NTNU
Det skapende universitet

Goal and evidence based dependability assessment

Eline Marie Bye Melkild

Medical Technology
Innlevert: juni 2013
Hovedveileder: Torbjørn Skramstad, IDI

Norges teknisk-naturvitenskapelige universitet
Institutt for datateknikk og informasjonsvitenskap

Problem description

The development of Safety Critical Systems is guided by standards. Many standards require the development of a safety case to demonstrate the acceptability of Safety Critical Systems. The safety case must provide confidence that the system is deemed safe enough to operate. For software where it is not possible to quantify the associated risks, current standards in the aerospace, rail and defence sectors identify design and safety processes for different Safety Integrity Levels (SILs) or similar. Software is shown to be fit for use primarily by appeal to the standards, supported with appropriate evidence. The assumption is that software developed against the process requirements of higher SILs will be less prone to critical failures and thus have a lower impact on the overall system safety.

It has been proposed that an evidence-based approach should be taken to software. An evidence-based or "goal-based" approach can be implemented by using a framework for articulating software safety arguments, based on categorisation of evidence, which is largely independent of the development process.

The following tasks are included in the project:

Task 1: A survey of different tools to build safety cases.

Task 2: Based on one of the tools: Propose a way to structure the safety case in order to give sufficient evidence of safety.

Task 3: Discuss the experience of using the tools (could include semi-structured interviews with experts in industry about their use of safety case)

Oppgavebeskrivelse

Utvikling av sikkerhetskritiske systemer er styrt av standarder. Mange standarder krever utvikling av en safety case for aa demonstrere aksept av sikkerhetskritisk utstyr. En safety case maa gi visshet om at systemet er trygt nok til aa operere. For programvare der det ikke er mulig aa tallfeste den tilknyttede risiko, gjeldende standarder innen luftfart, jernbane og forsvarssektorer, identifiseres design- og sikkerhetsprosesser for ulike Safety Integrity Levels (Sil's) eller lignende. Programvare blir sagt aa vaere egnet for bruk hovedsaklig ved aa appellere til standardene, og stoettet med passende bevis. Forutsetningen er at programvare som er utviklet etter prosesskravene om hoeyere Sil's, vil vaere mindre utsatt for kritiske feil, og dermed har en lavere innvirkning paa det totale systemets sikkerhet.

Det har blitt foreslaatt at en "bevisbasert" tilnaerming boer brukes paa programvare. En "bevisbasert" eller "maalbasert" tilnaerming kan implementeres ved hjelp av et rammeverk for aa uttrykke programvaresikkerhets-argumenter, basert paa kategorisering av bevis, som i stor grad er uavhengig av utviklingsprosessen.

Foelgende oppgaver er inkludert i prosjektet:.

Oppgave 1: En undersoekelse av ulike verktoey for aa bygge safety case'er.

Oppgave 2: Basert paa et av verktoeyene: Foreslaa en maate aa strukturere en safety case for aa gi tilstrekkelig bevis om sikkerhet.

Oppgave 3: Diskuter erfaringen ved aa bruke de verktoeyene (kan inkludere semi-strukturerte intervjuer med eksperter i bransjen om deres bruk av safety case'er).

Preface

This report was written as my master's thesis at the Department of Computer and Information Science (IDI) at the Norwegian University of Science and Technology (NTNU), and has been a continuation of my specialization project in the fall 2012.

I would like to thank my supervisor Torbjorn Skramstad for his support and guidance through the work on this thesis and the project during the fall semester. I also want to thank my friends and colleages for motivating me during the process, and making the work place a positive place to be. And lastly I would like to thank my family for believing in me, and getting me where I am today.

Abstract

This report presents a survey of current safety case research related to software, and different tools available to build saftey cases. These tools are developed to help structuring safety cases, making them more understandable and easy to grasp. The aim of the report is to propose a way to structure safety cases, by using one of the tools and a goal based approach.

There are several different ways to define a safety case, one contributing factor is the industry it belongs to, but the purpose of it can be defined in the following terms:

A safety case should communicate a clear, comprehensive and defensible argument that a system is acceptably safe to operate in a particular context

Many safety cases written and used today are long, textual documents which makes it difficult to pick up the main arguments. In this report, I have looked at goal structuring notation, which is a technique to structure the safety cases and make them clearer and easier to understand. Together with the safety case tool ASCE, I propose a way to structure a former case study from the healthcare industry. The experience from using these tools is then discussed, with advantages and disadvantages presented, in regards to the concept of safety cases and safety critical systems. Finally I discuss the usefullness of safety cases in regards to the healtcare industry.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

Safety cases are structured bodies of information that are crucial to delivering today's dependable systems. The safety case should convincingly argue that a particular system or service will meets its safety requirements, due to the way in which it is designed, operated and managed. The safety case comprises claims, arguments and the supporting evidence, and is an an assurance case addressing safety.

The development of safety critical systems is guided by standards. Many standards require the development of a safety case to demonstrate the acceptability of safety critical systems. The safety case must provide confidence that the system is deemed safe enough to operate. For software where it is not possible to quantify the associated risks, current standards in the aerospace, rail and defence sectors identify design and safety processes for different Safety Integrity Levels (SILs) or similar. Software is shown to be fit for use primarily by appeal to the standards, supported with appropriate evidence. The assumption is that software developed against the process requirements of higher SILs will be less prone to critical failures and thus have a lower impact on the overall system safety [42].

It has been proposed that an evidence-based approach should be taken to software. An evidence based or "goal based" approach can be implemented by using a framework for articulating software safety arguments, based on categorisation of evidence, which is largely independent of the development process. Goal setting approaches focus on desired, measurable outcomes, rather than required product features or prescriptive processes, techniques, or procedures. The certification authority specifies a threshold of acceptable performance and often but not always a means for assuring that the threshold has been met. Basically, the standards set a goal, which may be a risk target, and usually it is up to the assurer to decide how to accomplish that goal [33].

The term Safety Case has become popular recently as a solution to the problem of regulating safety-critical systems. The term arises from the HSE (Health and Safety Executive) in the U.K., but different definitions seem to be used. To avoid confusion, this paper uses the term safety case to denote the use of a structured argument for why the system is safe [33]

There seem to be an increasing amount of safety critical systems, but the argument of if they are safely made and can be trusted, is still an ongoing question. What is the best way to develope them, and how can one know for sure if they are safe? The use of safety cases has also increased the last years, and they seem to be the answer for many. A safety related system must have a safety case; this is explicitly required by licensing regulations in a wide range of industries and equivalent requirements are given in many standards such as IEC 61508 [25]. The safety case should:

- demonstrate an adequate level of safety
- ensure safety is maintained throughout the lifetime of the system
- minimise project risk

Despite these requirements, there is little published guidance on developing safety cases.

Today's complex systems are often sophisticated socio-technical systems, integrating data and services from a wide range of sources. They are developed and delivered by diverse teams of designers, engineers, operators and managers, working hard to ensure the system reflects changes in customer requirements and in their operational environment. Managing the safety case and keeping it in step with the system is often a significant challenge on a project. At the same time, the safety case must clearly communicate to the range of stakeholders how and why the system is adequately safe [3].

1.2 Project goals

The starting point of this thesis was a litterary study I conducted in the fall of 2012, where I explored the background of the safety case. In this master thesis I want to look more cloesly at different tools available to build safety cases, and goal based approaches to implementing them, and see if it is possible to use some of the methods and tool available to structure an adequate safety case. The main motivation for this will be with regards to using safety cases within the healthcare sector, mostly in Norway. There is a huge need for system upgrades in the healthsector in Norway, and a lot of big challenges to come over the next years when more and more of this sector will be upgraded. Can safety cases be something to include when developing safety critical systems in this aspect?

1.3 Methodology

The methodology used in this report are mostly internet based, starting with a survey of the different safety case tools available. Searches on Google Scholar[21] and BIBSYS[5] were the two main databases used to find apropriate litterature surrounding the topic, and the search for tools specifically where performed on Google [20]. Typical search words used were "Safety case tool", Assurance case tool" and "Safety Case Patterns". I chose three of the tools to look at closer and eventually try out.

Research question that were followed to support this work were

- Is it possible to structure a safety case by using one of the existing tools?
- Looking from a healthcare perspective, is safety cases a good idea to include in the development of new systems?

1.4 Outline of the document

Capter 2 gives a presentation of the safety case, what it is and how it is structured

Capter 3 presents methods and tools to build safety cases

Capter 4 presents a relevant case study and a suggested goal based safety case solution

Capter 5 contains an evaluation of the different methods and a discussion about safety cases within the healthcare sector

Capter 6 presents a conclusion to this thesis

2 Background - Safety Case

2.1 Introduction

This chapter gives a description of what a safety case is, and why it is relevant in regards to developing software and especially healthcare software.

2.2 The safety case

In Europe, over recent years, there has been a marked shift in the regulatory approach to ensuring system safety, especially in regards to safety critical systems were it is absolutely crucial [28]. Safety critical systems are systems whose failure most importantly can result in loss of life, but can also cause significant damage to properties or the environment. Examples of safety critical systems are medical devices, aircraft flight control, weapons, and nuclear systems[31].

Previously prescriptive safety codes and standards were the performed methods, Firstly, with prescriptive regulations, the service provider is only required to carry out the mandated actions to discharge his legal responsibilities. If these actions then prove to be insufficient to prevent a subsequent accident, it is the regulations and those that set them that are seen to be deficient. Thus safety is viewed as the responsibility of the regulator and not the service provider whose responsibility, in law, it actually is [39].

With such approaches, operators claim safety through satisfaction of the regulators requirements. With the introduction of safety cases, the responsibility is shifted back to the operators. It is up to the operators to demonstrate that they have an adequate argument of safety. Previous approaches have focused primarily on prescriptive safety requirements, e.g. construction codes. With such approaches, operators claim safety through satisfaction of the regulator's requirements. With the introduction of safety cases, the responsibility is shifted back to the operators. It is up to the operators to demonstrate that they have an adequate argument of safety [28].

During the last twenty years the responsibility has been moved back onto the developers and the operators to argue that their systems achieve acceptable levels of safety. These arguments, presented with supporting evidence, are called safety cases [28].

A number of serious accidents such as the Piper Alpha Off-shore Oil and Gas Platform Disaster [14] and Clapham Rail Disaster [16] have been instrumental in prompting a reconsideration of how safety is managed in the safety-critical sector [28].

2.3 Definition

There are several different ways to define a safety case, the purpose of it can for example be defined in the following terms

A safety case should communicate a clear, comprehensive and defensible argument that a system is acceptably safe to operate in a particular context

Studying the safety standards relating to different sectors, it is possible to identify a number of definitions of the safety case, some clearer than others. The definition given above attempts to cleanly define the core concept that is in agreement with the majority of the definitions that exist. The following are important aspects of the above definition [28]:

Argument the safety case communicates an argument, to demonstrate how someone can reasonably conclude that a system is acceptably safe from the evidence available.

Clear the safety case is meant to communicate ideas and information to a third party, and must therefore be as clear as possible.

System the safety case refers to a system of some kind.

Acceptably the safety case will never be able to guarantee absolute safety, but are there to convince someone that the system is safe enough compared to a definition of tolerable risk.

Context the safety case must define the context within which safety it is to be argued. Context-free safety is impossible to argue, and almost any system can be unsafe if used in an inappropriate way.

2.3.1 Elements of a safety case

The main elements of the safety case are

Claim - about a property of the system or some subsystem

Evidence - which is used as the basis of the safety argument, can be facts, assumptions or subclaims

Argument - linking the evidence to the claim, which can be deterministic, probabilistic or qualitative

Inference - the mechanism which provides the transformational rules for the argument

The use of these elements is shown in figure 1 [6]

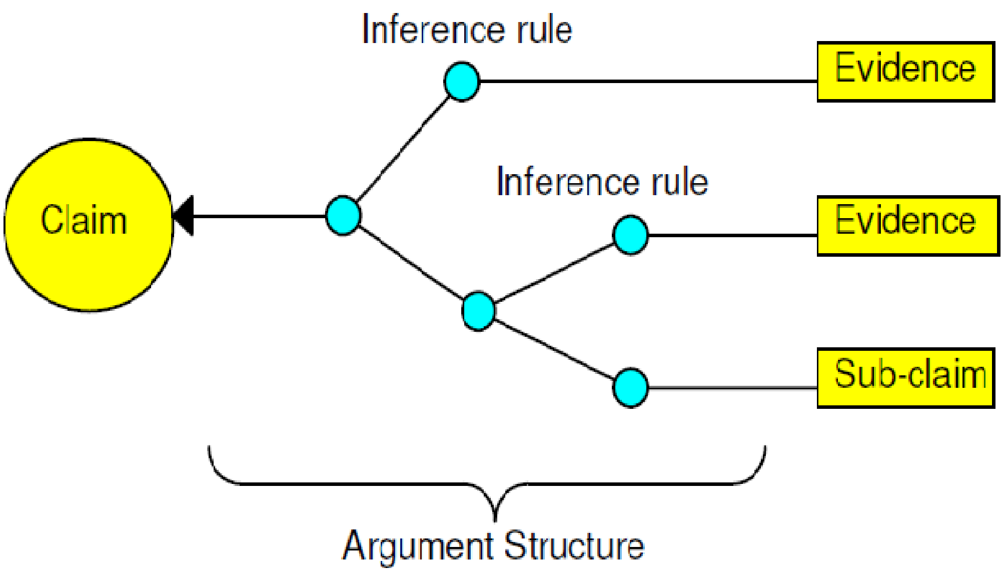


Figure 1: Safety case elements [6]

2.3.1.1 Types of Claims

Adelard [3] brakes the safety case down into claims about different attributes for the various sub-systems, for example:

- reliability and availability
- security (from external attack)
- functional correctness
- time response
- maintainability
- usability (by the operator)
- fail-safety
- accuracy
- robustness to overload
- modifiability

2.3.1.2 Types of Arguments

Different types of arguments can be used to support claims for the attributes [3]:

Deterministic: application of predetermined rules to derive a true/false claim (given some initial assumptions), e.g. formal proof of compliance to a specification, or demonstration of a safety requirement (such as execution time analysis or exhaustive test of the logic)

Probabilistic: quantitative statistical reasoning, to establish a numerical level (reliability testing) or sub claims, derived from a lower level sub argument.

Qualitative: compliance with rules that have an indirect link to the desired attributes (e.g. compliance with QMS standards, staff skills and experience)

The choice of argument will depend on the available evidence and the type of claim. For example claims for reliability would normally be supported by statistical arguments, while other claims, like maintainability, might rely on more qualitative arguments such as adherence to codes of practice.

2.3.1.3 Sources of Evidences

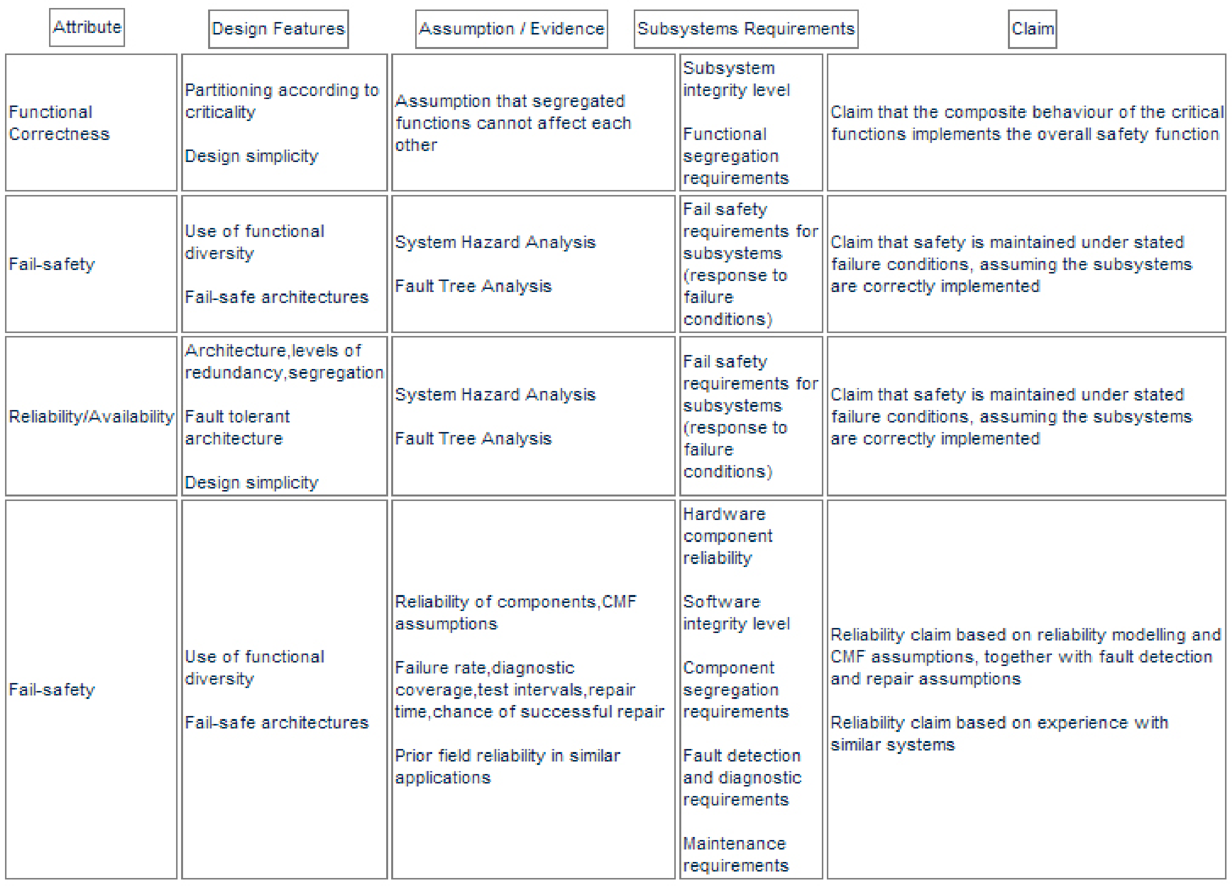
The arguments themselves can utilise evidence from the following main sources:

- the design
- the development processes
- simulated experience (via reliability testing)
- prior field experience

The choice of argument will depend in part on the availability of such evidence, e.g. claims for reliability might be based on field experience for an established design, and on development processes and reliability testing for a new design.

2.3.1.4 Example Arguments

Some example arguments for different claims, using different types of evidence, are shown in the following table.



Attribute	Design Features	Assumption / Evidence	Subsystems Requirements	Claim
Functional Correctness	Partitioning according to criticality Design simplicity	Assumption that segregated functions cannot affect each other	Subsystem integrity level Functional segregation requirements	Claim that the composite behaviour of the critical functions implements the overall safety function
Fail-safety	Use of functional diversity Fail-safe architectures	System Hazard Analysis Fault Tree Analysis	Fail safety requirements for subsystems (response to failure conditions)	Claim that safety is maintained under stated failure conditions, assuming the subsystems are correctly implemented
Reliability/Availability	Architecture,levels of redundancy,segregation Fault tolerant architecture Design simplicity	System Hazard Analysis Fault Tree Analysis	Fail safety requirements for subsystems (response to failure conditions)	Claim that safety is maintained under stated failure conditions, assuming the subsystems are correctly implemented
Fail-safety	Use of functional diversity Fail-safe architectures	Reliability of components,CMF assumptions Failure rate,diagnostic coverage,test intervals,repair time,chance of successful repair Prior field reliability in similar applications	Hardware component reliability Software integrity level Component segregation requirements Fault detection and diagnostic requirements Maintenance requirements	Reliability claim based on reliability modelling and CMF assumptions, together with fault detection and repair assumptions Reliability claim based on experience with similar systems

Table 1: Safety case elements [3]

2.4 Safety Integrety Level (SIL)

The term safety-related is used to describe systems that are required to perform a specific function or functions to ensure risks are kept at an accepted level. Such functions are, by definition, safety functions. Two types of requirements are necessary to achieve functional safety:

- safety function requirements (what the function does) and
- safety integrity requirements (the likelihood of a safety function being performed satisfactorily).

The challenge is to design the system in such a way as to prevent dangerous failures or to control them when they arise. Dangerous failures may arise from []

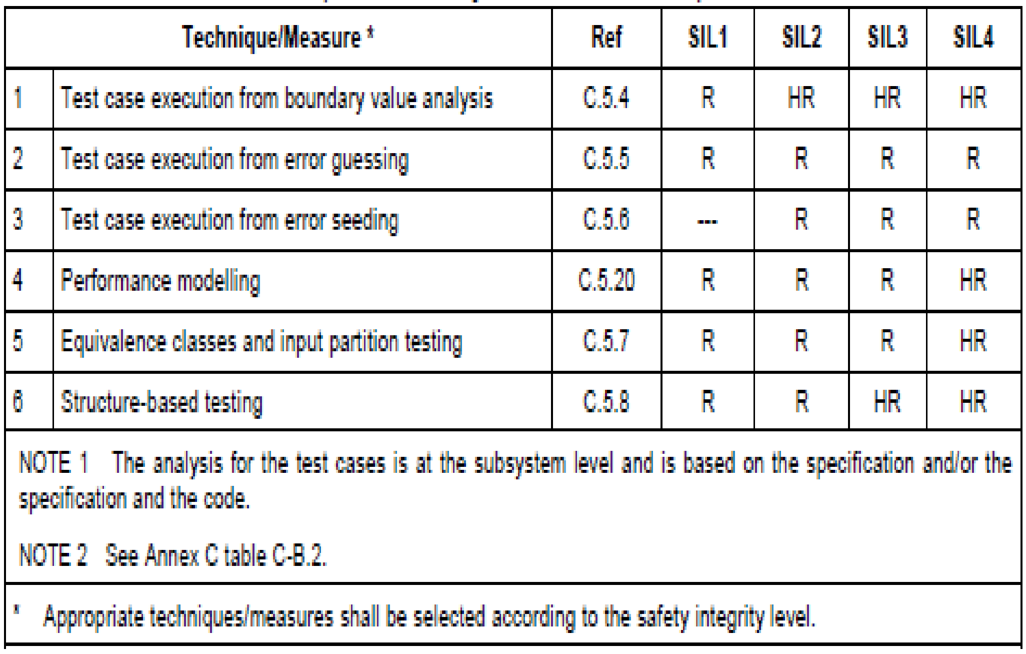
- incorrect specifications of the system, hardware or software
- omissions in the safety requirements specification (e.g. failure to develop all relevant safety functions during different modes of operation)
- software errors

- common cause failures
- human error

To what extent can a process be expected to perform safely? And, in the event of a failure, to what extent can the process be expected to fail safely? These questions are answered through the assignment of a target Safety Integrity Level (SIL). SILs are measures of the safety risk of a given process. According to IEC 61508, the SIL concept must be related to the dangerous failure rate of a system [25].

IEC 61508 specifies 4 levels of safety performance for a safety function. Safety integrity level 1 (SIL1) is the lowest level of safety integrity and safety integrity level 4 (SIL4) is the highest level. The standard details the requirements necessary to achieve each safety integrity level. These requirements are more rigorous at higher levels of safety integrity in order to achieve the required lower likelihood of dangerous failure [40].

Part three og IEC 61508 seven parts, IEC 61508-3, Software requirements, can be used in the structuring of safety cases. It contains several specifications as to what safety meassures to follow according to the different SIL levels. Table 2 below shows an example of a specification in part three of the standard, dynamic analysis and testing. The higher the SIL level, the higher safety precautions are made.



Technique/Measure *		Ref	SIL1	SIL2	SIL3	SIL4
1	Test case execution from boundary value analysis	C.5.4	R	HR	HR	HR
2	Test case execution from error guessing	C.5.5	R	R	R	R
3	Test case execution from error seeding	C.5.6	---	R	R	R
4	Performance modelling	C.5.20	R	R	R	HR
5	Equivalence classes and input partition testing	C.5.7	R	R	R	HR
6	Structure-based testing	C.5.8	R	R	HR	HR

NOTE 1 The analysis for the test cases is at the subsystem level and is based on the specification and/or the specification and the code.

NOTE 2 See Annex C table C-B.2.

* Appropriate techniques/measures shall be selected according to the safety integrity level.

Table 2: Table B.2 Dynamic analysis and testing in IEC 61508 [25]

2.5 Relevant industries

The concept of the safety case has already been adopted across many industries, including defense, aerospace, railways, and medical. Safety cases are already requirements in many safety standards. Explicit safety cases are required for military systems, the off shore oil industry, rail transport and the nuclear industry [6].

It is important that an adequate safety case is produced for a system, at the same time as the need to demonstrate safety can involve significant direct costs and indirect costs if the overall project is delayed due to too extensive safety assessments. In regulated industries such as the nuclear industry, the need to demonstrate safety to a regulator can be a major commercial risk. For example the computer-based Darlington Reactor Protection System in Canada required around 50 man years of software assessment effort, which delayed the reactor start up and millions of dollars of income were lost[6]. In this case, a safety case could have shortened the safety assessment significantly, and thus saved them a lot of money.

Computers are used in medicine far more widely than most people realize. Microprocessors control the insulin pumps and pacemakers are basically computers monitoring the patients heart. Also in the surgical procedures are computerized equipment used to a large extent. In cases such as hip replacements, spinal surgery, and ophthalmic surgery, computer controlled robotic devices are replacing the surgeons traditional tools, and providing substantial benefits to patients [31]. It is crucial that this equipment work as intended and that safety is secured.

Aircraft is another industry with highly safety critical systems. In 2002 the Boeing 777 was described by Boeing as *The Most Technologically Advanced Airplane in the World* [31]. Today, the aircrafts are even more technologically advanced, but they are still highly dependent on the systems being safe enough according to their requirements, before they are taking into use.

Many modern information systems are becoming safety critical in a general sense because financial loss can also cause great devastations. Bank systems for example might not cause the loss of lives, but they have a lot of responsibility in regards to peoples everyday lives. If something were to go wrong in a influential bank system, it could affect other industries as well.

2.5.0.5 Safety cases related to the health industry

During the last decade, there has been an increasing use of IT in healthcare, aiming to improve the healthcare quality as well as safety delivered to patients. Creating a safety case has been in practice in numerous domains and is now starting to be adopted in the health IT domain [12]. There is a lot at stake, both with patient safety and in protecting personal information about the patients, when developing software for the healthcare sector. Therefore, the use of safety cases could be a much needed tool.

Quality, is an aspect of IT that can be apparent in all every day devices, but in healthcare, poor IT quality may have safety implications [12].

The ISB 0129 standard (DSCN 14/2009): Application of clinical risk management to the manufacture of health software defines the safety case as:

Accumulation, through, the lifecycle of the health software system, of product and business process documentation and of evidence structured such as to enable a safety argument to be developed to provide a compelling, comprehensible and valid case that a system is, as far as *the clinical risk management process can realistically ascertain, free from unacceptable clinical risk for its intended use*

Hazard analysis and risk assessment can be thought of as two of the most fundamental activities of safety. During hazard analysis the system stakeholders identify conditions arising during the operation of the system that can have safety related consequences. Risk assessment is the process during which the hazards are assessed in terms of their severity and likelihood. Despotou and Kelly [12] argue that with respect to the health domain, these activities provide a number of difficulties. This is because hazards are defined in relation to consequences during the operation of a system and therefore they should be distinguished from faults.

They present a health IT example, arguing that a hazard could be administering the wrong dose to patient which directly can result in an accident, at the same time can a corrupt data record be thought of a system specific fault and contributing factor to the hazard. But even though a fault within the system may be a contributing factor to an accident, it may not necessarily be a hazard. Hazards are the states that result from the manifestation of one or the combination of more than one fault. Furthermore, in complex systems hazards may occur without the presence of a fault. Figure 2 shows how system failures result in hazards and accidents, called a bow tie diagram [12].

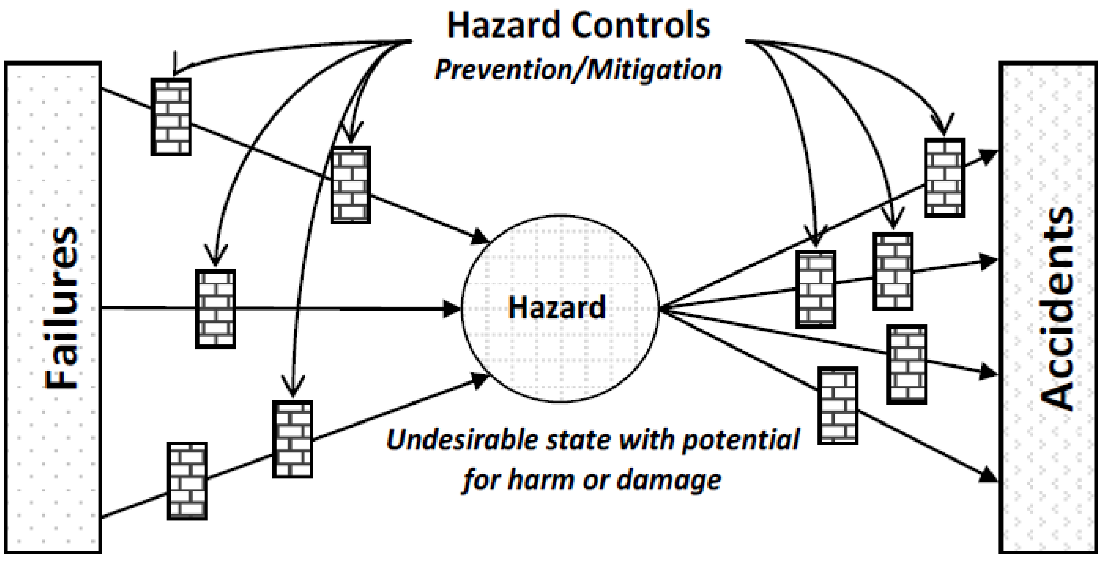


Figure 2: Bow tie diagram

Managing the hazards involves both prevention (e.g. checklists) and mitigation (e.g. recovery procedures) of hazard.

Risk assessment is a difficult activity during the safety lifecycle of a system as it is very difficult to quantify the likelihood of an event; in particular more scarce events. Whereas the likelihood of more frequent events can be captured, calculation of the likelihood of more rare events depends on a number of assumptions. This can result in inconsistencies of risk between different developers, whereas ideally there should be consistent approaches that can be communicated to and evaluated by the regulator easily [12].

2.6 Safety Case Patterns

The safety case consists of three principal elements: Requirements, Argument and Evidence. The relationship between these three elements is depicted in Figure 3 [30].

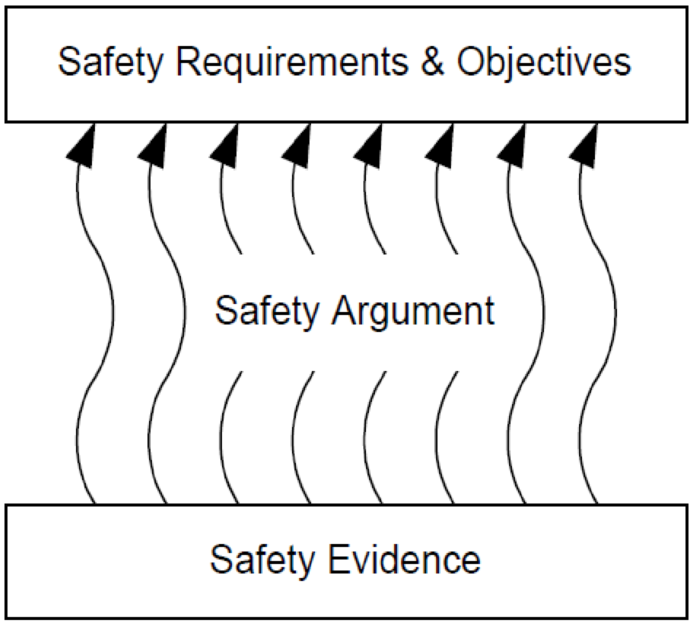


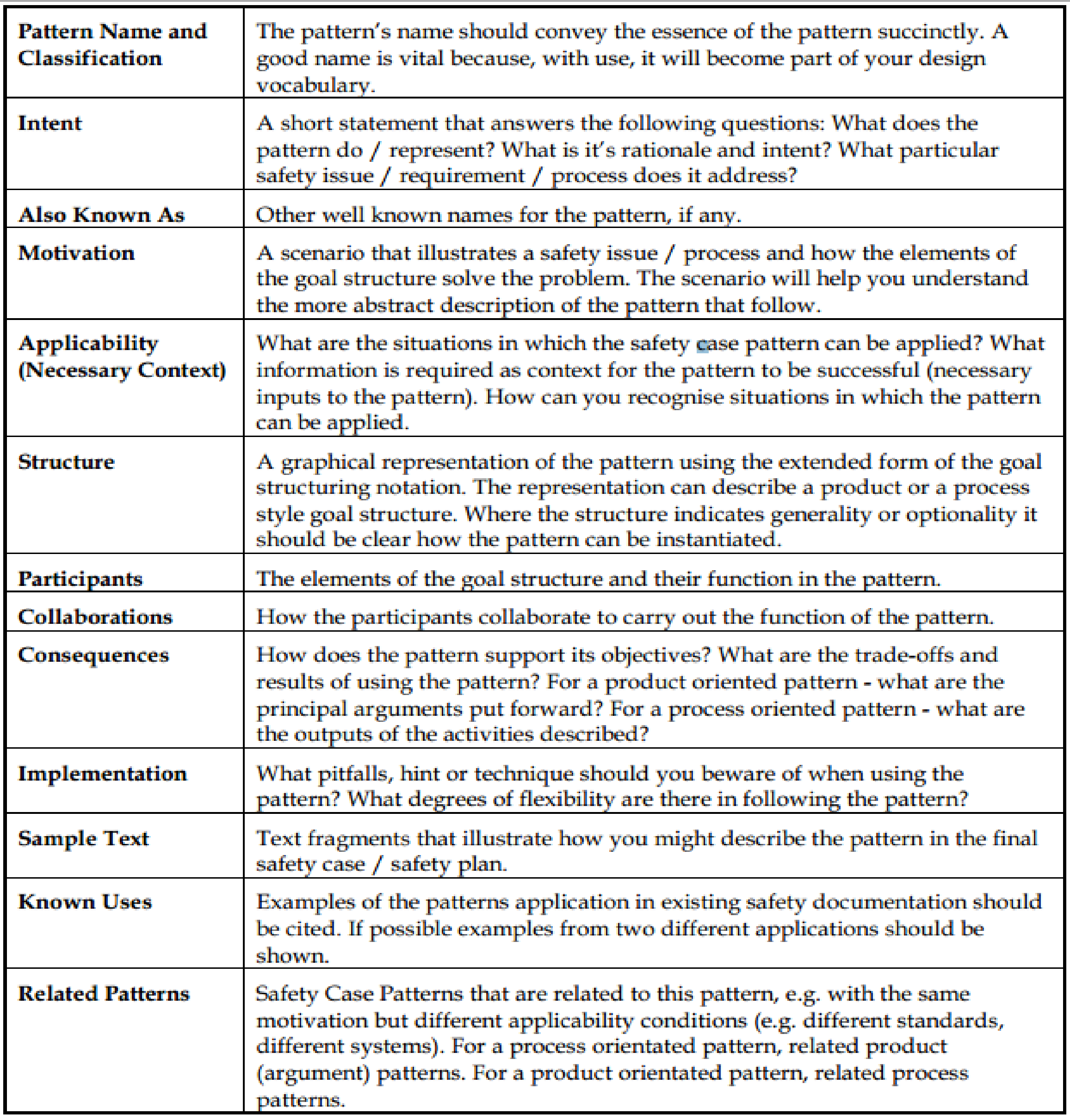
Figure 3: The role of safety argumentation

The safety argument communicates the relationship between the evidence and objectives, however, a commonly observed failing of safety cases is that the role of the safety argument is often neglected. In such safety cases, many pages of supporting evidence are often presented, but little is done to explain how this evidence relates to the safety objectives. The reader is often left to guess at an unwritten and implicit argument [30].

When constructing a safety argument, not everyone is equally good at expressing themselves. This can lead to poorly structured and difficult to understand arguments. Safety case management has no value if they have no clear meaning or shared understanding. The use of patterns can improve safety case construction, and therefore increase the level of understanding of the safety cases [30]. The idea of the patterns is that the approach supports the reuse of safety arguments between safety cases. For example, patterns extracted from a safety case built for a specific product can be reused in constructing safety cases for other products that are developed via similar processes [4]

It is difficult to propose a standard safety case structure that may be valid for most systems. However, some of the argumentation will be the same for many systems, such as all safety requirements have been realized or the like. Such argumentation structures, or so called safety case patterns, may be reused in several safety cases for different systems. By using such patterns, safety cases can be devised much faster. Similarly, safety case anti pattern can be used to express weak and flawed safety arguments [41]. A safety case pattern should be a simple and efficient solution to a particular problem, whether it be the execution of a safety process or the construction of a particular safety argument.

Kelly and McDermid [29] compares the design patterns of Gamma et. al. [19] to the description of safety cases, and says that ”safety case patterns are an attempt to capture solutions that have evolved over time”. He presents different safety case patterns in table 3.



Pattern Name and Classification	The pattern's name should convey the essence of the pattern succinctly. A good name is vital because, with use, it will become part of your design vocabulary.
Intent	A short statement that answers the following questions: What does the pattern do / represent? What is it's rationale and intent? What particular safety issue / requirement / process does it address?
Also Known As	Other well known names for the pattern, if any.
Motivation	A scenario that illustrates a safety issue / process and how the elements of the goal structure solve the problem. The scenario will help you understand the more abstract description of the pattern that follow.
Applicability (Necessary Context)	What are the situations in which the safety case pattern can be applied? What information is required as context for the pattern to be successful (necessary inputs to the pattern). How can you recognise situations in which the pattern can be applied.
Structure	A graphical representation of the pattern using the extended form of the goal structuring notation. The representation can describe a product or a process style goal structure. Where the structure indicates generality or optionality it should be clear how the pattern can be instantiated.
Participants	The elements of the goal structure and their function in the pattern.
Collaborations	How the participants collaborate to carry out the function of the pattern.
Consequences	How does the pattern support its objectives? What are the trade-offs and results of using the pattern? For a product oriented pattern - what are the principal arguments put forward? For a process oriented pattern - what are the outputs of the activities described?
Implementation	What pitfalls, hint or technique should you beware of when using the pattern? What degrees of flexibility are there in following the pattern?
Sample Text	Text fragments that illustrate how you might describe the pattern in the final safety case / safety plan.
Known Uses	Examples of the patterns application in existing safety documentation should be cited. If possible examples from two different applications should be shown.
Related Patterns	Safety Case Patterns that are related to this pattern, e.g. with the same motivation but different applicability conditions (e.g. different standards, different systems). For a process orientated pattern, related product (argument) patterns. For a product orientated pattern, related process patterns.

Table 3: Safety case patterns [29]

2.7 Implementing a Safety Case

The first important aspects to consider before creating a safety case is experience and understanding, both in terms of the system being developed and in structuring safety cases. That means taking the knowledge and reapplying it to similar environments or new situations, and at the same time recognising the details which might give it some different needs [18].

At an early stage of the development, the safety argument will only be an outline of the final safety argument. It is useful to include potential solutions (or markers) for goals even though these solutions are at a high level. The review of the early safety argument can assist in both providing a sound framework for ongoing development and assist with buy in from the client and/or regulatory authorities [10].

To implement a safety case, we need to[6]:

- Make an explicit set of claims about the system
- Produce the supporting evidence
- Provide a set of safety arguments that link the claims to the evidence
- Make clear the assumptions and judgments underlying the arguments
- Allow different viewpoints and levels of detail

To benefit from safety case management, it is important to consider the safety case throughout the project. Some regard the safety case as a "bolt on" accessory to the system, and might even produce it after the system is built, which can prove to be an expensive affair. To implement the safety case it has been have advocated the integration of safety case development into the design process. By including the safety case and its possible costs in the design trade-offs, unsuitable designs can be avoided together with their attendant costs in safety case construction, project delays and long-term support overheads[3]. [**?**] recommend an approach where the safety case is taken into account during the whole process:

- Integration of the safety case into the design and development process
- "Layered" safety cases, i.e. a top level safety case with subsidiary safety cases for subsystems
- Traceability between system and subsystem levels
- "Design for assessment" which takes into account the costs and complexity of the safety case as well as the design

2.7.0.6 Safety case developing life cycle

It is increasingly recognised by both safety case practitioners and many safety standards that safety case development, contrary to what may historically have been practised, cannot be left as an activity to be performed towards the end of the safety lifecycle. This view of safety case production being left until all analysis and development is completed is depicted in the figure 4 [28].

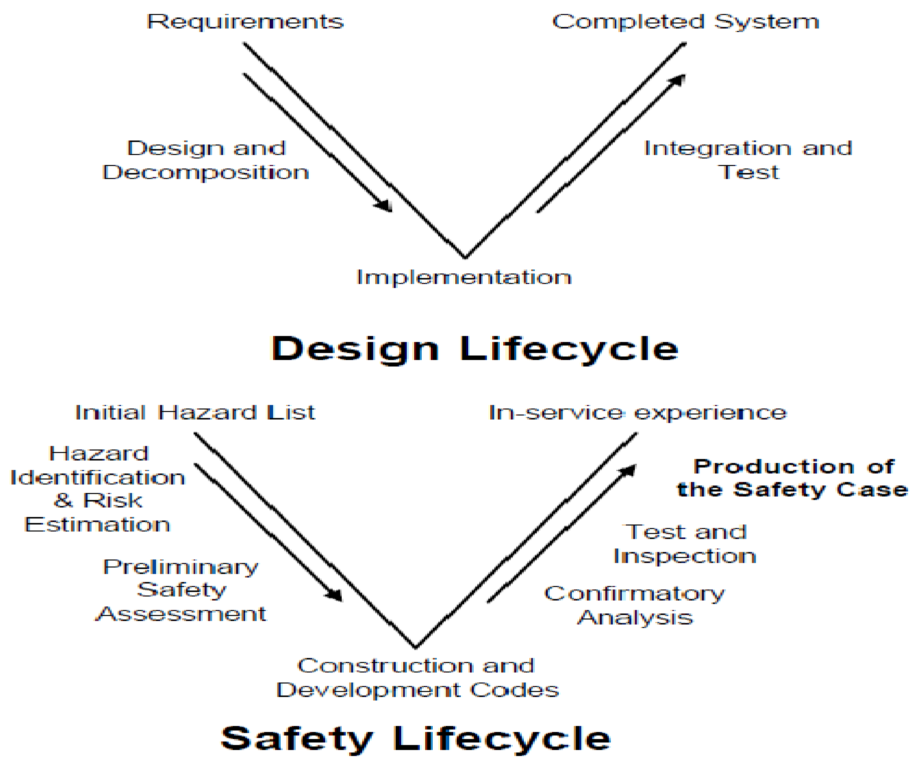


Figure 4: A historical view of safety case development

Kelly presents several problems with this way of implementing the safety case to the development process

- Large amounts of redesign apon realizing that a satisfactory safety argument can not be constructed
- Less robust safety arguments being presented in the final safety case
- Lost safety rationale. The rationale concerning the safety aspect of the design is best recorded at design time.

Kelly continues to present an *evolving safety argument* [28]. At the heart of the concept of phased safety case production is the presentation of an evolving safety argument. At the Preliminary Safety Case stage the aim is to present an outline safety argument showing the principal objectives, approach to arguing safety and the forms of evidence anticipated. At the Interim stage the argument should be evolved to reflect the increased knowledge concerning the detailed design and specification of the system. At the Operational stage the argument can again be evolved further to reflect evidence concerning the system as implemented and tested. The integration between the production of these safety cases and the traditional development lifecycle is depicted in the figure 5.

Kelly argues that early identification of safety objectives allows the design to be influenced as system development progresses in order that a more compelling safety case may be established [28].

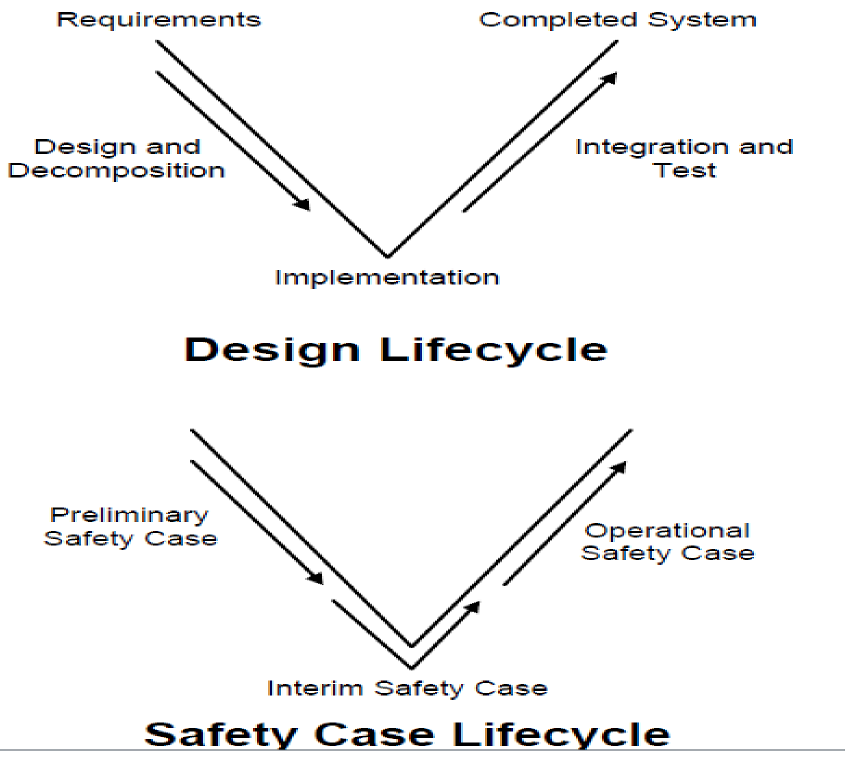


Figure 5: An integrated view of safety case development

Figure 6 show an example of a layered safety case which starts at the top level with the overall safety target and then transforms into the derived requirements for subsystems [6].

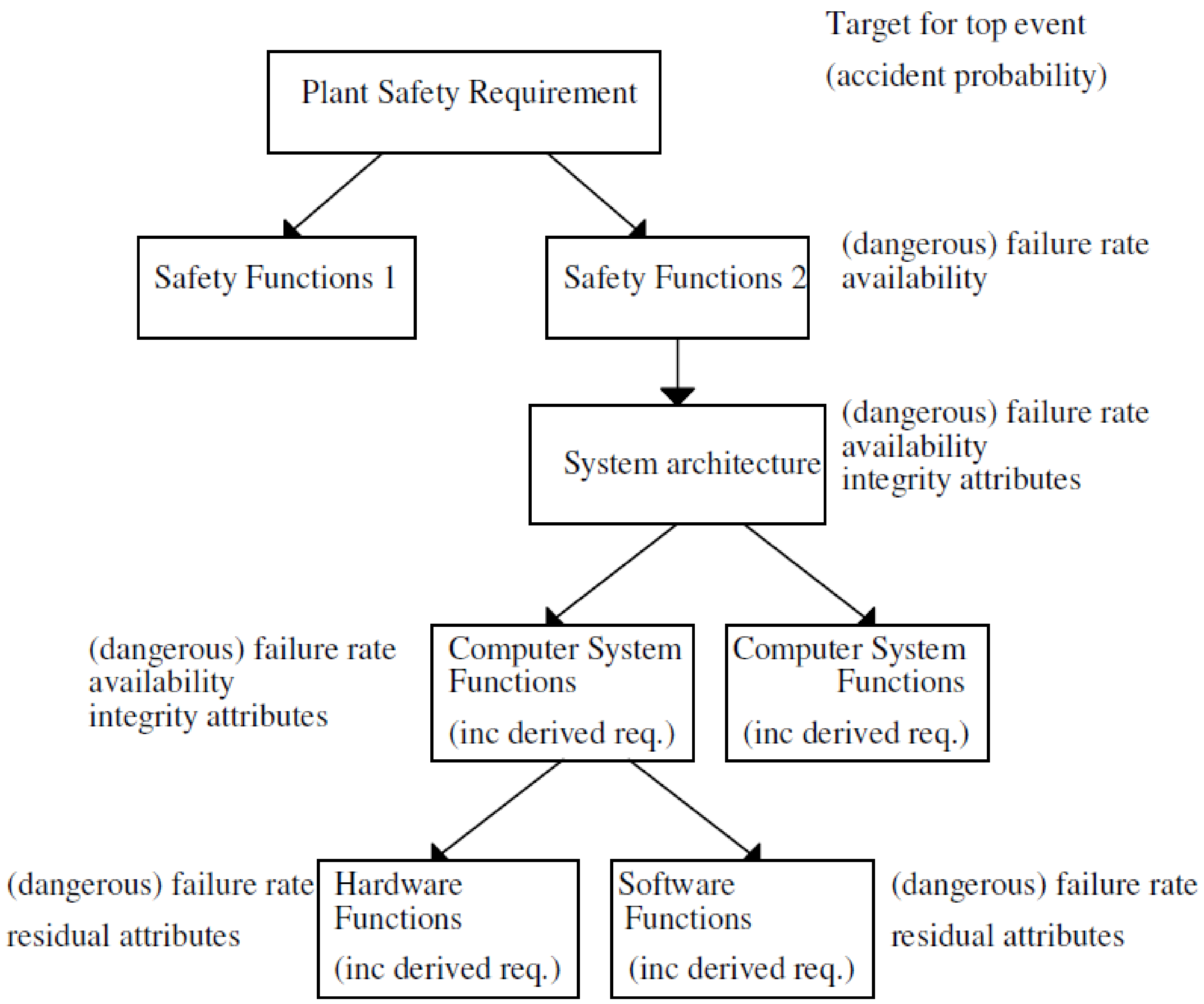


Figure 6: Example of a layered safety case [6]

Adelard [8] explains safety assurance in terms of a triangle comprising:

- The use of accepted standards and guidelines.
- Justification via a set of claims/goals about the system's safety behaviour.
- An investigation of known potential vulnerabilities of the system.

This is illustrated in Figure 7 below.

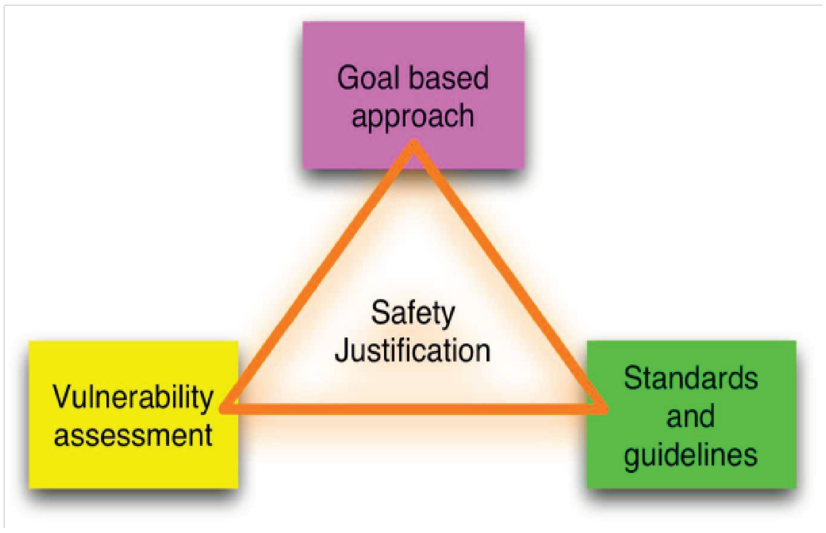


Figure 7: The safety case triangle [8]

The first approach is based on demonstrating compliance to a known safety standard. This is a common strategy, for example the Emphasis tool produced by the nuclear industry supports an initial assessment of compliance with IEC61508 [25]. The second approach is goal based where specific safety goals for the systems are supported by arguments and evidence at progressively more detailed levels. This would typically be implemented using Claims Argument Evidence (CAE) or goal structuring notation (GSN) notations. The final approach is a vulnerability based argument, where it is demonstrated that potential vulnerabilities within a system do not constitute a problem. This is essentially a bottom up approach as opposed to the top down approach used in goal based methods [8].

3 Existing methods and tools

3.1 Introduction

In this chapter I will present two types of structures which can be used as methods when constructing safety cases, and three different types of tools which have been developed to create safety cases.

3.2 Structuring with patterns

In order for assurance cases to be developed, discussed, challenged, presented and reviewed amongst stakeholders, and maintained throughout the product lifecycle, it is necessary for them to be documented clearly. The documented argument of the assurance case should be structured to be comprehensible to all safety-case stakeholders. It should also be clear how the evidence is being asserted to support this argument [22].

Software safety argument patterns provide a way of capturing good practice in software safety arguments. Patterns are widely used within software engineering as a way of abstracting the fundamental design strategies from the details of particular designs. The use of patterns as a way of documenting and reusing successful safety argument structures was pioneered by Kelly in [27]. As with software design, software safety argument patterns can be used to abstract the fundamental argument strategies from the details of a particular argument [23].

During the last decades, systematic approaches to express safety cases have been introduced. The Goal Structuring Notation (GSN), the Claims Argument Evidence (CAE) notation and Argumentation Metamodel (ARM) are examples of such approaches [11]. These are the patterns I have chosen to take a closer look at.

3.2.1 Goal Structuring notation

Goal Structuring Notation (GSN) is a technique that is increasingly being used in safety critical industries to improve the structure, rigor, and clarity of safety arguments [28].

GSN explicitly represents the individual elements of any safety argument (requirements, claims, evidence and context) and the relationships that exist between these elements (i.e. how individual requirements are supported by specific claims, how claims are supported by evidence and the assumed context that is defined for the argument). The principal symbols of the notation are shown in Figure 8 (with example instances of each concept) [30].

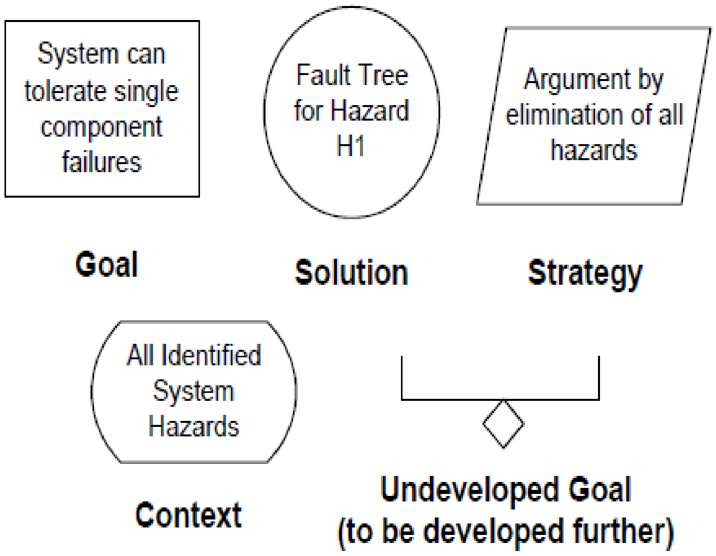

System can tolerate single component failures
Goal
Fault Tree for Hazard H1
Solution
Argument by elimination of all hazards
Strategy
All Identified System Hazards
Context
Undeveloped Goal (to be developed further)

Figure 8: Principal Elements of the Goal Structuring Notation [30]

At the heart of GSN is the explicit documentation of a hierarchy of claims [22]. When the elements of the GSN are linked together in a network, they will be a goal structure. The main principal of a goal structure is to show how the goals can be broken down into sub goals, until it can be supported by direct reference to available evidence, also known as solutions. Figure 9 shows an example of a goal structure, with the top level goal *C/S (Control System) Logic is fault tree* first broken down intro strategies, then goals and finally four available solutions [30].

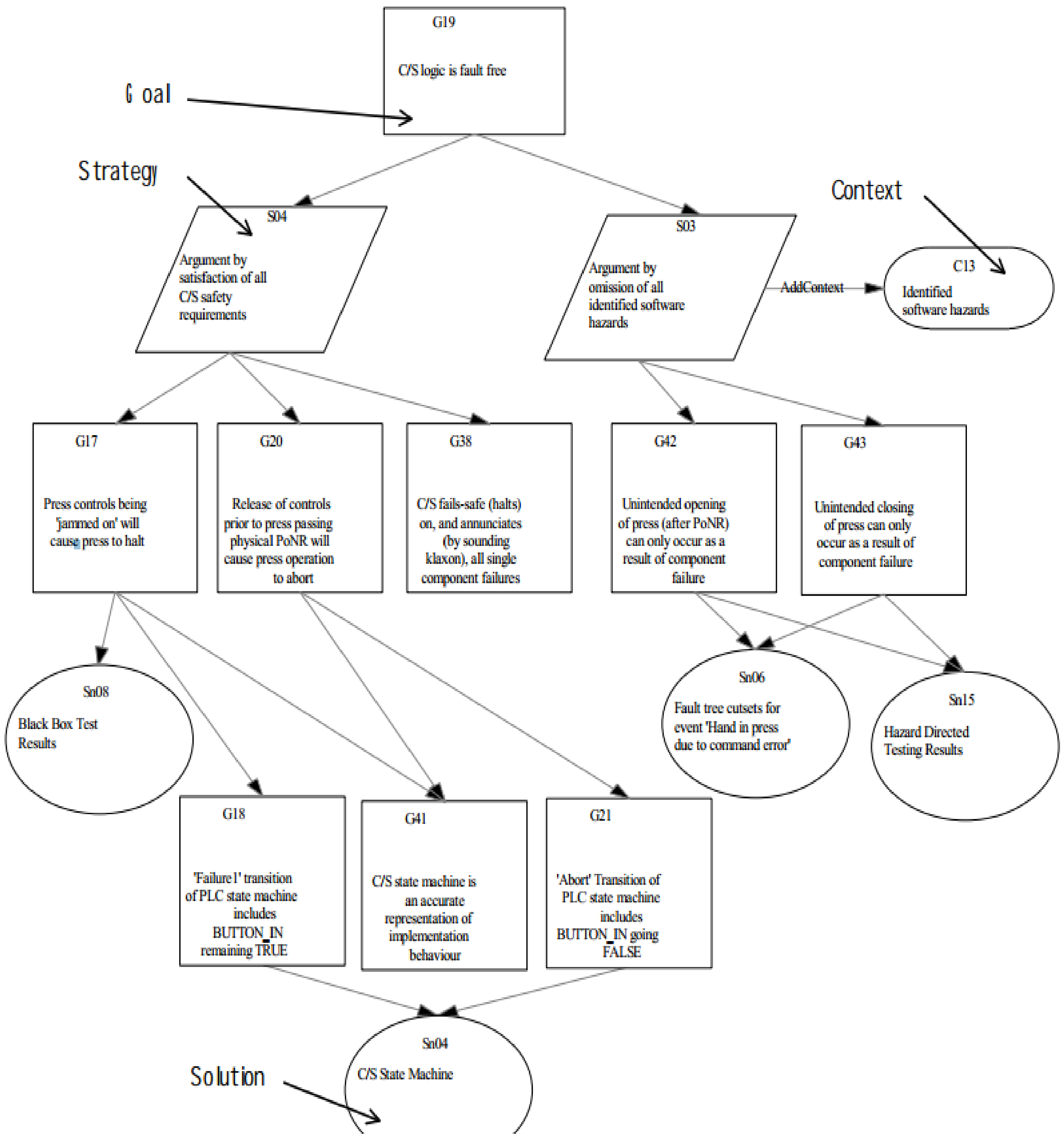

Goal
Strategy
Context
Solution
G19
C/S logic is fault free
S04
Argument by satisfaction of all C/S safety requirements
S03
Argument by omission of all identified software hazards
AddContext
C13
Identified software hazards
G17
Press controls being 'jammed on' will cause press to halt
G20
Release of controls prior to press passing physical PoNR will cause press operation to abort
G38
C/S fails-safe (halts) on, and annunciates (by sounding klaxon), all single component failures
G42
Unintended opening of press (after PoNR) can only occur as a result of component failure
G43
Unintended closing of press can only occur as a result of component failure
Sn08
Black Box Test Results
Sn06
Fault tree cutsets for event 'Hand in press due to command error'
Sn15
Hazard Directed Testing Results
G18
'Failure1' transition of PLC state machine includes BUTTON_IN remaining TRUE
G41
C/S state machine is an accurate representation of implementation behaviour
G21
'Abort' Transition of PLC state machine includes BUTTON_IN going FALSE
Sn04
C/S State Machine

Figure 9: Example of a Goal Structure [29]

A particular GSN decomposition proposed by the EU project EASIS [9] organizes the argumentation into a product branch and a process branch, claiming that a system is safe if the process is safe and the product is safe. The safety of the process can be assured through application of certified development standards, such as the IEC 61508 [25].

On the product side, we can decompose the claim the product is safe into the sub goal the safety requirements are traceable which turns the satisfaction of our safety case into a traceability problem. We can argue, that if all safety related aspects of our system can be traced to their origin and to their realization, the system is safe, given the process is safe (proof of the process branch). Traceability is a prerequisite for assessment and validation of the safety goals, which we refer to as proof of safety requirements. We can then decompose the traceability goal further into proof of safety requirements, origin of safety requirements documented, and safety requirements realized [41].

3.2.1.1 GSN with patterns

Patterns can emerge at many different levels in the safety argument and at varying degrees of specificity. At the highest level it is possible to identify a number of basic argument structures that are used to decompose ill defined system safety requirements. For example, against the ultimate top level requirement *System X is safe* two possible argument approaches could be applied [29]

- Hazard Directed Argument
- Functional Decomposition Argument

The figure below shows the GSN pattern representing a hazard directed argument. In this pattern, the implicit definition of safe is hazard avoidance. The requirement G1 is addressed by arguing that all identified hazards have been addressed (S1). This strategy can only be executed in the context of some knowledge of plausible hazards. With this information (C1), identifying n hazards, n sub goals of the form G2 can be constructed. The argument then develops from this hazard avoidance goals [29]. Figure 10 shows a hazard avoidance pattern.

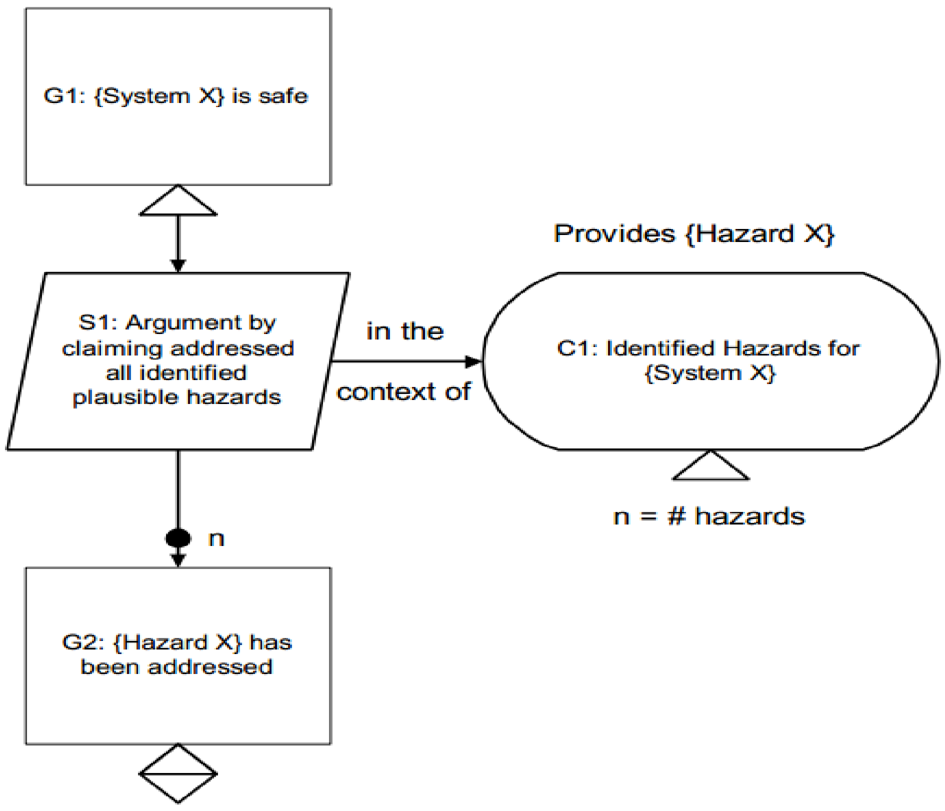

G1: {System X} is safe
S1: Argument by claiming addressed all identified plausible hazards
in the context of
Provides {Hazard X}
C1: Identified Hazards for {System X}
n = # hazards
n
G2: {Hazard X} has been addressed

Figure 10: Hazard Avoidence Pattern

At lower levels in the safety case argument, patterns also emerge. Figure 11 shows an example pattern that could be found in the lower levels of a safety case argument [29].

In this pattern, the claim that the software element in a system is fault free (G1) is supported by two main strands of argument (S1 and S2). First, over a list (C3) of identified hazardous software conditions the m sub-goals of the form G3 are expressed, to argue that these hazards can only occur through physical component failures. Second, over a list (C4) of identified software requirements the n sub goals of the form G2 are expressed to argue that these properties are enforced in the software. In order that this pattern will be appropriately applied, the context of the pattern is made clear through the elements C1 and C2 - both defining key terms in the top level claim [29].

3.2.1.2 Advantages and disadvantages to Goal Structuring Notation

Adopting a goal based regulation, as opposed to using a prescriptive regulation, gives a greater freedom in developing technical solutions and accommodating different standards. It ensures that the service provider is responsible for the safety instead of it being the regulators responsibility. A goal based regulation also makes it possible to use the best engineering practice as seen fit at the time of development, and not the best practice at the time the safety case was written. Technologies are constantly changing, and this should not stand in

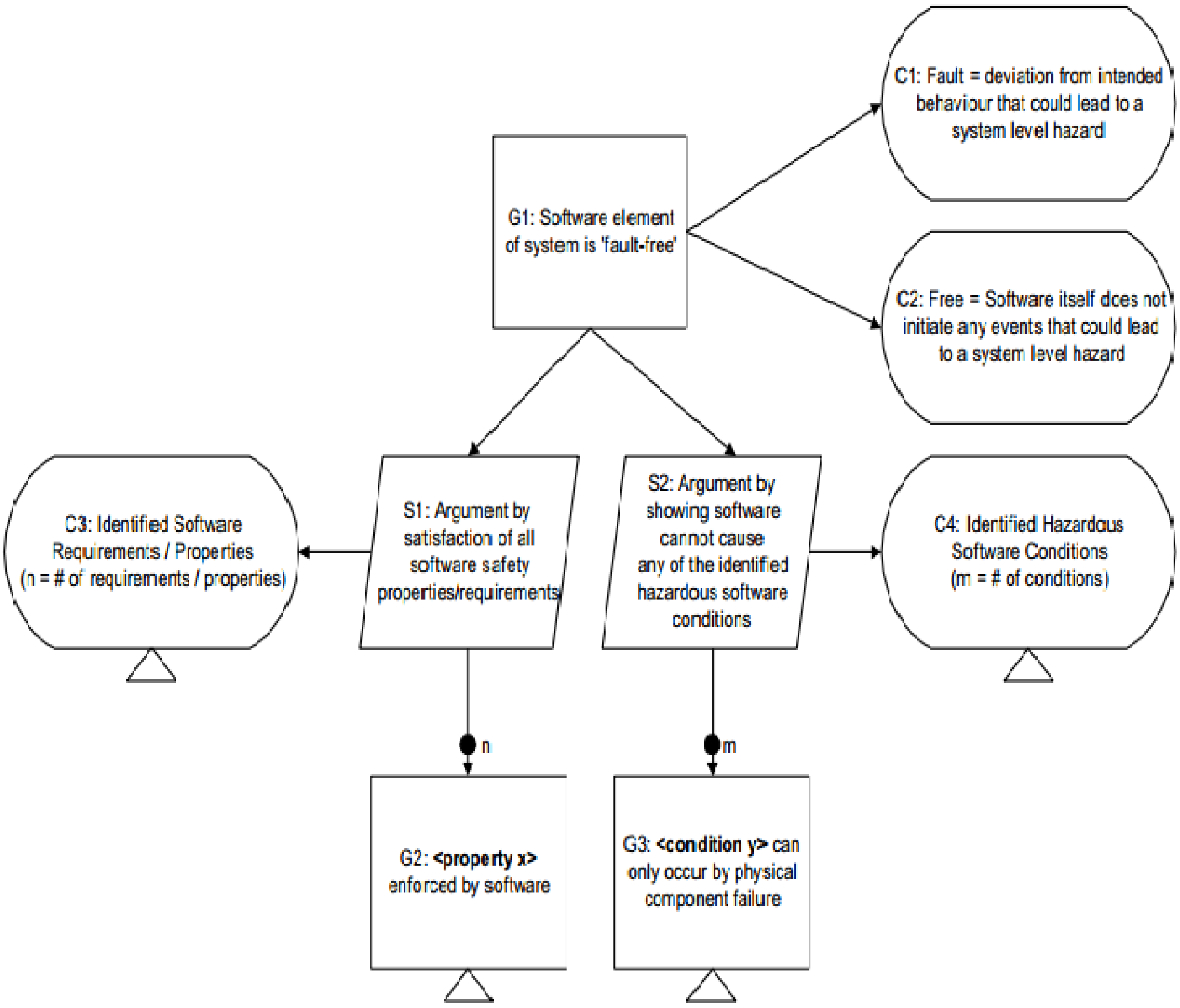

G1: Software element of system is 'fault-free'
C1: Fault = deviation from intended behaviour that could lead to a system level hazard
C2: Free = Software itself does not initiate any events that could lead to a system level hazard
S1: Argument by satisfaction of all software safety properties/requirements
S2: Argument by showing software cannot cause any of the identified hazardous software conditions
C3: Identified Software Requirements / Properties (n = # of requirements / properties)
C4: Identified Hazardous Software Conditions (m = # of conditions)
n
m
G2: <property x> enforced by software
G3: <condition y> can only occur by physical component failure

Figure 11: Fault Free Software Pattern

the way of anything [39].

Prescriptive regulations are often based on past experience and can therefore end up being inappropriate, or even worse to create unnecessary dangers in industries that are technically innovative. It is the innovator that is best placed to ensure the safety of their design, not the regulator [39]. A suggestion for a hybrid between a goal based and a prescriptive approach has been made. It bases itself on the fact that goal based approaches can be difficult to apply because they require a lot of work, competence and experience [17].

3.2.2 Claims Argument Evidence (CAE)

The claim argument evidence was created by Adelard as a simple way to present safety case arguments, and it is is a simple yet effective notation for presenting and communicating a safety argument. Adelard developed CAE as a straightforward notation for structuring safety cases, and it is one of the key notations available in their software ASCE [3].

Claims, Arguments and Evidence is a simple yet effective notation for structuring arguments to communicate how a system is adequately safe in its environment. Adelard says users find CAE to be a powerful tool to both structure their safety arguments, and to communicate them to a range of project stakeholders [3]. The figure below shows the key notions of CAE.

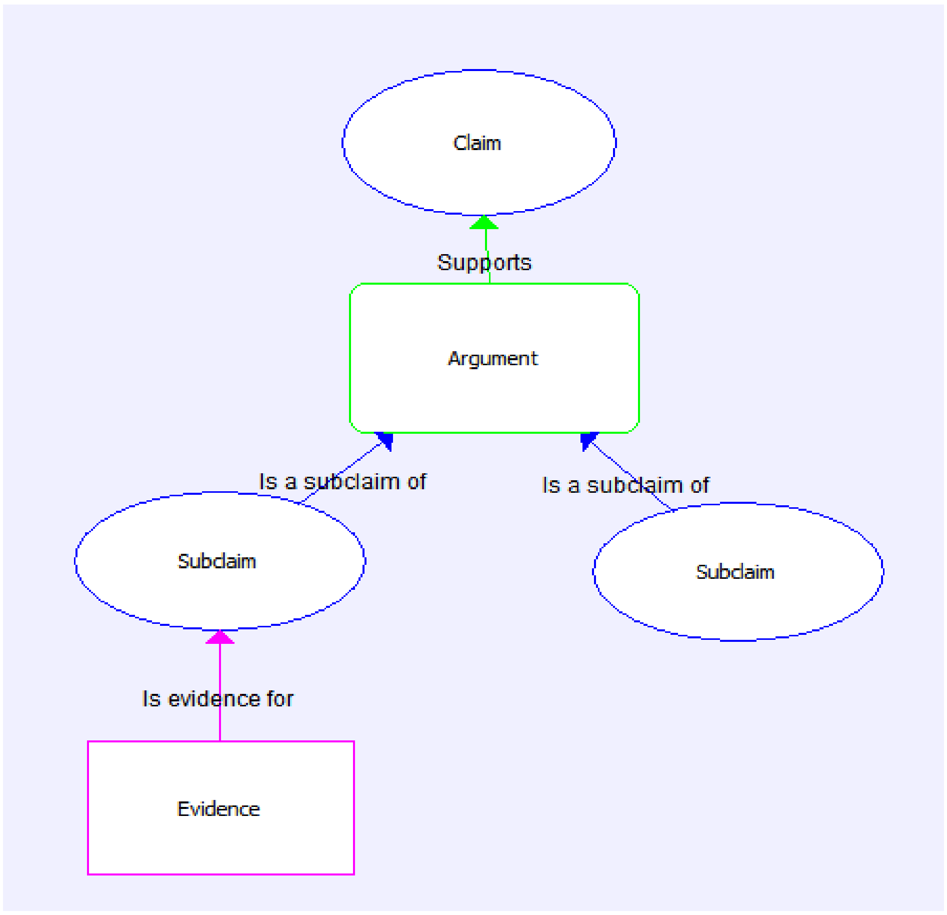


Figure 12: CAE screenshot [3]

3.2.2.1 Elements of CAE

CAE can structure the overall argument into the following elements [3]

Claim - a statement asserted within the argument that can be assessed to be true or false. Each claim is be supported by a number of sub claims, arguments or evidence.

- "System X is adequately safe during the shut down phase of operation"

Argument - a description of the argument approach presented in support of a claim. This element is optional, but often it is good practice to include to explain the approach to satisfying the parent claim

- "Argue by considering safety of subsystems"

Evidence - a reference to the evidence being presented in support of the claim or argument. Usually the evidence node will summarise and link out to the relevant report containing the evidence

- "Interlock design documentation"

3.2.3 Argumentation Metamodel (ARM)

Argumentation Metamodel is an OMG standard, which has been specified in response to the need for safety and security cases, and its basic elements can be traced to GSN, having also been influenced by another safety case notation, the Claims Argument Evidence (CAE) notation [6].

The Argumentation Metamodel (ARM) is an adopted standard by the OMG, and is the product of the OMGs system assurance task force [38]. The initial purpose of ARM was to represent the argument part of an assurance case, leaving capturing of evidence to the Structured Assurance Evidence Metamodel (SAEM), which also is an OMG standard. It was the initial intent that SAEM and ARM would work together to cap-ture an assurance case. ARM and SAEM do have some overlap, with the former able to capture basic concepts regarding evidence. A complete separation would make ARM impractical, as it would not be able to represent evidence. More recent work by the system assurance task force has resulted in consolidating the two metamodels in one; the Structured Assurance Case Metamodel (SACM), which however is not yet an adopted standard [11].

The main aim of the ARM is to align two major notations and facilitates the tool support. Unfortunately it only reects main constructs between the two, and some specic features, which are not compatible are missing from it. For instance, patterns are not included in the ARM [37].

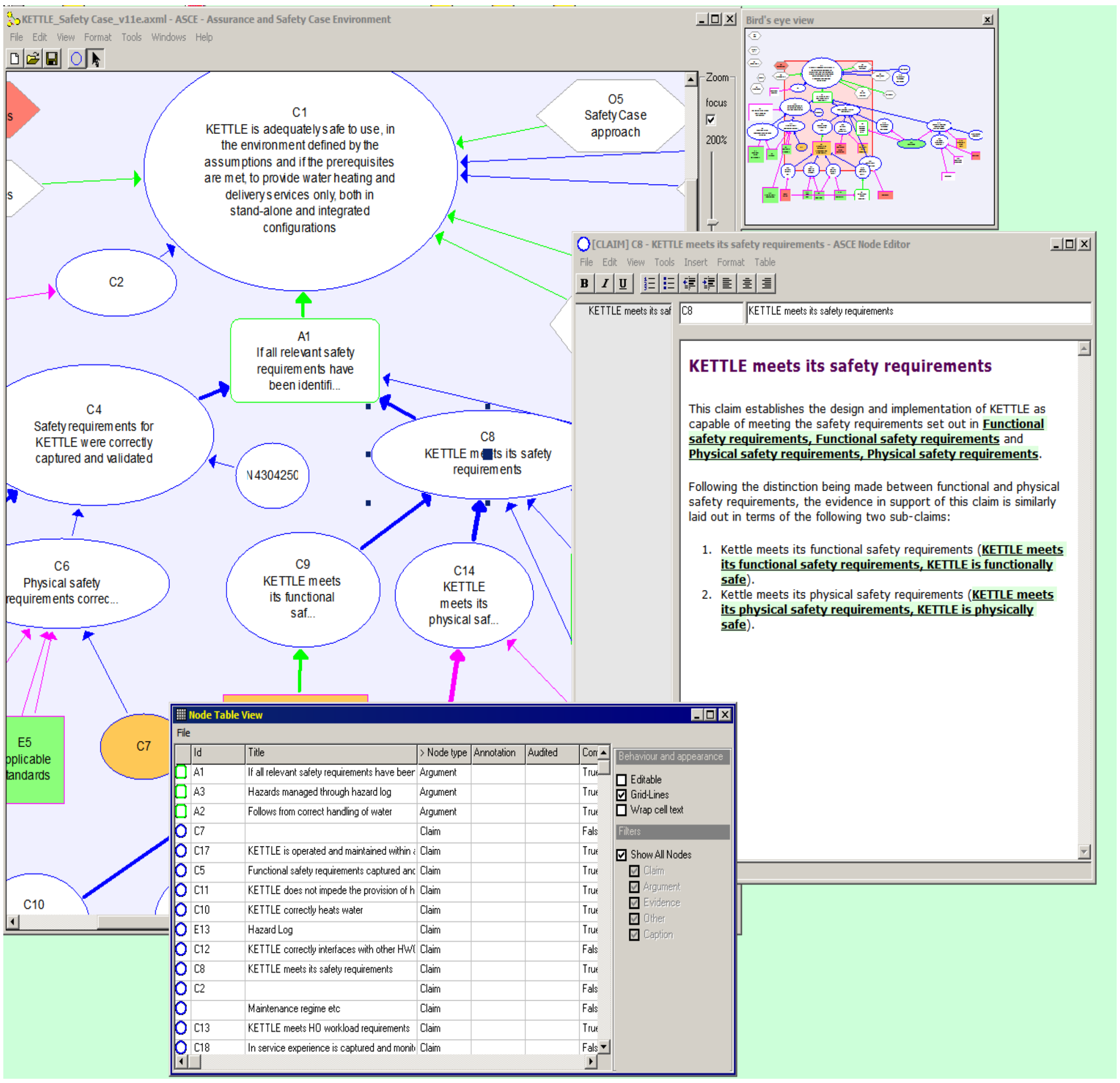
3.3 Safety case tools

3.3.1 ASCE

ASCE is a software tool to simplify the creation and management of your safety case, helping to reduce project and system risk through effective and straightforward communication of the safety argument and its associated evidence. It is an industry standard, used by hundreds of organizations world-wide [3].

ASCE lets you build robust arguments using recognised notations such as Claims Arguments Evidence (CAE) and Goal Structuring Notation (GSN). It targets problems with a proven approach to help you deliver robust safety cases. ASCE manages information complexity and communicates your argument to your stakeholders [3].

Figure 7 shows a screenshot of ASCE


KETTLE_Safety Case_v11e.axml - ASCE - Assurance and Safety Case Environment
File Edit View Format Tools Windows Help
Bird's eye view
Zoom
focus
200%
C1
KETTLE is adequately safe to use, in the environment defined by the assumptions and if the prerequisites are met, to provide water heating and delivery services only, both in stand-alone and integrated configurations
O5
Safety Case approach
C2
A1
If all relevant safety requirements have been identifi...
C4
Safety requirements for KETTLE were correctly captured and validated
N4304250
C8
KETTLE meets its safety requirements
C6
Physical safety requirements correc...
C9
KETTLE meets its functional saf...
C14
KETTLE meets its physical saf...
E5
pplicable tandards
C7
C10
[CLAIM] C8 - KETTLE meets its safety requirements - ASCE Node Editor
File Edit View Tools Insert Format Table
KETTLE meets its saf
C8
KETTLE meets its safety requirements

KETTLE meets its safety requirements

This claim establishes the design and implementation of KETTLE as capable of meeting the safety requirements set out in **Functional safety requirements, Functional safety requirements** and **Physical safety requirements, Physical safety requirements**.

Following the distinction being made between functional and physical safety requirements, the evidence in support of this claim is similarly laid out in terms of the following two sub-claims:

1. Kettle meets its functional safety requirements (**KETTLE meets its functional safety requirements, KETTLE is functionally safe**).
2. Kettle meets its physical safety requirements (**KETTLE meets its physical safety requirements, KETTLE is physically safe**).

	Id	Title	> Node type	Annotation	Audited	Con
	A1	If all relevant safety requirements have beer	Argument			True
	A3	Hazards managed through hazard log	Argument			True
	A2	Follows from correct handling of water	Argument			True
	C7		Claim			Fals
	C17	KETTLE is operated and maintained within a	Claim			True
	C5	Functional safety requirements captured anc	Claim			True
	C11	KETTLE does not impede the provision of h	Claim			True
	C10	KETTLE correctly heats water	Claim			True
	E13	Hazard Log	Claim			True
	C12	KETTLE correctly interfaces with other HW(	Claim			Fals
	C8	KETTLE meets its safety requirements	Claim			True
	C2		Claim			Fals
		Maintenance regime etc	Claim			Fals
	C13	KETTLE meets HO workload requirements	Claim			True
	C18	In service experience is captured and monit	Claim			Fals

Behaviour and appearance
- [ ] Editable
- [✓] Grid-Lines
- [ ] Wrap cell text

Filters
- [✓] Show All Nodes
  - [✓] Claim
  - [✓] Argument
  - [✓] Evidence
  - [✓] Other
  - [✓] Caption

Figure 13: ASCE screenshot [3]

3.3.2 ACedit

ACedit is an open source tool that resulted from a postgraduate project, and therefore it is an academic prototype and is provided on an as is basis. The tool has been developed as an Eclipse plugin and it implements the GSN and OMG ARM metamodels, including a transformation between the two. The tool supports the graphical construction of assurance cases in GSN and ARM and a number of model management techniques that apply to them [11].

A case is a conceptual entity initially conceived in the minds of the developers. In order to represent the case, a means of capturing the underlying reasoning is required. Two GSN extensions offer provisions for reusing arguments and these concepts are also implemented by

ACedit [11]

- Pattern - Patterns provide the concepts to capture a generalised argument.
- Modular GSN - Modular extensions allow cohesive arguments to be captured as argument modules, which can be referenced from other argument modules.

The tool is implemented as a set of eclipse plugins, and implements a graphical user interface which consists of the GSN and ARM editors and a number of model management tools. ACedit consists of a number of layers. At the bottom lies the core framework layer of the tool. The tool can be thought of as a set of plug-ins on top of the functionality offered by the core eclipse framework. The second layer consists of the eclipse plug-ins that were used to implement the tool. ACedit uses the Eclipse Modelling Framework (EMF), the Graphical Modelling Framework (GMF) [15], the Graphical Editing Framework (GEF), and Epsilon. The GMF and GEF plugins are utilized to implement the graphical user interface of the tool. EMF and Epsilon are responsible for the model development part. EMF was used for the specification of the metamodels and the generation the corresponding java code [11].

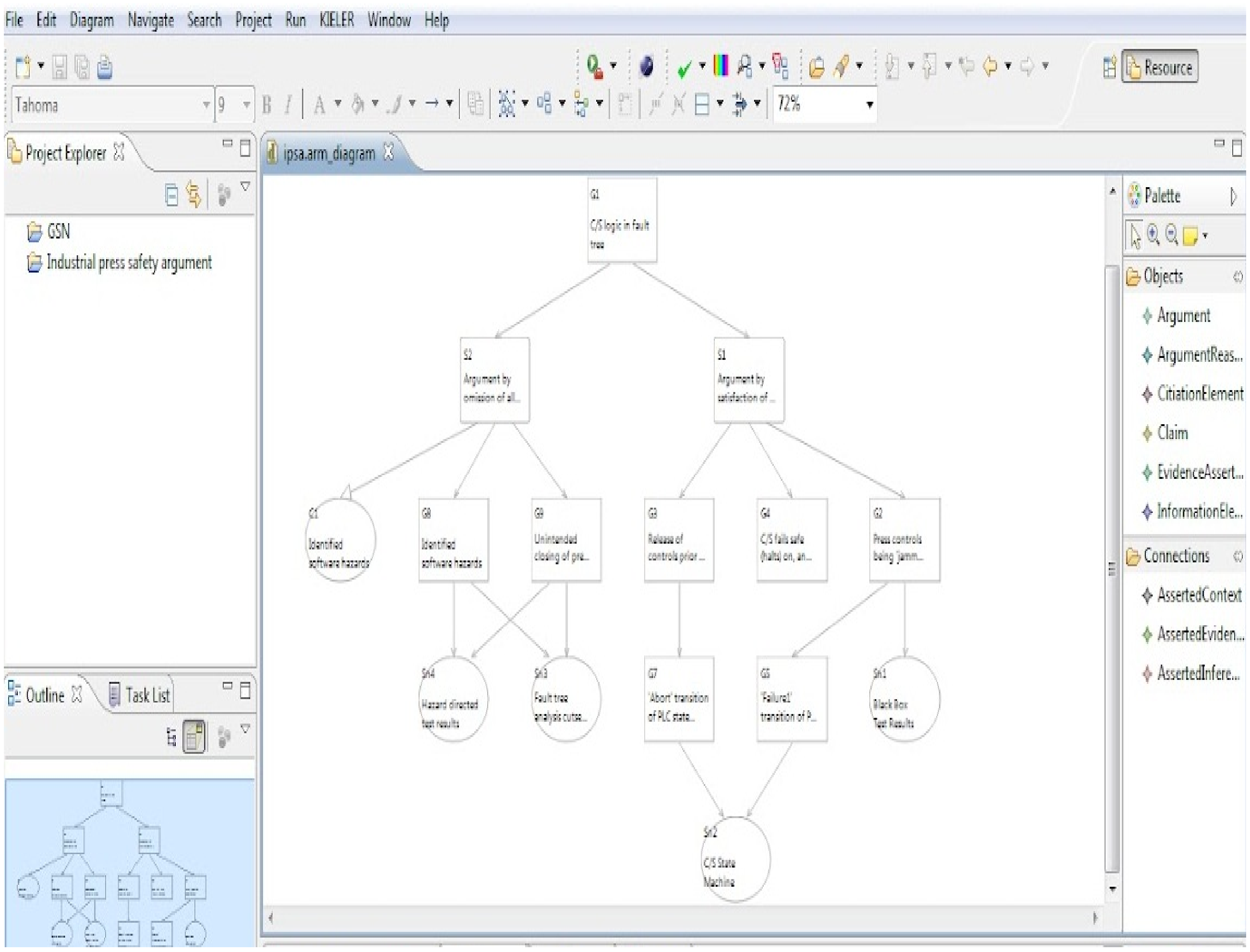

File Edit Diagram Navigate Search Project Run KIELER Window Help
Resource
Tahoma
9
72%
Project Explorer
GSN
Industrial press safety argument
ipsa.arm_diagram
G1 C/S logic in fault tree
S2 Argument by omission of all...
S1 Argument by satisfaction of...
C1 Identified software hazards
G8 Identified software hazards
G9 Unintended closing of pre...
G3 Release of controls prior...
G4 C/S fails safe (halts) on, an...
G2 Press controls being 'jamm...
Sn4 Hazard directed test results
Sn3 Fault tree analysis cutse...
G7 'Abort' transition of PLC state...
G5 'Failure1' transition of P...
Sn1 Black Box Test Results
Sn2 C/S State Machine
Palette
Objects
Argument
ArgumentReas...
CitiationElement
Claim
EvidenceAssert...
InformationEle...
Connections
AssertedContext
AssertedEviden...
AssertedInfere...
Outline
Task List

Figure 14: ACedit screenshot [1]

An assurance case exists to communicate an argument, supported by evidence, about the assurance one can have on the operation of a system. In order to represent the case, a means of capturing the underlying reasoning is required. GSN and ARM are two approaches used to represent the assurance case. ACedit implements and visualises an EMF version of the specification of the two standards as a GMF editor. ACedit offers model management infrastructure, and includes a number of rules often used by assurance case users including validation and transformation between the two standars [11].

3.3.3 D-Case Editor

D-case Editor is an safety cases editor being developed in DEOS (Dependable Embedded Operating System for Practical Uses) project funded by Japan Science and Technology Agency [36]. It is a prototype implementation of a dependability case editor, which has a pattern selection function. The D stands for dependability. The tool is based on the Eclipse GMF framework [15]. Its characteristics are

1. supporting GSN (Goal Structuring Notation)
2. GSN pattern library function and prototype type checking function
3. consistency checking function by an advanced proof assistant tool

Since it has been implemented on Eclipse, it is interesting to make a tool chain with existing development tools of Eclipse. To make assurance case more familiar to developers who are using open sources tools, the developers of D-case Editor released it as an open source safety case editor implemented on Eclipse GMF [37], since ecplise a widely used developing tool and also available as open source. The figure below shows a screenshot of D-Case Editor in use [36].

The figure below shows an example of the dependabiliy case patterns are included in D Case Editor [35].

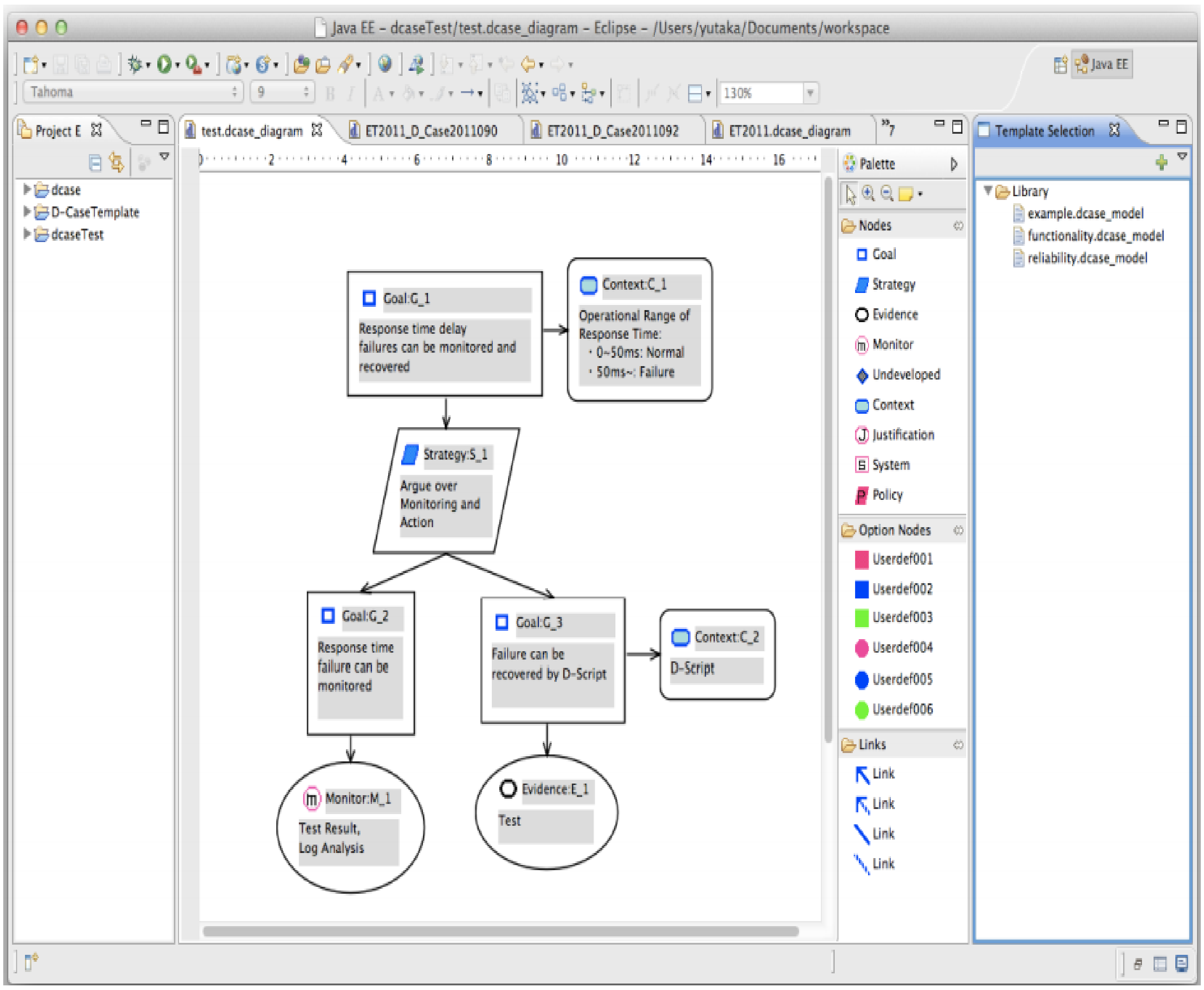


Figure 15: Screenshot of D-case Editor

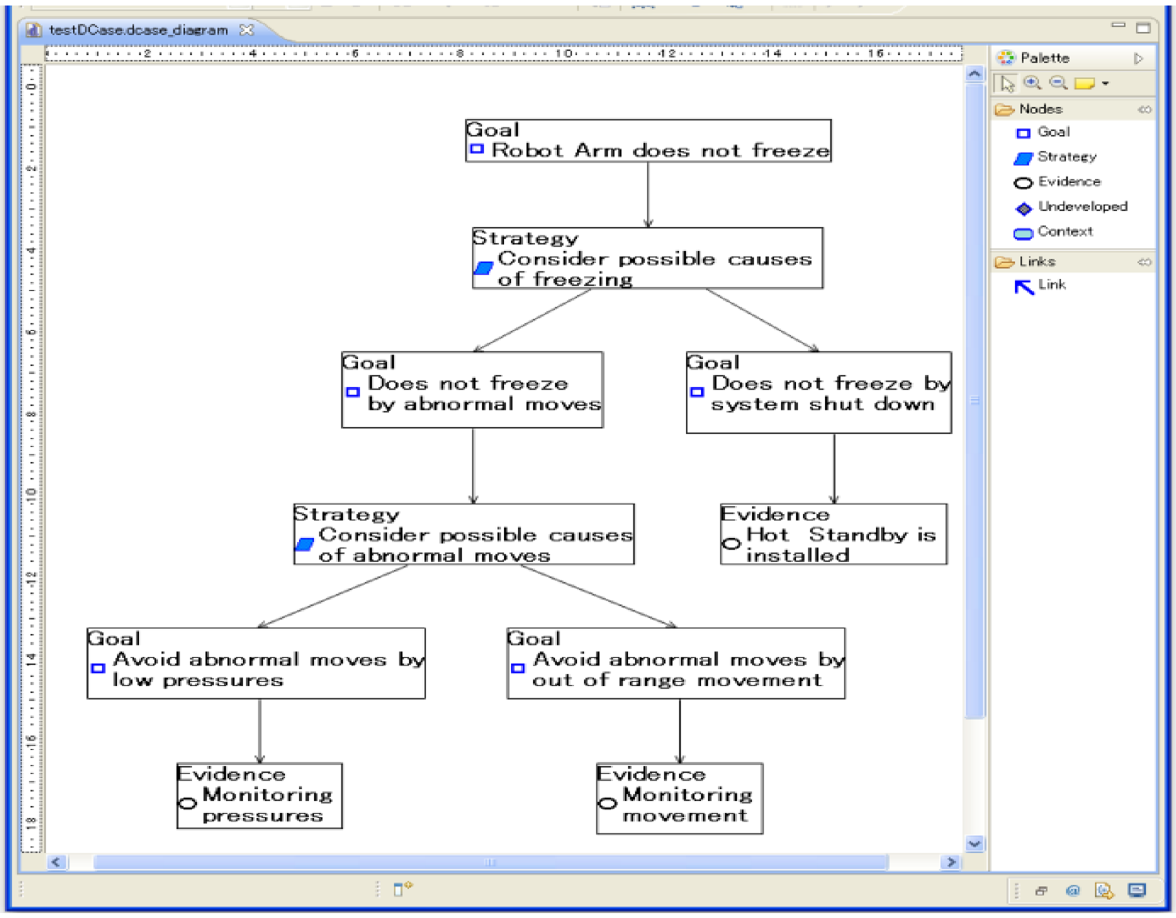

testDCase.dcase_diagram
Goal
Robot Arm does not freeze
Strategy
Consider possible causes of freezing
Goal
Does not freeze by abnormal moves
Goal
Does not freeze by system shut down
Strategy
Consider possible causes of abnormal moves
Evidence
Hot Standby is installed
Goal
Avoid abnormal moves by low pressures
Goal
Avoid abnormal moves by out of range movement
Evidence
Monitoring pressures
Evidence
Monitoring movement
Palette
Nodes
Goal
Strategy
Evidence
Undeveloped
Context
Links
Link

Figure 16: Example of patterns in D-case Editor

4 Building a safety case

4.1 Introduction

In this chapter, I will present a case study based on the development of a medical equipment without a safety case, and then I will suggest a top level, goal based structured safety case based on this case study.

4.2 Case Study - The Therac 25 a case study in safety failure

4.2.1 Introduction

Computers are increasingly being introduced into safety critical systems and, as a consequence, have been involved in accidents. Between June 1985 and January 1987, a computer controlled radiation therapy machine, called the Therac 25, massively overdosed six people. These accidents have been described as the worst in the 35 year history of medical accelerators [34].

4.2.2 The Machine

Medical linear accelerators (linacs) accelerate electrons to create high energy beams that can destroy tumors with minimal impact on the surrounding healthy tissue. Relatively shallow tissue is treated with the accelerated electrons; to reach deeper tissue, the electron beam is converted into X-ray photons. Using this double pass concept, AECL designed the Therac 25, a dual mode linear accelerator that could deliver either photons at 25 MeV or electrons at various energy levels. Compared with the Therac 20, the Therac 25 was notably more compact, more versatile, and arguably easier to use.

Several features of the Therac 25 are important in understanding the accidents. First, like the Therac 6 and the Therac 20, the Therac 25 was controlled by a PDP. However, AECL designed the Therac 25 to take advantage of computer control from the outset; AECL did not build on a stand alone machine. The Therac 6 and Therac 20 had been designed around machines that already had histories of clinical use without computer control.

In addition, the Therac 25 software had more responsibility for maintaining safety than the software in the previous machines. The Therac 20 had independent protective circuits for monitoring electron beam scanning, plus mechanical interlocks for policing the machine and ensuring safe operation. The Therac 25 relied more on software for these functions. AECL took advantage of the computer's abilities to control and monitor the hardware and decided not to duplicate all the existing hardware safety mechanisms and interlocks. This approach is becoming more common as companies decide that hardware interlocks and backups are not worth the expense, or they put more faith (perhaps misplaced) on software than on hardware reliability.

Finally, some software for the machines was interrelated or reused. In a letter to a Therac 25 user, the AECL quality assurance manager said, "The same Therac 6 package was used by the AECL software people when they started the Therac 25 software. The Therac 20 and Therac 25 software programs were done independently, starting from a common base." Reuse of Therac 6 design features or modules may explain some of the problematic aspects of the Therac 25 software. After a bug related to one of the Therac 25 accidents was found in the Therac 20 software, it was discovered that some Therac 20 routines were also used in the Therac 25.

The control console and printer etc. were all located outside the heavily shielded treatment room. Thus, when pressing the key to begin the treatment, the operator did not have any direct access to the machine or the patient. All the occurrences in the treatment room had to be observed through the TV monitor and the intercom. The figure below shows the facilities of Therac 25.

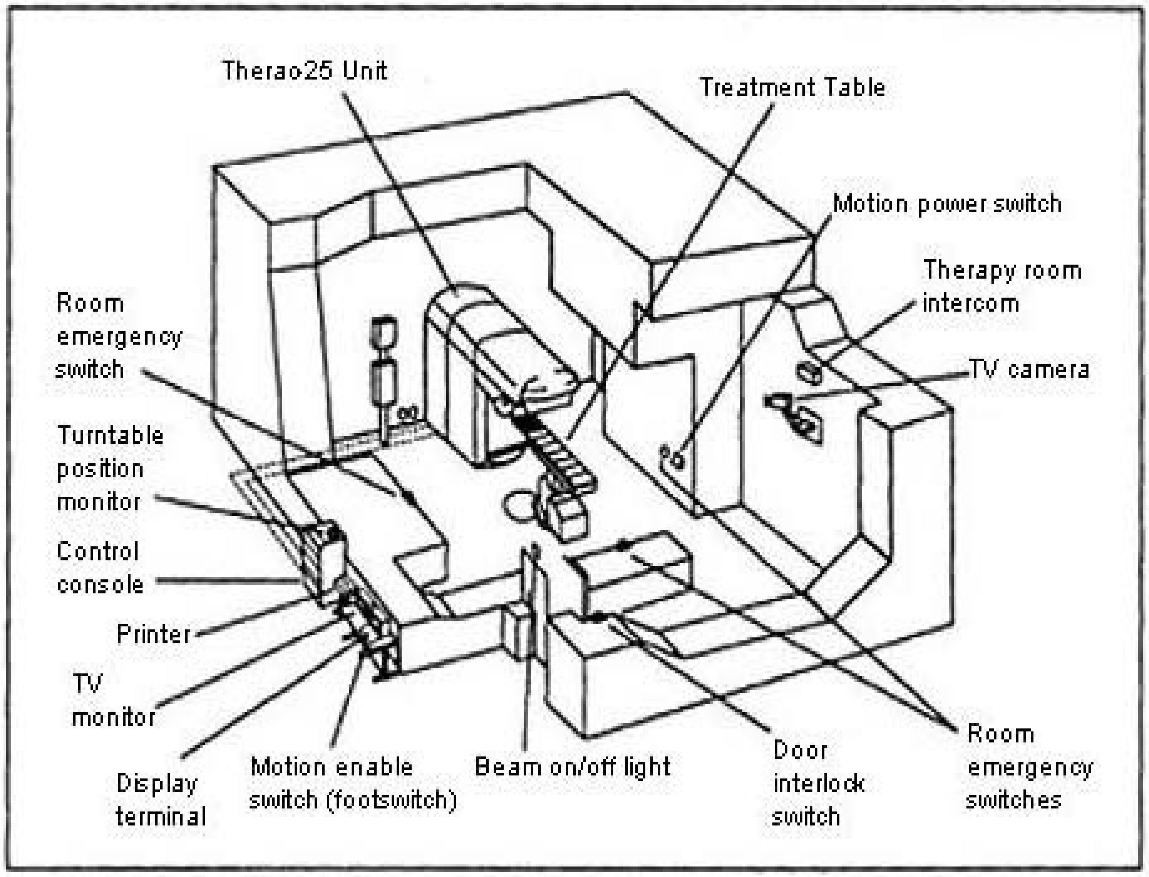


Figure 17: Therac facility [32]

4.2.3 The Turntable

Therac 25 is a dual mode machine. This means that it can treat the patient with relatively low energy electron beams or with X-ray beams. In addition, Therac 25 had a "field light" position that allowed a standard light beam to shine in the path of treatment to help the operator in setting up the machine. Thus there were three modes in which the Therac 25 could operate: electron beam and X-ray for treatment, and field light for setup.

Even though they are relatively low energy, the electron beams are too powerful in their raw form to treat the patient. They need to be spread thinly enough to be the right level of energy. To do this, Therac 25 placed what are called scanning magnets in the way of the beam. The spread of the beam (and also its power) could be controlled by the magnetic fields generated by these magnets. Thus for electron beam therapy, the scanning magnets needed to be placed in the path of the beam.

X-ray treatment requires a very high intensity electron beam (25 MeV) to strike a metal foil. The foil then emits X-rays (photons). This X-ray beam is then "flattened" by a device below the foil, and the X-ray beam of an appropriate intensity is then directed to the patient. Thus, X-ray therapy requires the foil and the flattener to be placed in the path of the electron beam. The figure below shows the turntable og Therac 25.

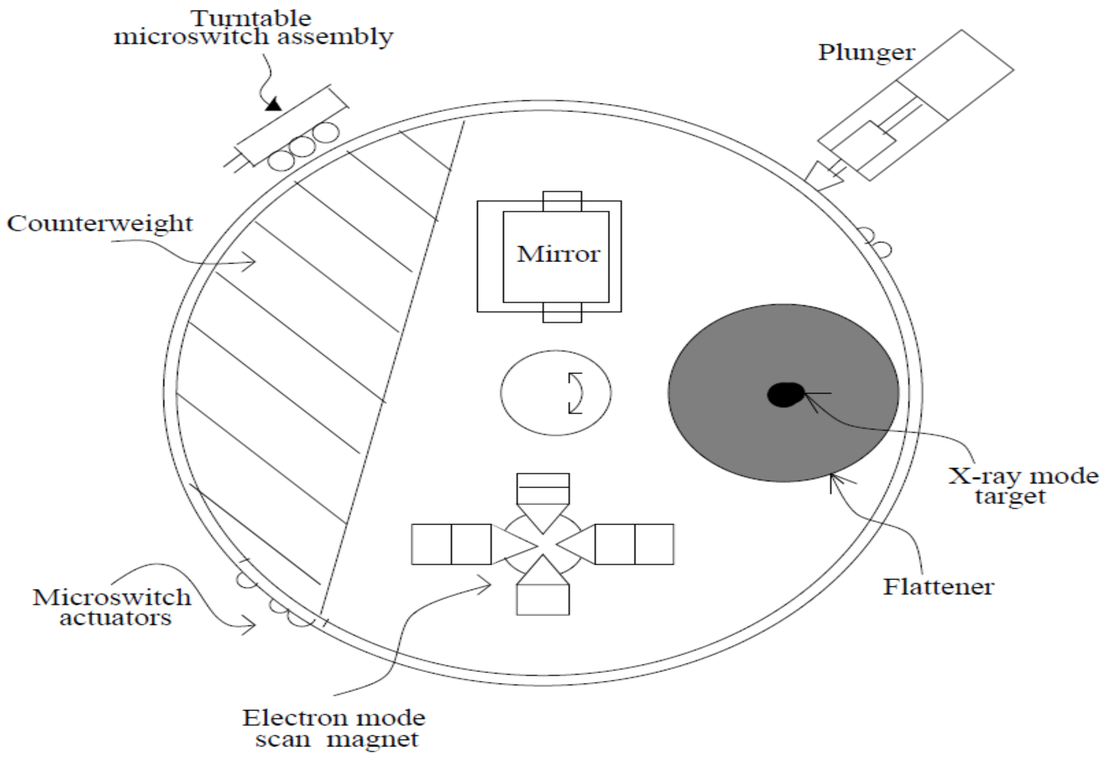


Figure 18: Turntable [34]

4.2.4 Use of Therac 25

Delivering a treatment was very time consuming. When a patient was put on the treatment table, the practionare would set the field size, gantry rotation and attach accessories to the machine. Then they would leave the patient in the room, and enter patient id, prescription, field size, gantry rotation and accessory info from outside the room. If alle the information matched, the machine would verify and indicate that the treatment could proceed. It was very common that it would not verify due to som small errors in the information, when this happened, they would have to go through the whole process again. Because of this, the developers implemented a feature which allowed ”enter” to be used to keep an existing entry unchanged.

During a treatment, there were two ways to stop the machine

- Treatment suspend (required a complete machine reset to start again)
- Treatment pause (required a single keystroke to resume treatment)

The usefulness of error messages were not a high priority, and therefore difficult to understand. Examples of an error message could be ”MALFUNCTION 47” or ”VILT”. Some of these could occur as much as 40 times per day, and often they did not involve in the patient safety. This resulted in the fact that the practioners did not pay much attention to them, and assumed the treatment. After several deaths attempts were made to locate the problems. It turned out that the error message ”MALFUNCTION 54” was an error which had existed in the software in Therac 20, but due to the hardware interlocks it had not resulted in any deaths, and therefore not been discovered.

4.2.5 Case Evaluation

In March 1983, AECL performed a safety analysis on the Therac 25. This analysis was in the form of a fault tree and apparently excluded the software. According to the final report, the analysis made several assumptions:

(1) Programming errors have been reduced by extensive testing on a hardware simulator and under field conditions on teletherapy units. Any residual software errors were not included in the analysis.

(2) Program software did not degrade due to wear, fatigue, or reproduction process.

(3) Computer execution errors were caused by faulty hardware components and by ”soft” (random) errors induced by alpha particles and electromagnetic noise.

The fault tree resulting from this analysis appeared to include computer failure, although apparently, judging from these assumptions, it only considered hardware failures.

Eleven Therac 25s were installed: five in the US and six in Canada. Six accidents involving massive overdoses to patients occurred between 1985 and 1987. The machine was recalled in 1987 for extensive design changes, including hardware safeguards against software errors. Related problems were found in the Therac 20 software, but these were not recognized until after the Therac 25 accidents because the Therac 20 included hardware safety interlocks and thus no injuries resulted [34].

4.2.6 Case Conclusion

The people who developed the Therac 25 relied too much on the technology from the two previous machines, without considering the changes that were made and the new technology which were introduced in the new machine. The process of developing the new machine did not go through a thorough development project, with planning, risk assessment, safety case construction and so on. It seems that they did not even consider "side effects" by chaning the basic formula. This goes back to Chapter 3.4 and the importance of experience and understanding, and being able to use that in new projects at same time tranforming it to fit the new. The two older machines had several hardware safety stops, to prevent some accidents to happen, but these were removed in the Therac 25 because it was believed that the software was better. Without even testing the new software, which was believed to be better, seems to be just plain foolish. This was not very safe, especially considering that it was a safety critical system, which proved to have catastrophic outcomes.

4.3 Building a safety case

Based on the case study about Therac 25, I will propose a way to structure a safety case using Goal Structuring Notation and Adelards Safety Case tool, ASCE. But first there are some key elements to sum up with this case study. There were a lot of mistakes done when developing and implementing this new system. First of all, a risk assessment was performed, but it did not consider the software. Secondly, the unquestionable reuse of the software from the previous machine without knowing if it had any faults. Thirdly, basically making a very cumbersome and hard to use system.

The most important system safety requirements that I can gather, without having done a risk analysis or hazard identification, are

- The error messages needs to be more specific and informative
- There should be a maximum limit to MeV electron treatments, 200 rad for example
- The system should not be able to resume after an error occurs, when something is definately wrong (will make it easier to understand with more explanatory error messages)
- the computer should not be able to give wrong energy
- the computer should not be able to select wrong modes

Without knowing this system and its requirements down to every detail, I have tried to make a top level safety case, using ASCE and its Goal Structuring Notation mode. The figure below shows a screenshot of this.

Naturally, a hazard identification and a risk analysis should have been performed prior to making a saftey case, so this proposal is not a real comprehensive solution, but it is a start and it tacles some of the major faults of the system. These are the requirements I presented above.

The layered structure of the safety case allows the safety case to evolve over time and helps to establish the safety requirements at each level. For large projects with sub-contractors, this "top-down" safety case approach helps to identify the subsystem requirements and the subsystem safety case can be made an explicit contractual requirement to be delivered by the sub-contractor [3]. This could definately apply to the Therac 25 case, if they had used safety cases when developing the system, it might have helped them discovering several of the faults

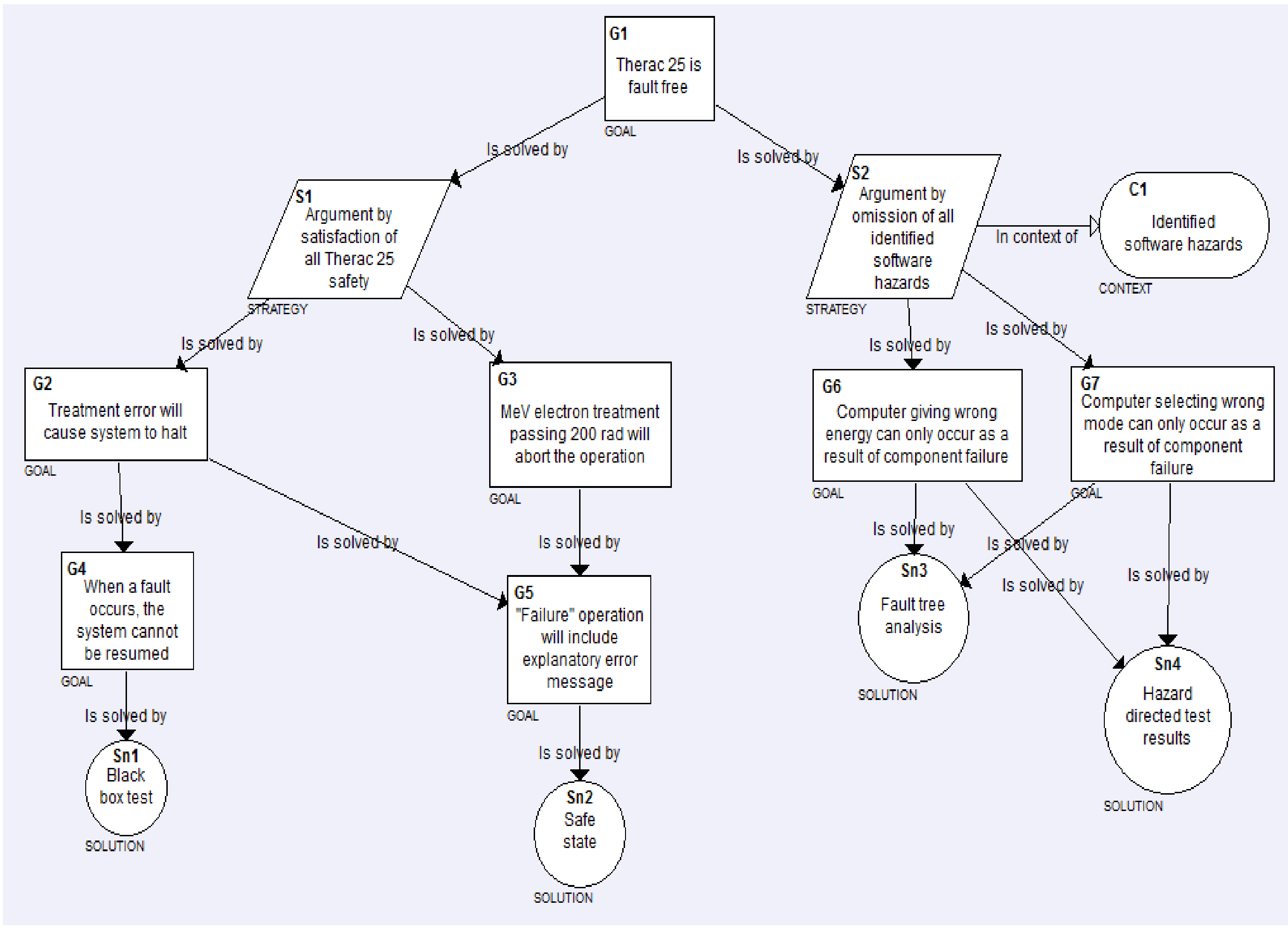

G1
Therac 25 is fault free
GOAL
Is solved by
S1
Argument by satisfaction of all Therac 25 safety
STRATEGY
Is solved by
S2
Argument by omission of all identified software hazards
STRATEGY
In context of
C1
Identified software hazards
CONTEXT
Is solved by
G2
Treatment error will cause system to halt
GOAL
Is solved by
G3
MeV electron treatment passing 200 rad will abort the operation
GOAL
Is solved by
G6
Computer giving wrong energy can only occur as a result of component failure
GOAL
Is solved by
G7
Computer selecting wrong mode can only occur as a result of component failure
GOAL
Is solved by
G4
When a fault occurs, the system cannot be resumed
GOAL
Is solved by
G5
"Failure" operation will include explanatory error message
GOAL
Is solved by
Sn3
Fault tree analysis
SOLUTION
Is solved by
Sn4
Hazard directed test results
SOLUTION
Is solved by
Sn1
Black box test
SOLUTION
Is solved by
Sn2
Safe state
SOLUTION

Figure 19: Therac 25 safety case

and maybe even saved til lifes og the people who were effected by their mistakes. We will never know.

5 Discussion

5.1 Introduction

In this chapter I will discuss the different safety case tools presented in chapter 3 and also explain why I chose the ASCE tool with Goal Structuring Notation to structure the safety case in chapter 4. I will also discuss the use of safety cases in general.

5.2 The safety case tools

5.2.1 ASCE

Adelards safety case tool ASCE [3] is a very suitable tool for building safety cases because it has been developed from years of experience in the businesss. Adelard is a very well adjustet company, and seem to be one of the biggest when it comes to safety cases. This of course gives them an advantage because their reputation and establishment leads a lot of customers to them, I would imagen. During the work of this master thesis, I always fond myself ending up back at Adelard in some way or another when doing research. They are connected to an incredible amount of work related to safety cases and Goal Structuring Notation and much more.

Asce was pretty easy to understand and navigate around, but they offer support and couring should it be necessary. In regards to major safety critical systems, taking a course is probably a good idea to be able to get all the benefits from using the programme in the best possible way. It does cost a considerable amount, at least for one person, but as long as it is for a company with several employees, it is a tolerable amount. Also, if everything goes after plan, they might end up saving money, by using the safety case to its full extent, and thereby developing good and reasonble safe system without having to discover major lackings at the end of the development process.

Another advantage to ASCE is that it is constantly being modified to follow the developments in research and marked. Since Adelard is a private company, earning money on ASCE, this should naturally be expected.

Since ASCE is a complete program on its own, it was very easy to start using it, and I did not encounter any problems when starting to use it. There only thing I had to was to download it and it was ready to use. This was not the only reason I chose to use ASCE over the two other tools. Adelard is such a big contributor in this area, therefore I wanted to try out their product for myself to see if it held up to their standards. In my opinion, it did, but I was lucky and got to try out their trial version, otherwise I would not have been able to afford it.

5.2.2 Acedit

ACedit is a open source plugin for Ecplise, which makes it acsessible for everyone. Eclipse is also very applicable, since it is used world wide by people in the development business environment. The disadvantage about the tool is that it was a result from a postgraduate project, and it seems to have stayed still since then (June 2012). Now, that is only one year ago, but a year can be a long time in computer and software world. Already there is a new version of Eclipse out there, and an often occuring problem with eclipse, is that plugins developed for one version, does noe always apply to the new version. I experienced this my self with ACedit. It is of course possible to download the older version, but it is not a good solution in the long run.

Once everything was in place, which took sometime because there were limited resources with information explaining all the steps, the tool worked fine. It was easy to create a goal based safety case with GSN. It has potential, and it is definitely a sought after tool.

5.2.3 D-Case Editor

As with ACedit, D-Case Editor is also made as a plugin for Eclipse, so several of the points made in the section above, applies with this tool as well. D-Case Editor actually uses an even older version of Eclipse (Galileo), which originally only was meant for 32-bit operating system. Today most computers have 64-bit, soon they might have an even higher option. This made it more challenging to be able to get the tool up and running, and since this also is an old tool, there was not much information about the problems I was facing to get a hold of.

Matsuno, at the University of Tokyo where D-Case Editor was created, claims in 2011 that there will be more features in the next release. Including OMGs ARM among other things [36]. In March 2012 it is announced to be a alpha version, and I can not seem to find anything more recent. It does not, of course, mean that it is not being developed any further, but seeing as it is a university project, it is being developed in the name of research and not economics. The strive for economic rewards is propably not as motivating as for a company meaning to sell the finished product. That being said, like ACedit, once it was properly installed, it was a good tool. The structure of it was similar to ACedit, as one can see from the screenshots in chapter 3.

5.2.4 Potential Limitations of Safety Cases

During the research for this thesis, I have not across a lot og negative talk about safety cases, except for Leveson [33] who questions this and the real safety of safety cases. She argues that although safety cases are often defined as "to denote an argument that the system will be acceptably safe in a given operating context". The problem is that it is always possible to find or produce evidence that something is safe. Unlike proving a theorem using mathematics (where the system is essentially complete and closed, based on definitions, theorems and axioms and nothing else), a safety analysis is performed on an engineered and often social system where there is no complete mathematical theory to base arguments and guarantee completeness.

Further, she argues that no system is completely safe, so the goal of the argument is untrue before starting. If instead the goal is to show the system is acceptably safe, the problem simply devolves to defining what is acceptable and to whom: to the producer of the system who is paying the cost of making it safe or to the potential victim? The main problem with the use of arguments for safety or acceptable safety in the safety case approach to certification lies in psychology and the notion of a mindset or frame of reference.

She brings up an important point, I think, by presenting that an important component of mindset is the concept of confirmation bias. Confirmation bias is a tendency for people to favor information that confirms their preconceptions or hypotheses regardless of whether the information is true. People will focus on and interpret evidence in a way that confirms the goal they have set for themselves. If the goal is to prove the system is safe, they will focus on the evidence that shows it is safe and create an argument for safety. If the goal is to show the system is unsafe, the evidence used and the interpretation of available evidence will be quite different. People also tend to interpret ambiguous evidence as supporting their existing position [33]. This is an important aspect to consider when working with safety cases.

The scope of the safety critical system concept is broad, and that breadth has to be taken into account when practitioners and researchers deal with specific systems. A closer examination of the topic reveals that many new types of system have the potential for very high consequences of failure, and these systems should probably be considered safety critical also. It is obvious that the loss of a commercial aircraft will probably kill people. It is not obvious that loss of a telephone system could kill people. But a protracted loss of 911 service will certainly result in serious injury or death[31]. There are probably several industries which include safety critical systems without them focusing on safety cases and standards in regards to it. But that does not mean that most industries should blindly embrace safety cases. It has been made clear that while safety cases can be a helpful tool when developing safety critical systems, that does not mean it automatically is the best solution. It is important to learn how to use it properly to reap the benefits, otherwise it will probably end up costing more money and time than it was intended to save (apart from human lifes of course).

[28] Asks: *What constitutes a good safety case?* Which in my mind is a valid question. It is difficult to meassure how good a document is, but it is possible to structure it better. Using goal structuring notation it can at least be easier to read it and get the main idea of it, which of course is the point. When used correctly, the idea if goal structuring notation seemd to work really well, and being presented with clear goals instead of having to look for them in a long text must nor only be easier, but also save time and maybe even money too.

There are several good points to using safety cases when developing safety critical systems, especially in combination with goal structuring notation, but there should also be focus on training when it comes to using it. There might also be a need for more research in the effects of the use of them. Leveson argues that careful evaluation and comparison between certification approaches has not been done and that most papers about safety cases express personal opinions or deal with how to prepare a safety case, but do not provide data on whether it is effective. As a result, there is no real evidence that one type of regulation or certification is better than another.

6 Conclusion

In this report I have looked at the safety case, different tools and methods to be able to build one and I have used ASCE and goal structuring notation to try and structure one myself.

Many modern systems depend on computers for their correct operation. Of greatest concern, of course, are safety-critical systems because their consequences of failure can be considerable. There are many applications that have traditionally been considered safety critical but the scope of the definition has to be expanded as computer systems continue to be introduced into many areas that affect our lives. The future is likely to increase dramatically the number of computer systems that we consider to be safety critical. The dropping cost of hardware, the improvement in hardware quality, and other technological developments ensure that new applications will be sought in many domains [31].

Security is becoming an increasingly important topic in the field of safety critical systems, and it must be addressed comprehensively if safety critical systems are to be operated successfully. The challenge here lies very much in the field of software engineering rather than security technology. The vast majority of security problems that arise in networked information systems arise because software defects make the systems vulnerable to attack [31].

There are definately several safety case tool out there, and I have only looked closer at three of them. In my experience, there seem to be a bit more challenges with ones that are free of charge and open source, but they all give pretty much the same results in the end.

My main intereset with safety cases has been in regards to the healthcare industry. It covers a great amount of safety critical systems allready and a potentially great deal more safety critical systems. This industry in Norway stands before a great upgrade need in many of the sectors. Big projects have already been announced, especially for some of the big hospitals, and the results rom them remains to be seen. According to my research in this thesis, the use of safety cases within the healthcare industries are starting to be more common in England, but here the use of safety cases in other major industries such as railway and military, and therefore the step over to healthcare might not be that big. Wether it is a good solution for the Norwegian healthcare industry, I am not sure of as of yet, but it could certainly be worth a try.

The Therac 25 case shows how the development of a safety critical system can go horribly wrong. This was over 20 years ago, but many of the issues are still highly relevant today. Looking back, it is always easy to have good hindsight and say that would never happen today. It is also impossible to say if the use of safety cases back then would have improved the quality of the system and the outcome. Non the least, it is clear that there will be more and more technology in our lives, and therefore probably also more and more safety critical systems. The need for safety cases, or at least something that will help develop safe systems, is definately there. When used correctly, safety cases can seem to be a helpful guide in the development process.

6.1 Further work

Further work on this topic should be to examine the healthcare industry in Norway more closely to see if safety cases could be something worth using. Interviews with leading companies who are developing safety critical systems is an important aspect that I did not have time to conduct. I t would be interesting to get their oppinions on the use of safety cases, and if they have considered making use of them.

There might be neccessary to take a bit more critical look upon the safety case as a whole, what are the alternatives to safety cases, how do they work? Have there been done any research as to the effects of the use of safety cases opposed to not using them? I would like to see more research in that direction

References

[1] ACedit tool website. Available: https://code.google.com/p/acedit/, 2013

[2] F. Altheide, S. Drfel, H. Drr and J. Kanzleiter. An Architecture for a Sustainable Tool Integration.Workshop on Tool Integration in System Development, pp. 2932, Helsinki, Finland, September 2003.

[3] Adelardl website. Available: http://www.adelard.com , 2013

[4] A. Ayoub, B. Kim, I. Lee and O. Sokolsky, "A Safety Case Pattern for Model-Based Development Approach", The 4th NASA FORMAL METHODS SYMPOSIUM (NFM2012), Virginia, USA, April 3-5, 2012.

[5] BIBSYS website. Available: https://ask.bibsys.no/ask2/html/ , 2013

[6] P. Bishop and R. Bloomfield, "A Methodology for Safety Case Development", Adelard, 1998

[7] P. Bishop, R. Bloomfield and P. Froome, "Justifying the use of software of uncertain pedigree (SOUP) in safety related applications", Adelard, 2001

[8] R. Bloomfield and P. Bishop, "Safety and Assurance Cases: Past, Present and Possible Future an Adelard Perspective", Adelard LLP, 2010

[9] O. Bridal and others. Deliverable D3.1 Part 1 Appendix E: Safety Case, Version 1.1. Technical Report, EASIS Consortium (www.easis-online.org), February 2006.

[10] T. Cockram and B. Lockwood, "Electronic Safety Case: Challenges and Opportunities", Praxis Critical Systems and Raytheon Systems Limited, 2003

[11] G. Despotou, A. Apostolakis and D. Kolovos, "Assuring Dependable and Critical Systems: Implementing the standards for Assurance Cases with ACedit", Department of computer Science University of York, UK, 2012

[12] G. Despotou, T. Kelly, S. White and M. Ryan, "Introducing Safety Cases for Health IT", University of York, Connecting for Health (NHS-CfH) and Rotheram NHS Foundation Trust, 2012

[13] D-Case Editor installation process, DEOSC, 2011

[14] W. D. Cullen, "The Public Enquiry into the Piper Alpha Disaster", Department of Energy, London, HMSO, 1990

[15] Eclipse foundation, EuGENia GMF Tutorial, http://www.eclipse.org/epsilon/doc/articles/eugenia-gmf-tutorial/

[16] C. Edwards, "Railway Safety Cases", Springer London, 1995

[17] S. Elgfoss, A. Aas, T. Skramstad, V. Nicolaysen and G. Merkesvik, "Dependability qualification of complex software-intensive architectures - prescriptive or goal based use of safety standards", DNV, NTNU, FMC, 2010

[18] M. Ellims, "Saefty Analysis: Thoughts on Methods and Experience", Pi-Shurlock, Cambridge, UK, 2008

[19] E. Gamma, R. Helm, R. Johnson and J. Vlissides, ”Design Patterns: Elements of Reusable Object-Oriented Software”, Addison-Wesley, 1995

[20] Google website, Available: http://google.com , 2013

[21] The Google Scholar website, Available: http://scholar.google.no/ , 2013

[22] GSN Working Group, GSN Community Standard v.1., http://www.goalstructuringnotation.info/archives/76, November 2011

[23] R. Hawkins, ”Overview of the Tiered Safety Argument Patterns”, The GSN Working Group Online, 2009

[24] R. Hawkins, T. Kelly, J. Knight and P. Graydon ”A New Approach to Creating Clear Safety Arguments”, Department of Computer Science, The University of York, Department of Computer Science, University of Virginia, 2011

[25] IEC, ”IEC 61508 Functional safety of electrical /electronic/programmable electronic safety-related systems”, 1998

[26] Automotive Standards Committee of the German Institute for Standardization. ”ISO/WD 26262: Road Vehicles Functional Safety. Preparatory Working Draft”, Technical Report, October 2005.

[27] T. Kelly, Arguing Safety A Systematic Approach to Managing Safety Cases, PhD Thesis, Department of Computer Science, The University of York, 1998.

[28] T. Kelly, ”A Systematic Approach to Safety Case Management”, SAE International, 2003

[29] T. Kelly and J. McDermid, ”Safety Case Construction and Reuse using Patterns”, University of York, 1997

[30] T. Kelly and R. Weaver, ”The Goal Structuring Notation - A Safety Argument Notation”, Department of Computer Science and Department of Management Studies, University of York, 2004

[31] J. Knight, ”Safety Critical Systems: Challenges and Directions”, Department of Computer Science, University of Virginia, 2002

[32] N. Leveson, ”Safeware: System safety and Computers”, Addison-Wesley, 1995

[33] N. Leveson, ”White Paper on the Use of Safety Cases in Certification and Regulation”, Massachusetts Institute of Technology, 2011

[34] N. Leveson and C. Turner, ”The Investigation of the Therac-25 Accidents”, Computer, 1993

[35] Y. Matsuno, H. Takamura, Y. Ishikawa, ”Dependability Case editor with Pattern Library”, University of Tokyo, 2010

[36] Y. Matsuno, ”D-Case Editor: A Typed Assurance Case Editor”, University of Tokyo, 2011

[37] Y. Matsuno and K. Taguchi, ”Parameterised Argument Structure for GSN Patterns”, University of Tokyo and Industrial Science and Technology, Japan, 2011

[38] Object Management Group (OMG), Argumentation Metamodel (ARM) (FTF - Beta 1), http://www.omg.org/spec/ARM, August 2010

[39] J. Penny, A. Eaton, PG. Bishop and RE. Bloomfield, "The Practicalities of Goal-Based Safety Regulation", SRG and Adelard, 2001

[40] The International Electrotechnical Commission (IEC) , " Functional Safety and IEC 61508: A basic guide", Projex Solutions Limited, 2002

[41] W. Ridderhof, HG. Gross and H. Doerr, "Establishing Evidence for safety Cases in Automotive Systems - A Case Study", Software Engineering Research Group, 2007

[42] R. Weaver, "The Safety of Software - Constructing and Assuring Arguments", PhD Thesis, University of York, 2003