notations in grey. Figure 8.7 provides an internal view of the *Contr_LOOBS1* contract which is derived for the event *LOOBS1* as an example.

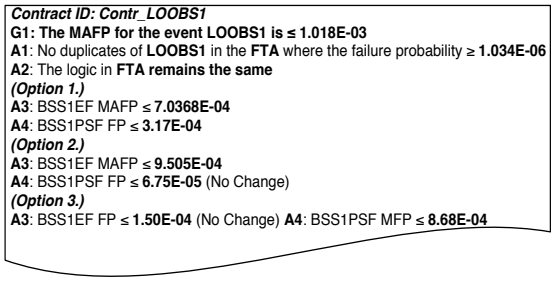

Figure 8.7: A derived safety contract

Step 3. Associate the derived contracts with the safety argument: In this example, we use a GSN argument fragment to show the association. Figure 8.6 shows how the derived safety contracts from FTA are associated with a safety argument fragment for WBS using the proposed contract notation in Figure 8.3-a. We do not want to affect the way GSN is being produced but we want to bring additional information for developers' attention. It is worth mentioning that a safety contract should be associated with all claims that are related to the event which the contract is derived for. For example, the safety contract *Contr_SWFSTS2PAS2F* should be associated with any articulated claims about the state when Switch Failed Stuck to System 2 Position and System 2 Fails.

Now, let us assume some change scenarios that can resemble real life change requests.

Change request scenario (1): The WBS developers have received a change request from the senior management asking to replace the current installed power supplies in BSCU 1 and 2 by a different model. Based on the provided product specifications by the new power supplies manufacturer, the FP of that model is *3.00E-04*, which means that it is less reliable than the FP of the current model in use (i.e., *6.75E-05*). Subsequently, step 4 should be followed to assess the impact of the given change scenario.

Step 4. Check the ability of FTA to contain greater FP(s) than those already exist: As a quick check, we want to update the FPs of the affected events based on the new given FPs and calculate the new FP of the top event. The new FP of the top event after the replacement is **1.646E-06** and since $1.646E\text{-}06 < 3.3E\text{-}05$, the increments to the FPs of *BSS1PSF* and *BSS2PSF* are tolerated (i.e., containable) in the FTA but the question is: ***Where can they be contained?***

To answer this question we need to specify the affected contracts by the change and check whether or not they still hold in the light of the new FP. The change request will affect four contracts, namely, *Contr_LOOBS1*, *Contr_LOOBS2*, *Contr_LOOBS1_D* and *Contr_LOOBS2_D*. Each derived contract contains different options in the assumptions list (as shown in Figure 8.7). We choose *(Option 1.)* in the four contracts and check if the MAFPs of *BSS1PSF* or *BSS2PSF* can contain the new FP. Since *3.16E-04* (MAFP) $>$ *3.00E-04* (new FP), the increments to *BSS1PSF* and *BSS2PSF* are contained in the four contracts and they still hold. This implies that replacing the power supply is rated as a GREEN change which means (according to Table 8.1) that there is no need to make any structural changes to the system design nor the safety argument. However, a manual check for the argument is still needed to replace the information of the old power supply with new valid information. For example, the description which the context *CxtPSDesc* refers to (in Figure 8.6) is out of date and should be replaced by the new power supply description.

Step 5. Re-balance the FPs of the FTA's events as a preparation for future changes: The reduction in the margins of the *BSS1PSF* and *BSS2PSF* FPs should be shared by all of the events in the FTA. That is, all current FPs should contribute to make up the contraction of *BSS1PSF* and *BSS2PSF* FP margins due to the power supply replacement, as follows:

1. Find $\Delta FP_{(Topevent)}$ which is the difference between the required FP (i.e., *3.30E-05*) and the new $FP_{Current(Topevent)}$ after containing the change which we have determined earlier (i.e., *1.646E-06*). $\Delta FP_{(Topevent)}$= *3.136E-05*.

2. Repeat Step 1-ii (i.e., the SANESAM+ approach which we have already mentioned under Step 1 in this Subsection) to distribute *3.136E-05* over all FPs' margins in the FTA. The grey squashed rectangles in Figure 8.5 represent the new MAFPs after the change.

Step 6. Update the affected safety contracts: Since new MAFPs have been calculated for all of the events, all derived contracts should be updated to reflect the new MAFP values.

Change request scenario (2): This scenario is similar to scenario (1). The only difference though is the FP value of the new power supply model, which is in this case equals to *5.00E-03* and thus it has less reliability than the current FP and even lesser than the one from the first scenario. As a quick check, the FP of the top event after introducing the change is ***2.8106E-05***, which means that it

is $< FP_{Required(Topevent)}$ and thus the change is tolerable. By applying the same steps we did in the previous scenario we will find out that *Contr_BSS1&2DNO* is the contract which contains the change.

Change request scenario (3): This scenario is similar to the previously discussed scenarios (2) and (3). The difference here is that the FP value of the new power supply model is *6.00E-03*, which means that it has less reliability than the current FP and it is the least reliable in this the three scenarios. The new calculated FP for the top event of this scenario is ***3.9432E-05*** and it is $>$ *3.3E-05* (the MAFP for the top event). That is, the resultant change effects due to replacing the power supply by this specific model is not containable and the entire FTA is going to be impacted. Hence, the WBS cannot meet its current safety requirements without considering major structural changes or updates.

Figure 8.8 shows a high level view of the change effects in the FTA that is caused by replacing the power supply in the three discussed change scenarios. The figure also shows how the safety contracts are used to highlight the affected parts in the WBS design and the safety argument. More detailed description on how the technique is applied to the three scenarios is available in [21].

8.6 Conclusion and future work

In our previous works [6, 5] we introduced SANESAM and SANESAM+ as techniques to facilitate the maintenance of safety cases using safety contracts. In this paper, we use the key principle of SANESAM and SANESAM+ to introduce a new technique that can save huge efforts in re-verification or re-certification due to some design changes. The technique can serve as a first impact analysis layer that helps system's developers to estimate the size of effort needed to accommodate a design change. The technique can also guide the developers to avoid massive re-engineering efforts when it is not really needed. Although the technique can be effective in maintaining safety systems and safety cases, the scope of the changes addressed by it may seem limited in the general maintenance scenario. However, these types of changes are the most critical from a safety perspective and they are worth making the emphasis. Future work will focus on considering different properties other than failure probabilities (e.g., timing) in order to consider additional types of changes. In addition, development of an automation tool is considered as a potential direction. We also intend to perform a case study to validate both the feasibility and efficacy of the technique.

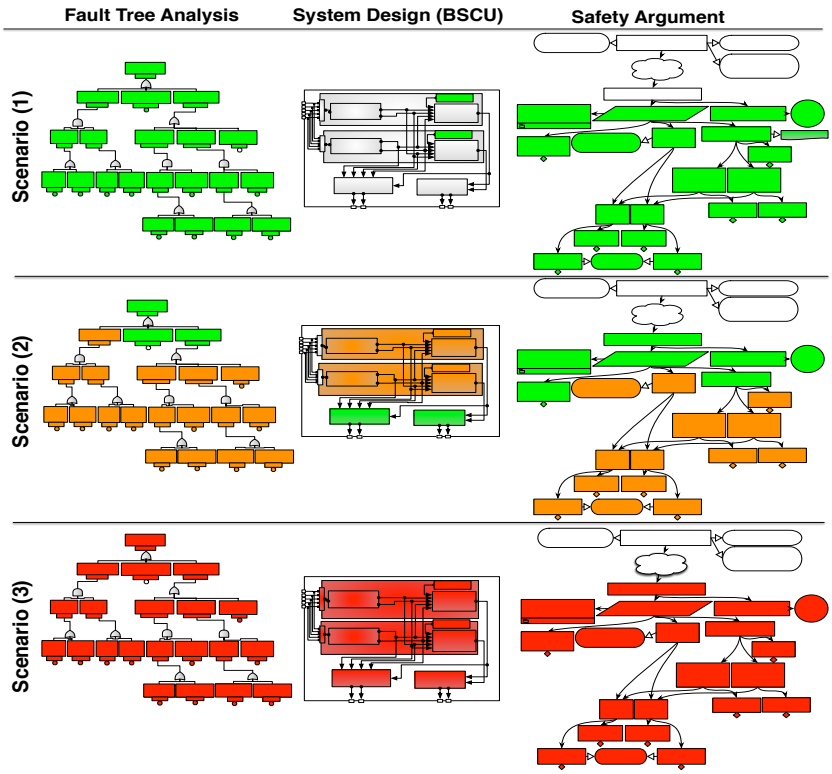

Figure 8.8: The effect of change on the FTA, system design and the safety argument: An overview of the three scenarios

Acknowledgment

This work has been partially supported by the Swedish Foundation for Strategic Research (SSF) (through SYNOPSIS and FiC Projects) and the EU-ECSEL (through SafeCOP project). We thank Sasikumar Punnekkat for his fruitful help and comments.

Bibliography

[1] R. Cleaveland. Formal certification of aerospace embedded software. In *National Workshop on Aviation Software Systems: Design for Certifiably Dependable Systems*, 2006.

[2] Neil R. Storey. *Safety Critical Computer Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1996.

[3] Iain Bate, Hans Hansson, and Sasikumar Punnekkat. Better, faster, cheaper, and safer too – is this really possible? In *Proceedings of the 17th IEEE International Conference on Emerging Technologies for Factory automation*, 2012.

[4] 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.

[5] 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.

[6] Omar Jaradat and Iain Bate. Deriving hierarchical safety contracts. In *The 21st IEEE Pacific Rim International Symposium on Dependable Computing (PRDC 2015)*, November 2015.

[7] SAE ARP4761 Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment, 1996.

[8] U.K. Ministry of Defence. *00-56 Defence Standard — Safety Management Requirements for Defence Systems*, December 1996.

[9] 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)*, UK, 2014.

[10] Goal Structuring Notation working group, November 2011.

[11] M. Stamatelatos, W. Vesely, J. Dugan, J. Fragola, J Minarick, and J. Railsback. Fault Tree Handbook with Aerospace Applications. Handbook, National Aeronautics and Space Administration, 2002.

[12] Prabhu Shankar, James Mathieson, Raveesj Ramachandran, Joshua D. Summers, and Gregory M. Mocko. Can design evaluation tools predict/prevent change propagation? In *Proceedings of Tools and Methods of Competitive Engineering*, 2012.

[13] A. Saltelli. *Global sensitivity analysis: the primer*. John Wiley, 2008.

[14] John Pate, Keith R. Edlin, and Kenneth I. Kawano. Sensitivity analysis of hardware-in-the-loop (HWIL) simulation systems. In *Proceedings of Modelling and Simulation*. ACTA, May 2005.

[15] L. Breierova and M. Choudhari. An introduction to sensitivity analysis. Technical report, Massachusetts Institute of Technology (MIT), 1996.

[16] C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, October 1969.

[17] 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.

[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] 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.

[20] M. Rausand and A. Høyland. *System Reliability Theory: Models, Statistical Methods and Applications*. Wiley-Interscience, NJ, 2004.

[21] O. Jaradat and I. Bate. Using safety contracts to guide the maintenance of systems and safety cases: An example. Technical report, Mäalardalen University, http://www.es.mdh.se/publications/4726-, April 2017.

[22] 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.

Chapter 9

Paper D: Using Safety Contracts to Verify Design Assumptions During Runtime

Omar Jaradat and Sasikumar Punnekkat.
In Proceedings of the 23rd International Conference on Reliable Software Technologies (Ada-Europe), Lisbon, Portugal, June 2018.

Abstract

A safety case comprises evidence and argument justifying how each item of evidence supports claims about safety assurance. Supporting claims by untrustworthy or inappropriate evidence can lead to a false assurance regarding the safe performance of a system. Having sufficient confidence in safety evidence is essential to avoid any unanticipated surprise during operational phase. Sometimes, however, it is impractical to wait for high quality evidence from a system's operational life, where developers have no choice but to rely on evidence with some uncertainty (e.g., using a generic failure rate measure from a handbook to support a claim about the reliability of a component). Runtime monitoring can reveal insightful information, which can help to verify whether the preliminary confidence was over- or underestimated. In this paper, we propose a technique which uses runtime monitoring in a novel way to detect the divergence between the failure rates (which were used in the safety analyses) and the observed failure rates in the operational life. The technique utilises safety contracts to provide prescriptive data for what should be monitored, and what parts of the safety argument should be revisited to maintain system safety when a divergence is detected. We demonstrate the technique in the context of Automated Guided Vehicles (AGVs).

9.1 Introduction

Safety critical systems are those systems whose failure could result in loss of life, significant property damage or damage to the environment [1]. Factories are often categorised as safety critical systems since failures of these systems, under certain conditions, can lead to severe consequences [2]. Assuring safety for such systems should provide justified confidence that all potential risks due to system failures are either eliminated or acceptably mitigated. Hence, all failures which might expose the manufacturing processes to hazards shall be analysed and controlled as part of pre-deployment safety assurance and monitored and controlled as part of operational phase.

Developers of some safety critical systems build a safety case to demonstrate the safety aspect of their system by identifying all unreasonable risks and describing, in the light of the available evidence, how these risks have been eliminated or adequately mitigated. Typically, a safety case comprises both safety evidence (e.g. safety analyses, software and hardware inspection reports, or functional test results) and a safety argument (i.e., reasoning) explaining that evidence. The safety argument shows which claims the developer uses each item of evidence to support and how those claims, in turn, support broader claims about system behaviour, hazards addressed, and, ultimately, acceptable safety [3].

An organisation building a safety case should be accountable for the ownership of the risks to be controlled by adopting an appropriate safety management system, performing a hazard assessment, selecting appropriate controls, and implementing them [4]. In order to help building a sufficient and credible (i.e., on a scientific basis) confidence in the safe performance of a system, its safety case shall always communicate the actual safe performance of the system, and shall always contain only acceptable items of evidence that this system meets its safety requirements. However, an item of evidence is valid only in the operational and environmental context in which it is obtained or to which it applies. More clearly, as the system evolves after deployment, there could be a mismatch between our communicated understanding of the system safety by the safety case and the safety performance of the system in actual operation, which might invalidate many of the prior assumptions made, undermine the collected items of evidence and thus defeat safety claims [5]. Despite the improvements in operational safety monitoring, there is insufficient clarity on how to utilise the analysis results of the monitored data on the documented confidence in safety cases.

In safety critical systems, failure rates are sometimes used as quantitative

criteria while performing safety assessment (i.e., Probabilistic Safety Assessment (PSA)). Failure Rate (FR=λ) is defined as the probability per unit time that a component experiences a failure at time "*t*", given that the component was operating at time "0" and has survived to time "*t*" [6]. Failure rates can be deemed as a reliability prediction that together with the consequences (*Risk = probability of failure * consequence of failure*) determine the Safety Integrity Level (SIL), which in turn specifies a target level of risk reduction that should be considered by a safety function or instrument. The quality of the failure rate measure determines the quality of the PSA. Hardware components are usually provided by generic failure rates which are derived by the statistical analyses of the failure frequency [7]. Failure frequency is usually obtained by the test results and the historical data of the components. Although the calculation of a generic failure rate is based on complex models which include factors using specific component data such as temperature, environment, and stress [6], it is, at its best, just a probability that is still subject to a percentage error even if it is used in the same context as in specifications. Assuming the perfection of the failure rate calculations is not judicious and can be misleading. Hence, a minimum level of fault tolerance in the architectural design of the safety functions should be considered. For example, the functional safety standards IEC 61508 [8] and IEC 61511 [9] recognise that there is always some degree of uncertainty in the assumptions made in calculation of failure rate and probability [10].

In this paper, we propose a novel technique to detect the discrepancies between the failure rates of system's components during their operational life and their generic failure rates used for analysis and assurance during the design time. Since it is infeasible to monitor the failure rates of all components of a system, the technique utilises probabilistic Fault Tree Analysis (FTA) to evaluate the criticality of the system components, and selects the most critical ones for monitoring. The technique derives safety contracts for the selected components and associate them with the relevant events in the FTA and the relevant parts in the safety case. If a discrepancy is detected between an observed failure rate (λ_O) and a generic failure rate (λ_G) of the same component, where $\lambda_O > \lambda_G$, then the relevant contract should be flagged and the referred parts of both the FTA and the safety case should be revisited.

Our hypothesis is that using safety contracts for monitoring the failure rates during the operational life of a system can help to provide essential feedback on the overall confidence in safety. More clearly, getting more precise measure of failure rates than the predicted ones will 1) improve the efficacy of the system design to reduce the risk (mitigate by design), 2) define stronger evidence

(e.g., refine or rectify the test results) and 3) highlight the required preventive, corrective, perfective or adaptive maintenance for safer operation

In this paper, we specifically make the following four contributions:

1. A novel technique to continuously reassess the failure rates and use the results to suggest system changes or maintenance
2. A new way to derive safety contracts to facilitate the traceability between the system design, safety analysis and the safety case
3. An example of how to argue more compelling over the failure rate in the light of the derived evidence from the operational phase
4. An example of how to carry out a through-life safety assurance

The rest of the paper is organised as follows: In Section 9.2, we present our approach to verify the design assumptions during runtime by safety contracts. In Section 9.3, we apply our technique to an AGV system to illustrate the main steps. In Section 9.4, we discuss how the suggested approach enables a through-life safety assurance. Finally, we conclude and describe the future directions in Section 9.5.

9.2 Using Safety Contracts to Verify Design Assumptions During Runtime

Failures of components in safety critical systems are typically divided into four modes, namely, Safe Detected (SD), Safe Undetected (SU), Dangerous Detected (DD), and Dangerous Undetected (DU). DD and DU failures can cause loss of a safety function while we believe that we are protected and this might happen in fraction of diagnostic interval in case of DD failures or during the unknown downtime in case of DU failures [11]. DU failures are typically due to either random or systematic failures. In this paper, we specifically focus on dangerous failures (DD and DU). Whenever FTAs are constructed to evaluate hazards, the basic event failure data must describe only failures that contribute to that hazard and thus only dangerous failure rates (λ_D) should be included for the basic events, where $\lambda_D = \lambda_{DD} + \lambda_{DU}$.

In this section, we propose a technique that aims to determine the λ_D of particular HW components in their operational life (observed $\lambda_D = \lambda_{D\_O}$) and compare the results with the design assumptions of these components (generic $\lambda_D = \lambda_{D\_G}$) to ultimately highlight any discrepancies between λ_{D_O} and λ_{D_G}. The

technique uses criticality importance measure to rank the components from the most to the less critical so that safety engineers can select particular components for monitoring when it is infeasible to monitor all of them. The technique also uses sensitivity analysis to determine whether a highlighted discrepancy is acceptable or not. The technique heavily depends on probabilistic FTAs, and it comprises 8 steps as follows:

9.2.1 Determine the PFD or the PFH in the FTA

In this step, we calculate the PFD (Probability of Failure on Demand) or the PFH (Probability of Failure per Hour) using a probabilistic FTA where each component is specified by its λ_{D_G}. The selection between PFD and PFH is based on the demand of a safety function. More clearly, if the safety function will be working in a continuous mode, then we have to select PFH [8]. However, if the safety function is expected to work once per year (at most), then PFD should be selected [8]. To calculate the PFD or PFH of an FTA, four sub-steps should be performed as follows:

A. Calculate the Failure Probability of the Basic Events:

There are different formulas used to calculate PFD depending on different factors, such as system's structure (*K*-out-of-*N* structures), Common Cause Factor (CCF), operational maintenance, safety standards obligations, etc. For example, Exida (a leading product certification and knowledge company) provides a realistic formula to calculate the PFD [12]. However, the difference between PFD formulas will not be influential in our technique. For the sake of simplicity, we adopt the PFD formula given in [13]. Formula 9.1 shows how we calculate the PFD for the basic events:

$$PFD(i) = \lambda_{D,i} * \tau \tag{9.1}$$

where i denotes the basic event and τ is the proof test interval. The component reparation or replacement time is assumed to be short and thus it is negligible.

The main difference between calculating PFD and PFH is in the logic of determining the probability of failures for the basic events. To calculate the PFH for the FTA's events, Formula 9.1 should be replaced with Formula 9.2, which is basically the famous unreliability exponential equation where only λ_D is considered. Unreliability in the context of functional safety is interpreted as the probability of a function to fail during a given time interval.

$$PFH(i) = 1 - e^{-\lambda_D t} \tag{9.2}$$

For calculating the PFD or PFH, we assume the failure rates of all components are constants, independent and have the same τ. We also assume that all potential CCFs are explicitly modelled as basic events in the FTAs. The rest of the sub-steps (B, C and D) are the same irrespective of we use PFD or PFH.

B. Determine Minimal Cut Set (MCS) in the FTA:

The MCS is defined as: *"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]. There are several algorithms to find the *MC*. We apply Mocus cut set algorithm [14].

C. Calculate the Failure Probability of the Determined MCS:

Calculating the probability of occurrence for the top event in a FTA with many MCS requires calculating the probability of those MCS. The failure probability of each determined MCS in the previous sub-step should be calculated according to formula 9.3 [11], as follows:

$$\breve{Q}_j(t) = \prod_{i \in C_j} q_i(t) \tag{9.3}$$

where $q_i(t)$ denotes the probability of basic event i at time t, $\breve{Q}_j(t)$ is the probability that minimal cut set j is in failed state at time t, $i \in C_j$ denotes the minimal cut set j that contains the basic event i.

D. Calculate the PFD or PFH of the Top Event:

We calculate the actual PFD or PFH by the *upper bound approximation formula* 9.4 [11] using the determined MCS, as follows:

$$PFD_{Act}(Top), PFH_{Act}(Top) = \sum_{j=1}^{k} \breve{Q}_j(t) \tag{9.4}$$

So far, all PFD or PFH calculations are based on λ_{D_G}. We refer to the result of the probability calculation based on λ_{D_G} as *Actual or Act*. The PFD_{Act}(Top) or PFH_{Act}(Top) are design assumptions which will be compared with the observed λ to check the correctness/validity of the design assumptions.

9.2.2 Identify the Most Critical Components

Monitoring every single component in safety critical systems is infeasible especially since such systems become bigger and more sophisticated over time. However, some components in a system are more critical for the system safety than other components. The objective of this step is to identify the most critical components in a system w.r.t the FTA. There are different measures through which FTA's events can be ranked based on their importance (e.g., Birnbaum, Criticality Importance, Fussel-Vesely Importance, Risk Achievement Worth (RAW)). In our technique, however, we are interested to rank the components based on their contributions to system safety. More specifically, we are interested in the components whose failures have the maximum impact on system safety. RAW is a measure that focuses on the 'worth' of the basic event in 'achieving' the present level of risk and indicates the importance of maintaining the current level of reliability for the basic event [14]. RAW is often used as an importance measure to rank components in terms of safety significance [15] and hence we will adopt it for our work .

The failure probability of the component i at time t may be described as:

$$P(i) = \begin{cases} 0 & \text{if the component is functioning at time } t \\ 1 & \text{if the component is in a failed state at time } t \end{cases}$$

The RAW, I^{RAW}(i|t) is the ratio of the (conditional) system unreliability if component i is $P(1)$, and it is calculated as follows [14]:

$$I^{RAW}(i|t) = \frac{1 - h(0_i, p(t))}{1 - h(p(t))} \text{ for } i = 1, 2, ..., n \tag{9.5}$$

where $h(0_i, p(t))$ is the probability of top event with component i = $P(1)$, and $h(p(t))$ is probability of top event. All basic events should be ranked from the most important to the less important. The most important event is the event for which Formula 9.5 has the maximum value.

9.2.3 Refine the Identified Critical Parts

The idea of this step is to discuss with system developers (e.g., safety engineers) and refine the ranked list of the critical components. This step is important, since it embeds the system level knowledge and experience of engineers regarding the uncertainty in a generic λ as well as helps as a validation step in the decision making process. For example, it could be the case that a high

ranked critical component in the list has a stable λ_G and systems engineers decide not to monitor it. That is, it is envisaged that some events may be removed from the list or the rank of some of them change. Moreover, the list can be extended to add any additional events by the developers.

9.2.4 Perform Sensitivity Analysis

The idea of this step is to determine the maximum allowable λ_D (λ_{D_Max}) of the system components which are selected for monitoring. More specifically, we need to define the upper- and lower bounds of the acceptable λ_D of each event in the MCS, where PFD_{Act}(Top) or PFH_{Act}(Top) is less than or equal to the required probabilities PFD_{Req}(Top) or PFH_{Req}(Top), respectively. The required probability is described as safety requirements by the safety standards (e.g., SIL, ASIL and DAL). It is important for our technique to determine to which extent PFD_{Act}(i) or PFH_{Act}(i) can be deviated while PFD_{Act}(Top) or PFH_{Act}(Top) still satisfies PFD_{Req}(Top) or PFH_{Req}(Top), respectively. To this end, two main activities should be performed, as follows:

Determine the maximum allowable $q_{i,Max(t)}$ for each component

The $q_{i,Max}$(t) for each component should be determined with respect to PFD_{Req}(Top) or PFH_{Req}(Top). Formula 9.6 should be used to determine $q_{i,Max}$(t) for each component at a time.

$$\frac{PFD_{Req}(Top), PFH_{Req}(Top) - (\sum \check{Q}_{i \notin C_j}(t))}{\sum \check{Q}_{i \in C_j}(t) \neg q_i(t)} = \frac{\sum \check{Q}_{i \in C_j}(t)}{\sum \check{Q}_{i \in C_j}(t) \neg q_i(t)} \tag{9.6}$$

where $i \notin C_j$ denotes the minimal cut set j that does not contain basic event i.

Determine λ_{D_Max} for Each Component

Once we have $q_{i,Max}$(t) for a component it is easy to determine its $\lambda_{D,Max}$. Formula 9.7 determines $\lambda_{D,Max}$ in case of PFD, as follows:

$$\lambda_{D,Max} = \frac{q_{i,Max}(t)}{\tau_i} \tag{9.7}$$

Formula 9.8 determines $\lambda_{D,Max}$ in case of PFH, as follows:

$$\lambda_{D\_Max} = \frac{-\ln(q_{i,Max}(t))}{\tau_i} \tag{9.8}$$

After calculating $\lambda_{D,Max}$ for all events, the latter should be ranked from the most sensitive to the less sensitive to change. The most sensitive event is the event for which Formula 9.9 is the minimum:

$$Sensitivity(\lambda_{D_i,G}) = \frac{\lambda_{D_i,Max} - \lambda_{D_i,G}}{\lambda_{D_i,G}} \tag{9.9}$$

9.2.5 Derive Safety Contracts

In this step, safety contracts should be derived from FTAs. The main objectives of deriving safety contracts are: 1) highlight the most important components to make them visible up front for developers attention [16], and 2) record the thresholds of $\lambda_D(i)$ to continuously compare them with the monitoring results (λ_{D_O}). Hence, if λ_{D_O} of component i exceeds the guaranteed $\lambda_{D\_Max}(i)$ in the contract of that component, then we can infer that the contract in question is broken and the related FTA should be re-assessed in the light of the λ_{D_O}. Another objective to derive safety contracts is to associate these contracts with safety arguments as reference points so that developers know the related part of the argument when they review a FTA and vice versa. To this end, we introduce two templates to derive contracts. The first contract template is for deriving a contract for the top event only. The *top event safety contract* is annotated with the abbreviation "**TE**" in the upper-right corner of the contract to denote that this contract is derived for a **T**op **E**vent as shown in Figure 9.1-A.

The second contract template is for deriving a safety contract for each event in the MCS (i.e., events related to important components). This type of contracts is referred to as "*monitoring safety contracts*" and it is is annotated with the abbreviation "**BE**" in the upper-right corner to denote that this contract is derived for a **B**asic **E**vent as shown in Figure 9.1-B.

9.2.6 Associate Safety Contracts with Safety Arguments

In this step, all safety contracts which were derived in Step 4 should be associated with safety arguments. This step assumes that the safety argument should come down to a claim that the "probability of failure of hazard H due to component failure is acceptable", in turn supported by a context element about what that probability is in the context of an applicable definition of acceptable, in turn supported by the FTA as evidence. An Assurance Claim Points (ACP) [17] should be created between the claim about the acceptable probability and the evidence, where a separate confidence argument should extend this

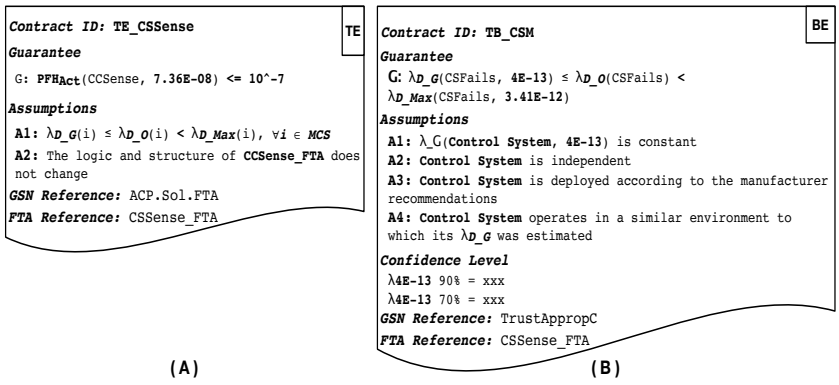

Figure 9.1: **A.** Contract template: Top Event. **B.** Contract template: Basic Event

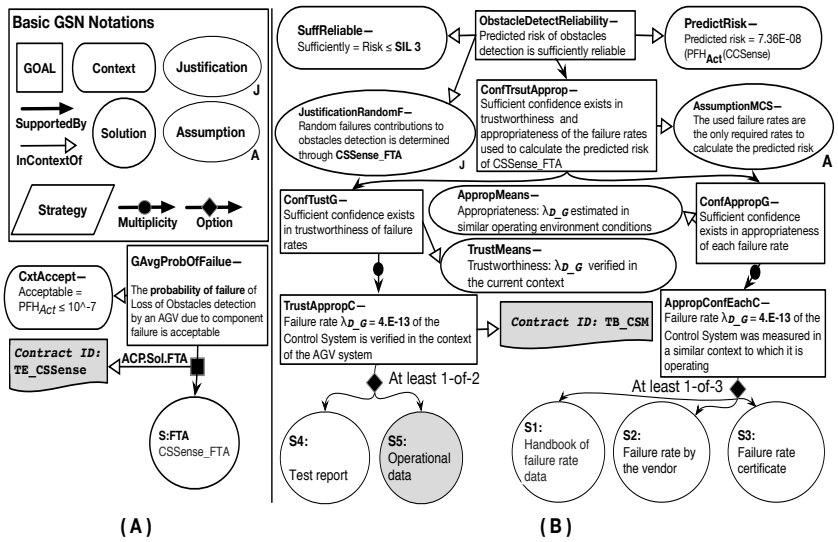

Figure 9.2: **A.** A probability of failure argument with an association of a top event safety contract. **B.** Confidence argument with an association of a monitoring safety contract

ACP to argue over the quality of the used failure rates to calculate PFD_{Act}(Top) or PFH_{Act}(Top).

It is necessary that the argument should be clearly structured and the items of evidence to be clearly asserted to support the argument [18]. There are several ways to represent safety arguments (e.g., textual, tabular, graphical, etc.). In this paper, we use the Goal Structuring Notation (GSN) [18], which provides a 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 basic notations of GSN are shown in Figure 9.2 (in the upper left side corner). A goal structure shows how goals are successively broken down into ('solved by') sub-goals until eventually supported by direct reference to evidence. GSN can 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.

Assertions in a safety argument relate to the sufficiency and appropriateness of the inferences declared in the argument, the context and assumptions used and the evidence cited [17]. For example, when an item of evidence is used to support a claim, it is asserted that this evidence is sufficient to support the claim. However, a simple 'SolvedBy' relation between the evidence and the claim will not satisfy a reviewer's concerns to reach a certain level of confidence, such as, 'why the reviewer should believe that the evidence is appropriate for the claim?' or 'whether it is trustworthy'.

Hawkins et al., [17] introduced "An assured safety argument" as a new structure for arguing safety in which the safety argument is accompanied by a confidence argument that documents the confidence in the structure and bases of the safety argument. Hawkins suggests that instead of decomposing the arguments further to argue over the appropriateness and trustworthiness of the supporting evidence, an ACP can be created to indicate an assertion in the safety argument. An ACP is indicated in GSN with a named black rectangle on the relevant link and a confidence argument should be developed for each ACP [17]. Three types of assertions were defined as ACPs as follow:

1. Asserted inference: the ACP for an asserted inference is the link between the parent claim and its strategy or sub-claims

2. Asserted context: the ACP for asserted context is the link to the contextual element

3. Asserted solution: the ACP for asserted solutions is the link to the solu-

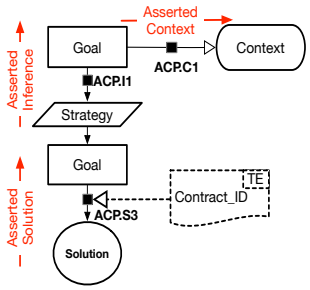

Figure 9.3: Types of ACPs with an example of each usage [17]

tion element

In this step, we suggest to use the principle of the ACP. Hence, the *top event safety contract* should be associated with the ACP (i.e., asserted solution) between the GSN goal which claims the acceptability of the hazard probability due to a component failure and the GSN solution which refers to the relevant FTA. Whereas, each *monitoring safety contract* should be associated with a GSN goal about the relevant component in the confidence argument. Figure 9.2-A shows a pattern of PFD or PFH argument and an example of top event safety contract association. Figure 9.2-B shows a confidence argument pattern with an association of a monitoring safety contract. Figure 9.3 instantiates an example of each ACP type and it also represents our suggested traceability means which associates the derived contracts from FTAs with safety arguments (the dotted part in the figure).

9.2.7 Determine λ_{D_O} Using the Data from Operation and Compare it to the Guaranteed λ_{D_Max} in Safety Contracts

In this step, λ_{D_O} of specified components should be obtained during the components' runtime. Using runtime monitors is one way to obtain data from operation. There are many proposed architectures to detect or test a system (or parts of it) for bad behaviour [19]. We provide a monitoring logic which requires two parameters (inputs) from any monitoring framework, namely, the number of recorded failures (i.e., DD and DU) as well as τ in time unit (e.g., hours). Algorithm 1 should be used to determine λ_{D_O} using the data from operation

Algorithm 1: The monitoring logic to determine λ_{D_O} and compare it to λ_{D_Max}

Data: MissionTime, τ, λ_{D_Max}, λ_{DU_O}, DUfailures = 0, λ_{DD_O}, DDfailures = 0, λ_{D_O}, Num_Comp, CL90, CL70;
Result: Determine λ_{D_O} and compare it to λ_{D_Max}

```
TotMonTime = clock();  \\Comment: start monitoring the mission time
while TotMonTime ≤ MissionTime do
    Test_Interval_Monitor = clock();  \\Comment: start the monitoring time of the test interval time
    while Test_Interval_Monitor ≤ τ do
        if a DD failure is found then
            DDfailures++;  \\Comment: add an observed failure from a diagnosis log file
        end
        if a DU failure is recorded then
            DUfailures++;  \\Comment: add an observed failure which was inserted manually
        end
        λ_DU_O = 1/((TotMonTime * Num_Comp) / DUfailures);  \\Comment: calculate λ_DU_O
        λ_DD_O = 1/((TotMonTime * Num_Comp) / DDfailures);  \\Comment: calculate λ_DD_O
        λ_D_O = λ_DU_O + λ_DD_O;  \\Comment: calculate λ_D_O
        CL70 = Chi-Squared(X^2_{70%,2(DUfailures+DUfailures+1)})/(2*Num_Comp*TotMonTime);  \\Comment: λ_D_O 70%
        CL90 = Chi-Squared(X^2_{90%,2(DUfailures+DUfailures+1)})/(2*Num_Comp*TotMonTime);  \\Comment: λ_D_O 90%
        if λ_D_O ≥ λ_D_Max then
            Contract [C] is broken;  \\Comment: highlight the broken contract whenever λ_D_O ≥ λ_D_Max
        end
    end
    Test_Interval_Monitor = 0; \\Comment: reset the τ timer to start a new one
end
```

and compare it to the guaranteed λ_{D_Max}. The more we monitor a component and record its failures the more confident we will be in its actual λ_D in a specific context. The calculated level of confidence can reveal how long we still need to monitor a component to reach a certain level of confidence. Hence, our algorithm also calculates the confidence level of λ_{D_O}(i)70% and λ_{D_O}(i)90% continuously and cumulatively using the *Chi-Squared distribution*. The calculated levels of confidence of a monitored component are automatically inserted into its "*monitoring safety contract*" and get updated continuously so that developers and assessors can review them in the FTA and the safety argument.

9.2.8 Update the Safety Contracts and Re-visit the Safety Argument

If a *monitoring safety contract* is broken it means that there is at least one broken *top event safety contract* as well. In this case, the broken safety contracts should be used to trace the FTA events and elements of safety arguments (for which the contracts were derived). As a result of doing this, developers can specify the entry point of the impact of failure in the safety analysis and the safety argument. It is worth mentioning that we assume the existence of a redundant component of the failing component. Hence, a broken safety contract does not necessarily lead to a total system failure.

9.3 Motivating Example: Automated Guided Vehicles (AGVs)

AGVs are being extensively used for more than 40 years now. They are used for intelligent transportation and distribution of materials in warehouses and auto-production lines. There are different setups and operational assumptions for each application of AGVs in industry. In our example, however, the AGVs are a number of battery-powered vehicles whose movements are autonomous. The AGVs are interfaced to automated warehouse and holding area, and to the machine tools, so that stock movement requirements can be fulfilled. The plant, in our example, is not fully automated so that people cannot be fully excluded from the areas where the AGVs work. Clearly, one of the most important safety features of the AGV vehicles is their ability to detect obstacles and stop quickly in order to avoid a collision with humans, hazardous objects (e.g., flammable materials, electrical resources, other AGVs, etc.). After performing safety analysis, a number of safety hazards were identified. In this

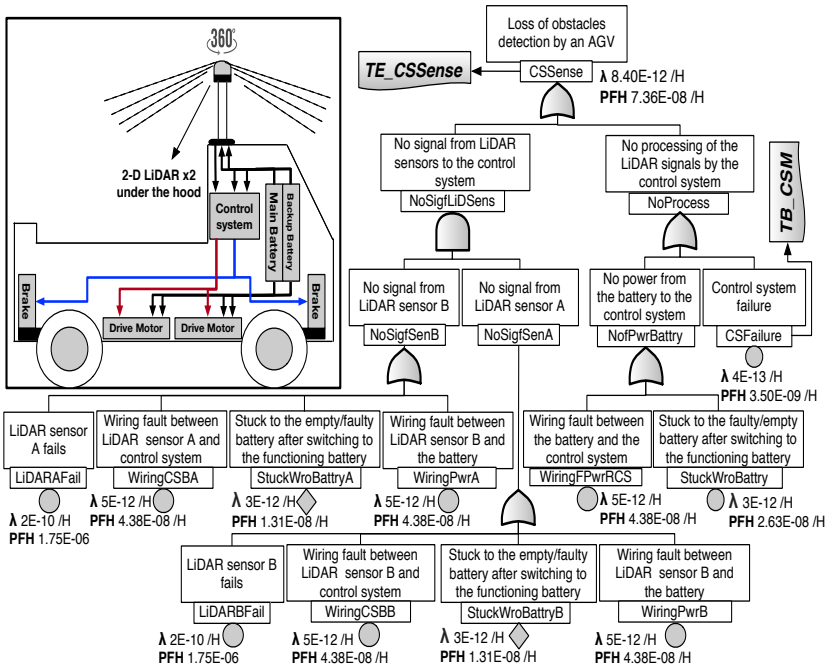

Figure 9.4: An overview of AGV's and its probabilistic FTA (CSSense_FTA)

paper, we will focus on one hazard, which is: *Loss of obstacle detection while the vehicle is in motion*. A redundant 2-D LiDAR sensor with all-round (360°) visibility is used for detecting obstacles within up to 30 meters range. Information about detected obstacles are sent to the control system to determine the manoeuvring strategy to ultimately avoid any potential collision.

According to the likelihood of occurrence, potential consequences and other safety countermeasures in the AGVs, the obstacle detection function is assigned **SIL 3** (Safety Integrity Level) according to IEC 61508. Moreover, since the function under discussion operates in a high demand (i.e., in a continuous mode), the allowable frequency of dangerous failure according to the same standard is PFH $< 10^{-7}$. The proof test interval τ is assumed as 1 year (i.e., 8760 hours) for all components. Figure 9.4 shows an overview of the AGV design (on upper left-hand corner). The figure also shows the FTA of

Table 9.1: A summary of the results of applying the steps 1-5

			STEP 1	STEP 2	STEP 3			STEP 4	STEP 5
No.	**Events**	λ_{D_G}	**PFH**	**RAW**	**Max PFH**	λ_{D_Max}	**Sensitivity**	**Refine**	**Contract**
1	CSSense (Top)	8.4E-12	7.36E-08		10^{-7}				TE_CSSense
2	CSFails	4E-13	3.50E-09	13589269.0946	2.99E-08	3.41E-12	7.5380	1	TB_CSM
3	WiringFPwrRCS	5E-12	4.38E-08	13589268.5470	7.02E-08	8.02E-12	0.6030		
4	StuckWroBattry	3E-12	2.63E-08	13589268.7851	5.27E-08	6.02E-12	1.0051	3	
5	LiDARAFail	2E-10	1.75E-06	26.3559	1.42E-02	1.63E-06	8137.5	2	
6	WiringCSBA	5E-12	4.38E-08	26.3559	1.42E-02	1.63E-06	325499		
7	StuckWroBattryA	3E-12	2.63E-08	26.3559	1.42E-02	1.63E-06	542499	3	
8	WiringPwrA	5E-12	4.38E-08	26.3559	1.42E-02	1.63E-06	325499		
9	LiDARBFail	2E-10	1.75E-06	26.3559	1.42E-02	1.63E-06	8137.5	2	
10	WiringCSBB	5E-12	4.38E-08	26.3559	1.42E-02	1.63E-06	325499		
11	StuckWroBattryB	3E-12	2.63E-08	26.3559	1.42E-02	1.63E-06	542499	3	
12	WiringPwrB	5E-12	4.38E-08	26.3559	1.42E-02	1.63E-06	325499		

the system where the top event together with the basic events are specified by λ_{D_G}.

Applying the first 5 steps in Section 9.2 is straightforward. Table 9.1 provides the results of the steps 1-5. The *Refine* column reflects the experts judgment that is supported by the RAW and Sensitivity ranking. For the sake of giving a clear example of what should be done next, we assume that *Control system* got the highest priority for monitoring (row No. 2 in Table 9.1). Hence, two contracts should be derived in the case: 1) TE contract *TB_CSM* and, 2) BE contract (i.e., monitoring contract) *TE_CSSense*.

Step 6 requires associating the derived contracts with the safety argument. For AGV system example, we use our suggested GSN patterns in Section 9.2.6 to create the confidence argument first and then associate the contracts with it through an ACP. Figure 9.2 presents our safety argument and the role of the proposed monitoring technique to provide supportive evidence for the articulated claims about the failure rates in the argument. Figure 9.1 shows the derived TE and BE for the top event *CSSense* and the basic event *CSFails*. The figure also shows the GSN and FTA references which reveal the associations (or traceability) of the contracts with the safety argument and the FTA, respectively.

9.4 A Through-life Safety Assurance Technique

Denney et al., [5] introduced the term "Dynamic Safety Cases (DSCs)" as a novel operationalisation of the concept of through-life safety assurance. The main motivation for introducing DSCs is that the appreciable degree of certainty about the expected runtime behaviour of a system might not be precise or it perhaps over- or underestimate the actual behaviour, which can create de-

ficiencies in the reasoning about the safety performance of that system. Hence, there is a need for a new class of safety assurance techniques that exploit the runtime related data (operational data) to continuously assess and evolve the safety reasoning to, ultimately, provide through-life safety assurance [5]. The suggested lifecycle of DSCs comprises four main activities as follows [5]:

1. **Identify** the sources of uncertainty in a safety case.
2. **Monitor** the runtime operation of the related system to collect data about system and environment variables, events, and assurance deficits in the safety argument(s).
3. **Analyse** the collected operational data from the former activity to examine whether the defined thresholds are met, and to update the confidence in the associated claims
4. **Respond** to operational events that affect safety assurance. Deciding on the appropriate response depends on a combination of factors including the impact of confidence in new data, the available response options already planned, the level of automation provided, and the urgency with which certain stakeholders have to be alerted.

In this section, we explain how using the described technique in Section 9.2 enables a through-life safety assurance, where we 1) identify a source of uncertainty, 2) provide a runtime monitoring mechanism, 3) analyse the collected operational data, and 4) suggest a response to the operational events.

1. Identify a source of uncertainty: Evidence supporting a claim about a prediction of a hardware failure rate may be obtained from different sources. Handbooks produced by commercial, military or government sources can support a claimed prediction of a hardware failure rate. A hardware vendor or an expert might also support such claims. The explicit logic of a claim about a failure rate prediction and its supported evidence is that the predicted likelihood of component C to fail during time T of operation is λ because a handbook, a vendor or an expert "says so". The implicit assumption of such claims is that the actual λ will conform to the predicted λ during the operational life. This assumption is an obvious source of *uncertainty* (i.e., lack of confidence) which can influence the level of confidence in the safety argument. Hence, it is particularly important to know whether or not the actual failure rate of a component during the operational life will be similar to the predicted (i.e., generic) rate as the evidence suggests.

2. Monitor the actual failure rate: Algorithm 1 provides the runtime monitoring logic through which the number of failures of a hardware component is continuously calculated during runtime.

3. Analyse the collected operational data: Algorithm 1 also analyses the calculated number of failures by comparing it with a predefined threshold.

4. Respond to operational events: If an observed λ exceeds the generic λ and it is not tolerated by the maximum allowed λ, then a safety contract is broken. The monitoring algorithm highlights broken contracts indicating that an additional safety countermeasure should be considered, such as replacing a hardware component with an ultra reliable component or add a redundant component. Since the contracts under monitoring by the algorithm is associated with ACPs in the safety argument, a broken contract indicates the affected GSN elements in the argument.

9.5 Discussion and Conclusion

Numerous studies and data analysis have shown either a decreasing or increasing failure rate with time. Runtime monitoring enables a new source of data which improves our perception of some functions, components, and behaviours within safety critical systems. Monitoring a property of interest of a system component and analysing the collected data enable us to know more about this component (e.g., the way it behaves, fails, etc.). As a result, we can improve our confidence in safety based upon more conscious reasoning that replaces the intuitive evidence by more cognitive one. Some safety standards require monitoring and re-assessing the reliability parameters which were used during the design time. For example, IEC 61511-1 [9] requires operators to monitor and assess whether reliability parameters of the Safety Instrumented Systems (SIS) are in accordance with those assumed during the design time [10]. Although runtime monitoring is not a new technique, there is no single way to specify what to monitor, why and how. Safety contracts, on the other hand, are useful for building, reusing or maintaining safety critical systems. The cost of maintaining system components can be drastically reduced by using contracts as system developers may rework the components with knowledge of the constraints placed upon them [20].

In this paper, we proposed a novel technique to monitor the runtime of a system and detect the divergence between the failure rates (which were used in

the safety analyses) and the observed failure rates in the operational life. The technique enables through-life safety assurance by utilising safety contracts to provide prescriptive data for what should be monitored, and what parts of the safety argument should be revisited to maintain system safety when a divergence is detected. Future work will focus on creating a more in-depth case study to validate both the feasibility and efficacy of the technique for software and hardware applications. We also plan to formally define safety contracts and to fully automate the application of the technique.

Acknowledgment

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

Bibliography

[1] J.C. Knight. Safety critical systems: Challenges and directions. In *Proceedings of the 24rd International Conference on Software Engineering (ICSE).*, pages 547–550, May 2002.

[2] Omar Jaradat, Irfan Sljivo, Ibrahim Habli, and Richard Hawkins. Challenges of safety assurance for industry 4.0. In *European Dependable Computing Conference (EDCC)*. IEEE Computer Society, September 2017.

[3] 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)*, UK, 2014.

[4] Patrick J. Graydon and C. Michael Holloway. An investigation of proposed techniques for quantifying confidence in assurance arguments. *Safety Science*, 92(Supplement C):53 – 65, 2017.

[5] E. Denney, G. Pai, and I. Habli. Dynamic safety cases for through-life safety assurance. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 2, pages 587–590, May 2015.

[6] Reliability prediction basics. Technical report, ITEM Software, Inc., 2007.

[7] Paolo Pittiglio, Paolo Bragatto, and Corrado Delle Site. Updated failure rates and risk management in process industries. *Energy Procedia*, 45(Supplement C):1364 – 1371, 2014. ATI 2013 - 68th Conference of the Italian Thermal Machines Engineering Association.

[8] *Functional safety of electrical/electronic/programmable electronic safety-related systems*. IEC 61508-4:2010.

[9] *Functional safety – Safety instrumented systems for the process industry sector*. IEC 61511-1:2016.

[10] M. Generowicz and A. Hertel. Reassessing failure rates. Technical report, I&E Systems Pty Ltd, 2017.

[11] Marvin Rausand. *Reliability of safety-critical systems: theory and applications*. John Wiley & Sons, 2014.

[12] Iwan van Beurden and William M. Goble. The Key Variables Needed for PFDavg Calculation. White paper, Exida, Sellersville, PA 18960, USA, July 2015.

[13] William M. Goble. *Control System Safety Evaluation and Reliability*. 2nd edition, 1998.

[14] Marvin Rausand and Arnljot Høyland. *System Reliability Theory: Models and Statistical Methods and Applications*. John Wiley & Sons, Inc., 2004.

[15] M van der Borst and H Schoonakker. An overview of PSA importance measures. *Reliability Engineering and System Safety*, 72(3):241 – 245, 2001.

[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] Richard Hawkins, Tim Kelly, John Knight, and Patrick Graydon. *A New Approach to creating Clear Safety Arguments*, pages 3–23. Springer London, London, 2011.

[18] GSN Community Standard Version 1. Technical report, Origin Consulting (York) Limited, November 2011.

[19] Aaron Kane. *Runtime Monitoring for Safety-Critical Embedded Systems*. PhD thesis, Carnegie Mellon University, September 2015.

[20] 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.

Chapter 10

Paper E: A Safety-Centric Change Management Framework by Tailoring Agile and V-Model Processes

Abdallah Salameh and Omar Jaradat. [1]
In Proceedings of the 36th International System Safety Conference (ISSC), Arizona, USA, August 2018.

[1] The author's names are listed in alphabetical order

Abstract

Safety critical systems are evolutionary and subject to preventive, perfective, corrective or adaptive changes during their lifecycle. Changes to any part of those systems can undermine the confidence in safety since changes can refute articulated claims about safety or challenge the supporting evidence on which this confidence relies. Changes to the software components are no exception. In order to maintain the confidence in the safety performance, developers must update their system and its safety case. Agile methodologies are known to embrace changes to software where agilists strive to manage changes, not to prevent them. In this paper, we introduce a novel framework in which we tailor a hybrid process of agile software development and the traditional V-model. The tailored process aims to facilitate the accommodation of non-structural changes to the software parts of safety critical systems. We illustrate our framework in the context of ISO 26262 safety standard.

10.1 Introduction

Many safety critical systems are subject to compulsory or advisory certification process which often necessitates building the systems in compliance with domain-specific safety standards [1]. Safety standards are becoming the main guide of the development and maintenance of hardware and software parts of safety critical systems. Safety standards, also, form the basis for the approval and certification of those systems [2]. Software systems, in general, are subject to different types of changes (e.g., preventive, perfective, corrective or adaptive) during the different stages in their life-cycle. In order to maintain the confidence in safety after accommodating a change, developers are required to update the safety case, which in turn requires identifying, re-analysing, and re-checking the impacted parts of the system and generate a new valid set of evidence [1]. Despite the obvious recommendations to adequately maintain and review the systems and their safety cases by different safety standards, the latter offer little or no advice on how such operations can be carried out [3]. There is an increasing need for globally-accepted methods and techniques to enable easier change accommodation in safety critical systems without incurring disproportionate cost compared to the size of the change. However, since broader re-verification and re-validation require more effort and time, it is important for any proposal aims to facilitate system changes to delimit the impact of changes.

Safety standards in many safety critical system domains adopt the traditional V-model as a development process for building the systems. Despite the effectiveness of validation and verification that the V-model provides, in addition to other advantages (e.g., easy to estimate costs, create timeliness, and stick to deadlines), the model has a well-known drawback when it comes to handling system changes. This is particularly true when it comes to changes to software systems and their requirements. Following the V-model implies that changes to software components requires re-visiting the system requirements and all later stages to perform a broad and costly impact analysis process. Hence, accepting software changes while using a V-model based process is not a trivial task.

Unlike the series of isolated phases in the V-model, agile methods depend on iterative and incremental development of software to enable reduction in cost, acceleration of time to market in addition to the focus of providing more maintainable code [4][5]. Software developers who follow agile methods breakdown their project into manageable fragments which enables a rapid responsive ways to handle software changes. The Agile way of working min-

imises the shortcomings of traditional sequential methods and improves the software development process in a more cost-efficient way [5]. The alignment of the development process with a dynamic environment is a critical motivation for adopting Agile Software Development (ASD) [6]. Test Driven Development (TDD) is an important agile process that brings many benefits such as reducing the potential consequences of software defects. TDD protects the system from future failures proactively, which leads to an acceleration of the maintenance process [7]. Thats is, TDD helps to build a continuous confidence in the implemented or modified code and whether or not it will behave as intended.

The work in this paper does not seek to conduct a comparative study between agile methods and the V-model. The main contribution of this paper, however, is to propose *XP-Kan-Safe* as a novel maintenance framework to facilitate the accommodation process of software non-structural changes in safety critical systems by utilising the strengths of agile methods and the V-model. More clearly, we reconcile the known effective validation & verification process of the V-model to the known effective practices and the TDD process of agile methods. We exploit the usage of safety contracts [8] as: 1) stitches that connect the V-model, Extreme Programming (XP) and Kanban into our tailored process, and 2) means to enable a tri-directional impact analysis process. The hypothesis we make is that ASD can resolve some observed maintenance challenges in the V-model while maintaining software parts of safety critical systems.

10.2 Background and Motivation

10.2.1 Safety cases and safety arguments

A safety case (also known as assurance or safety assurance case) is: *"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"* [9]. A safety case shall comprise both safety evidence (e.g. safety analyses, software inspections, or functional tests) and a safety argument explaining that evidence [10]. Safety cases might contain an implicit safety argument but some safety standards require an explicit argument that is usually expressed in terms of a defined hierarchy of safety claims and sub-claims that are supported by a body of evidence [9]. There are several ways to represent safety arguments (e.g., textual, tabular, graphical, etc.). In this pa-

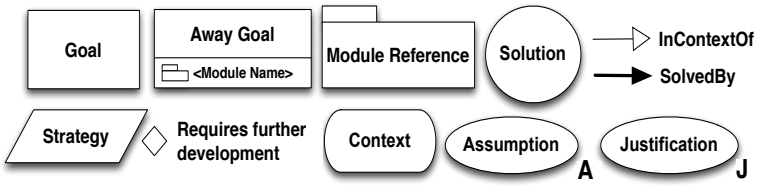

Figure 10.1: Notation Keys of the GSN

per, we use the Goal Structuring Notation (GSN) [11], which provides a graphical means of communicating (1) safety argument elements, claims (goals), argument logic (strategies), assumptions, context, evidence (solutions), and (2) the relationships between these elements [12]. Figure 10.1 shows the main notations of the GSN.

10.2.2 Maintenance of safety critical systems and their safety cases

Change requests should be assessed before decision makers decide whether or not to accept them. The assessment should reveal if the change can cause unreasonable risks, and the required cost to implement the change. Hence, system developers should understand the change and the potential risks that it might carry before they identify the impacted parts. 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. For example, the effect of a change can propagate to other parts of the system — creating a ripple effect — and cause unforeseen violations of the acceptable safety limits. If the impact of change is not clear, developers might be conservative and do wider analyses and verification (i.e., check more elements than strictly necessary), and this will exacerbate the cost problem of safety cases. It is also necessary for the developers to describe how the change affects the system parts in order to correctly estimate the cost of the response to that change. Otherwise, the response to a change might generate unplanned further changes to which the system must again respond, and this requires more cost than originally estimated.

10.2.3 ISO 26262 safety standard

ISO 26262 [13] regulates the automotive domain and it is intended to be applied to safety-related systems that include one or more electrical and/or electronic systems. The following parts are summarised descriptions of the safety requirements decomposition directly from ISO 26262 guidelines:

1. After identifying hazards, the standard recommends to formulate Safety Goals (**SGs**) to eliminate or mitigate hazards. The standard defines a safety goal as a top-level safety requirement resultant of the hazard analysis and risk assessment. Safety goals are not expressed in terms of technological solutions, but in terms of functional objectives.

2. Identification of SGs leads to the functional safety concept. The objective of the functional safety concept is to derive the Functional Safety Requirements (**FSRs**) from the SGs, and to allocate them to the preliminary architectural elements. At least one FSR shall be specified for each SG. Derivation of FSRs can be supported by safety analyses (e.g., Failure modes and effects analysis (FMEA), Fault Tree Analysis, Hazard and Operability Study (HAZOP)) in order to develop a complete set of effective functional safety requirements.

3. The functional concept leads to the technical safety concept. The first objective of the latter is to specify the Technical Safety Requirements (**TSRs**) and their allocation to system elements. The second objective is to verify that the technical safety requirements comply with the functional safety requirements. TSRs are used to derive Software Safety Requirements (**SSRs**).

10.2.4 Safety contracts

Contract-based design [14] is defined as an approach in which 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. Hence, contracts are intended to describe functional and behavioural properties for each design component in form of *assumptions* and *guarantees*. A contract is said to be a *safety contract* if it guarantees a property that is traceable to a hazard. Using contracts in development of safety critical systems is not a novel idea since there are many works utilise contracts for building, reusing or maintaining safety critical systems (e.g., [8][15][12]). The cost of maintaining, reusing and changing software components is lessened

while using contracts as developers may rework software components with knowledge of the constraints placed upon them [16]. In this paper, we use contracts to support the maintainability of safety critical systems. We also suggest to include additional information into safety contracts in order to enable effective traceability.

10.2.5 Agile Software Development (ASD)

Compared to traditional software engineering approaches, ASD targets complex systems and product development with dynamic, non-deterministic and non-linear characteristics. ASD methods (e.g., XP, Kanban, Scrum) evolve through collaboration between self-organising and cross-functional teams by sharing the same philosophy and utilising the appropriate practices for their contexts.

Each agile method has its own set of features (e.g., practices, terminologies, and tactics) and those features should reflect ASD values and principles. However, agile methods vary when it comes to the strategies they adopt to reflect those values and principles. For example, *Kanban* is known to have a rapid response to software requirement changes since it allows the team to instantly postpone some change requests to start with other emergent requests. *Scrum* might do the same but not after the completion of a sprint planning meeting and team commitment. XP teams are much more amenable to change within their iterations as long as a team has not started work on a particular feature that needs to be exchanged with the new feature. There is no standard recommendation as to how an agile method should implement its features [17].

Organisations, typically, adapt software development methodologies to be in line with their needs and contexts, while covering the full spectrum of the software development life-cycle [18][4]. In fact, there is no single agile method that can be adopted for any arbitrary context or to efficiently cover all phases in the development life-cycle. Hence, organisations might not adopt an entire agile method, but rather they combine different processes from different agile methods based on their needs and contexts.

10.2.6 Agile tailoring

The process in which an agile method is adapted for a specific project situation in a responsive way to accommodate the encountered challenges and to cover the indented interplay between contexts in a dynamic way, is called Agile Tailoring. There are two main approaches to tailor agile methods: the contingency

factors and the method engineering theory [17]. The first approach handles the tailoring by choosing multiple methods to be on *standby* in an organisation (i.e., Crystal family [19]). The selection of any standby method is based on project size and criticality, as well as the development context, such as uncertainty level, impact and structure. The second approach is based on meta-method processes and proposes the creation of a new method to be applied on specific contexts based on existing method fragments (a fragment represents a set of practices) [17]. Despite the flexibility of this approach, it introduces challenges such as how to control the fragments or how to assemble the method for a context specific situation by bringing the appropriate fragments and integrating them into one framework [17]. In this paper, we tailor our framework using the method engineering approach.

10.2.7 The Kanban method

Kanban is based on lean principles; it tries to remove the waste of the production process by embracing rules to limit Work In Progress (WIP) and measures the time to finish the tasks [17]. Kanban does not prescribe a specific set of roles or process steps, but rather it encourages its users to start from the existing context by understanding and emphasising the customers' needs [20]. Kanban is deemed as an approach to process change for organisations by providing sufficient visibility and understanding of the workflow and its progress. Kanban is all about visual signs (aka *Kanban Cards*) which represent individual work items accompanied with their critical information. Those cards move across a board (aka *Kanban board*). The latter is partitioned by vertical lanes which are titled, typically, according to the names of the development life-cycle phases (e.g., Analysis, Development, testing). These lanes can be partitioned further to specify the current state of each phase (To Do, Doing and Done). The location of a card on the board indicates the progress of the work and its current state. Kanban shows the assigned work for each team member, communicates priorities and highlights bottlenecks via cycle or lead time and the cumulative flow diagram [20][17].

10.2.8 The XP method

The XP method intends to improve software quality and responsiveness to the changing customer requirements. XP is considered a lightweight agile method that focuses on cost savings, unit tests before and along code activities, frequent full system integration and frequent releases [17]. XP comprises five

phases: exploration, planning, iterations to release, productionising, maintenance and death [4]. The exploration, planning and iterations to release are the only phases involved in our tailored framework.

During the exploration phase, the customers describe the features they wish to have in the first release of their system by writing each of them into a story card [21]. Our tailored framework is designed to deal with changes to a system that has been already built by the V-model. Hence, the features are considered as changes to the software system in our case. More clearly, safety engineers (who represent the customers) write change requests into story cards and discuss them with the team manager. During the planning phase, the story cards should be prioritised, an agreement on the first small release should be made and the time span required to implement the story cards should be estimated [21]. In the iteration and release planning phase, each release should be incremented by exactly one iteration. The development team should break down requested features (i.e., requested changes in our case) into several small releases. The customer selects the stories that should be implemented in a specific iteration. XP *Planning Game* is a close interaction between the customer and the development team. The latter should estimate the effort needed to implement the stories.

10.3 A maintenance framework to facilitate change management

In this section, we build upon Section 10.2 to propose a new framework which aims to streamline the change management process of non-structural software changes in safety critical systems. The framework is referred to as *XP-Kan-Safe* and it comprises two main processes: The Preliminary Process and The Change Management process.

Figure 10.2 provides a conceptual model of the framework. The conceptual model encompasses three phases: 1. Analysis phase to cover the derivation of safety contracts, 2. Planning phase to cover the game planning, and 3. Implementation phase to cover the TDD in addition to other XP practices. It is worth noting that the grey background of the model represents the Kanban board.

10.3.1 The Preliminary Process

This process is preparatory and should be performed before handling changes. The main objective of this process is to derive safety contracts and enrich them

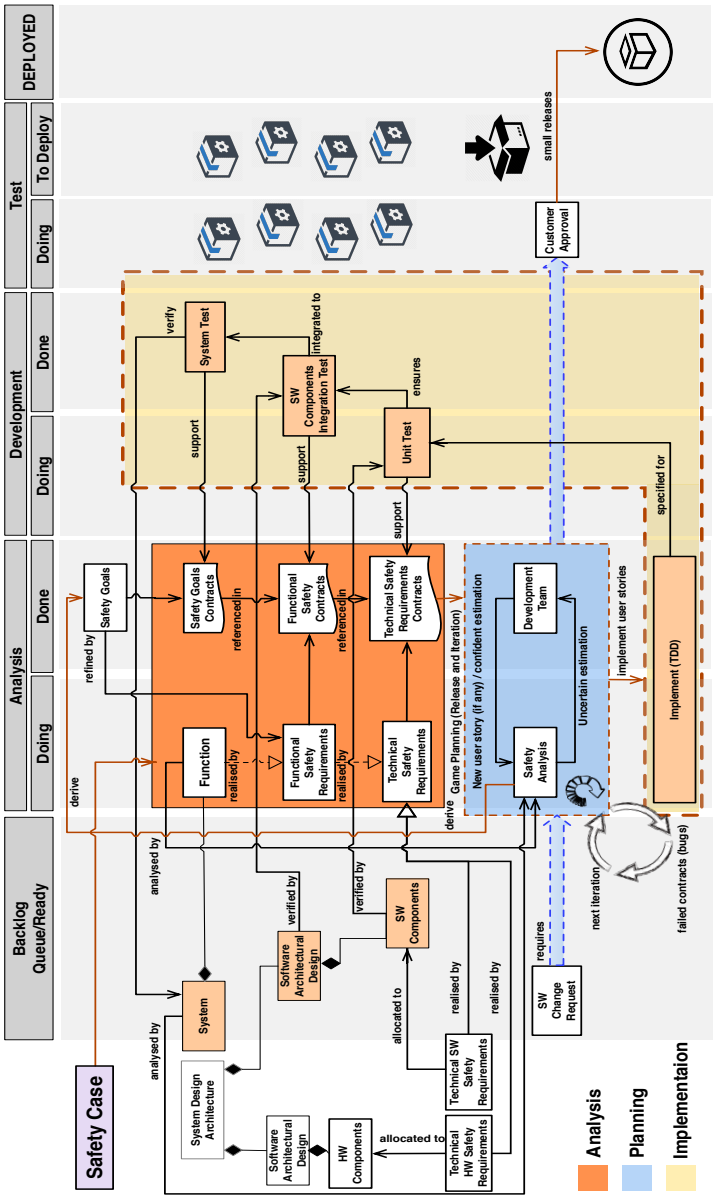

Figure 10.2: A conceptual model of *XP-Kan-Safe* framework

with additional information to increase the traceability between the requirements (i.e., guarantees) and different related artefacts. The activity of deriving safety contracts should start from the safety analysis phase. Safety analysis, however, is typically performed on different levels such as system, subsystems and components levels. The preliminary process enables system developers to derive contracts from safety analyses on the highest level down to lower levels. The preliminary process is applicable to any approach, and aims to decompose and specify safety requirements. The work in this paper, however, is designed to comply with ISO 26262 thus the derivation of safety contracts starts from the safety analysis through which SGs are derived.

After completing the safety analysis on the system level, safety contracts should be derived to guarantee the resultant SGs. A safety contract that guarantees a SG is referred to as "*SG contract*". The assumptions of a SG contract should capture the FSRs that fulfil the guaranteed SG. Furthermore, a contract should be derived to guarantee every assumed FSR in SG contracts after completing the safety analysis on the safety function level. A safety contract that guarantees a FSR is referred to as "*FSR contract*". The assumptions of a FSR contract should capture the TSRs that implement the guaranteed FSR. Finally, a contract should be derived to guarantee every assumed TSR in FSR contracts; such contracts are referred to as "*TSR contracts*". The assumptions of a TSR contract should capture the Software Safety Requirements (SSRs) that implement the guaranteed TSR after completing the safety analysis on the software components level.

Failure modes and effects analysis (FMEA) is recommended by many safety standards (including ISO 26262) as a safety analysis tool to identify potential failures modes. We enable the derivation of safety contracts from FMEAs by adding an extra column to the FMEA table so that safety analysts, together with requirement engineers, should cite their derived contracts in it. FMEA might have a deficiency when it comes to a multiple failures investigation. Hence, safety analysts might use different tools, such as Fault Tree Analysis (FTA) to search for the effects of multiple failures. Our preliminary process takes this into account and manages the derivation of safety contracts from FMEAs and FTAs. Figure 10.3 shows the connection between FTA and FMEA in addition to an example of a derived safety contract.

A guarantee in a contract and its related assumptions are the main elements of the contracts and they help to understand the relationships and the dependencies among the safety requirements. However, they might not be enough for analysts to identify the impacted artefacts and the elements in the GSN safety argument due to changes because they do not provide information as how the

different parts are related to each other. For instance, identifying an impacted TSR will not directly lead to the impacted test cases and the items of evidence which need to be replaced. In order to enhance the traceability between the requirements (i.e., guarantees) and other related artefacts as well as GSN elements, safety contracts should be enriched with additional information. To this end, system developers should include additional information into the derived contracts as follows:

1. Elements in the system architecture: all derived safety requirements should be allocated to elements of the system architecture. However, since the changes we are after in this work are non-structural, we assume that the changes have no effect on the system architecture.

2. Test cases: potential failure modes for which a safety requirement is derived should be considered as testing criteria during the verification phase to ensure the prevention of those failures. Including a reference to test cases in safety contracts enables direct traceability between safety analyses (i.e., FMEA and FTA), safety requirements (i.e., guarantees) and test cases. This traceability enables a top-down change impact analysis from the safety analysis down to the test cases. This top-down analysis represents the **first** direction of the tri-directional impact analysis process in our maintenance framework. While documenting the safety contracts, the reference of test cases might not be available as the test cases themselves might not be built yet. System developers are required to revisit each contract and add the corresponding test case references whenever they are made available. Furthermore, given that the test cases are available and complete, system developers can annotate them with the contracts' references. The annotations in the source code of the test cases are important to establish a traceability that enables a bottom-up impact analysis from the test cases up to the safety analysis. This bottom-up analysis represents the **second** direction of the tri-directional impact analysis process in our maintenance framework.

3. Elements of safety arguments: each safety contract should contain a reference to the related goals, contexts or items of evidence from safety arguments. Whenever GSN references are made available, system developers are required to revisit each contract and add the corresponding GSN reference to it. Including a reference to GSN elements in safety contracts enables direct traceability between a system and its safety case. This traceability enables a bi-directional impact analysis from the sys-

tem to its safety case and vice versa. More clearly, an affected guarantee can lead to an affected GSN element. Since the safety case presents

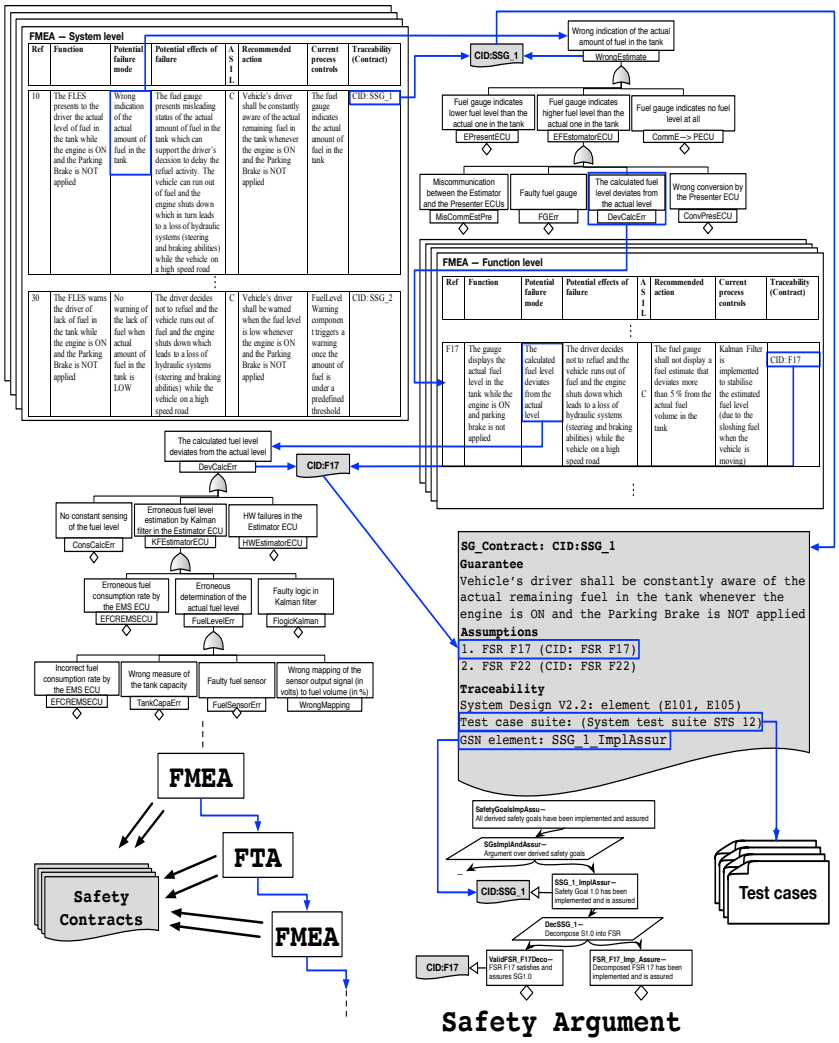

Figure 10.3: An illustration of a contract derivation by the Preliminary Process

the logic of how different artefacts are related, impact analysts might use it to highlight the change impact in the related system. The bi-directional change impact analysis represents the **third** direction of the tri-directional impact analysis process in our framework.

Figure 10.3 highlights the suggested traceability information and connects them to specimen artefacts and a GSN element.

10.3.2 The Change Management Process

In this section, we describe the second process of *XP-Kan-Safe*. This process and its activities represent the result of tailoring ASD and the V-model. The main objective of this tailored process is to guide whoever involved in the change management activities from the arrival of a change until the generation of a new test results report. Figure 10.4 presents the flow of these activities. The Change Management Process activities are described as follows:

Activity 1: Understand the change and its impact in the system and its safety case. Once a change request is placed, *Activity 1* should be followed in which the safety engineers should understand the nature of the change and determine its potential effects in the system and its safety case. In order to initiate the Kanban management process, safety engineers should create a *card* that describes the change request in more technical specifications and visualise it as a WIP in the analysis phase. The outcome of this activity should provide plausible data about the impacted parts of the system and its safety case.

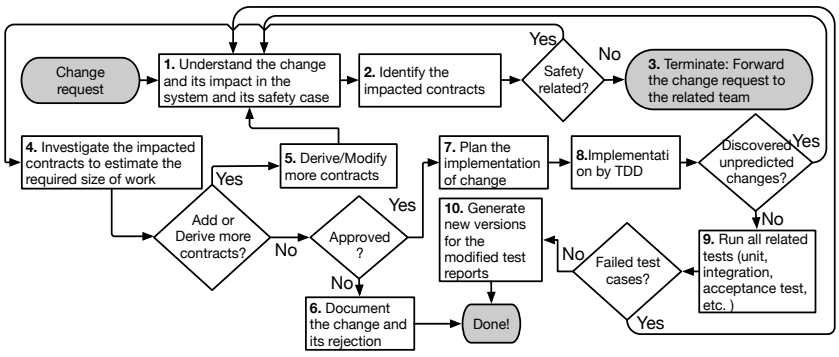

Figure 10.4: The change management process of *XP-Kan-Safe* framework

Activity 2: Identify the impacted contracts. In this activity, all related safety contracts to the change should be identified. The benefit of applying the first process of *XP-Kan-Safe* (i.e., preliminary) will be more realised in this activity since using safety contracts should help to provide a systematic impact analysis through the utilisation of the tri-directional impact analysis. The identified impacted contracts should be listed in the Kanban card.

Activity 3: Terminate: Forward the change request to the related team. If there is no safety contract identified as suspect in *Activity 2*, this implies that the change request has nothing to do with the functional safety in the system (no safety requirements are affected). In this specific case, the change request should be forwarded to the relevant team and no further continuation of the change management process is needed.

Activity 4: Investigate the impacted contracts to estimate the required size of work. There is no perfect impact analysis that can determine the effects of a change in the system and its artefacts at the first glance. That is, it is unlikely that the team will find out what might, precisely, get impacted merely by looking at the documented requirement and without iterating the impact analysis process. Hence, further investigation should be conducted to gain sufficient confidence in the perceived impact of a change. To this end, this activity should be followed to make further investigation of the impacted contracts. During this activity, a preliminary meeting should be carried out in which safety engineers, who represent the on-site customer with respect to XP, together with the development team, should determine the possibility of identifying more impacted contracts. Any additional identification of safety contracts should be added to the Kanban card. Safety contracts should support the collaboration between safety engineers and the development team to delimit the impacted parts of a system through the tri-directional impact analysis process. It is worth mentioning that any need to modify an existing contract or derive a new one will necessitate the application of this activity.

Activity 5: Derive new contracts or modify existing contracts. Since changes might introduce other changes, this might lead to modifying or deriving other contracts (i.e., requirements) that were not thought of earlier in *Activity 2*. In this activity, safety engineers and system developers derive new contracts or modify the existing ones to capture the newly introduced requirements or to update the already captured requirements, respectively. An initial cost of the change accommodation and its timeframe are two among several other factors upon which the approval decision is made. The involvement of the development team in the *Activities 4 and 5* should cover the estimation of the initial amount of work and the time needed to complete it. Safety engineers

and system developers should agree on: 1) what should be changed or added (i.e., size of the work) and 2) the acceptance of the accompanied potential risk on safety functions. Subsequently, they should submit their agreement to the management where the latter can either decline or accept the change request. Submission of the agreement concludes the Analysis Phase and this means that the Kanban card should move on the board from (Analysis $\rightarrow$ Doing) to (Analysis $\rightarrow$ Done).

Activity 6: Document the change and its rejection. If the change request receives a rejection by the management, the change request, the performed investigation and the management decision should be documented [13]. The rejection implies that the Kanban card should be closed.

Activity 7: Plan the implementation of change. If the change request receives an approval by the management, the Kanban card should be available for development. The adopted planning method, in our change management process, complies with XP. This implies that the implementation of the change request is initiated by the planning game. The input of the planning game is the estimated work and the impacted safety contracts. The output are more fine-grained estimated tasks than the earlier estimated tasks in *Activity 4*.

Activity 8: Implementation by TDD. In this activity, the implementation of the change is carried out using TDD. For those contracts that are subject to modification, system developers should find the related test cases (using the parameters that refer to them in the contracts) and modify them accordingly. Since modifying a contract might require creating new test cases, system developers should cite the newly added test cases in the corresponding contracts and vice versa. This is particularly important to support bi-directional traceability between the test cases and the contracts while is deemed as a preparation for future changes. Citing the newly added test cases in the contracts applies to the derived contracts during the impact analysis process — after the preliminary process — as well *Activity 5*. Moreover, after implementing required production code to satisfy the derived test cases, other already existing test cases might get impacted by newly added code. If the solution is to modify or add new requirements, system developers should inform the safety engineers about the suggested changes to the requirements. In this case, the suggested changes by system developers should be declared as *unexpected changes*. Afterwards, safety engineers and system developers should arrange an on-the-fly meeting to investigate the discovered unexpected changes *Activity 4*. The meeting should reveal 1) whether or not the suggested changes might introduce unreasonable risks (i.e., criticality level) and 2) the size of work required to cope with the suggested changes. The size of work is defined, in this context, based

on its influence on the earlier gaming plan *Activity 7* so that big work means a modification of the release planning is required. If the suggested changes are non-critical, system developers should implement them or forward them to the relevant team. If the suggested changes are critical, one of two possible actions should be performed:

1. If the size of work is small, developers should do the fixes on-the-fly and cite the related test cases in the contracts and vice versa.

2. If the size of work is big and critical, developers should either follow the exchange strategy by XP to re-prioritise the tasks within the current iteration of the planned release or plan the tasks for the next release.

Activity 9: Run all related tests. In this activity, system developers should utilise the continuous integration as a first step, according to XP, to avoid delays caused by integration problems. Subsequently, a continuous testing process should be initiated to obtain immediate feedback on the possibility of violating safety countermeasures to prevent unreasonable risks associated with a software release. The scope of testing should be extended from a bottom-up assessment (from test cases to safety requirements) to validate safety goals. In case of any violation of safety requirements after running the continuous testing, system developers should follow *Activity 8*.

Activity 10: Generate new versions for the modified test reports. This activity should be followed once the continuous testing is completed successfully. New reports of the test results should be generated to replace the out-of-date reports in the safety case. It is significant to update the references of these reports in the safety contracts of both the system and its safety case.

10.4 Discussion and conclusion

Maintaining safety critical systems due to changes is a challenging process because of: *1)* the lack of awareness of the change's effects and the ripple of these effects on the system, *2)* the lack of documentation of dependencies among the generated artefacts during the development process, and *3)* the lack of traceability between a system and its safety case. Following the V-model to accommodate system changes might be very strict, which might be justifiable for structural system changes since many parts get impacted and there is no precise clue about the size of work needed to maintain the system. For software non-structural changes, this might not be justifiable. ASD can provide

promising methods to maintain software changes. For example, XP puts great emphasis on the technical aspects (e.g., TDD, continuous integration and code refactoring). Also, Kanban brings the visibility of the workflow and improves the communication and collaboration among the stakeholders. Using ASD for maintaining safety critical systems can be promising but it still needs to comply with the current safety standards. In this paper, we introduced *XP-Kan-Safe* as a novel framework in which we tailor a hybrid process of ASD and the traditional V-model. The tailored process exploits safety contracts to connect ASD and the V-model, and enable a tri-directional impact analysis process. Future work will focus on creating a more in-depth case study to validate both the feasibility and efficacy of the process as well as to fully automate its application.

Acknowledgment

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

Biographies

- Omar T. Jaradat, Ph.D. candidate, School of Innovation, Design and Engineering, Mälardalen University, Högskoleplan 1, SE-72123, Västerås, Sweden, Tel: +46 21101369, Fax: +46 21101460 e-mail: omar.jaradat@mdh.se.
  Omar Jaradat is a Ph.D. candidate in the Innovation, Design and Engineering department at Mälardalen University. His research interests include safety argumentation for safety critical systems, where the main focus is on maintenance of safety-critical systems and safety cases.

- Abdallah M. Salameh, Ph.D. candidate, School of Computing, Science and Engineering, University of Salford, Manchester, UK, Tel: +46 721844015, e-mail: a.salameh@edu.salford.ac.uk.
  Abdallah Salameh is a senior developer at Bambora Group AB - Sweden and a Ph.D. candidate in the School of Computing, Science and Engineering at the University of Salford, U.K. His research interests include agile software development, where the main focus is on tailoring the processes in large-scale software intensive organisations.

Bibliography

[1] Omar Jaradat and Iain Bate. Using safety contracts to guide the maintenance of systems and safety cases. In *Proceedings of the 13rd European Dependable Computing Conference (EDCC)*, pages 95–102, Sept 2017.

[2] E. Denney, G. Pai, and I. Habli. Dynamic safety cases for through-life safety assurance. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 2, pages 587–590, May 2015.

[3] 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.

[4] Abdallah Salameh. On Process Tailoring - An Agile Example. Master's thesis, Chalmers University, 2011.

[5] S. Tarwani and A. Chug. Agile methodologies in software maintenance: A systematic review. volume 40, pages 415–426. Slovene Society Informatika, 2016.

[6] Lan Cao, Balasubramaniam Ramesh, and Tarek Abdel-Hamid. Modeling dynamics in agile software development. *ACM Trans. Manage. Inf. Syst.*, 1(1):5:1–5:26, December 2010.

[7] Daniël Knippers. Agile software development and maintainability. In *15th Twente Student Conf*, 2011.

[8] Iain Bate, Richard Hawkins, and John McDermid. A contract-based approach to designing safe systems. In *Proceedings of the 8th Australian Workshop on Safety Critical Systems and Software - Volume 33*, SCS '03,

pages 25–36, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc.

[9] U.K. Ministry of Defence. *00-56 Defence Standard — Issue 6, 2015. Safety Management Requirements for Defence Systems — Part 1: Requirements and Guidance. (UK) Ministry of Defence*, June 2015.

[10] 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)*, UK, 2014.

[11] Goal Structuring Notation working group, November 2011.

[12] 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.

[13] ISO 26262:2011. Road Vehicles — Functional Safety, Part 1-9. International Organization for Standardization, Nov 2011.

[14] 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.

[15] O. Jaradat, I. Bate and S. Punnekkat. Facilitating the maintenance of safety cases. In *Proceedings of the 3rd International Conference on Reliability, Safety and Hazard - Advances in Reliability, Maintenance and Safety (ICRESH-ARMS)*, Luleå, Sweden, June 2015.

[16] 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.

[17] Amadeu Silveira Campanelli and Fernando Silva Parreiras. Agile methods tailoring – a systematic literature review. *The Journal of Systems & Software*, 110:85–100, December 2015.

[18] Lise Heeager and Jeremy Rose. Optimising agile development practices for the maintenance operation: nine heuristics. *Empirical Software Engineering*, 20(6):1762–1784, December 2015.

[19] Pekka Abrahamsson, Juhani Warsta, Mikko T. Siponen, and Jussi Ronkainen. New directions on agile methods: A comparative analysis. In *Proceedings of the 25th International Conference on Software Engineering*, ICSE '03, pages 244–254, Washington, DC, USA, 2003. IEEE Computer Society.

[20] Muhammad Ovais Ahmad, Jouni Markkula, and Markku Oivo. Kanban in software development: A systematic literature review. pages 9–16. IEEE, September 2013.

[21] Pekka Abrahamsson, Outi Salo, Jussi Ronkainen, and Juhani Warsta. Agile software development methods: Review and analysis. *CoRR*, abs/1709.08439, 2017.