Identifying Effective Improvements to Software Safety Practice

Matthew Steven Osborne

University of York

Department of Computer Science

Doctor of Philosophy

November 2024

Abstract

Good safety management means that continuous attempts are made to improve all aspects of safety engineering practice. This includes the work required to assure the safe contribution of software to acceptable system safety (software safety practice). These improvements are often through creating interventions to perceived problems with software safety practice.

Historically, improvements to software safety practice have resulted in interventions which seem to have been largely ineffective. This suggests that they may not be addressing the real impediments to good software safety practice. It is not argued that existing tools for improving software safety practice are necessarily deficient, rather that the notion of whether they are being employed to correct the actual causes of impediments to better practice is challenged.

Software safety practice 'As Observed' (the actual software safety engineering activities performed) is informed by defined processes (software safety practice 'As Required'). These processes aim to ensure practice achieves the best safety outcomes (software safety practice 'As Desired'). For many different and complex reasons 'As Observed' software safety practice may not be equivalent to software safety practice 'As Required'. Similarly, software safety practice 'As Required' may not be equivalent to software safety practice 'As Desired'. Any, or all of these discrepancies could play a significant role in poor software safety practice. By exploring these discrepancies it becomes possible to understand the causes of deficiencies in practice, and to start to propose effective interventions.

This thesis defines a framework and process for understanding and assessing software safety practice based around modelling software safety practice 'As Desired', 'As Required', and 'As Observed', and the interactions between these elements.

The process is defined, described, instantiated and evaluated. Use of this framework and process for understanding software safety practice is an effective means by which an organization can identify currently existing impediments to the achievement of software safety best practice.

Acknowledgements

Sam, you are my rock. You are the reason I ‘do what I do’. Without you, this thesis would never have been written. Thank you and our wonderful sons - Lewis and Samuel - for supporting me through thick and thin (my thanks must also go to the wonderful NHS for the ’thin bit’, too).

Richard (Hawkins) and Mark (Nicholson); thank you for your advice, guidance, challenge, and friendship. Your unwavering support has been incalculable over the years.

My thanks to Rob (Alexander) for becoming my mentor on academic writing. I’m not sure I ever asked you officially, but it is a role you excelled in regardless. I hope I remembered at least some of your guidance.

Finally my thanks to friends and colleagues in the Assuring Autonomy International Programme and Centre for Assuring Autonomy. Your supportive nature is limitless, and your intelligence unbounded. Special thanks must go to Philippa, Katrina, Ana, Chrysle, Dawn, and John (you know why).

Declaration

I declare that this thesis is a presentation of original work and I am the sole author. This work has not previously been presented for an award at this, or any other, University. All sources are acknowledged as References.

Some of the material presented in this thesis has previously been published in the following papers:

- Matt Osborne, Mark Nicholson, and Richard Hawkins. Empirical evaluation of the impediments to an "as desired" model of software safety assurance. *Systems and Covid-19.* 2021.
- Matt Osborne and Mark Nicholson. Skills for assuring the safe adoption of emerging technology. 2023.
- Matt Osborne, Richard Hawkins, Mark Nicholson, and Rob Alexander. Understanding safety engineering practice: Comparing safety engineering practice as desired, as required, and as observed. *Safety science*, 172:106424, 2024.

All of the work contained within this thesis represents the original contribution of the author.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Space

Many safety engineers, safety engineering consultants, safety managers, and academics have suggested that current software safety practice observed in organizations undertaking software projects[1] rarely achieve the best safety outcomes. It should be our collective ambition to try to establish whether (and why) this is true, and if so to recover the gaps between the software safety practice carried out and good software safety practice.

A variety of stakeholders, such as companies, organizations, standards committees, and researchers, have tried to implement changes to software safety practice over the last few decades to 'fix' issues. Examples of changes to software safety practice range from those proposing a change to the software safety lifecycle (e.g. Safe Scrum [153] or spiral software development [10]); using 'Duration Calculus' to improve software safety requirements [45]; attempts to reduce the errors in software safety requirements [89]; using Fault Tree Analysis to automatically derive software safety requirements [99]; and the proposed use of patterns [4] and new models for development [131].

These fixes seem to have been largely ineffective(despite the expenditure - and assuming that the fixes have been adopted) and evidence (including anecdotal evidence) suggests that problems still persist (see [94], [83], [47], [56], [101], [74], for example). Often, proposed 'fixes' offer nothing more than a new variant of existing analysis methods which are only evaluated in terms of how they *could* have prevented a notable accident/incident (decried as YAAPing by Rae et al in [135]).

Before researchers, practitioners, or managers embark on a program of software safety practice improvement they must be as confident as possible that they

[1] A Project can emanate from any industry and application, and its level of design abstraction can range from a full product, through systems, items, or down to the hardware and software levels. A Project may involve one or more organizations.

have both identified where any issues lie, and that they are addressing the complete set of issues. Moreover, they must be confident that they are addressing appropriately-distal causes of issues rather than only proximal causes or indeed just symptoms.

Without empirical evidence, one cannot be sure where any issues with software safety practice lie. To understand the current state of software safety practice, we argue that one must first identify and understand the elements that constitute software safety practice, and the relationships between these individual elements. This thesis discusses what these elements are later.

To understand software safety practice fully, an analyst needs a process which can be used to understand and assess software safety practice in a robust manner that allows the different elements of practice to be represented effectively and consistently. This thesis develops a framework which is capable of linking the different elements together, and a process for evaluating the relationships between the different elements, and ways of revealing and presenting any nuances and subtleties therein.

1.2 Thesis Scope

The research is not focussed on the effectiveness, nor currency of a specific safety engineering technique (such as the work of Leveson and Thomas [81], [82] to improve the status quo of safety engineering), but is concerned with the phenomenon [164] of software safety practice in safety critical systems. We describe ′practice′ as the set of processes which describe the activities carried out. The software safety process is merely a subset of the system safety process [79], and as such, the term ‘software safety’ is used to describe the contribution of software to safety in its system context [95]. ‘Software safety practice’ means the activities carried out to assure and demonstrate the safe contribution of software to a system. A ‘system’ is a combination of interacting elements organized to achieve one or more stated purposes [66]. This software safety practice is a socio-technical endeavour [154], and is predominantly the work carried out by software safety engineers or software engineers with responsibility for system safety; but also involves many other engineering disciplines (e.g. hardware, systems, reliability, and requirements engineers), and other specializations such as programme and project managers, and those within commercial, contractual, and legal services. For brevity and clarity, the multi-disciplinary professionals working on the activities which constitute software safety practice are referred to as ‘software safety practitioners’.

The research presented in this thesis seeks to answer the question of how an organization can make effective improvements to its software safety practice. To make effective improvements, an organization will first need to understand its software safety practice. By ‘understand’ we mean identify the elements of the organization’s project’s software safety practices, and determine their interrelationships. Once an organization understands its software safety practice, it can then assess it. By 'assess' we mean identify the relationships between the elements and determine the effectiveness of those relationships (including against appropriate measures of 'good' practice).

Practice of any kind is impacted by the culture of the organization. Organizational culture [20], of which safety culture is a subset [35], bears a predictive relationship with safety; and particular kinds of organisational culture affect safety - both positively and negatively [167], [143], [62], [35], [143]. The term 'safety culture' has been used since the Chernobyl disaster of 1986 [35], [20], and has been attributed as a causal factor in many well-known accidents (see [35] and [62]). Whilst we acknowledge it has a reciprocal, and causal effect on safety outcomes [143], [20], we do not seek to contribute to the debate on the efficacy and usefulness of safety culture as a phenomenon (see [143], and [62] for example). Rather, we acknowledge the influence of safety culture on the attainment of good software safety practice, and provide a framework and process by which any impediments to good practice can be identified and remedied.

Having understood and assessed its software safety practice, an organization can identify potential impediments to achieving good practice for software safety practice, and then undertake targeted investigations into whether any impediments exist, and derive effective mitigations. The notion of good practice considers ensuring the safe contribution of software to system safety throughout the system lifecycle. By 'lifeycycle' we mean the "activities occurring during a given time interval that starts when a system is conceived and ends when the system is no longer available for use, is decommissioned and is disposed of" [63].

There are many different types of system lifecycle models, ranging from the V-Model development lifecycle required by ARP 4754A [147], through to Iterative Incremental Development lifecycle models such as Scrum (see [153] for example) or DevOps [72]. We do not make any judgements on the model of system lifecycle used by an organization, but do restrict our focus to the design and development phase of the lifecycle (i.e. up to, but not including the point that a system enters service).

1.3 Thesis Motivation

Personal experience of the author in delivering consultancy to a plethora of design and development organizations reveals that many organizations have preconceived notions on where issues with their software safety practice lie.

Organizations often assert (with varying degrees of confidence) where impediments to best practice for software safety emanate from, and yet it is often the case that such assertions relate to symptomatic manifestations. As such, the motivation for the research presented in this thesis is to enable an organization to identify appropriately-distal causes of any impediments to achieving good practice for software safety.

1.4 Thesis Aims

This thesis aims to provide a framework through which an organization can gain an understanding of the elements which constitute their software safety practice, and a process by which that organization can therefore understand and assess its software safety practice. To our knowledge, no process exists for understanding and assessing software safety practice within an organization.

The Thesis Aims are supported by:

- a state of the art review into the elements and activities of software safety practice associated with the Research Objectives and Research Questions (Chapter 2)
- the development of a framework for understanding the elements that constitute software safety practice (Chapter 3)
- the development of a process for understanding and assessing software safety practice (Chapter 3)
- an adapted modelling symbology by which software safety practice is represented and subsequently assessed (Chapter 4)
- an instantiation of the framework and process (Chapter 5), and
- an evaluation of the framework and process (Chapter 6).

We deliberately avoid theorising as to whether and why poor software safety practice exists within current projects. Instead, we adopt a phenomenon-based research approach suggested by von Krogh, Rossi-Lamastra and Haefliger, and create a novel process which will allow an organization to identify and gather relevant data that enables them to assess their own software safety practice [164].

1.5 Thesis Structure

Chapter 2 provides a Literature Review in support of defining the research objectives and research questions. This concludes the aspects of this thesis which represent existing knowledge. Based upon the literature review a set of research objectives are defined.

The unique contribution of this thesis commences from Chapter 3, which (using the research questions from Chapter 2) outlines the framework and process by which the elements of software safety practice can be identified, and the interrelationships of the elements can be understood, and assessed.

Chapter 4 evaluates the options for modelling software safety practice (predicated on the framework and process defined in Chapter 3), and argues over the selection and adaptation of the modelling symbology and ontology to be used in the process to understand and assess software safety practice.

The framework is used, and the processes is instantiated in Chapter 5, with the outputs of the process to understand and assess software safety practice presented as evidence of its effectiveness. The instantiated framework and process uses the symbology and ontology selected and argued over in Chapter 4.

Chapter 6 evaluates the framework and process, and argues over the 'goodness' of the framework and associated process. This evaluation considers the extent by which the research questions from Chapter 2 have been met, by arguing over the evidence and its inference to meeting the evaluation claims.

The thesis concludes in Chapter 7. This final Chapter also contains recommendations for future work which will provide an improved framework and process to understand and assess software safety practice, as well as generating more evidence for the evaluation claims made in Chapter 6.

This chapter has outlined and contextualized the problem space, has provided an overview of the solution space, and provided the contents and structure for the thesis.

Chapter 2

Current State of the Art of Software Safety Process Improvement

This chapter presents the extant academic and industrial practice knowledge with regard to the following areas:

- Existing Investigations of Software Safety Practice (to assess the state of the literature)
- Models of Software Safety Practice (to determine what modelling exists)
- The concept of 'Idealised Software Safety Practice versus Reality' (to consider what the literature reveals about the notion of 'best practice'; what it is constituted by; and how it is interpreted in practice)
- Functional Safety Standards (to review what Standards Committees require projects to undertake to assure the safety of software in a system)
- Considering the Solution Space (to assess what models or processes exist which could be used or adapted for our research)
- Empirical Research Methods (to understand best practice for undertaking the planned empirical research).

The rationale for considering each area relating to the problem space (and how they relate to the thesis aims) is discussed in turn as the chapter progresses. As a result of the findings of the literature review, the chapter concludes by defining the research objectives and research questions addressed by this thesis.

2.1 Existing Investigations of Software Safety Practice

Before embarking on a drive to improve software safety practice, it is prudent to first assess the state of the literature with regard to empirical investigations into software safety practice. By examining what is argued to represent good practice for software safety, we can undertake empirical research to reveal whether and how it is being followed. If we can establish whether recognized good practice for software safety is already agreed and stipulated, we can plan research to discover whether it is being followed, and to uncover the reasons why it may not.

Although investigations exist into incidents and mishaps where software is asserted to be a causal factor (such as the Therac-25 accident [84], the loss of the Mars Climate Orbiter [9], or the loss of Ariane 5 [85]), the literature review provided in this chapter revealed that (to the author's knowledge) no empirical investigations have been carried out to specifically assess the effectiveness of software safety practice itself. In the absence of any empirical investigations into software safety practice specifically, attention now turns to the existing investigations of safety practice generally (to establish what software safety practice could learn/adopt from safety practice).

State other the Art Summary 1: No empirical investigations into the effectiveness of software safety practice exists.

Surprisingly few empirical investigations have been undertaken into what constitutes good continuous improvement, or fixes for software safety engineering practice, nor why improvement attempts have historically not been effective. Anecdotal evidence suggests that issues with general safety engineering practice itself also persists. This may be for instance, due to only a limited subset of the elements of practice being considered.

One example of focusing on a limited subset of the elements of safety engineering practice is the use of checklists [39]. With checklists, the aim is to prevent any gaps between between 'work as-required' and 'work as-done'. Previously used in the realm of pilots who use them as part of a 'pre-flight check' routine, their use is potentially without evidence of effectiveness, nor evidence that the issues lie with either 'work as-required', or 'work as-done'. Checklists have also increasingly been adopted by industries outside of aviation as a means of ensuring work as-done done corresponds to work as-required. Guwande used checklists in nursing [39] to improve adherence to the steps of a procedure. However, adherence may not be the problem. It might be that the procedure was inappropriate or incorrect. If this is true, any fix may instead need to focus on the identification and control of processes. Rasmussen [139] focused on the control of work processes to avoid "accidental side effects causing harm to people, environment, or investment".

Rasmussen also noted that at the highest level, safety was controlled, influenced, and motivated by rules, regulations, and instructions. Rasmussen later acknowledged however, that owing to the localized contingencies associated with tasks and procedures, work instructions can never be "followed to the letter", because to be operational, the rules have to be "interpreted and implemented in the context of a particular company, considering the work processes and equipment applied". He further argued that the complexities, freedom of choice, and the interpretations of work carried out by an operative are a reason why classical prescriptive work processes (such as those required by an organization) could never be effective. Rasmussen observed that "modelling human behaviour in terms of a stream of acts will be unreliable for a dynamic environment when behaviour is context dependent" [139].

This theory that prescriptive work processes can never be followed to the letter is supported by Rooksby et al [144] who set out to ethnographically observe the relationships between the documented or expected procedures of software testing and the 'reality' of practice, and what the testing "actually involved". In doing so they were not concerned with having pre-conceived ideas on what "ought to be done", rather they set out to observe and characterize the socio-technical issues emanating from the practice of software testing.

State of the Art Summary 2: There is a lack of explanations as to why good practice for software safety has not been achieved.

Any, all, or none of these examples could provide appropriate explanations as to why good practice for software safety is not being achieved, but we currently do not have the evidence. We need empirical data to establish whether software safety practice is sub-optimal, and this lack of research to date could be reducing the effectiveness of practitioners in improving practice [132]. This lack of empirical data may be due to one of the inherent problems with safety science research, insofar as there are many ethical and practical issues with studying interventions [157].

Empirical investigations of safety work are sparse, and those that do exist appear to be centred on the safety of working practices, and the work of assuring the safety of working practices (e.g. [133]). Both types of investigations have investigated (Safety) **<Work As> <X>** concepts:

- Work as **Desired**: how people would like work to take place [137]
- Work as **Imagined**: what people expect everyday work to be [60]
- Work as **Done**: that carried out by the workforce [133], or
- Work as **Documented, or Observed**: an observed and documented assessment of work carried out by the workforce [58]

For example, Hollnagel has described the differences between the construct of "Work as Imagined" and "Work as Done", noting that this could be extended further through analyses of different 'lenses' of 'Work as X' (such as between work as documented and work as observed) [58]. Such a theoretical construct allows the analyst to ethnographically identify differences between Work as Imagined and Work as Done (such as the work of [133] in considering the safety of work and safety work), thereby making changes to either—in order to improve safety and resilience.

A potential limitation of these investigations is that they tend to be centred on investigating theoretical discrepancies between two elements of safety practice. These investigations tend to compare two elements of **<Work As> <X>** as an omni- or bi-directional relationship, which makes an implicit assumption that any issues with practice emanate from *only* the elements under consideration. Examples of such investigations of safety practice include Hollnagel's description of the construct of 'Work as Imagined' and 'Work as Done' [155], and the empirical investigations of this construct by researchers such as Provan, Rae, and Dekker in [133]. The terms used in these investigations (**<Work As> <X>**) are heavily entrenched in the discipline of Occupational Health and Safety (OHS), or the "safety of work"[133]. They do not yet extend to consider the work of a software safety practitioner working to assure the safety of a complex safety-critical system.

Each of these omni- or bi-directional approaches (Hollnagel's characterizations of Work as Imagined and Work as Done, Provan et al's considerations of the differences between the work of safety and the safety of work, and Gawande's use of checklists) are focused on assessing specific elements of safety practice. Should changes be made as a result of these assessments, we cannot currently be sure whether such approaches are identifying and fixing real problems, as not all elements of safety practice have been considered concurrently. Nor do we know whether such 'fixes' are introducing new issues, or undermining other elements of safety practice. Notwithstanding, such approaches have become the accepted norm when issues with safety practice are suspected. Noting that academics have pre-conceived ideas on what the issues and fixes may be [144], and that "safety professionals are not confident operationally of how to create safety improvement" [133], the question therefore arises, how do we improve software safety practice?

What has not yet been produced is a credible process for understanding safety practice per se. Myriad papers have hinted at providing a 'framework' on which such a process could be built, but we seem to be 'separated by a common language' in this regard. For example, the prevailing use of the term 'framework' in the safety science literature does not in fact refer to a framework as a 'structure on which something is built', but provides attributes such as concepts (e.g. [6]), notations (e.g. [70]), new tools ([55], [27], [171]), [148], [162]), models ([92]), principles (e.g. [52]), patterns (e.g. [170]), checklists (e.g. [96]), or ontologies ([138], [137], [73]). To prevent confusion with any of these definitions, the term 'framework' is used in this thesis to describe the elements of practice, and the relationships between them.

Noting that fixes to safety practice have been implemented despite surprisingly little empirical investigations [135], this lack of investigation may question the effectiveness of these fixes in addressing the real impediments to good safety practice. We argue that many implemented changes to safety practice were ineffective perhaps because the analysis that revealed their necessity was limited to a consideration of only a limited subset of all the elements that constitute safety practice in its entirety.

Improvements to software safety practice are needed to prevent further failures such as the emergency shut-down of the Hatch Nuclear reactor [168]; misinterpretations such as those leading to the loss of the Mars Climate Orbiter [9]; and incidents such as the fatal radiation overdoses administered at the Panamanian National Oncology Institute [12].

Before embarking on a programme of software safety practice improvement however, we believe we must be as confident as possible that we have both identified where any issues lie, and that we are addressing the complete set of problems and their causes. This thesis therefore starts from the position taken by Rae et al [135] in that we must describe current work before changes are prescribed. It also aims to “capture, describe, and document, as well as conceptualize” [164] software safety practice so that appropriate theorizing can proceed. In other words, we need to first understand software safety practice.

We could have looked to empirical investigations into the ‘good practice’ of engineering disciplines other than software safety and safety engineering in general. We argue however, that the discipline of safety engineering and its relationship with the field of safety science makes safety practice 'unique amongst its peers'. Safety science and safety engineering are described by Shorrock in [75] as “islands in a common sea”, and cognisant that safety relies on many non-engineering stakeholders for a successful ‘outcome’, this renders any comparison with other engineering disciplines null. We do not know if any process to understand any other engineering discipline exists.

To understand safety practice fully, we need to have a means by which we can discuss and evaluate software safety practice in an ontologically robust manner that allows the different elements of safety practice to be represented equally. We need a form of model.

2.2 Models of Safety Engineering Practice

By model, we take the INCOSE definition of a "representation of a system of interest from a particular viewpoint" [25], and extend this to also include ‘phenomenon’ . The literature did not in fact reveal any existing models which have been created specifically to understand the *practice itself* of software safety (nor of safety practice generally), and so attention initially turned to how safety practice is currently controlled and directed.

Safety practice is nominally controlled and directed through processes and procedures expressed in Open and/or Closed standards. The term ‘Open’ (Standard) refers to the fact that there are no Intellectual Property Rights (IPR) that preclude paid access to the standard. 'Closed' (Standard) refers to those created and used by an organization - who restrict access to their employees, thereby protecting the invested IPR held by the organization. Examples of Closed Standards include organizational procedures and processes, and examples of Open Standards include BS EN 61508 [13] which describes functional safety processes for the design of software and complex electronic hardware used in safety functions, and the ARP 4754A suite of publications [147] [1] which describe safety processes for the design of aerospace systems.

Many of these standards comprise multiple sections ([13] having 7 volumes as but one example), as they aim to consider all phases of development, from concept to disposal/termination. Understanding large, predominantly text-based standards presents many weaknesses. These weaknesses include the need for multiple cross-references, and the need to be re-read numerous times to decipher meaning; as Weaver discovered when considering text-based safety arguments [165]. Another weakness of lengthy text-based processes and procedures stems from the complexity of being able to clearly and succinctly describe the complex interrelationships between processes or activities. These weaknesses may suggest that textual representation is not a suitable form of representing software safety practice.

State of the Art Summary 3: Existing means by which (software) safety practice is currently controlled and directed is limited to textual representations. Text-based representations suffer from inherent weaknesses.

Research Design Decision: Owing to the weaknesses of text-based representations, a graphical representation with a suitably-defined structure, syntax, and taxonomy would be a critical enabler for this empirical research.

There are many potential options for representing practice / processes graphically. One option is a 'pattern'. In his 1998 book, Ambler defines a Process Pattern as "a pattern which describes a proven, successful approach and/or series of actions (for developing software)" [3]. To define what a process pattern should constitute, Ambler examines the definitions of both 'process' and 'pattern'; a process being defined as a "series of actions in which one or more inputs are used to produce one or more outputs"; and a pattern being "the description of a general solution to a common problem or issue from which a detailed solution to a specific problem may be determined" [ibid].

When considering Ambler's further definition of a 'Task Process Pattern'; a subset of process patterns that "depicts the detailed steps to perform a specific task" [ibid], this resonates with an aim of producing a workflow, or pattern of required activities that can be used repeatably as a means to understand software safety practice. Although process patterns can be written in text, it is common practice to employ a graphical representation. There are many tools that provide visual representations, and Chapter 4 contains a thorough review of those

[1] This thesis considers ARP 4754A. It is noted that ARP 474B now exists - which primarily aims to clean up the distinction between 'process' and 'techniques and methods' considered in ARP 4761A.

available. Beforehand, we must first consider the specifics of what needs to be represented when considering software safety practice.

Whilst extant lifecycle representations portray the activities to be undertaken, they do not currently consider the attributes of such activities e.g. timing constraints [49], timing requirements [163], commercial/contractual complexities [152], [141], the resources required to undertake the activities (nor the attributes thereof), nor the intricate interrelationships and interdependencies between activities. They rely instead on a simplistic overview portrayed in any accompanying visual representations.

Software safety practice is constituted by many elements (and their interrelationships); including philosophy, intent, and activities. As such, any graphical representation must support the creation of a model that can be employed as part of a robust and repeatable process for understanding practice; whose constituent parts represent all elements of the practice under analysis.

This thesis aims to provide a means by which an organization can understand and assess their software safety practice - in order that the organization can assure that best practice is being/will be achieved. The first enabling step in the creation of such a process is to identify all elements that constitute software safety practice. To define these disparate elements, attention turns to how idealized practice is conceptualized, how it is described, and how it compares with the reality of practice.

2.3 Idealized Software Safety Practice versus Reality

To understand the current state of software safety practice, we argue that an analyst must first identify the elements that constitute practice as indicated above, and the relationships between these disparate elements. We define ‘elements’ as any ’constituents which make up the whole’ [2], and assert these elements to be the discrete building blocks which combine to constitute practice (see Figure 2.1).

We have already noted that existing investigations into safety practice generally are limited, and have been referred to using the form (Safety) **<Work As> <X>**, such as:

- Work as **Desired:** how people would like work to take place [137]
- Work as **Imagined:** what people imagine everyday work to be [59]
- Work as **Done:** that carried out by the workforce [133]
- Work as **Documented, or Observed:** an observed and documented assessment of work carried out by the workforce [57].

We have identified three elements of software safety practice, spanning the idealized concept of what practice *should* be, the manner in which it is imparted

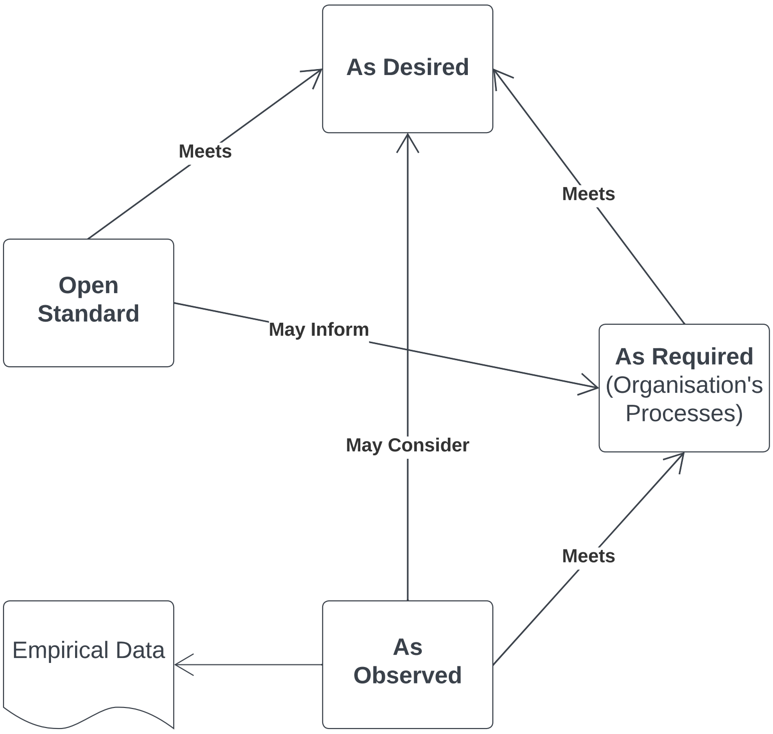


Figure 2.1: The 3 Elements of Safety Engineering Practice.

to practitioners, and the reality of practice as carried out by practitioners. These elements are represented in Figure 2.1. All existing software safety (**<Work As> <X>**) can be mapped onto these three elements, and whilst it is argued this is necessary, it cannot yet be argued whether this is complete:

- Software Safety Work as Desired - a representation of how an organization desires software safety work to be, in the abstract
- Software Safety Work as Required - a representation of how an organization explicitly requires its personnel to carry out software safety work
- Software Safety Work as Observed - a representation of how an organization's software safety practitioners carry out their work.

The term 'software safety practice (as X)' is now used to describe the different elements by which it can be categorized. We now consider each of these elements of software safety practice in turn.

2.3.1 Software Safety Practice As Desired

Software safety practice as-desired is characterized by a set of safety objectives that are held by stakeholders within a project. The stakeholders who hold these objectives are the individuals, teams, organization, or classes thereof, having an interest in a system [67] - specifically those who are responsible and accountable for system and / or software safety. Together these safety objectives embody the software safety philosophy and risk appetite of the organization. Complete and correct compliance with software safety practice as-desired should manifest in a product which is acceptably safe to operate in a given operating environment. Of course, poorly identified or articulated, or incompatible safety objectives may be a source of software safety process failures.

There is no ‘one-size-fits-all’ approach to defining software safety practice as desired across all sectors and applications, and we do not prescribe what as-desired practice is constituted by in all cases. We do provide criteria for as-desired practice in Chapter 3.1.1 however, and offer these criteria to projects for consideration and further adaptation.

For an organization that wishes to understand software safety practice, defining what constitutes as-desired practice may be the most complex and challenging element. Even if the responsible and accountable stakeholders in an organization believe that their as-desired practice is clearly defined and described, it may not actually be explicitly documented anywhere - or at least it may not be described in a manner which allows its philosophical attributes to influence an engineered design. There are many potential reasons for this.

From experience, Open Standards (such as [13]) are often held to represent safety practice as-desired in the form of codified expertise [5]. We argue that this cannot in fact, be considered to be sufficient for as-desired practice. Such standards contain a mixture of normative requirements, informative guidance, and safety philosophy. Whilst normative requirements can be measured and even audited against, it is not immediately clear how the required intent and safety philosophy of a Standard can be validated and verified in a design. Nor can the committee responsible for an Open Standard presuppose the safety philosophy and risk appetite of an organization which aims to comply with it.

Whilst compliance with requirements can be measured through qualitative and quantitative means; and processes can be assessed as to their correctness and completeness, it is not possible to measure the alignment of intent and philosophy of a standard in an auditable manner. This presents an open question of whether a safe design is achieved by following a standard, or whether a safe design is achieved because the software safety practitioners cared enough to deliver it, despite the standards’ deficiencies.

Standards committees therefore face the challenge of ensuring that their standard will represent the overarching intent and safety philosophy for any organization that elects to comply with it as a means of assuring the safety of their design. Consider BS EN 61508 [13], which was established as a unified, generic standard that aims to achieve functional safety by minimizing risk through the application of Safety Integrity Levels (SIL) to safety functions. A challenge that arises through offering such a ‘pan-industry’ approach is whether the standard can appeal to the different safety philosophies of disparate organizations across industry, many of whom will also have their own disparate supply chain.

Different Open Standards have been designed with variations in the means of their intended application, and predicated on different principles of risk management. For example, whilst BS EN 61508 [13] may be formally certified against for compliance, the standard is not designed to be applied in tandem with a regulatory or certifying body specifically. This is in contrast with the ARP 4754A (ARP) suite of standards [147], which provide "safety recommended practice". These practices have been adopted as acceptable means of compliance for certification by regulatory bodies (such as the FAA and EUROCAE).

The ARP's safety philosophy is to moderate the severity of outcome through the application of Design Assurance Levels (DAL) to components and systems. A challenge for the Standard's body here is ensuring that its safety philosophy is applicable to all civil manufacturers of aerospace systems - both rotary and fixed wing (and indeed to organizations outside of the civil aerospace sector which have adopted it). Organizations and regulatory bodies in these disparate sectors may have differing approaches to risk which are predicated on factors other than the moderation of the severity of outcome (alone). They may also have differing safety philosophies.

Standards such as BS EN 61508 and ARP 4754A are predicated on philosophies that safety is achieved through the moderation of risk or severity of outcome, yet an organization's philosophy may not be founded in risk or severity reduction per se. The as-desired software safety practice of an organization may be founded on principles (e.g. [49]), systems theory (e.g. [82]), Normal Accident Theory (as discussed in [40]), or predicated on specific attributes such as resilience (e.g. [61]), or high levels of reliability (e.g. [142]). Further, an organization may also operate a safety management system which is based on either centralized control (i.e. 'Safety I'), or guided adaptability (i.e. 'Safety II' [134]).

Whilst we accept that an organization can impart some aspects of idealized practice through normative requirements, a final challenge concerns the efficacy of the processes, procedures, techniques, and methods that manage these requirements. Consider again BS EN 61508 [13], for which compliance is met by achieving its objectives. These are held to have been met if the applicant meets the requirements (clauses); which in turn can be instantiated by following recommended techniques and measures (with accompanying levels of importance predicated on the SIL). What is not known is whether this process (objectives met by requirements, which are instantiated by techniques and measures) manifests in a safe design directly (referred to as the 'inductive quality gap' by Habli, Hawkins, and Kelly [43]). These objectives, requirements, and techniques and measures are not derived from an evaluation of empirical data, but predicated on expert judgement and opinion - based on the experience of the standard's committee members.

Similarly, UK Defence Standards require both clauses and objectives to be met. Defence Standard 00-055 [160] requires not only that its specific clauses be met, but demands compliance with its objectives (expressed as five Principles).

One clause requires the Contractor to select an Open Standard to comply with. Once selected, the proposed Standard is agreed with the MoD and complied with. Additional work is then required to conform with the appropriate 'Military Delta'. The 'Military Delta' stipulates additional requirements to recover perceived shortfalls in the selected standard when used in a military context. Whilst the five Principles are laudable (predicated on [49]), it is not immediately clear how the Principles are met by compliance with the clauses of the Standard itself, the selected Open Standard, nor the Military Delta required by the Defence Standard.

Research Design Decision: Noting the complexities associated with establishing as-desired practice for software safety, this thesis therefore defines and expresses software safety practice as-desired. The description of as-desired practice is given in Chapter 3.1.1; where we also explain how as-desired practice can be represented and assessed for compliance.

2.3.2 Software Safety Practice As Required

Software safety practice as-required is constituted by a set of processes which are designed to instantiate software safety practice as-desired when followed by a software safety practitioner. Software safety practice as-required is a representation of how an organization explicitly requires its personnel to carry out software safety practice. Used by standards bodies and organizations alike, software safety practice as-required describes the processes required to be followed by software safety practitioners.

The activity to convert software safety practice as-desired into software safety practice as-required is not considered as part of this thesis as there exist many projects emanating from disparate industries and applications, and there can be no 'one-size-fits-all' approach to this. The next research design decision therefore relates to the means by which an organization can understand its safety practice.

Research Design Decision: A multi-directional mechanism should be provided to enable a project to understand its software safety practice.

There are two types of as-required software safety practice, the first being an Open Standard such as the ARP 4754A suite of publications, which requires the adoption of a 'V-Model' lifecycle that aims to show the interaction between safety processes and design and development processes, and is used in an iterative and concurrent manner from 'Platform' level down to 'Item' and 'Software' levels. Another example of as-required practice expressed by a standard is that specified by BS EN 61508, and whilst this standard doesn't require a specific development lifecycle model, it requires that its objectives are met. Each objective is achieved through compliance with the standard's requirements (clauses) - which may in turn be instantiated by the use of selected techniques and methods.

Standards Committees design and compile their Open Standards in a manner that expresses a set of lifecycle activities which - when adopted by an organization, and followed by the organization's software safety practitioners - aims to comply with the standard's version of as-required safety practice.

We must also look to the practice required by organizations who employ software safety practitioners within a manufacturing, design, or procurement setting. Such practice is normally documented by the developing / acquiring organization, and expressed as a lifecycle of processes, and resulting documentation that are undertaken throughout the product lifecycle (referred to as 'Closed Standards'). An organization may develop its own practice predicated on the requirements and informative guidance expressed by an Open Standard, as the organization seeks to demonstrate due diligence based on conformance with the selected Open Standard [42] by incorporating the Standard's knowledge within its processes [4]. Typically, organizations do not directly follow an Open Standard - they create their own set of internal processes and procedures which may be derived with the intent of meeting a specific and selected Open Standard. An organization *may* select an Open Standard and create their processes and procedures such that they will comply with the as-required practice required in the Open Standard. Open Standards have their own challenges, and these challenges may be masked or exacerbated as an organization develops their processes in a manner that may lose the intent of the Open Standard.

Closed Standards may offer different representations of software safety practice to Open Standards, but they are protected by IPR which prevents, for example, the widespread sharing of practice and issues arising. When considering the development of complex 'Systems of Systems' (SoS), the constituent systems / components may also be developed by differing organizations who adhere to different Open / Closed safety Standards. Should these differences between Open and / or Closed standards exist, we need to understand the reasons for this, and what impact they may have on software safety practice as-observed across the SoS development process.

Akin to Morley et al's argument that research into policies for safely implementing Artificial Intelligence (AI) into the health care domain requires a prospective approach that is cognizant of the complexities of individuality, and social and organizational structures; any subsequent mitigation research into improving the effectiveness of Open / Closed Standards should take into account the complex interrelationships of all elements of software safety practice [100].

The challenges associated with the creation and maintenance of Standards are well-documented (see [42]), but we offer no a priori hypotheses regarding any contribution made by the use of as-required processes [164]. A challenge for any organization is to define as-required processes and procedures which meet their as-desired software safety practice AND to express this in a way that clearly and explicitly imparts the requirements, intent, and philosophy to those individuals charged with implementing it.

Individuals control a lot of detailed software safety practice, and individuals form both organized and unorganized groupings within an organization and will learn, modify behaviours, and create their own norms and rules for self-governance [164]. Any means by which an organization can understand and assess its software safety practice must therefore be capable of eliciting such nuanced practices and behaviours, and identifying how they relate to an organization's processes [164].

2.3.3 Software Safety Practice as Observed

Software safety practice as-observed is a representation of how an organization's software safety practitioners carry out their work. This software safety practice as-observed is nominally controlled and directed through defined processes, and there will often be multiple sources of these processes which "can be represented, with imperfect fidelity, through standardized models and procedures" [135] (or software safety practice as-required).

For example, we may observe in an organization that work on eliciting software safety requirements starts before component-level safety requirements are fully established. This may deviate from the lifecycle model that portrays a chronological and sequential hierarchical decomposition of safety requirements.

For many different and complex reasons, software safety practice as-observed may not be equivalent to software safety practice as-required, and this could contribute to achieving poor software safety engineering outcomes. We must therefore strive to ensure software safety practice as-observed aligns with software safety practice as-required. Further we must show that both elements of practice are fit for purpose (i.e. that software safety practice as-required, and software safety practice as-observed are aligned with software safety practice as-desired as well).

Of course, it may be the case that those charged with undertaking as-observed practice are aware of shortcomings in software safety practice as-required and that they have made subtle (perhaps hidden) improvements on software safety practice as-required in an attempt to align with software safety practice as-desired. For example, in the example above of software safety requirements being partly derived before component-level requirements are finalized, this may be a positive deviation, in that it allows the design activities to progress within the required timescales in the specific context of the project that does this. Because deviations may be positive, any evaluation should be capable of identifying any subtle improvements made by those carrying out software safety practice as-observed over that stated in the as-required processes; including any changes resulting in meaningful engagement with internal or external communities of practice [5].

It is of course possible that any such perceived 'improvements' may not actually improve software safety practice, and may instead undermine other elements of practice, or may introduce new issues. This observation is aligned with the systems-theoretic approach [78], [80] as it acknowledges the existence of emergence as a system property.

By considering all elements of software safety practice (and the interrelationships between them) in a holistic manner, it should be possible to identify and mitigate the risk of unintentionally undermining software safety practice.

It is also possible that even when gaps, conflicts, or differences do not exist, emergent behaviour [82] could still lead to 'bad' practice due to phenomena manifesting at an organizational level. These deviations could create deficiencies in practice.

Beyond deficiencies in the as-required process, software safety practice as-observed may contain elements of what Dekker refers to as "malicious compliance" - where those charged with implementing safety practice as-required carry out processes that they know are inadequate [23]; or instances of work that don't contribute to achieving or demonstrating safety (or "safety clutter" [133]). As-observed practice may also have instances of what Provan refers to as "role retreat" (where workers just perform their role only as defined (i.e. work to role)), or covert work systems (where work as-observed is hidden from 'outsiders' due to the fear that it will be stopped or changed, thereby making work more difficult for front-line teams) [134].

Safety practice as-observed is normally measured by auditing against work as-required. This misses the nuances and intricacies of what actually happens 'as done'. Indeed, Provan notes that safety work needs to adapt and deviate from plans, rules, roles, and procedures because of the dynamic and emergent nature of complex systems [134].

Having considered the contribution of Standards to, and their relationship with the elements which constitute software safety practice, attention now turns to the efficacy of Functional Safety Standards themselves.

2.4 Functional Safety Standards

Functional Safety Standards are compiled by regulatory or standardization bodies such as those which publish BS EN 61508 [13], and together present variations of goal-based and process-based processes. Prescriptive, process-based assurance processes (in isolation of a product-based assurance process, or robust assurance of safety requirements) are not supported by the literature however for modern, complex, software-derived safety critical systems [152], [43], [94], [53], [29], [54].

Open Standards are often adopted as a means of appealing to Recognized Good Practice (RGP); with organizations adopting the recommendations made by them into their internal processes. Their applicability is neither pan-industry, nor pan-technology however, and whilst there are commonalities within Open Standards, there are also major variations across sectors and countries [94].

Open Standards also present inherent weaknesses in terms of the guidance they offer on vital aspects of a software safety lifecycle, such as the decomposition of software safety requirements [4]. Of equal issue is the fact that Open functional safety standards, and regulatory/Defence Standards do not necessarily align with realistic software safety requirements engineering practices, nor the evolution of software safety requirements throughout the product/system lifecycle [87].

Open safety standards such as BS EN 61508 [13], and ARP 4754 [147] that advocate 'traditional V-Model' Lifecycles do not support pragmatic software safety requirements engineering practices, as:

- Many still (unrealistically) assume that all requirements are known 'up front' [55]
- Some encourage the 'finalization' of requirements before the design work commences [87]
- Offer weak guidance on the decomposition of requirements [4]
- Their implementation varies widely between the Open Standards – making integration into complex systems awkward and complicated [163], [36], [103], and their implementation is difficult in practice [93]
- They inadvertently encourage tokenism [71] and naïve implementation [152]; which may lead to a lack of any tangible safety benefit [87] – especially when focussing purely on prescriptive compliance [70]
- They can present an inductive qualitative gap when considering the assurance of software (vice integrity) [43]
- Have inherent weaknesses in the implementation of the 4+1 Principles [71]
- Ignore the basic principles of systems engineering (i.e. simplicity) [152]
- Do not take into account the commercial and/or contractual complexities in the procurement of safety-critical systems [152], [141].

In 2013, Stålhane et al asserted that V-model development processes are weak as they cannot handle changes to requirements and / or customer needs [153] - although they acknowledged that safety requirements are not particularly volatile (but are often impacted through changes in functional requirements and emerging hazard analyses). Implicit in hierarchical representations such as the V-Model lifecycle, is that software safety requirements are derived specifically from component or hardware safety requirements (item requirements being the previous development/design tier in the overall lifecycle process). Habli and Kelly however, urge caution against not considering software requirements at the system level [44]; asserting that a lack of system-level requirements review by software analysts may give rise to common mode failures.

Although iterative in nature, the sequential and hierarchical lifecycle inferred by the V-Model is not supported by the state of the literature as to the links between requirement elicitation activities and the software safety lifecycle. In the literature, software safety requirements are considered at a far higher level of abstraction and earlier phase of the design lifecycle. Furthermore, in complex socio-technical systems, emergent properties (and therefore hazards) are generated from a complex web of interdependencies that span systems, sub-systems, components and the environment [153]. Iterative, yet chronological models are insufficient for mitigating and managing such complexity. Indeed, sociotechnical systems that are both tightly coupled and interactively complex cannot, in the long run, be managed in a safe manner [40]. The continued management of safety during the run-time of such complex systems (in the presence of inherent uncertainty) is not typically given sufficient consideration by open safety standards.

In 2001, McDermid and Pumfrey observed that some projects that have been developed to 'certification standards' were, in essence, "developed three times" with the rework due to "late discovery of requirements or design flaws" [94]. They further highlighted that development standards assume a lifecycle for "completely new systems, and generally ignore the change and development of existing systems"; and also, that processes such as those described in ARP 4761 would be enhanced by mandating an extension to the classic functional failure analysis to the software level (through techniques such as a Software HAZOP) [ibid].

Boehm urged a re-consideration of development lifecycles as far back as 1988 [11]; warning that "many software projects. . . have come to grief because they pursued their various development and evolution in the wrong order". In a more recent Systematic Literature Review, Vilela et al highlighted the importance of identifying requirements as early as possible in the lifecycle in order to prevent the propagation of safety issues through subsequent phases of development [163]. This would also reduce costs significantly – by addressing the issues when it is cheapest to do so.

Defence Standards present similar weaknesses with respect to requirements engineering; in terms of a lack of guidance for the decomposition of safety requirements [4]. They do not take into account commercial or contractual complexities; and the requirements are often difficult to impart to suppliers and sub-suppliers in the supply chain [52].

In many functional safety standards, no consideration is given to non-functional requirements such as timing constraints, or requirements of activities that derive artefacts; nor to any contractual principles (between acquirer and supplier) or limitations thereof [53]. Vilela et al [163] note further that "safety standards. . . do not explicit [sic] highlight which information should be specified early in the development process". Examples of such information may include human interaction with software, and integrity constraints (which may also inform an organization's make/buy decision at the simplest level).

The type of V-lifecycle model portrayed by many Open Standards may no longer be valid therefore, and we suspect that people and projects are not actually following 'traditional' safety lifecycles such as this, as:

- The model is neither complete nor sufficient to deal with the complexities of socio-technical system safety assurance; AND/OR

- People/organizations may be willfully detracting from the model as it is no longer fully fit for purpose for the complex socio-technical (and software-intensive) systems that are being designed.

Having considered the issues, deficiencies and nuances involved in determining good practice for software safety assurance, attention now turns to the potential modelling solutions, and the research questions this thesis aims to answer.

2.5 Modelling Approaches

This section of the thesis presents the state of literature with regards to how the solution space highlighted in Sections 2.1, 2.2, 2.3 and 2.4 can be defined. The concept of 'solutions' are returned to as the thesis progresses.

As we have discussed in Section 2.2, to understand software safety practice fully, we need to have a graphical representation which can be used to understand and assess software safety practice in an ontologically-robust manner that allows the different elements of practice to be represented equally. This thesis therefore presents a framework which is capable of linking the different elements together. This allows an analyst to evaluate their relationships, and reveal any nuances and subtleties.

It is noted that the implementation of any new models or process patterns will face their own impediments to adoption. Some of these impediments may by socio-technical in nature. This includes those impediments incurred as organizational structures change (people churn) [46], and any association a new model/process may have with a previously 'failed' product [3].

We did not envisage that an entirely new model, or tool would be required however, as the literature reveals representations and notations that – although not currently adopted in a safety-related context – may be improved or adapted for use within this empirical research. The review of models and graphical representations is contained in Chapter 4 as it provides the link between the framework and process in Chapter 3 and their instantiation in Chapter 5. Attention now turns to how the empirical research is to be conducted.

Empirical Research Methodology

This research adopts a 'soft' normative approach [48], and starts from the position that an identified difference between elements of software safety practice is not necessarily positive nor negative (until further analysis and assessment is satisfactorily completed).

The assurance of software safety practice is supported currently by compliance or conformance with a project's Closed Standard that has been designed specifically to meet software safety practice as-desired. Closed Standards may also have been informed by the requirements and objectives of an appropriate Open Standard (relevant to their industry, technology, and intended use).

It is logical, therefore to assume that any impediments to achieving good practice for software safety manifest from the identification or translation into practice of software safety practice as-desired, the characteristics of software safety practice as-required, and/or software safety practice as-observed. These elements of software safety practice will be modelled and assessed using the framework and process outlined in Chapter 3. Chapter 4 presents an assessment, selection, and adaptation of the graphical representation used in the Case Studies [169].

Software safety practice as-observed presents a distinct challenge for empirical research however, as it must represent the practice of people performing software safety activities. Rather than relying on a suite of documents, eliciting softare safety practice as-observed requires a form of ethnographic study [109]. Ideally this would be carried out as an impartial ethnographic observation of software safety practice. To our knowledge however, such studies are not carried out due to the substantial time required (dependent on the technology involved, and the length of the project/programme); the need for wholly impartial and independent observers; and/or the cost of such an undertaking. As Lipshitz et al note "it takes a team to study teams in context" [86]. This leads us to another research design decision.

Research Design Decision: Undertake a series of interviews to create a model of as-observed practice.

As multiple ethnographic studies are not feasible to complete in the course of a single PhD programme, a series of interviews are employed with representatives of projects that had submitted their processes for modelling and assessment. Representation at interviews was fulfilled by team members who were either software safety engineers, or software engineers with a responsibility for safety.

As responses from interviewees could have been influenced by the created models of safety practice, the models of practice were not shared with the interviewees. Instead, the interviews were conducted from a single initiating request:

"In your role, please describe the software safety engineering activities you carry out"

Care was taken when designing the interviews, including planning for follow-up questions, considering the structure, manner and concepts of the interview, its location, sensitivity awareness (from the interviewer), question sets that avoid ambiguity, questions of a leading nature, double negative/positives, double-barreled questions, and inherent biases [151], [30], [90].

The pre-defined follow up questions were as follows:

"Are the activities defined?"

and..

"Are you able to carry them out verbatim?"

As this thesis is wholly reliant on a robust set of empirical data, the data had to be compelling and ethically derived, and the inferences and interpretations derived from it demonstrably sound. This required a clear strategy for deriving and defining (inter alia):

- The presence of existing, relevant empirical data that may be utilised to become 'theory aware' [151]
- Interviews:
    (a) Structure, manner, contexts and concepts [151],
    (b) Objectivity
    (c) Location considerations and decision criteria [151], [30] [73]
    (d) Interviewee categorizations and selection criteria
    Sensitivity awareness (closely linked to social bias – below).
- Question sets [151], [90] – avoiding:
    (a) Ambiguity
    (b) Leading questions
    (c) Double negatives/positives
    (d) 'Double-barrelled' questions
    (e) Antagonistic questions
    (f) Biases.
- Reliability of data
- Validity of data:
    (a) Statistical validity
    (b) Lack of social biases.
- Data capture, storage, and retention
- Ethical questions (neutrally framed so as not to be 'leading')
- Ethical data mining (can one legitimately make the inferences from the data?).

This chapter has presented a review of the pertinent literature that relates to both the problem space and the potential solution space for modelling solutions. Analysis of the literature reveals a paucity of empirical investigations into software safety practice, and despite many posited theories as to where the impediments to achieving good practice may emanate from, no evidence yet exists as to their source, and no current process exists by which empirical evidence can be generated and assessed.

It is argued that these gaps in both the literature and data can begin to be closed through fulfilling the Research Objectives and specific Research Questions therefore.

2.6 Research Objectives and Research Questions

The current state of the literature reveals that we do not know confidently whether good practice for software safety exists. If it does exist, we do not know whether it is being followed. As there are no existing empirical investigations into software safety practice, we have no available means by which an empirical investigation can be fulfilled. This thesis therefore identifies two Research Objectives to fill this knowledge gap:

1. **OBJECTIVE ONE:** To provide a process by which an organization can understand and assess the disparate elements that constitute their software safety engineering practice
2. **OBJECTIVE TWO:** To provide a process by which an organization can identify potential impediments to achieving best practice for software safety practice, in a manner that gives confidence that any potential impediments are appropriately-distal, and enable effective remedies to be derived.

These two research objectives can be decomposed into measurable, and atomic research questions which, if answered, fulfill the two objectives of the research. Based on the Thesis Aims, the research questions are as follows:

- **RQ1** How can an organization understand its software safety practice?
- **RQ2** How can an organization assess its software safety practice?
- **RQ3** How can an organization identify true impediments (i.e. appropriately-distal and confirmed as being causal rather than just symptomatic) to achieving best practice for its software safety practice?
- **RQ4** How can an organization derive effective mitigations for the identified impediments to software safety engineering best practice?

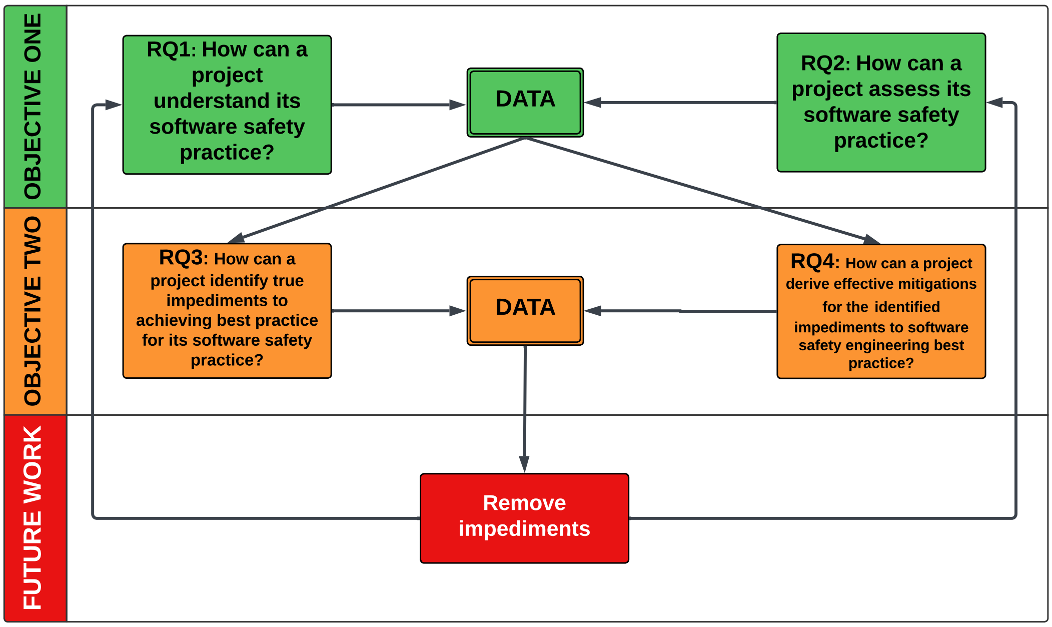

OBJECTIVE ONE
OBJECTIVE TWO
FUTURE WORK
RQ1: How can a project understand its software safety practice?
DATA
RQ2: How can a project assess its software safety practice?
RQ3: How can a project identify true impediments to achieving best practice for its software safety practice?
DATA
RQ4: How can a project derive effective mitigations for the identified impediments to software safety engineering best practice?
Remove impediments

Figure 2.2: Research Objectives and Research Questions

Whilst the proposed novel framework and process provides the means by which an organization can glean data to address the four research questions, this thesis only provides empirical data in support of the first two. This is illustrated in Figure 2.2 by the use of 'traffic light' colour-coding:

- **GREEN:** Research Questions 1 and 2 will generate empirical data in a format which can be used by an organization in Objective Two.
- **AMBER:** Research Questions 3 and 4 use the data outputs from Objective One (i.e. the models and reports created by the framework and process). Through investigations, an organization uses this data to identify true impediments are correct and appropriately-distal. The organization then embarks on mitigation research to remove the impediments.
- **RED:** The final steps are to implement the identified mitigation(s) to remove the impediments. As indicated by the arrows, aspects of the process must now be repeated to confirm whether practice has been improved, and no aspects have been detrimented. This element is a future work activity

The proposed novel framework and process provides an organization with instructions for how to complete all of the above steps, but the thesis only provides data in support of Objective One. Other than operating with the constraints of a single PhD programme, the reason for the omission of all later process steps is because the answers to questions 3 and 4 are application-, technology-, and sector-specific. They are also entirely dependent on the means by which each element of practice is described, imparted to practitioners, and interpreted. As such there can be no one-size-fits-all approach to determining whether potential impediments are true and appropriately-distal. For the same reasons, this thesis cannot derive what constitutes effective mitigations, nor confirm the removal of impediments. This requires specific, dedicated research to be planned and enacted; and is therefore not in scope.

What this thesis is capable of achieving however, is to gather data amassed from the output of research questions one and two, and present it in a manner by which it supports research questions three and four (future work).

Research Questions 1 and 2 are answered by providing an organization with a framework and process with which software safety practice can be understood and assessed. The framework and process is provided in Chapter 3. The framework and process requires a graphical notation which facilitates the modelling. The selection and adaptation of the graphical representation is explained in Chapter 4. Use of the framework and process is explicated by an illustrative example in Chapter 5.

2.7 Summary

The current state of the art is summarized as follows:

- No empirical investigations into the effectiveness of software safety practice exists
- There is a lack of explanations as to why good practice for software safety has not been achieved
- Existing means by which (software) safety practice is currently controlled and directed is limited to textual representations. Text-based representations suffer from inherent weaknesses.

As a result of the investigations into the state of the art, the following research design decisions have been taken:

- Owing to the weaknesses of text-based representations, a graphical representation with a suitably-defined structure, syntax, and taxonomy is a critical enabler for this empirical research
- Noting the complexities associated with establishing as-desired practice for software safety, this thesis therefore defines and expresses software safety practice as-desired
- A multi-directional mechanism should be provided to enable an organization to understand its safety software safety practice

- Undertake a series of interviews to create a model of as-observed practice.

Having discussed the state of the literature, and defined the research objectives and questions answered by this thesis, attention now turns to the framework of the different elements which constitute software safety practice, and the means by which they can be understood and assessed.

Chapter 3

Software Safety Practice Framework and Process

In answering the research objective from Chapter 2, the framework and process to understand and assess software safety practice must answer two research questions:

- **RQ1** How can an organization understand its software safety practice?
- **RQ2** How can an organization assess its software safety practice?

This chapter therefore describes a theoretical framework of the elements which constitute software safety practice, and posits how these elements interact. The framework which illustrates the elements of software safety practice and their interrelationships is constructed, discussed, and its ontology described in Section 3.1. The chapter also describes the required modelling and assessment activities required to instantiate the framework. The modelling and assessment activities are defined in Section 3.2, with step-by-step process instructions provided in Section 3.3.

The process to understand and assess software safety practice will generate a substantial amount of information for action by a project. The actions required of an organization are considered in Section 3.4. The models created by the process will require maintenance throughout a project's life, and this maintenance is considered in Section 3.5. The 'trigger points' in a project's life which will require parts of the process to be repeated by a project are considered in 3.6. The chapter concludes in Section 3.7 by considering the potential outcomes from applying the process, and what these outcomes may suggest.

3.1 A Framework of Software Safety Practice

We have created a theoretical framework which defines the elements which combine to constitute software safety practice. This framework also defines how the elements of software safety practice relate to and influence each other. The aim of the framework is to enable an understanding of how software safety practice is carried out, and why it is done in the manner that it is.

Specifically, we seek an understanding of how software safety practice is desired to be, how the desired practice is imparted to those required to enact it, and how software safety practice as desired is interpreted and implemented by software safety practitioners. The resulting framework model is useful because it creates a detailed representation of software safety practice whilst remaining as simple as possible.

We deliberately avoid theorising as to whether and why poor safety practice exists, rather we adopt the phenomenon-based research suggested by von Krogh, Rossi-Lamastra and Haefliger, and create a process (Section 3.2) which can be used to identify and gather relevant data using this innovative framework [164].

We noted in Chapter 2.3 the three elements which constitute software safety practice, and that these three elements improve on the simpler model of 'work as imagined versus work as done' [137] discussed in Section 2.1. Before we can understand and assess the elements which constitute software safety practice, we must first be able to describe and represent each element of practice as comparable models that describe each element as accurately and as simply as possible. Before we recommend any changes to software safety practice, we must be as confident as possible that it has identified genuine impediments which are appropriately-distal. This requires a representation of what constitutes software safety practice, and an understanding of the relationships between the disparate elements of practice.

Framework Design Decision: Define a framework which articulates the disparate elements of software safety practice and their interrelationships.

The first decision in designing the framework was to decide how the elements of software safety practice can be broken down in a way that makes them amenable to modelling and assessment. A number of approaches to doing this were identified in the literature review in Chapter 2.3, and the proposed framework employs the three identified elements of software safety practice, spanning the idealised concept of what best practice *should* be, the manner in which it is imparted to practitioners, and the reality of practice as carried out by practitioners. This framework is shown in Figure 3.1, where each of the elements, along with the relationships between them is shown:

- Software Safety Practice **as-desired**
- Software Safety Practice **as-required**

- Software Safety Practice **as-observed**.

We now consider in turn each of these elements of software safety practice identified in Chapter 2.3.

3.1.1 Software Safety Practice As-Desired

In the instantiation of this framework, a project will determine and assert its own version of software safety practice as-desired. This as-desired practice may be defined by establishing and expressing the objectives by which it will deliver a product that is safe to operate within a stated system and given operating environment. Such objectives will likely be influenced by the organization's risk appetite, and legislative and regulatory obligations; and the need for an organization to simultaneously minimize expenditure and maximize profitability. An organization's version of as-desired practice may or may not be documented by the organization. Understanding and modelling an organization's as-desired software safety practice does not guarantee that practice is complete, nor correct; rather it provides a mechanism to assess its 'goodness' — as we will demonstrate.

Software safety practice as-desired is an idealized representation of what software safety practice is to achieve. The 4+1 Principles are an example of what could constitute software safety practice as-desired, because they are claimed to be "constant across domains and across projects, and can be regarded as the immutable core of any software safety justification" [49]. The use of these principles is further justified as they continue to be widely adopted for use in system safety — including their incorporation as the overarching principles and objectives of UK Defence Standards such as [105].

Each of the 4+1 Principles are considered in turn, and we have defined a set of measurable criteria for each principle. Software safety practice must be capable of meeting each criterion to achieve compliance with that principle. To describe the decomposition of the design, the concept of design 'tiers' [51] are used. 'Tier n' describes the current design level, and 'tier n-1' and 'tier n+1' the previous and subsequent levels of design decomposition respectively. Each of the 4+1 Principles are considered in turn, and the criteria is defined for each.

The criteria for the 4+1 Principles serve two purposes. Firstly, they are offered as a unique contribution of this thesis, and are offered as a reasonable pattern which an organization may adopt and adapt further. The second purpose is to use the criteria to explicate the framework and process to understand and assess software safety practice (see Chapter 5).

Principle 1
The first principle states that "*software safety requirements shall be defined to address the software contribution to system hazards*". Software safety practice must therefore demonstrate that:

- A clear description of the software in the system will be provided
- The operating context of the system in which the software resides will be described
- A clear description of the system in which the software resides will be provided
- The system hazards to which software may contribute will be identified
- The specific failure modes by which software contributes to the identified system hazards will be described
- The software contribution to the identified system hazards will be acceptably managed through the elicitation of software safety requirements that specify the required behaviour(s); for each identified software contribution, to each system hazard
- All software safety requirements will be atomic, unambiguous, defined in sufficient detail, and be verifiable.

Principle 2
The second principle states that "*the intent of the software safety requirements shall be maintained throughout requirements decomposition*". Software safety practice must therefore demonstrate that:

- For each tier of design abstraction, the software safety requirements derived at the previous tier will be adequately addressed:

  (a) Either instantiated into, and mitigated by a design realization, OR

  (b) Further software safety requirements derived for implementation at tier n+1.
- The design for each tier of abstraction will be fully defined and understood
- Software safety requirements will be adequately allocated, decomposed, apportioned, and interpreted at each subsequent tier of design abstraction
- Software safety requirements at tier n will adequately capture the software safety requirements (and their intent) from tier n-1
- Design decisions taken at tier n will be appropriate to ensure that the software safety requirements at tier n-1 are maintained in the context of the potential hazardous failures identified at tier n-1 (for each design decision made)
- The balance of design decisions and derivation of further software safety requirements will be appropriate for the software safety requirements (and design as applicable) at tier n-1
- The design of tier n will be considered when defining the software safety requirements for tier n

- Full (forward and backward) traceability will be made through each tier of decomposition of software safety requirements
- Software safety requirements will be predicated on a reasonably stable design (at the earliest point in the design lifecycle)
- Software safety requirements will be re-validated at tier n following any resultant design changes at tier n.

Principle 3
The third principle states that *"software safety requirements shall be satisfied"*. Software safety practice must therefore demonstrate that:

- All software safety requirements will be verified as being fully instantiated into a design realization
- The appropriateness of the evidence set with respect to the software safety requirements at each tier of design will be demonstrated
- The sufficiency of the evidence set with respect to the software safety requirements at each tier of design will be demonstrated.

Principle 4
The fourth principle states that *"hazardous behaviour of the software shall be identified and mitigated"*. Software safety practice must therefore demonstrate that:

- Potentially hazardous failure modes of the software will be identified (i.e. in normal operation)
- Potential additional hazardous contributions at each tier will be identified (i.e. in faulted conditions)
- Potential additional hazardous contributions at each tier will be mitigated through the derivation or elicitation of further software safety requirements
- Appropriate software safety requirements will be elicited in mitigation of all identified potentially hazardous failure modes
- Design errors that could cause hazardous failure modes will not be introduced during the design process
- The design (and software code) will be analysed to ensure it does not contain design errors that could cause a hazardous failure mode
- Any changes to the design will not introduce potentially hazardous design (or code) errors.

Principle 4 + 1
The fifth principle states that *"The confidence established in addressing the software safety principles shall be commensurate to the contribution of the software to system risk"*. Software safety practice must therefore demonstrate that:

- The required confidence behind the attainment of each Principle (1 to 4) will be determined
- The most effort in generating evidence will be focussed on the areas with the highest risk from software's contribution to hazards (noting that the areas denoted as requiring the most effort are currently indicated in Open Standards by the notion of integrity/assurance levels)
- For each Principle (1 to 4), the required confidence that each has been met will be reflected in:

  (a) The appropriateness of evidence

  (b) The trustworthiness of each evidential artefact (the rigour in the approaches to be used):

    (i) Independence

    (ii) Resources and required attributes (personnel)

    (iii) Techniques and Methods used

    (iv) Audits and reviews

    (v)Tools

  (c) The type of evidence to be used.
- An understanding of the limitations with each type of evidence being used will be clearly understood.

If any alternative version of software safety practice is not expressed as a set of measurable criteria, gaps or ambiguities regarding the intent of as-desired practice may exist. In such cases, it is important to note that any lack of detail in a project's as-desired model may be an oversight, but it may also be deliberate — reasonably relying on other elements of software safety practice to add the detail.

3.1.2 Software Safety Practice As-Required

As noted in Chapter 2, there are two elements of as-required software safety practice. The first element is that represented in Open Standards such as [147] or [13]. Standards such as these prescribe a set of lifecycle activities that are argued to represent good practice. The second element is those practices described by organizational processes (Closed Standards), and these may, or may not have been designed as a means to implement the described lifecycle of a specific Open Standard. Our framework is designed to allow both ways to be represented, and any relationships between them (and other elements of practice) to be evaluated.

3.1.3 Software Safety Practice As-Observed

Software safety practice as-observed is the element of software safety practice carried out by practitioners. Our framework is designed to represent this element of practice, and facilitates an evaluation of as-observed practice's relationships with the other elements of software safety practice.

3.1.4 The Framework

All the constituent parts of software safety practice can be mapped onto the three elements of practice we have identified (as-desired, as-required, and as-observed), and whilst it is argued this mapping is necessary, it cannot yet be argued whether this is complete - although further instantiations of the framework will reveal the levels of confidence in its completeness.

It is a reasonable challenge to argue that all of these elements of software safety practice (i.e. practice as-desired, as required, and as-observed), once represented in text or a graphical representation, are simply a form of 'Work as Imagined' [58] (the as-required process that one 'imagines' is carried out for example). Whilst this is a reasonable challenge to the framework, the transformation of practice into accurate, comparable models is a necessary compromise if we are to understand and assess software safety practice.

The main elements of software safety practice, and their relationships are shown in the framework in Figure 3.1. Each number on the framework denotes either a representation of an element of practice, or a relationship between elements. Each element and relationship is defined below.

The Framework:

1. **Desired:** the as-desired representation

2. **Required (Open):** representation of an Open Standard

3. **Required (Closed):** representation of a Closed Standard

4. **Observed** representation of practice as-observed

5. **Required (Closed) v Desired:** comparison of an organization's software safety process with software safety practice as-desired

6. **Required (Open) v Desired:** comparison of an Open Standard with software safety practice as-desired

7. **Observed v Required (Closed):** comparison of observed practice with the organization's software safety processes

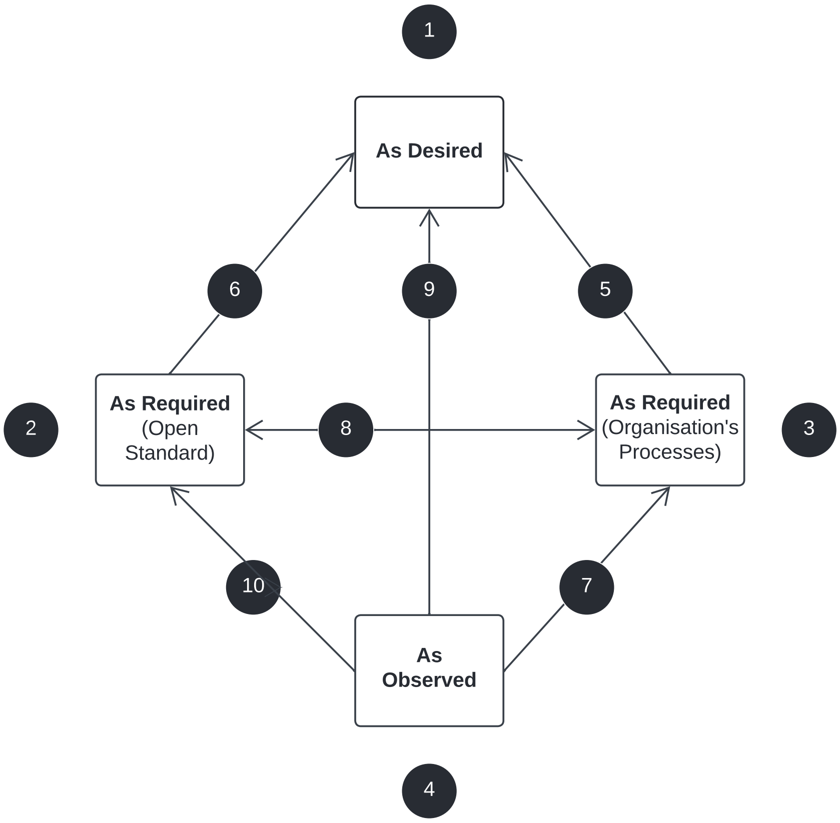


Figure 3.1: The Elements of Software Safety Practice.

8. **Required (Closed) v Required (Open):** comparison between the organization's software safety process and the Open Standard which may have informed its development

9. **Observed v Desired:** comparison of observed practice with software safety practice as-desired

10. **Observed v Required (Open):** comparison of observed practice with an Open Standard.

An activity is required to translate as-desired software safety practice into software safety practice as-required (Closed). This activity is sector, application, and technology specific. It will not be considered in our proposed framework.

Framework Design Decision: The translation of as-desired software safety practice into software safety practice as-required (Closed) is not considered in the framework.

Having established the framework of software safety practice, we next need to decide what needs to be included in the representations of each element of the framework.

Framework Design Decision: Define what must be included in each representation of practice, and all artefacts employed by each activity.

Instantiating the framework requires the creation of models that are a faithful representation of the key elements of software safety practice, but which are also as simple as possible. One of the weaknesses of many software safety process lifecycles is that although they portray the activities to be undertaken, they do not consider the attributes of activities such as timing constraints [50], timing requirements [163] commercial/contractual complexities [152], [141], the resources required to undertake the activities (nor the attributes thereof), nor the intricate interrelationships and interdependencies between activities. These weaknesses must be mitigated by our process. We also noted in Chapter 2.4 that the traditional 'V-Model' lifecycles were no longer valid, and that people were not necessarily following them anyway.

Framework Design Decision: The framework must be capable of representing any type of lifecycle (including those employed in 'Agile' lifecycles (see [140] for example)).

Framework Design Decision: The framework must be capable of representing all activities undertaken within an element of software safety practice.

As well as modelling the activities within each element of practice, and the artefacts employed by each activity, we can look to the ever-increasing body of knowledge on the attributes required to successfully complete each activity (see [57] for example). We cannot yet argue these attributes are a complete list of those required, but take confidence from this body of knowledge. Using this body of knowledge, we argue that the framework must be capable of modelling the following:

- Resources consumed by the activity (which includes both human resource and materiel)
- Inputs to the activity
- Output from the activity
- Methods/techniques that can be used to carry out an activity
- Controls that constrain or define the activity
- Time by which the activity must take place.

Framework Design Decision: Define the required attributes of each activity and artefact.

We need to provide a means of assessing and evaluating software safety practice. When considering conformity, '(lifecycle) Assessment' is defined by the ISO as a "compilation and evaluation of the inputs, (and) outputs...of a product system throughout its lifecycle" [65]. 'Evaluation' is subsequently defined as a "systematic examination of the extent to which a product, process or service fulfils specified requirements" [64]. As such we argue that the framework must therefore also be capable of:

- Representing the links between activities
- Representing levels of agreement (e.g. between work as-required (Open) and work as-required (Closed))
- Representing levels of compliance/conformance (with a standard for example)
- Representing identified deficiencies
- Describing optionality and multiplicity
- Quality attributes, including time (expressed as either a calendar date or phase in the lifecycle); personnel attributes including qualifications, experience, independence, authority, and role; and data format and contents.

As the model of practice progresses through activities, a level of abstraction is reached at which the inputs to, and/or outputs from an activity are merely artefacts that are consumed or produced by an activity. Examples of this would be a standard guidance document that controls how an analysis such as a HAZOP is to be undertaken, or a Project Management Plan that is not under the control of the safety team. For artefacts we expect to model the following attributes which are needed to meet the 'assessment' and 'evaluation' capabilities noted above:

- Human Resources
- Methods/techniques
- Data inputs (to activities)
- Data outputs (from activities).

Each of these attributes require specific quality criteria and time constraints to be defined for them if we are to assert/assess software safety practice as required. The required attributes of an artefact are discussed in Chapter 3.2.1, and the means of representing artefacts are shown in Chapter 4.5.

It is possible that assessments of software safety practice may reveal instances where the existence of an artefact is not explicitly stated by the element of practice under analysis. It is also possible that instances may be encountered where an artefact is required as part of the lifecycle, but where no producing activity for that artefact is explicitly stated. We propose to categorize these as *Inferred Activities*. Artefacts may also be inferred, and we have established three categories of modelled artefacts:

- **Explicit:** artefacts that are explicitly described, and have a consuming or producing activity that is clearly stated
- **Inferred:** artefacts that are discussed without any consideration of their creation or management, and with no consuming or producing activity that is explicitly stated. (e.g. if a standard says that "assumptions must be managed", we can infer the existence of an 'Assumptions' artefact)
- **Orphan:** artefacts that are explicitly described, but have no stated activity that produces them.

The means by which different types of artefacts are modelled is outlined in Chapter 4.5.

This concludes our discussions on our theoretical framework, but any theory needs to be tested. As such we need to ensure the framework design supports widespread use.

Attention now turns to the process which instantiates the framework.

3.2 Process to Understand & Assess Software Safety Practice

The framework introduced in Section 3.1.4 has ten elements constituted by four different representations of practice and their interrelationships. In this section, the process to apply each of these elements of the framework in order to understand and assess software safety practice in a particular project is described (addressing RQ1 and RQ2). To instantiate the framework, the process requires four modelling steps (1-4) and six analysis steps (5-10). Each analysis step takes a baseline model from Steps 1-4 and annotates it. Finally, there is a further activity not represented in Figure 3.1, and this concerns the maintenance of the process outputs throughout the life of a product/project. The maintenance of the created models is considered in Chapter 3.6.

To apply the framework for a particular software safety project the following modelling, comparison and annotation activities must be undertaken. The modelling process steps are provided in Chapter 3.2.1, and the means by which comparisons are to be annotated are provided in Chapter 4.5. Each activity is annotated to indicate the research question it seeks to address:

1. Model and represent the organization's as-desired model (RQ1)
2. Model and represent the relevant Open Standard (RQ1)
3. Model and represent the organization's software safety processes (as-required (Closed)(RQ1)

4. Model and represent software safety practice as carried out (as-observed) (RQ1)

5. Compare the organization's software safety processes (as-required (Closed)) with the as-desired model (RQ2)

6. Compare the selected Open Standard with the as-desired model (RQ2)

7. Compare software safety practice (as-observed) with the software safety processes formulated by the organizational lifecycle (as-required) (RQ2)

8. Compare the organization's software safety process (as-required (Closed)) and the Open Standard which informed/influenced its development (RQ2)

9. Compare software safety practice (as-observed) with the organization's as-desired model (RQ2)

10. Compare the organization's software safety practice (as-observed) with an Open Standard (RQ2).

Particular note should be taken of Step 9 which indicates that the assessment should compare software safety practice as-observed directly with respect to the as-desired model (rather than just with what is required by processes or standards). By performing this step, an analyst can identify whether software safety practitioners, through deviations from the defined processes and standards, are overcoming deficiencies in the as-required practice in order to improve safety practice (intentionally or otherwise). Step 9 ensures the assessment identifies any subtle improvements made by those instantiating an organization's software safety processes over what is specified by the as-required model. By considering all elements of software safety practice (and the interrelationships between them) our novel process is able to also identify, and mitigate the risk of unintentionally undermining safety practice by implementing 'fixes' which have negative impacts elsewhere in practice (RQ3 and RQ4).

Step 10 of Figure 3.1 is in place as it is entirely possible that software safety practitioners charged with implementing as-required practice may appeal to the requirements or guidance from an Open Standard they are familiar with rather than the organization's own software safety process. Practitioners could do this to recover shortfalls they perceive in the as-required processes, or could simply be a default to a standard they know well. Any identified differences or deviations between elements of practice will enable an assessment of whether, and how, software safety guidance and/or practices need to change.

3.2.1 Modelling Software Safety Practice

Before the process steps themselves are given, we first consider each element of software safety practice and describe what needs to be modelled for each element of practice, and how this is to be achieved. We then describe in Section 3.2.2 the

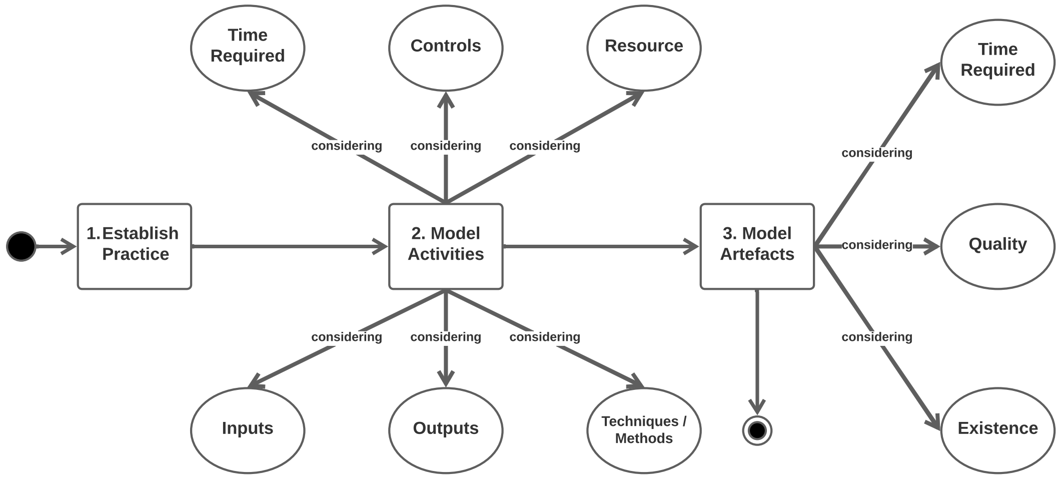


Figure 3.2: Modelling Safety Practice

activities required to facilitate an assessment of these models of practice, and their interrelationships. We now explain how the elements of software safety practice are to be modelled, along with the six attributes of each activity which must be defined. The first part of this process is illustrated in Figure 3.2, and concerns the modelling of software safety practice as-desired and as-required.

1. **Establish As Desired or As Required Practice**: This involves identifying standards / organizational practice, and as-desired practice relevant to the software safety practice being evaluated.

2. **Model Activities**: By scrutinizing the publication or criteria it is possible to identify and model all activities required by the lifecycle. Once these are modelled we can represent the aspects of each process activity (see Chapter 4.5):

   - Techniques/Methods: means by which an activity is fulfilled (e.g. carrying out a SHARD analysis)
   - Inputs/Outputs: stipulated inputs to and outputs from each required activity (for example design data as an input to an analysis, and a report as an output therefrom)
   - Time: expressed as the point by which the activity should start and/or be complete by (or perhaps not start until). This may be expressed as a calendar date, a dependent activity, or phase in the programme
   - Controls: aspects that control how an activity is undertaken (e.g. a recognized Open Standard that controls how functional safety analysis is to be undertaken)
   - Resources: person(nel) required to undertake the task, and any materiel required to complete it.

3. **Model Artefacts**: Artefacts represent the lowest level of abstraction, and are modelled as inputs to / outputs from an activity. Artefacts are deliverables or items that support or constrain an activity, or are produced as a result of an activity. Examples of artefacts include a description of the resource required in support of an activity, or a report produced as a result of an activity.

When discussing artefacts in the Structured Assurance Case Metamodel [106], OMG refer to the requisite properties of artefacts such as:

- Quality (completeness, consistency etc.)
- Its lifecycle / temporal properties (creation and / or modifications events)
- Relationships and dependencies (to and from the artefact)
- Resources expended in the production of an artefact
- Activities related to the management of artefacts, and
- Techniques associated with the production of the artefact.

The UK MoD's Acquisition Safety and Environmental Management System (ASEMS) [104] considers the characteristics of Timing, Independence, Method, and Personnel when recommending how 'proportionality' will be measured across acquisition projects.

Specifying the quality attributes of objects, or the characteristics of projects may not be sufficient for modelling software safety practice artefacts however, as we have already noted that the process needs to model:

- Human Resources
- Methods/techniques
- Data inputs (to activities)
- Data outputs (from activities).

All of these attributes require specific quality criteria and time constraints to be defined for them when representing software safety practice. The required quality criteria for aspects will differ - as illustrated by the examples given in Table 3.1 (noting that Time constraints are required for all – and is therefore not included in this table).

As can be seen in Table 3.1, the kinds of attributes which need to be defined for software safety practice artefacts are wide-ranging, differ on the type of artefact, and are not always required or relevant (for all artefacts). Defining every

Aspect	Quality Criteria	Examples
Human Resource	Qualifications	Safety Engineer with MSc
	Experience	5 Years experience
	Independence	Not involved in the design
	Authority	Authorised signatory
	Role	Software Safety Consultant
Method/ technique	Relevance (to an activity)	ISO Standard for HAZOP
	Approved/ Recommended status	Formally Issued at Rev C
Data (inputs/ outputs)	Format	DOORS export in Excel
	Specific Contents	Requirements Specification Measures of Performance Maturity Level
	Generating Resource	Requirements Engineer
	Receiving Resource	Owner

Table 3.1: Aspects and Quality Criteria

quality criteria for each artefact would be cumbersome in modelling terms, and many criteria may not be required for specific instantiations (i.e. 'Format' would not be required for a 'Resource' artefact).

Whilst activities are modelled with the inputs and outputs thereof, having both an input AND an output aspect of an artefact would be superfluous as we only need to represent the existence of the artefact – which can itself be assessed and analyzed using a pertinent colour scheme. To ensure an analyst can adequately model the required aspects of all types of artefact, the following aspects are modelled:

- **Time ('T'):** expressed as the point by which the artefact should be created by or supplied to an activity. This may be expressed as a calendar date or phase in the programme
- **Quality Criteria 'Q'):** quality attributes required of an artefact, such as the skills and experience required of the person charged with carrying out an activity, or the format and contents required of a report
- **Existence (positive/negative) ('E'):** does the artefact (yet) exist? This attribute is used to consider whether the artefact needs to be produced ahead of the supported activity (and therefore whether another activity should be modelled to create it; or a dependency placed on a department other than Software Safety Engineering); or whether a person exists within the project who has the requisite skills or independence when considering a 'Resource' artefact.

The accuracy of the represented model determines the robustness of the resultant assessment, and so it is vital that any model be as accurate and complete as possible. Organizations may have competitive advantage, or security concerns that leads them to withhold parts of their software safety processes from the analyst. Such instances must be explicitly disclosed by the respondent, along with an assessment from the analyst as to whether the strength of any inferences made on the model(s) are impacted by the absence of such data.

The second part of the modelling process considers software safety practice as-observed.

Software Safety Practice As Observed

Software safety practice as-observed represents the observed practice of practitioners performing software safety activities. Rather than relying on a suite of documents, modelling software safety practice as-observed requires a form of ethnographic study [109] of actual software safety practice. To our knowledge however, such ethnographic studies on the work of software safety practitioners are not carried out due to the substantial time required (dependent on the type of practice, the technology involved, and the length of the project/programme); the need for impartial and independent observers; and/or the cost of such an undertaking.

Should a full ethnographic study be infeasible, then an alternative method is to employ a series of interviews. Care must be taken if this approach is taken however, as this element of safety practice can no longer be considered as software safety practice as-observed, and instead morphs into safety practice "as-disclosed" [150], and presents many opportunities for bias to skew the data ([151], [169], [90]).

3.2.2 Assessing Software Safety Practice

Having modelled the relevant aspects of software safety practice, an assessment is performed to identify deficiencies in practice, and impediments to the adoption of software safety practice as-desired. The assessment process is illustrated in Figure 3.3, and the means of representing the outcome of the assessment is given in Chapter 4.5.

Assessing the Internal Completeness and Internal Consistency of Software Safety Practice As Required

This assessment has the following two steps:

1. **Evaluate Activities**. Each activity in as-required practice is assessed for:
    - Completeness and Consistency: are there enough activities commensurate with achieving the required outcome; and does each activity

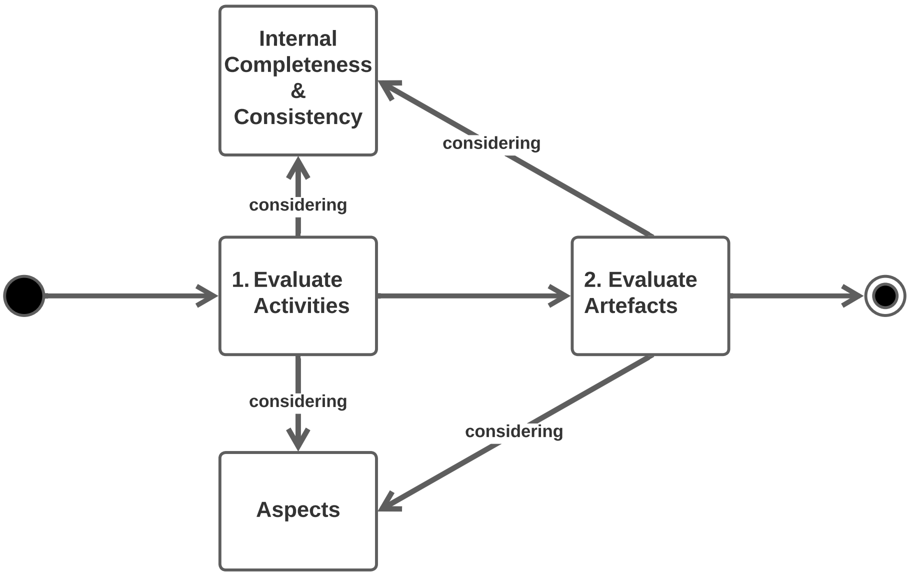


Figure 3.3: Assessing Safety Practice

have sufficient supporting sub-activities to ensure it can be completed to a sufficient level?

For example, to produce the artefact 'Software Common Cause Analysis Report', does the modelled software safety practice identify all activities that are reasonably required to produce it? Examples here would be 'Failure Modes Analysis', 'Assess Levels of Independence' and perhaps 'Carry out SHARD Analysis'. An analyst with an understanding of the activities required to produce such a software safety assessment will be able to assess whether it is reasonable that the safety assessment will be produced. Although this is perhaps a subjective assessment, it is an early indication of the sufficiency of the safety practice under review.

- Consideration of Aspects: is there sufficient detail in the description of the attributes required of / produced by the identified activities to have confidence that sufficient consideration is given to Inputs, Outputs, Time, Techniques and Methods, Controls, and Resources?

For example, the activity 'Review Software Safety Requirements' should have attributes that denote the time it should be completed by; a defined set of inputs and outputs; the techniques and methods by which the Software Safety Requirements are to be reviewed; any controlling procedures (such as a Process Instruction); and the resources expended by the activity.

2. **Evaluate Artefacts**: Each artefact is assessed for:

- Completeness and Consistency: are there enough artefacts to enable successful completion of all activities; does every activity produce an artefact; and does each activity have an adequate set of required inputs and outputs?

Many Open Standards do not explicitly document the specific artefacts that are consumed by an activity, but merely infer them (such as stating that the user should "check the validity of any assumptions", without ever referencing where such assumptions are derived from, nor what should be the arbiter of 'validity'. Open Standards also often fail to consider who needs an artefact, and in what format. This may not be an inadequacy on the part of the standard, however - which may reasonably rely on those implementing the standard's processes to consider such details. Whether this is a deliberate and reasonable assumption will be revealed through the modelling of organizational processes, and observations of practice.

- Consideration of Aspects: is there sufficient detail of the attributes to denote when they need to be produced or used; are sufficient quality attributes considered; and does the artefact yet exist, or does it need to be created (in which case further producing/consuming activities may need to be modelled)?

Whilst standards and organizational process denote the production of artefacts, they do not always specify when the artefact is needed. For aspects that are required inputs to activities, there isn't always consideration of when the artefact is needed by, who produces it, nor to what quality (such as format). On the occasion that resources are mentioned, the quality attributes of the resource are often not stipulated (such as qualifications, training, experience, and independence). Future analysis of organizational processes and Open Standards (subsequent steps in the process) will uncover whether this is an acceptable omission from an as-required perspective, and follow-up interviews with practitioners/management will reveal whether such potential shortcomings are overcome - intentionally or otherwise (RQ3 and RQ4).

Assessing Software Safety Practice As Required

An analyst next needs to compare the relationships between software safety practice as-required with software safety practice as-desired, so an assessment is undertaken to consider the levels of compliance between the as-required model and software safety practice as-desired (Steps 5 and 6 of Figure 3.1), as well as (where applicable) between the as-required (Closed) model and the model of the applicable Open Standard (Step 8 of Figure 3.1). When assessing the levels of compliance, we recommend the adoption of the following colour-coding scheme (further details for which are provided in Chapter 4.5):

- **GREEN**: The considered aspects of the activity meets the claims required of / agrees with the comparison model in full
- **AMBER**: The considered aspects of the activity only partially meets claims required of / partially agrees with the comparison model
- **RED**: The considered aspects of the activity meet no aspect of the claims required of / does not agree with any of the comparison model.

Assessing Safety Practice As Observed

Once the analyst has assessed as-required safety practice, they can then assess safety practice as-observed. Primarily, they should do this in terms of its its relationships with software safety practice as-required and as-desired. They can, again, use colour-coding to represent the level of consistency between as-required (Closed *and* Open) and as-desired practice:

- **GREEN**: The considered aspects of the activity agrees with that of software safety practice as-required/as-desired in full
- **AMBER**: The considered aspects of the activity partially agrees with that of software safety practice as-required/as-desired
- **RED**: The considered aspects of the activity does not agree with any part of software safety practice as-required/as-desired.

Once the analyst has assessed the as-observed practice in this way, they can then hold follow-up interviews with representatives of the organization whose processes are under analysis in the context of the respondent's industry sector, organizational hierarchy, and the product being created by the respondent's organization (RQ3 and RQ4). These follow-up interviews should aim to identify:

- The reasons for limited areas of agreement
- The reasons why there are areas of no agreement
- Evidence regarding the validity of the as-desired model (is the as-desired model complete and correct?)
- Any difficulties or complexities behind the areas of limited or no agreement
- Whether and why any areas of limited/no agreement contribute to meeting any perceived shortfalls with an organization's processes with respect to an Open Standard (Step 8 of Figure 3.1)
- Whether and why any areas of limited/no agreement contribute to meeting any perceived shortfalls with an organization's processes and that of the as-desired model (Step 9 of Figure 3.1)

As a result of the follow-up interviews, the analyst must determine whether and how the models and subsequent evaluations are challenged, and whether they need to be modified as a result of new information. These follow-up interviews are out of scope for this thesis (Research Questions 3 and 4).

Attention now turns to the Process Steps themselves. Each step of the process requires a Suitably Qualified and Experienced Person (SQEP) to undertake it. The argument of what constitutes a SQEP individual is outside of the scope of this research, however (see Recommendation 1).

3.3 Process Steps

This section now considers how each element of software safety practice described in the framework is transposed into explicit steps for an organization to understand and assess software safety practice. Each element of software safety practice framework is considered in turn (from 1 to 10). For each step we define its Objective(s), the Input(s), the Task(s), and its Output(s).

3.3.1 Step One: Create a Model of Software Safety Practice As Desired 1

We have already provided the criteria for software safety practice as-desired in Section 3.1.1. This process is applicable only if an organization wishes to determine an alternative model of as-desired practice.

Objective
Define software safety practice as desired for the organization wishing to understand and assess their software safety practice (RQ1, RQ2).

Inputs
An organization wishing to model software safety practice as-desired requires the following inputs:

- A Software Safety Philosophy
- A Risk (acceptance and tolerance) Policy
- A Software Safety Management Philosophy (which may be instantiated as a Management System and Plan(s))
- A suitably qualified and experienced Safety Manager to determine, represent and agree with the product owners the model of as-desired practice.

Tasks

1. Define software safety practice as-desired
2. Create a tangible and measurable representation of software safety practice as-desired.

Output
The output of this step is either a set of objectives, or a set of measurable criteria which can then be used to assess the other elements of safety practice for compliance.

Discussion

Asserting what constitutes software safety practice as-desired is perhaps the most challenging and complex part of understanding any aspect of software safety practice, as discussed in Chapter 2.3.1. Notwithstanding the complexities and challenges of this task, the organization must create a representation of as-desired practice which is both tangible and measurable.

It is important to note that any ambiguities in an as-desired model **may** be an oversight, but it may also be deliberate - reasonably relying on practice at the as-required or as-observed level to remove them.

This process to understand software safety practice does not guarantee that an organization's as-desired practice will be complete, nor correct; rather it provides a mechanism to assess its 'goodness'.

3.3.2 Step Two: Create a Model of Software Safety Practice As Required (Open) 2

Objective

There are two ways in which software safety practice as-required is currently modelled and represented in industrial practice. The objective of this step is to model the as-required lifecycle of an Open Standard.

Input
The single input for this task is the Open Standard which may have influenced the creation of organizational practice.

Tasks

1. Identify all the activities and produced / consumed documents and other artefacts required by the standard
2. Identify the sequences of linked activities and artefacts
3. Represent graphically the lifecycle required by the standard as a sequence of linked activities and artefacts
4. Compile a report which defines the:
   (a) Modelling process used
   (b) Modelling symbology used
   (c) Location of the model, and any proprietary software required to access it.

Outputs
Two outputs are created by this step:

1. A representation of the as-required (Open) model
2. The report accompanying the as-required (Open) model.

3.3.3 Step Three: Create a Model of Software Safety Practice As Required (Closed) 3

Objective

The second way in which software safety practice as-required is currently defined is that generated by a project in its 'Closed' Standard. The lifecycle of activities expressed in organizational practice may, or may not have been influenced by / designed as a means to implement the prescribed lifecycle of a specific Open

Standard. The objective here therefore, is to model the set of lifecycle activities described by the organizational processes and procedures (RQ1).

Input
The single input for this task is the Closed Standard which constitutes organizational practice.

Tasks

1. Identify all the activities and produced / consumed artefacts required by organizational practice (as specified in the organization's Closed Standard)
2. Identify the sequences of linked activities and artefacts
3. Represent graphically the lifeycycle required by the organization as a sequence of linked activities and artefacts
4. Compile a report which defines the:
   (a) Modelling process used
   (b) Modelling symbology used
   (c) Location of the model, and any proprietary software required to access it.

Outputs
Two outputs are created by this step:

1. The appropriately represented as-required (Closed) model
2. The report accompanying the as-required (Closed) model.

Discussion
The terms ‘Organizational Practice’ and ’practice As-Required (Closed)’ are synonymous.

3.3.4 Step Four: Create a Model of Software Safety Practice As Observed 4

Objective
Software safety practice as-observed is the model of safety work carried out by practitioners. Instead of relying on a suite of documentary articles, software safety practice as-observed necessitates a form of independent ethnographic study [109]. The objective here therefore, is to model the software safety activities carried out by software safety practitioners in a given organization (RQ1).

Input
The single input to this task is an empirical report of as-observed software safety practice as carried out by software safety practitioners.

Tasks

1. Identify all the activities and produced / consumed artefacts carried out by the software safety practitioner
2. Identify the sequence of activities carried out by the software safety practitioner(s)
3. Represent graphically the sequence of linked activities carried out and artefacts produced / consumed
4. Compile a report which defines the:
   (a) Modelling process used
   (b) Modelling symbology used
   (c) Location of the model, and any proprietary software required to access it.

Outputs
Two outputs are created by this step:

1. The appropriately represented as-observed model
2. The report accompanying the as-observed model.

Having identified, modelled and represented the elements of software safety practice, attention now turns to the process to assess software safety practice.

3.3.5 Compare Organizational Practice with Software Safety Practice As-Desired 5

Objective
Organizational practice must be capable demonstrably of complying with software safety practice as-desired. The objective of this step is therefore to assess the degree of compliance between organizational practice and software safety practice as-desired (RQ1, RQ2).

Inputs
Two modelling elements of the framework must have already been completed:

1. The as-required (Closed) model of software safety practice
2. The as-desired model of software safety practice.

Tasks

1. Create a copy of the model of as-required (Closed) practice created at Step 3
2. Using this copy, create a representation of where as-required practice complies with each of the as-desired criteria in turn
3. Taking each model element in turn, evaluate each contributing activity:

   Internal Completeness and Consistency: are the activities correct and commensurate with achieving as-desired practice? Do the right amount of activities exist; and does each activity have the correct amount of supporting / contributing activities to ensure it can be completed to the required level of compliance?

   Consideration of Attributes: is the information stated for the required attributes the correct information (i.e. Inputs, Outputs, Time, Techniques and Methods, Controls, and Resources); and is the correct amount of information given for the attributes for the as-desired practice to be met?
4. For each model element / criteria, evaluate each artefact produced / consumed by an activity:

   Sufficiency: is there the correct number of artefacts to enable successful completion of all activities, and are the artefacts the correct ones? Does every activity produce an artefact; and does each activity have the correct amount and type of artefacts (as inputs) to comply with the model of as-desired practice?

   Consideration of Attributes: is the information stated for the attributes correct (i.e. Time, Quality Criteria and Existence) to denote when they need to be produced or used? Are the correct number and set of quality attributes considered for each artefact, and are they the correct quality attributes for as-desired practice to be complied with?
5. Annotate the newly-created model indicating the degree of compliance with as-desired software safety practice.
6. Should potential deficiencies be evident, then follow-up research with the organization should be undertaken to establish the reasons why (RQ3, RQ4).
7. Compile a report which defines the:
   (a) Modelling process used
   (b) Modelling symbology used
   (c) Location of the model, and any proprietary software required to access it.

Outputs
Two outputs are created by this step:

1. The appropriately annotated model of as-required (Closed) practice compliance
2. The report accompanying the model of as-required (Closed) practice compliance.

Discussion
The means of representing degrees of compliance are evaluated, selected, and asserted in Chapter 4.

3.3.6 Step Six: Compare the Open Standard with Software Safety Practice As-Desired 6

Objective
A published Open Standard must be capable demonstrably of complying with an organization's software safety practice as-desired. The objective of this step is therefore to assess the levels of compliance between an Open Standard and software safety practice as-desired (RQ1, RQ2).

Inputs
For this step to proceed, two modelling elements of the framework must have already been completed:

1. The as-required (Open) model of software safety practice
2. The as-desired model of software safety practice.

Tasks

1. Create a copy of the model of as-required (Open) practice created at Step 2.
2. Using this copy, create a representation of how as-required (Open) practice complies with each criteria of as-desired software safety practice in turn.
3. Taking each model element / criteria in turn, evaluate each contributing activity:

   Internal Completeness and Consistency: are the activities correct and pertinent commensurate with achieving as-desired practice? Do the right amount of activities exist; and does each activity have the correct amount of supporting / contributing activities to ensure it can be completed to the required level of compliance?

Consideration of Aspects: is the information stated for the aspects the correct information (i.e. Inputs, Outputs, Time, Techniques and Methods, Controls, and Resources); and is the correct amount of information given for the aspects for the as-desired practice to be met?

4. For each model element / criteria, evaluate each artefact which is produced / consumed by an activity:

   Sufficiency: is there the correct number of artefacts to enable successful completion of all activities, and are the artefacts the correct ones? Does every activity produce an artefact; and does each activity have the correct amount and type of artefacts (as inputs) to comply with the model of as-desired practice?

   Consideration of Aspects: is the information stated for the aspects the correct information (i.e. Time, Quality Criteria and Existence) to denote when they need to be produced or used? Are the correct amount of quality attributes considered for each artefact, and are they the correct quality attributes for as-desired practice to be complied with?

5. Annotate the newly-created model with the degree of compliance with as-desired software safety practice.

6. Should potential deficiencies be evident, then follow-up research with the project should be undertaken to establish the reasons why (RQ3, RQ4).

7. Compile a report which defines the:

   (a) Modelling process used

   (b) Modelling symbology used

   (c) Location of the model, and any proprietary software required to access it.

Outputs
Two outputs are created by this step:

1. The appropriately represented model of as-required (Open) practice compliance

2. The report accompanying the model of as-required (Open) practice compliance.

Discussion
A Standard's Committee can also be considered as 'an organization' who wishes to understand the current state of their software safety practice.

3.3.7 Step Seven: Compare As-Observed Practice with Software Safety Practice As-Required (Closed) 7

Objective
Software safety practice as-observed may be different to, or the same as software safety practice as-required (Closed). The objective of this step is therefore to compare as-observed practice with organizational practice (RQ1, RQ2).

Inputs
Two modelling elements of the framework must have already been completed:

1. The as-required (Closed) model of software safety practice
2. The as-observed model of software safety practice.

Tasks

1. Create a copy of the model of as-observed practice created at Step 4.
2. Using this copy, compare the levels of agreement between software safety practice as-observed, and software safety practice as-required (Closed).
3. Annotate the newly-created model with the levels of agreement, and the differences between the two models of practice.
4. Should differences be evident, then follow-up research with the project should be undertaken to establish the reasons why (RQ3, RQ4).
5. Compile a report which defines the:
   (a) Modelling process used
   (b) Modelling symbology used
   (c) Location of the model, and any proprietary software required to access it.

Outputs
Two outputs are created by this step:

1. The annotated model of how as-observed practice compares with as-required (Closed) practice
2. The report accompanying the comparison of how as-observed practice compares with as-required (Closed) practice.

Discussion
This step is NOT a compliance check, as there is no assumption made on the completeness and correctness of organizational practice. Assessment of the interrelationships of the elements of this framework will reveal whether and how any element of software safety practice should or could change.

3.3.8 Step Eight: Compare As-Required (Closed) Practice with As-Required (Open) Practice 8

Objective
Software safety practice as-required (Closed) may be different to, or the same as the Open Standard which may have informed its development. The objective of this step is therefore to compare both models of software safety practice as-required (RQ1, RQ2).

Inputs
Two modelling elements of the framework must have already been completed:

1. The as-required (Closed) model of software safety practice
2. The as-required (Open) model of software safety practice.

Tasks

1. Create a copy of the model of as-required (Closed) practice created at Step 3.
2. Using the copy, compare the levels of agreement between software safety practice as-required (Closed), and software safety practice as-required (Open).
3. Annotate the copy with the levels of agreement, and the differences between the two models of as-required practice.
4. Should differences be evident, then follow-up research with the project should be undertaken to establish the reasons why (RQ3, RQ4).
5. Compile a report which defines the:
   (a) Modelling process used
   (b) Modelling symbology used
   (c) Location of the model, and any proprietary software required to access it.

Outputs
Two outputs are created by this step:

1. The appropriately represented model of how as-required (Closed) practice compares with as-required (Open) practice.
2. The report accompanying the comparison of the two models of as-required practice.

Discussion
This step is NOT a compliance check, as there is no assumption made on the completeness and correctness of any specific Open Standard. Assessment of the inter-relationships of the elements of this framework will reveal whether and how any element of software safety practice should or could change.

3.3.9 Step Nine: Compare As-Observed Practice with As-Desired Practice 9

Along with Step 10, this is a conditional step which may not necessarily have an output. The tasks for Steps 9 and 10 are identical, only the rationale behind any identified differences will differ.

Objective
Having completed the model of as-observed software safety practice, and completed the comparison with organizational practice, differences between the two models may have been identified. The objective here, therefore is to determine whether any differences in the as-observed model exist because those charged with implementing an organization's software safety practice are aware of deficiencies in organizational practice with regards to achieving as-desired practice. The step aims to determine whether any differences are additional activities to those required by organizational processes, or whether activities are carried out in a manner other than those required by organizational processes (RQ3, RQ4).

If no examples of additional or differing activities have been uncovered, then there is no action required for this step.

Inputs
Two modelling elements of the framework must have already been completed:

1. The model of how as-observed practice compares with as-required (Closed) practice.
2. The as-desired model of software safety practice.

Tasks

1. Determine whether any differences in the model of how as-observed practice compares with as-required (Closed) practice are a result of practitioners attempting to overcome deficiencies in the as-required practice in order to comply with software safety practice as desired.
2. Conduct further enquiries with the participant(s) in the observation of software safety practice to determine the reasons behind identified differences (RQ3).
3. Create a report that documents the reasons for the identified differences.

4. Carry out further investigations with the project whose processes are under analysis (RQ4).

Output
The single output from this step is a report outlining any differences between as-required practice (Closed) and as-observed practice which specifically aim to overcome deficiencies in the as-required practice in order to comply with software safety practice as desired.

Discussion
Any identified deviations may be owing to reasons other than a motivation to comply with as-desired practice. It is vital that all reasons for deviations are determined and mitigated appropriately (as identified through the other steps of this process).

3.3.10 Step Ten: Compare As-Observed Practice with As-Required (Open) Practice 10

Along with Step 9, this is a conditional step which may not necessarily have an output. The tasks for Steps 9 and 10 are identical, only the rationale behind any identified differences will differ.

Objective
Having completed the model of as-observed software safety practice, and completed the comparison with as-required (Open) practice, differences between the two models may have been identified.

The objective here, therefore is to determine whether any differences in the as-observed model exist because those charged with implementing an organization's software safety lifecycle are aligning practice with an Open Standard other than one associated with their organization.

There are many potential reasons why a practitioner may carry out activities required of an Open Standard (which are not required by, or differ from their organization's practice), perhaps even a standard other than the one which influenced organizational practice. Further data is required to establish whether this is a deviation from as-required (Closed) practice, and the reasons why (RQ3, RQ4).

Input
The model of how as-observed practice compares with as-required (Open) practice must already have been completed.

Tasks

1. Determine whether any differences in the as-observed model exist.
2. Conduct further enquiries with the participant(s) in the observation of soft-

ware safety practice to determine the reasons behind any identified differences.

3. Create a report that documents the reasons for the identified differences.
4. Carry out further investigations with the project whose processes are under analysis (RQ4).

Output
The single output from this step is a report outlining any instances of software safety practitioners undertaking working practice required by an Open Standard not associated with their organization. For these instances to be of concern, the working practices highlighted must be additional and / or different to aspects of as-required software safety practice associated with their organization.

Discussion
Any identified deviations may be owing to reasons other than appeal to an Open Standard. It is vital that all reasons for deviations are determined and mitigated appropriately (as identified through the other steps of this process).

3.4 Information to Action: Next Steps

The information produced by the process provides an organization with valuable insights into the state of its software safety practice. Some of the generated data may reveal the need for further, immediate recovery action, and some data may necessitate further research before any action is taken. Some potential outputs from the process and their implications are considered in Table 3.2 (RQ3, RQ4). Appropriate next steps are also considered. The implementation of these next steps is outside the scope of this empirical research.

Table 3.2: Potential Impediments and their Mitigation(s)

Identified Potential Impediment	**Next Steps**
Non-compliance between As-Required and As-Desired Practice	1. Clear deficiency (i.e. Lack of ‘activity x’ requires the creation of ‘activity x’) 2. Repeat Process Step 2/3 to ensure completeness and correctness 3. Repeat Process Step 5/6 to ensure deficiency has been cleared
Internal consistency deficiencies (insufficient information for activity to successfully conclude/for artefact to be produced)	1. Remove internal inconsistency 2. Repeat Step 2/3 to ensure deficiency is removed

Table 3.2: Potential Impediments and their Mitigation(s)

Identified Potential Impediment	**Next Steps**
Levels of disagreement between a project's process and the Open Standard which influenced its development (Not applicable if the project is a Standard's Committee)	1. Determine the reason for each disagreement considering the following: a. Is there evidence of safety clutter? b. Are the disagreements due to contractual/commercial complexities? c. Are the disagreements reasonable (i.e. is there an option asserted?) 2. Assess the impact of the disagreement in terms of whether this represents: a. An unsafe act b. A necessary deviation c. Surplus work activities 3. Identify potential mitigation options 4. Assess each mitigation option on the ability of the organisation to meet the as-desired criteria 5. Assess each mitigation option for whether an unintended consequence could manifest 6. Select mitigation(s) and implement 7. Repeat Process Steps 8 and 5

Table 3.2: Potential Impediments and their Mitigation(s)

Identified Potential Impediment	**Next Steps**
Non-compliance between As-Observed and As-Required (Closed) Practice	1. Determine the reason for each non-compliance considering the following (not exhaustive): a. Whether the non-compliance is an intentional deviation in order to meet As-Desired practice (i.e. recovering a perceived shortfall in As-Required (Closed) Practice) b. Whether there is a lack of clarity in the As-Required (Closed) Practice (an ambiguity or interpretation issue) c. Whether the non-compliance has an impact on safety (i.e. whether the non-compliance is predicated on removing clutter) 2. Identify potential mitigation options 3. Assess each mitigation option on the ability of the organisation to meet the as-desired criteria 4. Assess each mitigation option for whether an unintended consequence could manifest 5. Select mitigation(s) and implement 6. Repeat Step 7

Table 3.2: Potential Impediments and their Mitigation(s)

Identified Potential Impediment	**Next Steps**
Activities emanating from As-Observed Practice which comply with As-Required (Open) Practice – but which are not mandated by As-Required (Closed) Practice	1. Determine the reason for each activity considering the following (not exhaustive): a. Whether the activity is an intentional act in order to meet As-Desired practice (i.e. recovering a perceived shortfall in As-Required (Closed) Practice) b. Whether the activity has a positive/negative impact on safety 2. Identify potential mitigation options 3. Assess each mitigation option on the ability of the organisation to meet the as-desired criteria 4. Assess each mitigation option for whether an unintended consequence could manifest 5. Select mitigation(s) and implement 6. Repeat Steps 4 (partial) and 10 (as applicable)
Activities emanating from As-Observed Practice which comply with As-Desired Practice – but which are not mandated by As-Required (Open or Closed) Practice	1. Determine the rationale for the additional activities of the as-observed practice 2. Determine whether other activities required by as-required practice comply with the same requirement(s) of as-desired practice (using different activities) 3. Assess the data from #1 and #2 and establish which of the activities would be the most prudent to adopt or cease 4. Repeat Steps 2, 3, or 4 as appropriate

Having extracted the maximum value from the data generated by the process for understanding and assessing software safety practice, attention now turns to the management of the generated data.

3.5 Data, Information, and Knowledge Management

The results of the process must be consumable by the project and associated organization(s). To do this we must ensure that data is transformed to information, and that this information is in a form that can be consumed and turned into knowledge. We use the definitions from [21] to frame this element of the process:

- **Data:** a set of discrete objective facts
- **Information:** a message, usually in the form of a document
- **Knowledge:** a fluid mix of framed experience, values, contextual information, and expert insight that provides a framework for evaluating and incorporating new experiences and information.

Each step of the framework and process has at least one output, and the data created by these outputs must be communicated to the appropriate stakeholders as an information exchange. This information must be absorbed into organizational knowledge if maximum benefits are to be realized.

The process will generate the following data outputs contained in information:

- Models of each element of software safety practice
- Reports validating the creation of each model
- Compliance Models
- Compliance Reports (of elements of practice)
- Comparison Models
- Comparison Reports (between elements of practice).

Each 'message' represents valuable information to an organization, and managed in the right way this information will provide valuable organizational knowledge. Each data source created by the process has different customers with different needs (stakeholders), and this information should be managed and used as suggested in Table 3.3.

Table 3.3: Software Safety Practice Information Management

Data Source	Format	Stakeholder	Purpose
Model of Practice	Source format	Analyst	Maintenance
	Source format	Engineering Peers	Review
	pdf	Manager	Review & Authorisation
	pdf	Regulator / Auditor	Review & Endorsement
Model Validation Report	Office	Analyst	Maintenance

Table 3.3: Software Safety Practice Information Management

Data Source	Format	Stakeholder	Purpose
	pdf	Engineering Peers	Review
	pdf	Manager	Review & Authorisation
	pdf	Regulator / Auditor	Review & Endorsement
Compliance Model	Source format	Analyst	Maintenace
	Source format	Engineering Peers	Review
	pdf	Manager	Review & Authorisation
	pdf	Regulator / Auditor	Review & Endorsement
Compliance Reports	Office	Analyst	Maintenance
	pdf	Engineering Peers	Review
	pdf	Manager	Review & Endorsement Generate Action Plan
	pdf	Regulator / Auditor	Review & Endorsement
Comparison Model	Source format	Analyst	Maintenace
	Source format	Engineering Peers	Review
	pdf	Manager	Review & Authorisation
	pdf	Regulator / Auditor	Review & Endorsement
Comparison Report	Office	Analyst	Maintenance
	pdf	Engineering Peers	Review
	pdf	Manager	Review & Endorsement Generate Action Plan
	pdf	Regulator / Auditor	Review & Endorsement

Whilst the precise format and contents of each message is outside the scope of this thesis, care must be taken by the authors of these messages as to the contents and structure of each. It will be advantageous if maxims such as those created by Grice [37] are followed. Although Grice's maxims were originally created for efficient oral exchanges, they are applicable to all forms of communication and interaction, and not just conversation [98]. Miller discusses Grice's maxims in multi-modal communication terms, and these should guide the author of the data artefacts [98]:

- **Quality**: ensure the information is high quality. Do not say things that you believe to be false, and do not say things for which you have no evidence
- **Quantity**: provide the right amount of information. Be as informative as required, and no more informative than required
- **Relation**: only provide information related to the communication. Be relevant
- **Manner**: avoid obscurity of expression, avoid ambiguity, be brief (avoid unnecessary prolixity), and be orderly.

Avoiding the provision of too much data and associated information is vital if the messages aim to obtain or influence a decision. Too much data may mean that decision-makers can struggle to ignore irrelevant information. Irrelevant information can lead to less optimal decisions (referred to as the dilution effect in [68]).

3.6 Model and Assessment Maintenance

The models generated during the process to understand and assess software safety practice, and the assessments of the models provide an organization with a valuable resource as a safety management tool. We have noted the importance of establishing a model which is as accurate as possible, and this accuracy must be maintained through life for the models to retain their value. The currency and accuracy of the information must be maintained throughout the life of a project, but this should require a minimal maintenance overhead if the accuracy and currency are re-evaluated at regular, predetermined intervals.

Having completed the ten steps of the process, and completed any resultant analysis from the initial findings, a project will have established a 'final' model of each element of software safety practice:

- As-desired
- As-required (Open)
- As-required (Closed)

- As observed.

Table 3.4 considers the events which should trigger a re-evaluation of each model of software safety practice by an organization. The events listed should be considered as the minimum triggers for re-evaluation to which an organization can supplement / modify as required pertinent to their needs. The left hand column defines the triggers, and the remaining columns represent the models of practice. An 'X' in a column denotes the need to re-assess the model stated.

Whilst Table 3.4 highlights the triggers for models of practice to be re-assessed, subsequent steps in the process (i.e. Steps 5-10) must be repeated if the models of practice are amended as a result of the re-assessment. It may be beneficial for an organization to add such trigger events to their Safety Management System, or Safety Management Plan, but such considerations are outside the scope of this research.

Table 3.4: Triggers which Require Re-assessment of the Models of Software Safety Practice

Event Trigger	**As-desired**	**As-required (Open)**	**As-required (Closed)**	**As-observed**
Updated Safety Philosophy	X			
Changes to Legislation	X	X	X	X
Changes in Customer Requirements			X	
Lifecycle Process Changes		X	X	X
Introduction of a new (Process) Tool/Method			X	X
Introduction of a new (Organisation) Artefact			X	X
Change to Competence or Competencies	X		X	
Safety Incident/ Accident	X	X	X	X

In completing this chapter, it is prudent to consider the potential outcomes from the process, and consider what these outcomes may suggest.

3.7 Empirical Research Discussion

An assessment of software safety practice may reveal elements of practice which are in agreement or compliant with each other, or some areas of discrepancies between the different elements of software safety practice may exist. There are many potential reasons for any equivalence or discrepancy, and we now discuss some of these potential outcomes and reasons.

Organizational practice improves on practice required by an Open Standard, or as-desired practice. Should the as-required software safety practice of an organization's lifecycle improve on software safety practice expressed in Open Standards, or the representation of as-desired software safety practice, this may suggest that the practices required of Open Standards or the as-desired model are a cause of impediments to good software safety practice. Alternatively, perhaps the practitioner is implicitly or explicitly aware of the shortcomings of such standards and has evolved local processes in isolation of the standards. In such cases the organization may consider research into improving the as-required (Closed) practice and / or use this as a mechanism to research potential amendments to the practices extolled in as-required (Open) practice (RQ3 and RQ4).

Organizational practice is deficient when compared to the as-desired model. This may indicate that issues with software safety practice manifest in the interpretation of software safety practice as-desired into organization -described processes. Targeted interviews with the organization may indicate where the issues lie (RQ3).

Internal Inconsistency. It may be revealed that software safety practice as-required (by Open and / or Closed Standards) is inconsistent, preventing the software safety practice as-required from ever being successfully completed as software safety practice as-observed. Should this be revealed, it must be highlighted to the organization as part of follow-up interviews, and / or the relevant standard's committee should be notified (RQ3).

Software safety practice as-observed is equivalent to that stipulated in organizational processes. As the assessment process moves from the as-required software safety practice to software safety practice as-observed, software safety practice as-observed may be equivalent to as-required (Closed) practice. Equivalency between organizational practice and software safety practice as-observed will suggest that organizational processes are being fully implemented. However, follow-up interviews as part of evaluating software safety practice as-observed may identify difficulties in implementing the organization's processes, and the existence of such difficulties may suggest issues with the organization's processes exist (RQ3).

Software safety practice as-observed is not equivalent to organizational practice. Should software safety practice as-observed not be equivalent to organizational software safety practice, this may suggest those charged with implementing the organization's processes are aware of potential limitations, inefficiencies, inaccuracies, or unrealistic expectations of their organization's processes and have adopted approaches to compensate. There may even be elements of core and discretionary work. Such discretionary work may be owing to any deliberate vagaries of processes, and made by recourse to what a collective of engineers engaged with them believe is required [5]. There could also be assumptions made by the practitioner, or tensions arising through power relationships [135] that require investigation. Through targeted investigations as part of Steps 7 and 10 of Figure 5.1, it may be possible to identify any impediments or difficulties that have led to a circumvention of process; and ultimately characterize and suggest mitigation research accordingly (RQ3 and RQ4).

Any large gap between software safety practice as-required and software safety practice as-observed could suggest a breakdown in the coordination of the organizational system itself [134], and through follow-up questions and/or further interviews as part of Steps 7 and 10, the aim is to identify, characterize, and suggest research areas that are intended solve or mitigate any impediments or difficulties (RQ3 and RQ4).

Having outlined the framework and process to understand and assess software safety practice, and considered the potential outcomes of the process, Chapter 4 now considers the different means by which the elements of software safety practice can be represented graphically.

Chapter 4

Representing Software Safety Practice

Chapter 3 has described both the framework and the process to understand and assess software safety practice, but we have not yet considered the aspects of the framework and process which need to be represented graphically. The need to model and assess software safety practice graphically is pivotal in supporting Research Questions 1 and 2 (as we noted in Chapter 2.5). This chapter evaluates the current tools which could be used in a graphical representation for the framework and process. Having selected the most appropriate tool, the chapter then describes how the tool is to be used by a project to instantiate the framework. The chapter concludes by describing how the selected tool is used to represent the models, and undertake the assessment of the different elements of software safety practice required (as described in Chapter 3.3).

The current state of the art for graphical modelling is now assessed, followed by the steps taken to identify a suitable graphical notation. The chapter concludes by describing how each step of the process for understanding and assessing software safety practice is implemented using the selected notation.

4.1 State of the Art of Graphical Modelling

A plethora of model-based analysis / engineering tools exists which are perpetually amended and proposed as enhancements or extensions to systems, business processes, and safety engineering techniques. These emanate from a wide-range of academic and industrial sources (such as the examples given in Table 4.1). It is not argued that Table 4.1 represents a systematic literature review of the ever-increasing number of available modelling tools and graphical representations, but it is offered as a reasonable representation of the state of the literature nonetheless.

To compile the data shown in Table 4.1, academic databases (Science Direct, IEEE Explore, Scopus, Springer, ACM Digital Library, and Elsevier) were interrogated using the keywords “Safety Modelling”, “Information Model”, “Requirements Engineering and Safety Standards”, “Responsibility Modelling”, “SPEM Modelling”, “Socio-Technical Modelling”, “FRAM Notation”, “Patterns”, “Requirements Engineering Patterns”, “Process Patterns for Assuring Software Safety”, “Process Patterns for Requirements Evolution”, “Management of Software Safety Requirements”, “Safety Case Patterns”, “Standard Approaches to Information Modelling”, “Business Process Modelling”, “Compliance Models”, and “Modelling Safety Assurance”.

These search terms were selected as a result of the returns from initial searches for ‘process modelling’ and ’graphical representations’, and in combination with both our prior knowledge of software and safety engineering methodologies, and results from the literature reviews carried out early in the research. We judged that these search terms would cast a sufficiently wide net for our purposes. This was NOT intended to be a full, systematic literature review.

The results of these searches are presented in Table 4.1. The “Modelling Notation” column represents the primary or original title of the modelling notation, and the second column denotes known uses of the notation. The third column notes some of the extensions to the original notation, or languages used to complement the original notation which were revealed by the search.

Modelling Notation	Use(s)	Extensions/Languages
Goal Structuring Notation [38]	Safety Cases Assurance Cases Requirements Elicitation	UML [7] Multiplicity Optionality Entity (instantiated/ developed or not)
Bow Tie [24]	Safety Cases Safety Architectures	AdvoCATE [24]
Behaviour Trees [166]	Requirements Management	
NORMATIC [28]	Requirements Management	
i* (Goal based modelling) [131]	Requirements Elicitation Business Process Modelling	CREWS-SAVRE [156]

Modelling Notation	Use(s)	Extensions/Languages
UML UML 2.0	Many, including: Requirements Elicitation (from Use Cases for example) Object-Oriented Modelling Safety Modelling Safety Cases Development Model Driven Engineering Process Lifecycles Information Modelling Cost Modelling Product Line Development [41] Hazard Relation Diagrams [159] Business Compliance Modelling	SysML [8] SafeML [8] SAFE-RE [103] Ecore [22] SACM [106] COCOMO COCOMO II SPEM (see below) SAEM
MMI	Development Lifecycle	EPRAM [18] PSM [102] CMMI-DEV [103] which is itself extended by +SAFE [17] SRP CMMI [103]
IDEF0 [149] IDEF3 [149]	Business Process Modelling (functions)	
DFD [149]	Business Process Modelling (information)	
CoMOn [158]	Business Process Modelling (compliance)	
Fault Trees	Safety/Reliability Engineering	IFFS Model [26]
Event Trees Diagrams [161]	Safety Engineering	Cascade
State Charts	Functional Modelling	
Responsibility Model [88]	Responsibility Modelling	
ORDIT [19]	Responsibility Modelling	
SPEM [107] SPEM 2.0	Software/System Process Engineering Modelling Business Process Modelling	vSPEM [91]
CHESS-FLA (A plug-in within the CHESS toolset) [32]	Failure Logic Analysis	CONCERTO-FLA [32]
SAFE-Net [73]	Accident Modelling	
STAMP [77]	Accident Modelling	STPA

Modelling Notation	Use(s)	Extensions/Languages
FRAM [57]	Accident Modelling Process Modelling	
FRET [34]	Requirements Elicitation	

Table 4.1: Model-Based Analysis Tools

4.2 Graphical Representation Selection Process

In selecting a graphical notation the aim was not to empirically assess the merits of individual tools, but to argue over the suitability of the potential representation for use in modelling and assessing elements of software safety practice. Each modelling option shown in Table 4.1 therefore needed to be assessed for its suitability of use as a graphical representation.

In Chapter 3.1.4 it was noted that the framework must be capable of modelling defined attributes. Any graphical representation must therefore be capable of representing the:

- Resources consumed by the activity (which includes both human resource and materiel)
- Inputs to the activity
- Output from the activity
- Methods/techniques that can be used to carry out an activity
- Controls that constrain or define the activity
- Time by which the activity must take place.

It must be noted that the defined attributes we seek are not specific to system safety activities. The attributes considered are applicable to *any* form of activity, and are in no way related to the specific needs of safety activities.

Having identified a range of potential tools and notations, attention then turned to selecting the most appropriate tool or notation to be used in a graphical representation. We needed a robust selection method which could be used to evaluate each potential tool or notation.

Campos and Almeida [16] note a lack of guidance as to how different modelling options may be evaluated for selection. In response they have developed a somewhat complex scoring / weighting system as a route to model selection. Whilst their methodology may be considered as being 'overly scientific' for the needs of this research, their 6-step process does provide a framework on which to build a selection criteria and evaluation means. Their process involves a 4-point

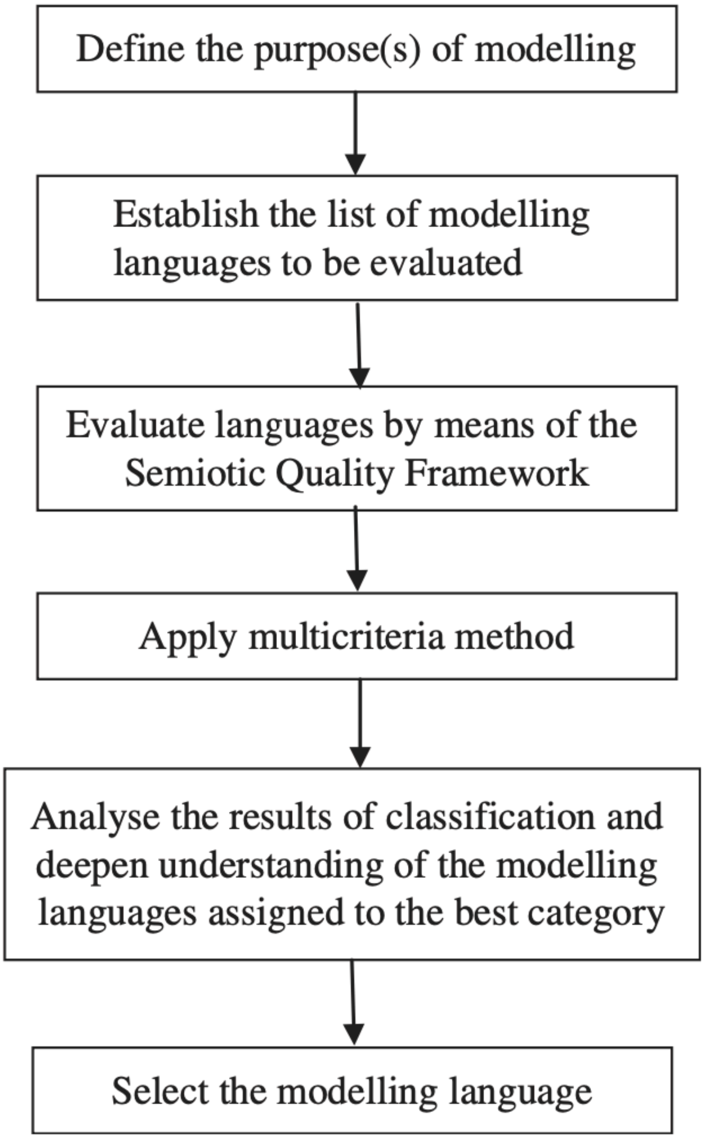

Define the purpose(s) of modelling
Establish the list of modelling languages to be evaluated
Evaluate languages by means of the Semiotic Quality Framework
Apply multicriteria method
Analyse the results of classification and deepen understanding of the modelling languages assigned to the best category
Select the modelling language

Figure 4.1: Framework for Selecting a Modelling Language [16]

scoring mechanism (score of 0 = not supported, 1 = partially supported, 2 = satisfactorily supported, and 3 = very well supported); and a weighting criteria for selection (see Figure 4.1).

Contrasted with the mathematical approach of Campos and Almeida, Kelemen et al [69] suggest a simple 5-step selection process and selection criteria. The process is given as follows:

1. Select a subset of candidate process modelling languages

2. Review the candidate process modelling languages

3. Define the comparison criteria for selection purposes

Goal Structuring Notation	Bow Tie
NORMATIC	CMMI
IDEF0	IDEF3
CoMOn	Fault Trees
Event Trees	State Charts
CHESS-FLA	SAFE-Net
STAMP/STPA	FRET

Table 4.2: Notations Eliminated after Considering Lifecycle Modelling Capability

4. Compare the modelling languages based on the criteria
5. Select process modelling language.

. . . and the suggested selection criteria are:

- Intelligibility
- Coverage of Process Elements
- Ability to express workflow patterns
- Software support
- Portable format
- Widespread use.

Using the required attributes which any graphical representation should be capable of, it was possible to establish a selection process and evaluation criteria by using [69] and [16] as a reasonable benchmark. The selection process is illustrated in Figure 4.2. The selection process considered each required attribute of the graphical representation in turn and is now used to select the most appropriate notation.

Capable of Modelling Lifecycle Activities?

The lifecycle activities are those activities performed within an element of software safety practice. A review of the notations in Table 4.1 resulted in the notations in Table 4.2 being discounted, as they are neither designed for, nor capable of modelling the activities of a software safety lifecycle:

Capable of Modelling Inputs/Outputs?

The remaining notations were then assessed as to whether they were capable of modelling inputs / outputs to activities. Inputs in this regard are the artefacts that enable the activity to occur, and outputs are those artefacts created as a result of the activity. This was not a question as to whether the notation was designed for this purpose – but an assessment as to whether the notation was capable of doing so. As a result, the following notations were discounted as they fail this criterion:

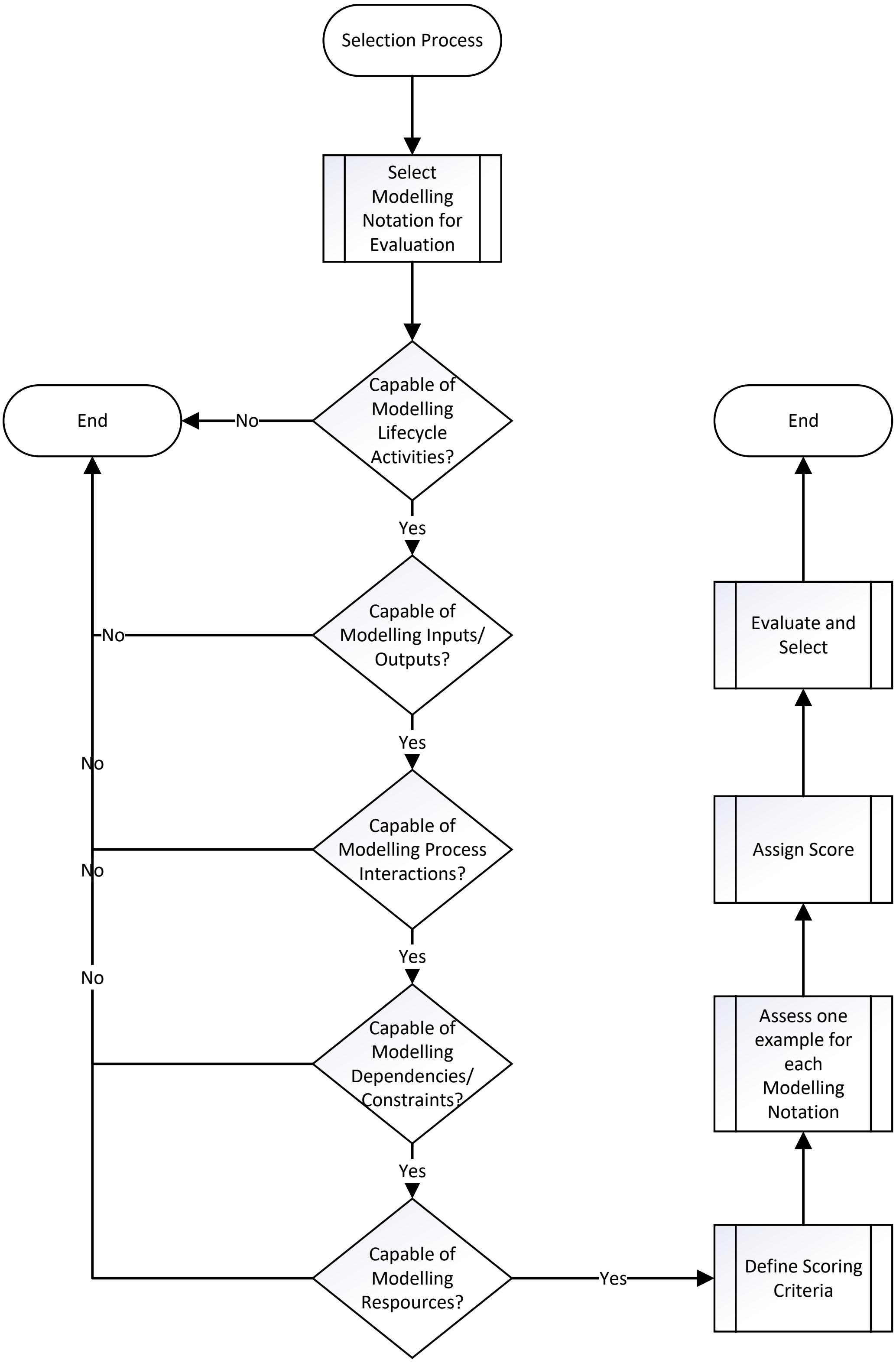

Selection Process
Select Modelling Notation for Evaluation
Capable of Modelling Lifecycle Activities?
No
End
Yes
Capable of Modelling Inputs/ Outputs?
No
Yes
Capable of Modelling Process Interactions?
No
No
Yes
Capable of Modelling Dependencies/ Constraints?
No
Yes
Capable of Modelling Respources?
Yes
Define Scoring Criteria
Assess one example for each Modelling Notation
Assign Score
Evaluate and Select
End

Figure 4.2: Graphical Representation Selection Process

- i*
- Responsibility Models
- ORDIT.

Capable of Modelling Process Interactions?
The previous step assessed whether a notation could model the requisite inputs/outputs of an activity undertaken in an element of software safety practice. This step assessed whether the interactions between different activities could be modelled (timing/sequencing imperatives, precursor activities etc.). As a result, Behaviour Trees (which are only capable of modelling clearly hierarchical or vertical interactions) and the DFD notation (which is only capable of modelling information flow) were discounted as they fail this criterion.

Capable of Modelling Dependencies and Constraints?
Each activity in an element of software safety practice has dependencies that enable the activity to take place (such as data inputs), constraints that affect its conduct (such as quality criteria), and required attributes of its output (deadlines for example). All three remaining notations (UML, SPEM, and FRAM) meet, or are capable of meeting this criterion.

Capable of Modelling Resources?
Each activity in an element of software safety practice consumes resources (materiel) as they are carried out. All three remaining notations (UML, SPEM, and FRAM) meet this criterion.

4.3 Defining the Scoring Criteria

Satisfied that the remaining notations all met the functional criteria, the next part of the selection process evaluated the non-functional capabilities of UML, SPEM, and FRAM.

We noted in Chapter 3.1.4 that the design of the framework must support widespread use. Considering also Keleman et al's selection criteria given in Section 4.2, we argue that the selected graphical representation should therefore also be:

- Ready for use with minimal adaptation
- Capable of use without the need for proprietary software
- Saveable in a portable format
- Capable of construction and analysis in the absence of formal modelling knowledge

- Understandable and interpretable in the absence of prior ontological knowledge/experience
- Capable of construction in the absence of complex background databases.

The scoring criterion for this final step of the selection process is binary: a score of 1 (meets the need), or a score of 0 (does not meet the need) – as shown in Table 4.3 below.

Criterion	**Meets the Need**	**Does Not Meet the Need**
Ready to use with minimal adaptation	1	0
Saveable in a portable format	1	0
Use without proprietary software	1	0
Construction without prior modelling knowledge	1	0
Interpretable/understandable without prior ontological knowledge	1	0
Does not require complex modelling databases	1	0

Table 4.3: Scoring Criteria

4.4 Assess One Example for Each Modelling Notation

The term 'one example' is used to signify that only one instance of each notation or tool (either the original notation, or an extension) was selected for assessment (see Table 4.1). Taking the remaining modelling notations in turn, individual results were compiled using the format in Table 4.4 below.

<Notation>	**Score**
Ready to use with minimal adaptation	
Use without proprietary software	
Saveable in a portable format	
Construction without prior modelling knowledge	
Interpretable/understandable without prior ontological knowledge	
Does not require complex modelling databases	
TOTAL SCORE	

Table 4.4: Notation Scoring Template

UML
Using the scoring criteria, UML is marked as per Table 4.5.

UML	Score
Ready to use with minimal adaptation	1
Use without proprietary software	0
Saveable in a portable format	1
Construction without prior modelling knowledge	0
Interpretable/understandable without prior ontological knowledge	0
Does not require complex modelling databases	0
TOTAL SCORE	**2**

Table 4.5: UML Evaluation

SPEM
Using the scoring criteria, SPEM is marked as per Table 4.6.

SPEM	Score
Ready to use with minimal adaptation	1
Use without proprietary software	0
Saveable in a portable format	1
Construction without prior modelling knowledge	0
Interpretable/understandable without prior ontological knowledge	0
Does not require complex modelling databases	0
TOTAL SCORE	**2**

Table 4.6: SPEM Evaluation

FRAM
Using the scoring criteria, FRAM is marked as per Table 4.7.

FRAM	Score
Ready to use with minimal adaptation	1
Use without proprietary software	1
Saveable in a portable format	1
Construction without prior modelling knowledge	1
Interpretable/understandable without prior ontological knowledge	1
Does not require complex modelling databases	1
TOTAL SCORE	**6**

Table 4.7: FRAM Evaluation

As illustrated by the results in Table 4.5, Table 4.6, and Table 4.7, FRAM [57] scored the highest of the remaining three notations and was selected as the basis for the graphical notation due to its inherent simplicity in representing process interactions, dependencies and constraints; and it supports fully our non-functional requirements for widespread use.

The next section of this chapter takes FRAM as the selected notation and shows how it has been modified for use in the framework and process to understand and assess software safety practice.

4.5 Refining the Selected Graphical Notation

Having selected FRAM to model and assess software safety practice, the next step was to establish and refine (where needed) the symbology for the graphical representation. This graphical representation is crucial for communicating the processes and the results with practitioners and stakeholders alike, and is also a vital enabler for answering Research Questions 1 and 2.

Whilst it is possible that other notations - either not revealed by the research, or developed at some point in the future - may prove to be more suitable and efficient, we assert that this adapted version of FRAM should be used in the process to understand and assess software safety practice. The reasons for this assertion are now discussed.

Using this FRAM-based notation to construct the models of practice enables the identification of agreements and disagreements between models in a simple manner, and the representation of complex empirical data in a manner that is easy to digest and comprehend. Further, FRAM is capable of representing simplistically not only the activities that should be carried out (depicted as 'functions' in the original ontology); but also has inherent ability to represent the prerequisite considerations (depicted as 'aspects') for each 'function' (see Figure 4.3). Aspects in FRAM are given as:

- Input
- Output
- Precondition
- Resource
- Time
- Control.

FRAM functions can be decomposed into sets of sub-functions – each of which have their own respective aspects. The FRAM notation can represent the links between the functions; and facilitates this linking via aspects [31] as shown in Figure 4.4.

FRAM was designed with the aim of assessing the safety of a system in terms of what is needed in performance terms - in order that "everything goes right" [57], and is a useful tool for modelling complex, socio-technical systems [33]. It has also already been subjected to adaptation. For example Leong et al [76] adapted FRAM as a means of arguing the management of epistemic uncertainty in socio-technical systems – adapting the syntax of 'aspects' for 'causal factors', 'functions' for 'objects', and the linking of functions/sub-functions being denoted as 'causal paths'.

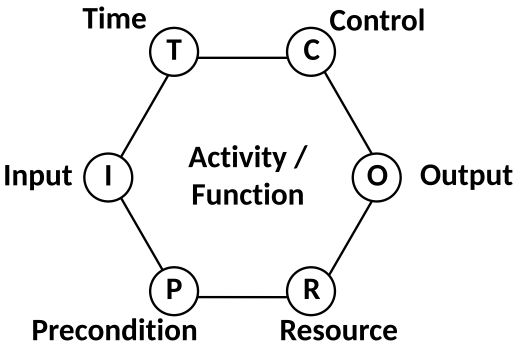


Figure 4.3: FRAM Notation Example [31]

In the adapted version of FRAM for this empirical research, the majority of the taxonomy from the original FRAM model is retained, but some of the syntax are modified. A 'layered approach' to the modelling is also adopted; and some new concepts to adapt and enhance the utility of FRAM for the specific research purpose are introduced (the comparison being made in Table 4.8). As each adaptation is introduced, the reasons behind it are explored.

The original FRAM symbology was designed to represent simplistically and assess functions / activities that occur sequentially / chronologically in a process. A function is depicted by a hexagon, and the circles on the edges of the hexagon denote the required attributes of a function (see Figure 4.3). These attributes are referred to as 'aspects', and are used to link activities and artefacts together as shown in the example on retrieving cash from an ATM in Figure 4.4. This example of FRAM shows the abridged activities required to withdraw cash from an ATM. Here preconditions are modelled using sub-activities shaded in grey. The customer requiring cash must first insert their card, and then enter their PIN. The entering of a PIN is a security check, and the precondition for entering the PIN is the ability to recall it. After entering the correct PIN, the customer specifies the amount to withdraw before collecting it – as long as they have sufficient funds in their bank account.

Whilst retaining most of the structure of the original FRAM, the adapted version used in this research has been modified to better represent the activities in an element of software safety practice. Further, specific instructions on how these shapes and linking lines are used are given as the instructions in the use of FRAM progresses.

Figure 4.5 shows the shapes which represent activities, and the aspects which represent their required attributes. The meaning of each aspect of an activity is explained in Table 4.8, which also compares the original FRAM with our adapted version. Our adapted version of FRAM uses 'aspects' to denote the characteristics, dependencies, and attributes / constraints required to undertake an activity,

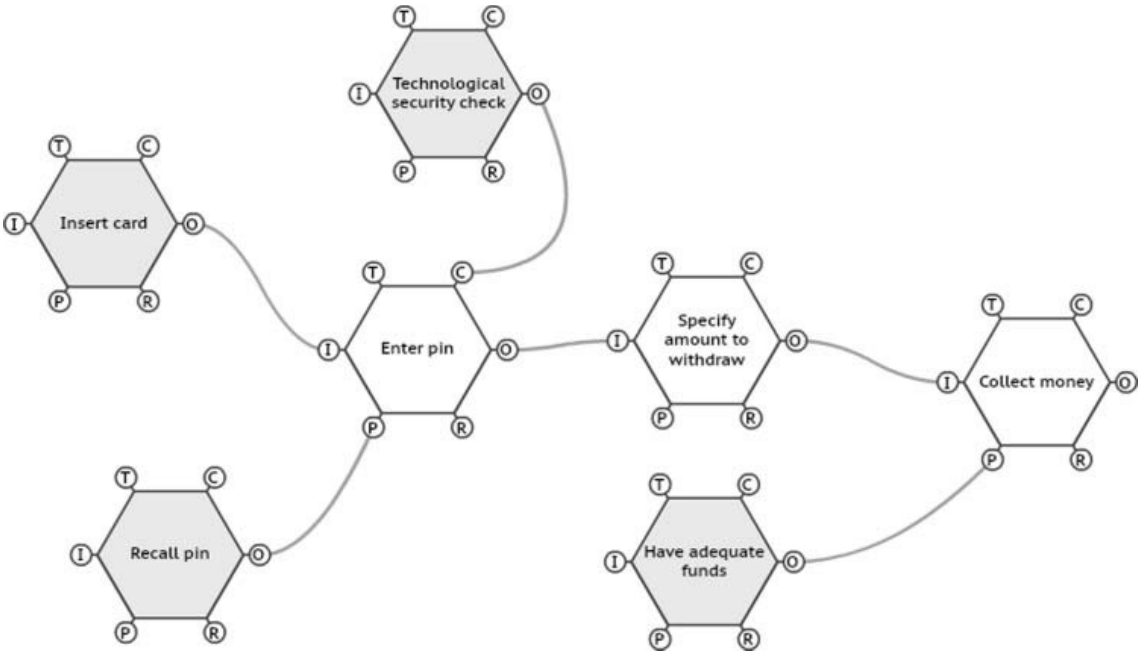


Figure 4.4: FRAM Instantiation Example [31]

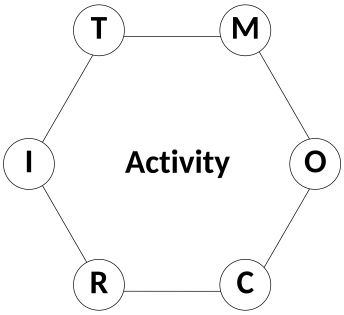


Figure 4.5: Activities and their Aspects

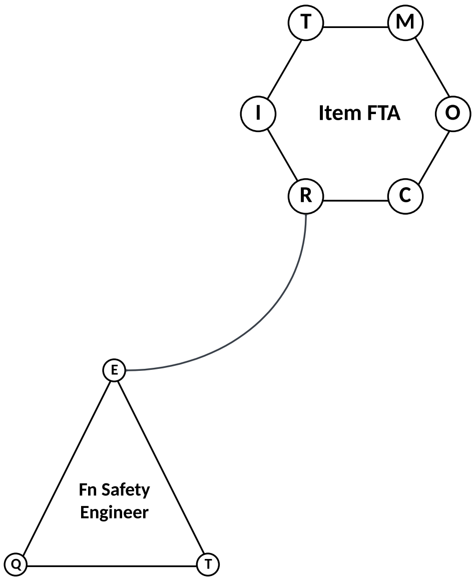


Figure 4.6: Modelling of Resources

or produce an output. Only 'Resources' are used (rather than 'Resources/Execution Conditions'), as 'Execution Conditions' are modelled by linking to the aspects 'Time', 'Control', and 'Resources'.

As shown in Figure 4.6, the activity 'Item FHA' expends the resource 'Fn Safety Engineer' in order to carry out the FHA. The representation of artefacts such as resources which are consumed or expended by activities are discussed shortly.

FRAM uses 'Preconditions' to model system states that must be true, or conditions that must be met before a function is performed. Our modified version of FRAM does not consider 'functions' this way – rather it models the activities associated with software safety practice. As such, the 'Precondition' aspect is replaced with that of 'Methods / Techniques', which is used to describe the means by which an activity is performed.

As system states are not modelled, the FRAM concept of preconditions is addressed through the consideration of the aspects 'Time' and 'Control', as shown in Figure 4.17. 'Time' describes a temporal relationship with the activity, and a 'Control' represents the aspect which supervises or regulates an activity - such as ARP 4761 controlling the way the 'Item FHA' is performed (see Figure 4.17).

Table 4.8 compares the aspects used in FRAM with those used in our adapted version. In the original FRAM, a general rule is that an aspect is described using a noun or noun-phrase. Another rule states that there can exist more than one link to / from an aspect. These rules are retained in our adapted version.

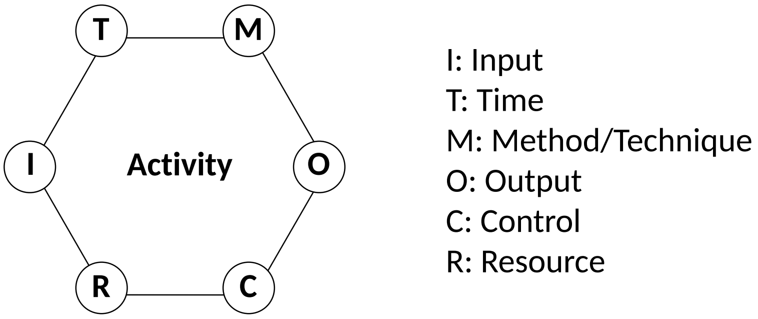


Figure 4.7: Modified FRAM Notation

Aspect	Original FRAM	Aspect	Adapted FRAM
Input	That which is used or transformed by the function to produce the Output, or that which activates or starts a function (always stated as a noun or noun phrase)	**Input**	That which is used or transformed by the activity to produce the Output, or that which activates or starts an activity (always stated as a noun or noun phrase)
Output	Describes the result of what the function does. The description of the output should be a noun or noun phrase. Something that is defined as an Output from one function must clearly also be defined as either an Input, Precondition, Resource, Control, or Time of another function(s)	**Output**	Describes the result of what the activity does. The description of the output should be a noun or noun phrase. Something that is defined as an Output from one activity must clearly also be defined as either an Input, Method, Resource, Control, or Time of another activity or activities

Aspect	Original FRAM	Aspect	Adapted FRAM
Precondition	System states that must be True, or as conditions that ought to be verified before a function is carried out. A Precondition must always be defined as an output from another function(s). The description of the precondition should be a noun or noun phrase	**Method**	A technique/method that can be employed to carry out the activity. The description of the Method should be a noun or noun phrase
Resources/ Execution Condition	Something that is needed or consumed while a function is carried out. A resource is consumed by a function. An execution condition is used to describe (for example) the competence of a resource to carry out a task. The description of the resource/execution condition should be a noun or noun phrase	**Resource**	Something that is needed or consumed while an activity is carried out. A resource is consumed by an activity. Where the original FRAM used this aspect to include Execution Conditions, characteristics such as this will be modelled by the aspects controlling an artefact. The description of the resource should be a noun or noun-phrase
Control	That which supervises or regulates a function so that it produces the desired Output, such as a plan, process, or guidelines. The description of the control should be a noun or noun phrase. Something that is defined as a Control for a function(s) must also be defined as an Output from another function(s)	**Control**	That which supervises or regulates an activity so that it produces the desired Output - such as a plan, process, or guidelines. A Control may also be a quality stipulation (expressed only by an artefact/free text). The description of the control should be a noun or noun phrase

Aspect	Original FRAM	Aspect	Adapted FRAM
Time	Temporal relations that represent the various ways in which time can affect how a function is carried out. It may relate to a function alone (i.e. elapsed time/clock time); or relate to a sequence of actions. It can also represent the point at/by which a function must occur. The description of Time should be a noun or noun phrase	**Time**	Temporal relations that represent the various ways in which time can affect how an activity is carried out. It may relate to an activity alone (i.e. elapsed time/clock time); or relate to a sequence of actions. It can also represent the point at/by which an activity must occur. The description of Time should be a noun or noun phrase

Table 4.8: Comparing the Use of Aspects between FRAM and our Adapted Version

This adaptation results in an updated notation of the activity and aspects from the original FRAM as depicted in Figure 4.7. Attention now turns to describing how our adapted version of FRAM is used to instantiate the process to both represent the outputs from the process steps, and used to assess software safety practice. Each step of the process from Chapter 3.3 is considered in turn. Only the modelling and assessment activities are considered, and the steps are not repeated here.

STEP 1: Define As Desired Software Safety Practice

Software safety practice as-desired is given as a set of criteria. As such no graphical representation is required for Step 1.

STEP 2: Model Software Safety Practice Practice As Required (Open)

Step 2 of the process in Chapter 3.3 requires the activities within software safety practice as-required (Open) to be represented graphically. This requires an analyst to represent the software safety activities of the Open Standard in graphical form. From the documents that constitute the selected Open Standard a set of activities and supporting artefacts must be generated. These activities and artefacts are represented using the modified FRAM notation as follows:

1. Create a hexagon shape for each activity required by the standard, placing the title of each activity in the centre of the shape (see Figure 4.5).

As a software safety lifecycle progresses beyond activities, a level of abstraction is reached at which the inputs to, and/or outputs from an activity are simply artefacts which are an output from and / or an input to an activity. These artefacts must be defined and be capable of representation. This is now discussed.

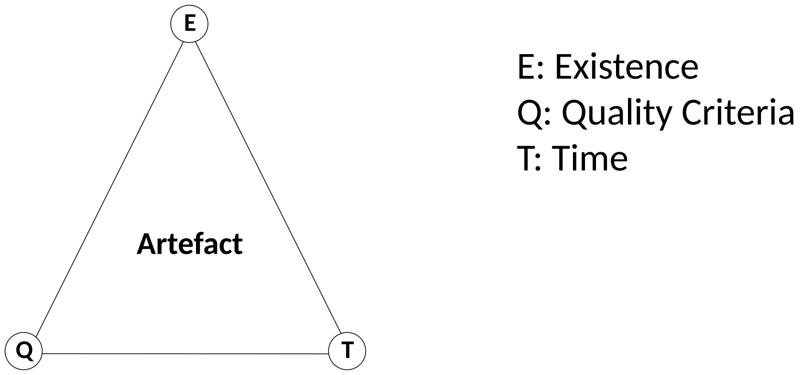


Figure 4.8: Artefact Symbol

An artefact represents an article that supports or constrains an activity, or is produced as the result of an activity. As such, it is the lowest level of abstraction that this process for modelling software safety practice will represent. The hexagon used to represent activities cannot be used to portray artefacts as one should not use the same symbology to represent different elements, and only three aspects of an artefact are required (as discussed in Chapter 3.2). As such, a simple triangle will be used – as shown in Figure 4.8. Having defined the modelling symbology for artefacts, the process is now returned to.

2. Create a triangle shape for each produced and consumed artefact, placing the title of the artefact in the centre of the shape.

3. Link the activities and artefacts together – connecting them using black, curved lines. The links should denote the role of the artefacts as either:

(a) Inputs to an activity

(b) Outputs produced by an activity (which may then provide an input to another concurrent activity)

(c) The time by which an activity should commence / complete

(d) Methods / techniques by which an activity is completed

(e) Controls which constrain/inform the way an activity should be undertaken, OR

(f) Resources expended.

An example of this is shown in Figure 4.9 where the output from the activity 'Create FDAL and IDAL Assignment' is the artefact 'FDAL and IDAL Assigment'; which in turn is an input to the activity 'Create Platform Safety Assessment'; which in turn creates the artefact 'Platform Safety Assessment'.

4. For each modelled artefact, represent the following attributes as aspects (as stated by the Open Standard):

(a) Time the artefact should be produced or delivered to enable an activity to commence

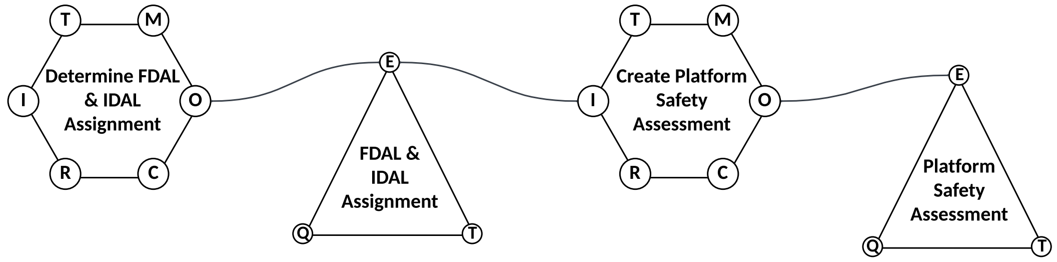

T
M
I
O
R
C
Determine FDAL & IDAL Assignment
E
FDAL & IDAL Assignment
Q
T
T
M
I
O
R
C
Create Platform Safety Assessment
E
Platform Safety Assessment
Q
T

Figure 4.9: Linking Activities Together

(b) Quality criteria required of an artefact (i.e. the skills and experience of a Resource, or the format of a report)

(c) Whether the artefact exists, or whether another activity should be modelled to create it.

Some nuances in the description of software safety activities have been uncovered during this research, and these have required further specific amendments to the adapted version of FRAM. These are now considered in turn.

Referenced Documents

During the modelling of the as-required software safety processes provided by organization's, it became evident that some organization's have a vast suite of process artefacts held in their repositories. Whilst the referenced documents were not all provided, they were referenced from documents that **were** provided as part of the empirical research.

There are many reasons why an organization may withhold documentary artefacts from analysis (such as confidentiality, security, competitive advantage, or for other commercial reasons). This thesis does not consider how referenced artefacts may be provided or utilized, but does provide a means of acknowledging their existence when it is prudent to do so.

Some of these referenced artefacts may contribute to the generation of a more complete (and therefore more accurate) model of practice, even though they were not supplied by the organization. As such, and where pertinent to do so, these referenced documents must be added to the graphical representation of practice. The positioning of these referenced documents in the Illustrated Example in Chapter 5 was based on a judgement of the author's personal experience of software safety engineering and software development lifecycles. An analyst undertaking the investigation on behalf of a project should apply the same personal judgement.

The motivation for adding these referenced (but not supplied) artefacts was twofold. Firstly, it is important to create as accurate a representation as possible, and secondly (for the purpose of this thesis) to enable the research to complete within the time limits of a single PhD programmme (as modelling scores of extra documents would be time-prohibitive).

The annotation of referenced documents in the adptated version of FRAM is facilitated by 'inverting' the colours of activities and artefacts to a black shape

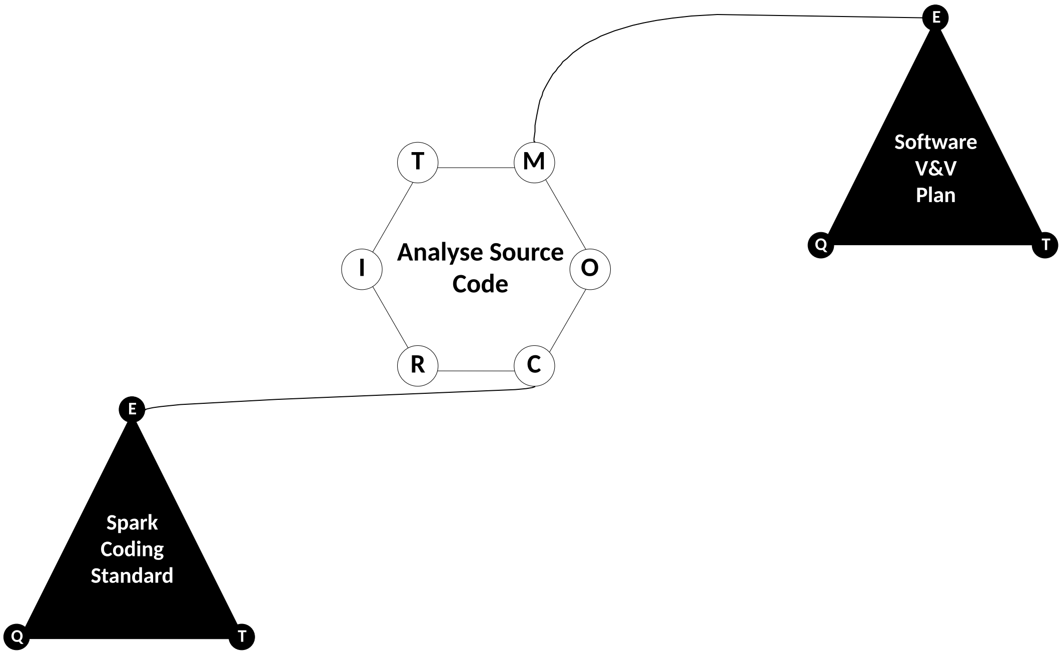


Figure 4.10: Referenced Documents

with white font (as shown in Figure 4.10).

Notes
Occasionally the modelling will require notes to be annotated to maintain the readability of the model. An example of how this is to be represented is shown in Figure 4.11. In this example Note 7 is used to (more fully) describe the specific data types that the artefact 'FDAL and IDAL Assignment' delivers in support of the activity 'Create Platform Safety Assessment'.

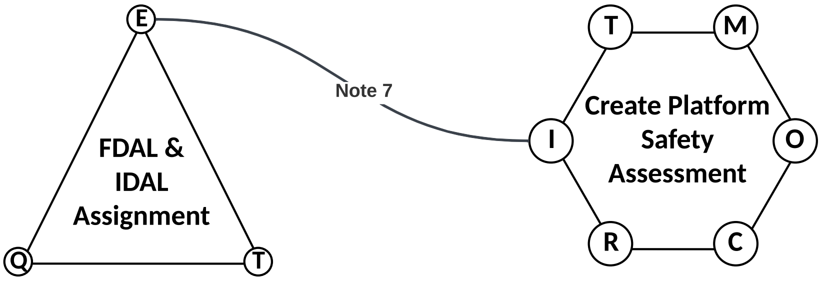


Figure 4.11: The Use of Notes

Levels of Abstraction

Open Standards such as ARP 4754A [147] portray lifecycle activities that span all layers of design abstraction – from the Platform- down to the Software-level. Modelling the activities and artefacts that traverse all layers of design abstraction results in a large, monolithic model that would span multiple pages of a portable document format such as Microsoft Word. This would detract from the readability and usability criteria on which FRAM was selected as a suitable modelling notation. To preserve the readability, usability, and portability of this adapted version of FRAM, the activities can be modelled in separate levels of design abstraction (one model per level), perhaps using the following as a guide:

- Platform
- System
- Item
- Software.

As activities and artefacts often span multiple levels of design abstraction, a means of linking the different (off-page) models was required. Two similar options for doing this has been created, both of which use an entirely grey-coloured activity/artefact:

- Off-page items whose title clearly denotes the level of abstraction to which the activity/artefact is linked
- Off-page item whose title does not clearly denote the level of abstraction to which the activity/artefact is linked.

The first option is to model an off-page reference using the 'Intersection' mathematical operator (∩) underneath the title of the activity/artefact (as shown in Figure 4.12). In Figure 4.12 the linked object emanates from/to the artefact 'System Architectures', and as the title of the artefact reveals its location (the System-level model), only the intersect symbol is used.

The second option is to model an off-page reference using the name of the page on which the activity or artefact is modelled – using italicised text within parentheses (Figure 4.13).

In Figure 4.13, the title of the object (Interface Requirements) does not reveal the location of the model from/to which the artefact emanates. As such the title of the model ('platform' in this case) is inserted in italicised text within parentheses.

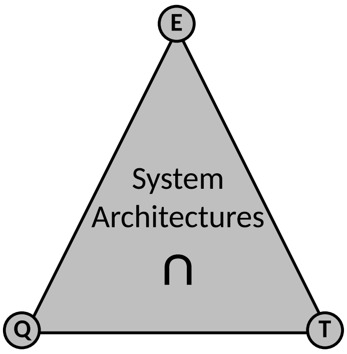


Figure 4.12: Off-page Link Using the Intersect Symbol

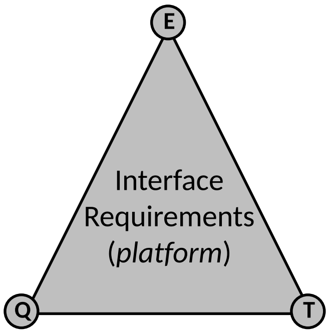


Figure 4.13: Off-page Link Using the Title in Italicised Text

STEP 3: Model Software Safety Practice Practice As Required (Closed)

The tasks for Step 3 are the same as for Step 2 - except that the element of software safety practice being modelled is as-required (Closed) practice. As such, the FRAM modelling process steps remain the same as for Step 2.

STEP 4: Represent Software Safety Practice As Observed

The tasks for Step 4 are the same as for Step 2 - except that element of software safety practice being modelled is as-observed practice. As such, the FRAM modelling process steps remain the same as for Step 2.

STEP 5: Compare Organizational Practice with Software Safety Practice As-Desired

Step 5 of the modelling process in Chapter 3.3 requires the model of software safety practice as-required (Closed) to be assessed for completeness and internal consistency. It also requires the models of software safety practice as-required (Closed) to be assessed for compliance with software safety practice as-desired.

When considering the completeness and consistency of activities and artefacts (with respect to meeting software safety practice as-desired), colour-coding is to be applied to the newly created model (the copy). The colour coding is to be

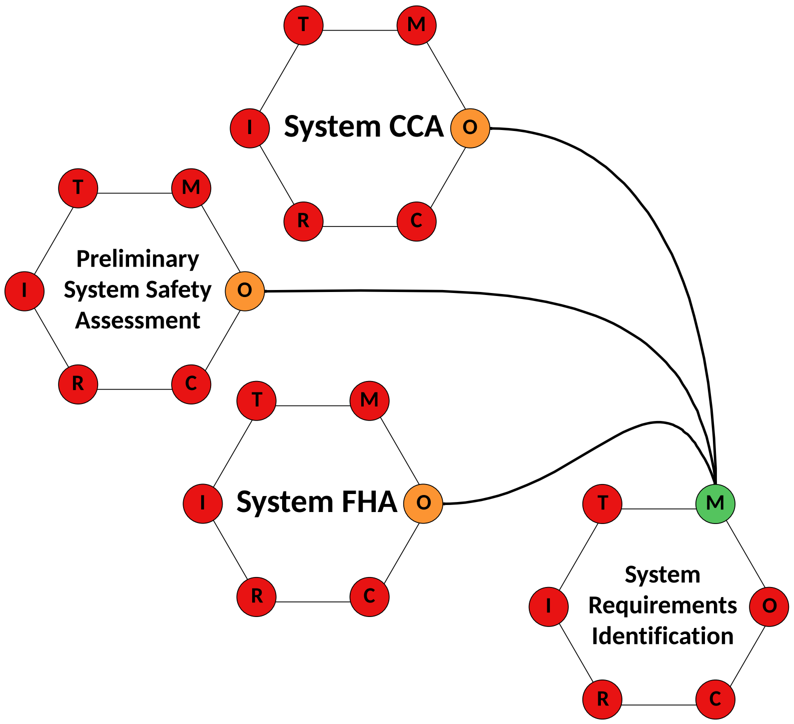

System CCA
T
M
I
O
R
C
Preliminary System Safety Assessment
T
M
I
O
R
C
System FHA
T
M
I
O
R
C
System Requirements Identification
T
M
I
O
R
C

Figure 4.14: Multiple Options in Support of Activities

applied to the aspects of the activities and artefacts as follows:

- **GREEN**: The aspect has been sufficiently defined to meet the criterion of the as-desired model in full
- **AMBER**: The aspect has been partially defined to meet the criterion of the as-desired model, but full compliance cannot be claimed.
- **RED**: There is no consideration given to the aspect.

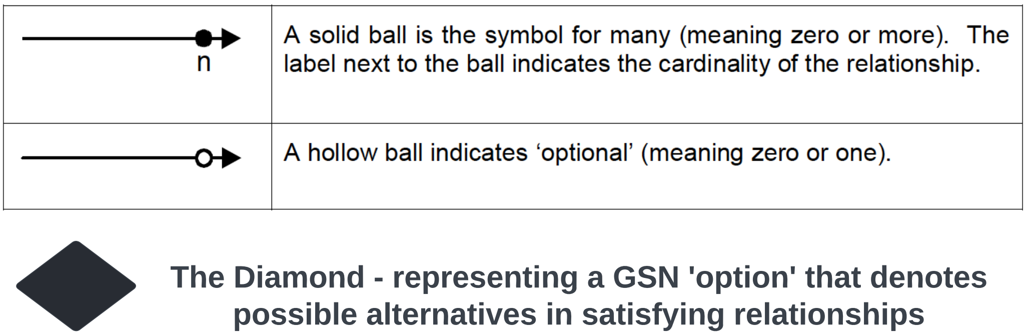
There may be occasions when applying colour coding to the aspects of activities and artefacts is not sufficient to represent certain ambiguities, however. These ambiguities and their means of representation in the models are now discussed.

Optionality and Multiplicity

During the modelling of the as-required representations as part of this research, it became clear that Open Standards often suggest that a number of activities may / should / could be used as a method for/input to another activities. An example taken from the modelling of [147] is given in Figure 4.14.

In this example, the activity of identifying Platform Requirements is carried out by completing the activities of Platform Functional Hazard Analysis (FHA), Preliminary Platform Safety Assessment, and Platform Common Cause Analysis (CCA). What is not clear from the text nor illustrations in the standard is whether some, or all of the sub-activities should be undertaken to completely and correctly carry out the activity of 'Platform Requirements Identification'.

We could colour the linking lines, but using colour coding of the linking lines would not offer a sufficient solution to represent optionality and multiplicity, however. Optionality and Multiplicity is managed in Goal Structuring Notation [38] using the extensions shown in Figure 4.15. Where optionality or multiplicity is to be denoted, the same iconography is to be used in the linking lines for this process.



A solid ball is the symbol for many (meaning zero or more). The label next to the ball indicates the cardinality of the relationship.

A hollow ball indicates 'optional' (meaning zero or one).

The Diamond - representing a GSN 'option' that denotes possible alternatives in satisfying relationships

Figure 4.15: Multiplicity and Optionality Extensions in GSN

Artefacts linked Directly to Artefacts

During the modelling of Open Standards as part of this research, it became evident that artefacts were occasionally linked directly to other artefacts without recourse to a consuming or producing activity. As artefacts are produced or consumed by an activity, an artefact cannot be linked to another without a resourced activity to facilitate this.

An example of a breach of this artefact-artefact prohibition is shown at Figure 4.16 (taken from the modelling of DO 178C), where the artefacts 'Object Code' and 'Executable Object Code' are created using 'Loading Data' (and other artefacts that are not shown here for brevity) without defining the specific activity that fulfils this.

As such where processes link artefacts together in the absence of a producing or consuming activity, the linking line between the artefacts should be coloured red - thereby highlighting the absence of an expending or consuming activity.

Aspects not Required

In certain circumstances, it is neither meaningful nor helpful to consider the aspects of an artefact. An example of this is shown in Figure 4.17 when considering 'Time'. Whilst time needs to be stated (e.g. phase / date etc.), it is not meaningful to define any quality attributes of time. In such cases, a simple free-text box should be used in this process ("By Phase 2a" in the example given at Figure 4.17).

Inferences

During the modelling of the as-required representations it emerged that Open Standards often infer an activity, or the reader must assume an input for an activity to take place. An example of this (from [147]) is the statement that "*additional assumptions*" will emerge. Although not explicitly clear from the text, this requires

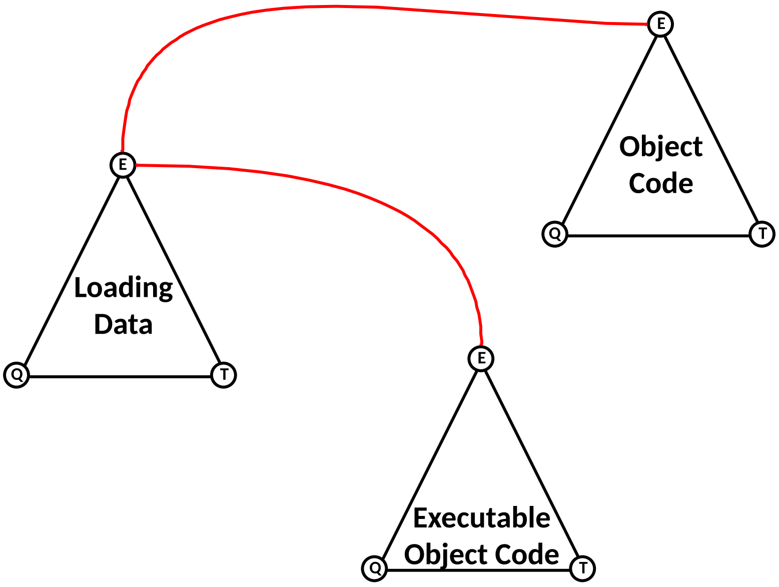


Figure 4.16: Artefacts Linked Without a Consuming Activity

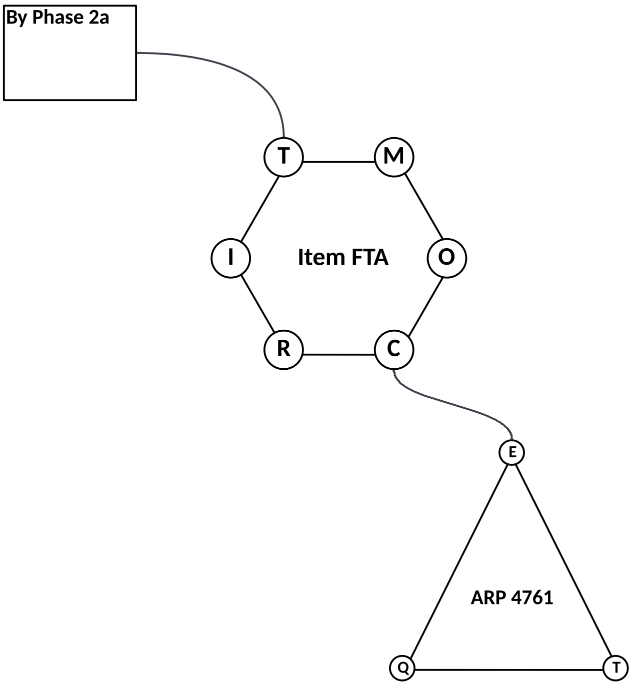


Figure 4.17: Modelling of Time

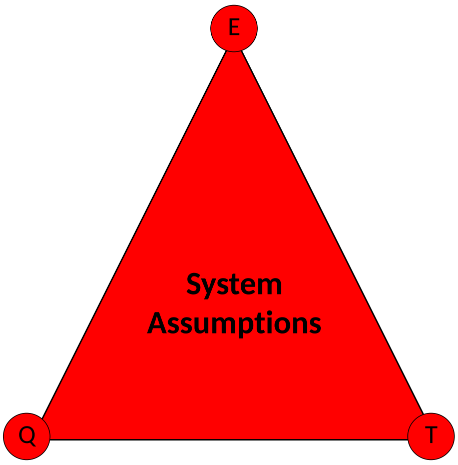


Figure 4.18: Using Colour to Denote the Assumed Existence of an Artefact

the creation of an artefact entitled 'System Assumptions' if internal consistency is to be achieved.

We have established three categories of artefacts in Chapter 3.1.4:

- **Explicit:** artefacts that are explicitly described, and have a consuming or producing activity that is clearly stated
- **Inferred:** artefacts that are discussed without any consideration of their creation or management, and with no consuming or producing activity that is explicitly stated. (e.g. if a standard says that "assumptions must be managed", we must infer the existence of an 'Assumptions' artefact)
- **Orphan:** artefacts that are explicitly described, but have no stated activity that produces them.

Inferred artefacts and activities, and inferred and orphan artefacts, and their associated aspects are to be fully shaded red – as shown in Figure 4.18.

STEP 6: Compare the Open Standard with Software Safety Practice As-Desired

The tasks for Step 6 are the same as for Step 5 - except that the element of software safety practice being assessed is different. As such, the use of FRAM for the modelling and assessment steps remains the same as Step 5.

STEP 7: Compare As Observed Practice with As Required (Closed) Practice

Step 7 of the process in Chapter 3.3 changes from a compliance assessment to a process which compares different elements of software safety practice.

Step 7 requires the model of as-observed practice to be compared with the model of as-required (Closed) practice. The levels of agreement, and any differences between the two models of practice are annotated using colour-coding. The following colour-scheme is to be applied to the aspects of the activities and artefacts of the newly created model (the copy):

- **GREEN**: The aspect of the activity / artefact is in full agreement with the as-required model.
- **AMBER**: The aspect of the activity / artefact is in limited agreement with the as-required model.
- **RED**: The aspect of the activity / artefact has no agreement with the as-required model.

STEP 8: Compare As Required (Closed) Practice with As Required (Open) Practice

The comparison tasks for Step 8 do not differ from Step 7 (only the source models differ). As such, the modelling and comparison steps remain the same as Step 7.

STEPS 9 and 10

Steps 9 and 10 are conditional comparison steps which may not necessarily have an output, and have no additional process instructions which need to be instantiated by the adapted version of FRAM.

Our proposed methodology is now complete. We have proposed a framework, created a modelling and assessment process based on the framework for software safety practice of a given project or organization, and have proposed a way of presenting the result of this activity via modified FRAM diagrams. Chapter 5 now provides an illustrative example that instantiates the framework in Chapter 3.1.4, using the process in Chapter 3.3, with the graphical notation described in this chapter.

Chapter 5

Applying the Framework and Process: An Illustrative Example

This chapter contains an illustrative example of instantiating the framework of software safety practice outlined in Chapter 3.1.4. It uses the process steps from Chapter 3.3 to produce the outputs created by following the process to understand and assess software safety practice. The models created, and the assessments of these models are represented using the graphical notation selected and adapted in Chapter 4. This instantiation of the framework and process to understand and assess software safety practice contributes to the demonstration that Research Questions 1 and 2 have been met.

5.1 Application of the Process

This section presents a step-by-step instantiation of the framework and process to understand software safety practice. The steps relate to the numbers in Fig 5.1 (repeated here from Chapter 3 for ease of reference).

1. Model and represent the organization's as-desired model
2. Model and represent as-required (Open)practice
3. Model and represent as-required (Closed) practice
4. Model and represent software safety practice as carried out (as-observed)
5. Compare the organization's (as-required (Closed)) practice with the as-desired model
6. Compare as-required (Open) practice with the as-desired model
7. Compare as-observed practice with as-required (Closed) practice
8. Compare as-required (Closed) practice and as-required (Open) practice
9. Compare as-observed practice with the as-desired model
10. Compare as-observed practice with an Open Standard.

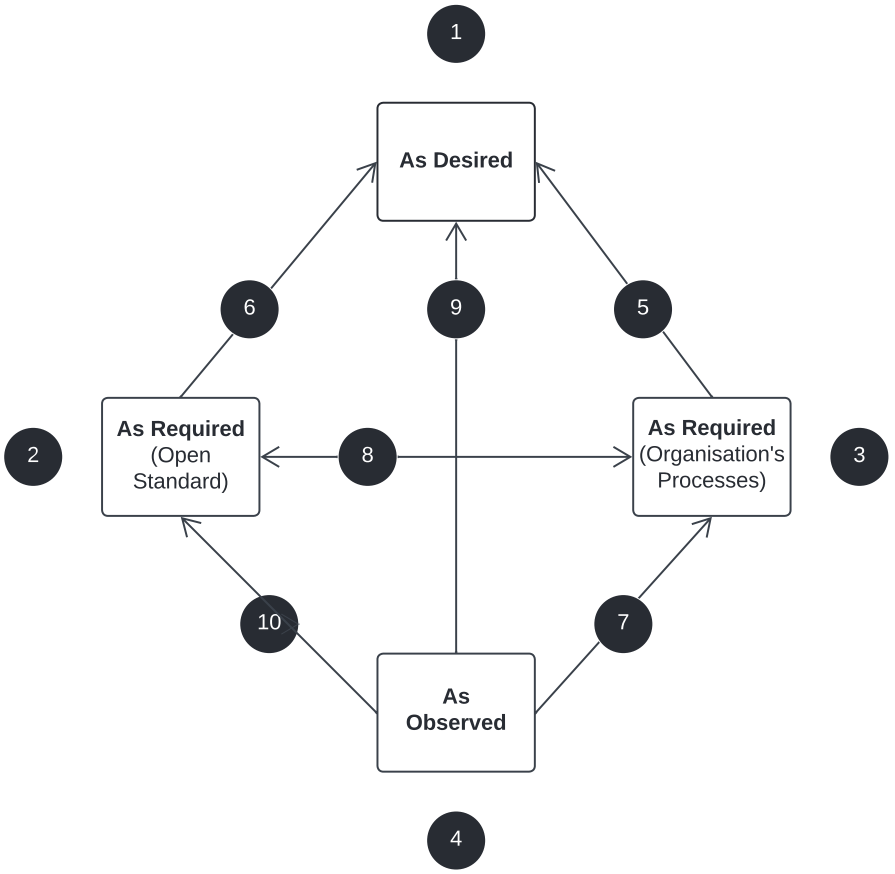

1
As Desired
6
9
5
2
As Required (Open Standard)
8
As Required (Organisation's Processes)
3
10
7
As Observed
4

Figure 5.1: The Elements of Safety Engineering Practice (Repeated from Chapter 3)

The organization and project which participated in this illustrative example is referred to as JB61834 to protect their identity. JB61834 are a global technology company who design and manufacture avionics equipment for use in military aircraft. They kindly supplied the software safety lifecycle processes from one of their aircraft development programmes.

5.1.1 Model Software Safety Practice As-Desired 1

As indicated in Chapter 3, any organization wishing to model safety practice as-desired requires as input a Safety Philosophy, a Risk (acceptance and tolerance) Policy, a Safety Management Philosophy and a suitably qualified and experienced Safety Manager.

JB61834 identified the 4+1 Principles [53], [49] as representative of their software safety practice as-desired. The criteria for meeting these principles are set out in Chapter 3.1.1, and are therefore not repeated here.

5.1.2 Model Software Safety Practice As Required (Open) 2

The Open Standard selected by JB61834 was ARP 4754A [147]. The four tasks for this process step were undertaken. The resultant as-required (Open) model is located at [111], and the accompanying report is at Appendix A. The modelling and assessment of the ARP 4754A lifecycle was presented at the Safety Critical Systems Club Conference, and positive feedback was received [130].

The lifecycle processes from JB61834 span more than two decades, during which time both Open and Defence Standards have changed considerably, and the project's processes have been influenced by differing safety philosophies. Despite this, it is argued that ARP 4754A remains valid as the selected Open Standard, not least as it is representative of Civil Aerospace system safety recommended practice. It is also an acceptable means of compliance for certification by regulatory bodies (e.g. FAA and EUROCAE). Its use is not merely confined to the aerospace sector, however.

ARP 4754A identifies the contribution made to system safety by software. Its objectives are therefore also in accord with Principle 1 of the 4+1 Principles [49] which manifest in UK Defence Standard 00-055 [160]. ARP 4754A is in fact a suite of documents as shown by the relationships in Figure 5.2 [147]. Figure 5.2 illustrates how the ARP has relationships with other documents in a suite of artefacts that combine to create lifecycle processes for the safety-assured development of systems, electronic hardware, and software for use in civil aviation. The development of aircraft and systems is expressed in ARP 4754A in the form of a 'traditional V-model' lifecycle that aims to show the interaction between safety and development processes – as shown in Figure 5.3.

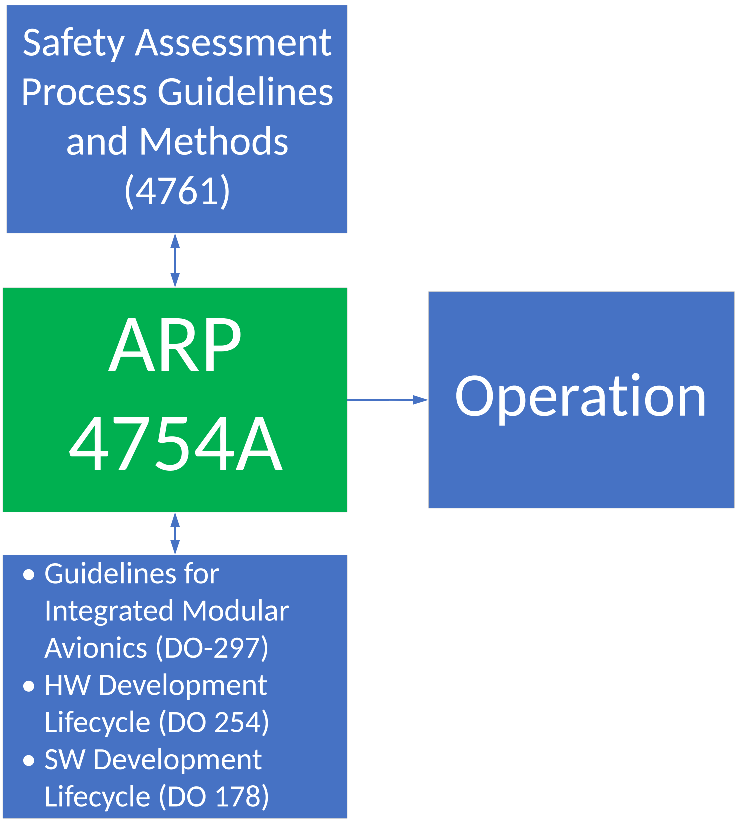

Safety Assessment Process Guidelines and Methods (4761)
ARP 4754A
Operation
• Guidelines for Integrated Modular Avionics (DO-297)
• HW Development Lifecycle (DO 254)
• SW Development Lifecycle (DO 178)

Figure 5.2: Suite of SAE Documents Covering the Development Phases [147]

The purpose of the ARP 4754A suite of publications is to provide guidelines

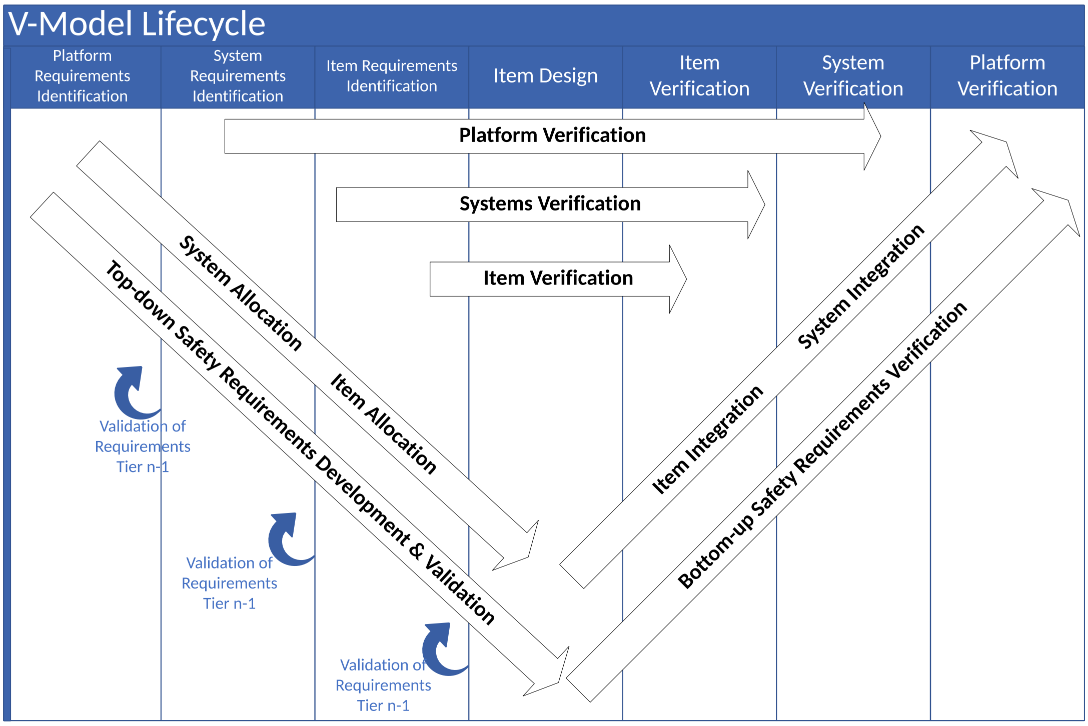

V-Model Lifecycle
Platform Requirements Identification
System Requirements Identification
Item Requirements Identification
Item Design
Item Verification
System Verification
Platform Verification
Platform Verification
Systems Verification
Item Verification
System Allocation
Item Allocation
Top-down Safety Requirements Development & Validation
System Integration
Item Integration
Bottom-up Safety Requirements Verification
Validation of Requirements Tier n-1
Validation of Requirements Tier n-1
Validation of Requirements Tier n-1

Figure 5.3: A Typical V-Model Lifecycle

directed at the systems that support aircraft-level functions whose failure modes affect the safety of the aircraft [147]. The ARP presents a V-model lifecycle process which is defined as being employed "in an iterative and concurrent fashion using both top-down and bottom-up strategies" [147]. It focuses "on the top-down aspect since it provides the necessary links between aircraft safety and system development" [ibid].

5.1.3 Establish As Required Practice (Closed Standard) 3

The single required input to this step is the Closed Standard which constitutes required organizational practice.

JB61834 do not have a single artefact which constitutes organizational practice; rather it is constituted by a suite of documentary artefacts. JB61834 provided the following eight software safety lifecycle documents for modelling:

1. Software Safety Plan
2. System Safety Programme Plan
3. Generic Requirements Management Process (for the entire project)
4. System Safety Working Practice Accident Mitigation Strategy
5. Software Safety Requirements Process Definition
6. Hazard Log
7. Software Development Plan

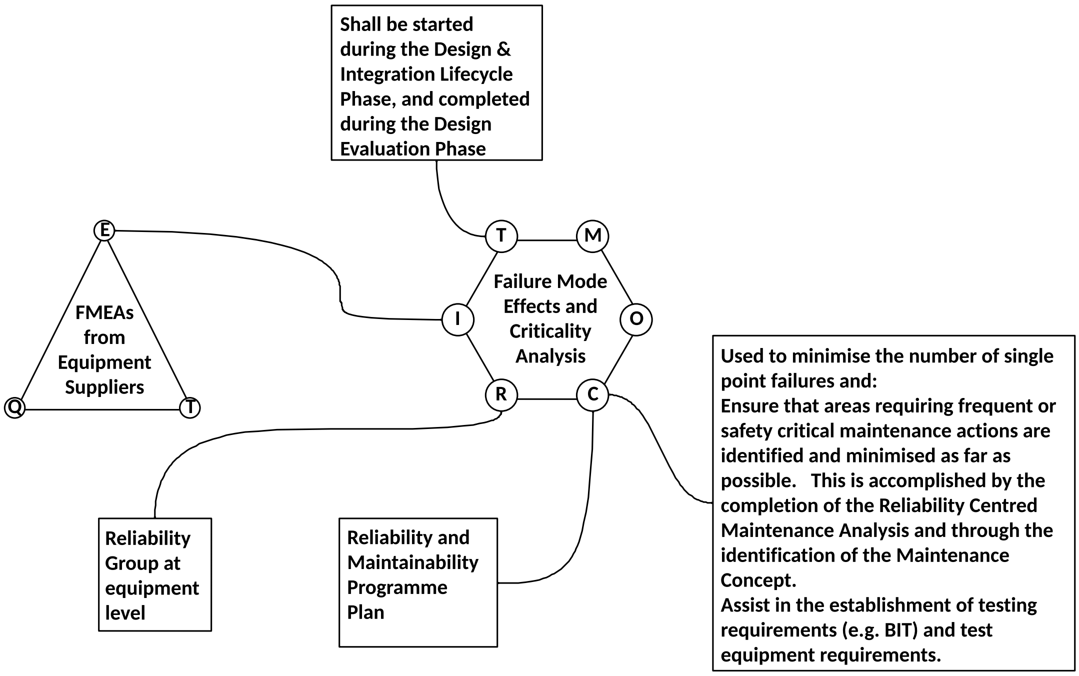

Shall be started during the Design & Integration Lifecycle Phase, and completed during the Design Evaluation Phase
E
FMEAs from Equipment Suppliers
Q
T
T
M
I
O
R
C
Failure Mode Effects and Criticality Analysis
Used to minimise the number of single point failures and:
Ensure that areas requiring frequent or safety critical maintenance actions are identified and minimised as far as possible. This is accomplished by the completion of the Reliability Centred Maintenance Analysis and through the identification of the Maintenance Concept.
Assist in the establishment of testing requirements (e.g. BIT) and test equipment requirements.
Reliability Group at equipment level
Reliability and Maintainability Programme Plan

Figure 5.4: Extract from JB61834 As-Required (Closed) Model

8. Software Testing Methods and Tool Application Standard.

Two outputs are created by this step - the appropriately represented as-required (Closed) model (found at [113]), and a report accompanying the as-required model (this section).

Evaluation of the supplied documents revealed a further 103 documentary artefacts that *could* contribute to the achievement of functional safety, and the initial modelling resulted in five referenced documents being appended to the completed graphical representation.

The full modelling of JB61834's organizational practice for software safety can be found at [113], and this initial stage of modelling made no judgement as to the efficacy of their software safety practice. An extract of the model showing the activity 'FMECA' is provided in Figure 5.4.

The models of JB61834's practice served only to feedback to the project to ensure it was an accurate representation of their processes. Follow-on meetings with representatives of JB61834 confirmed that model was an accurate portrayal of their activities and artefacts.

Of note, JB61834 have started to use the model of the practice as a basis on which to establish Training Needs Analysis (TNA). This TNA will be facilitated by the *further* consideration of the 'Resource' aspect, including the quality attributes required of the resources.

5.1.4 Model As Observed Software Safety Practice 4

The single required input to this task is an empirical report of as-observed safety practice (i.e. that which is carried out by either a software safety engineer, or a software engineer with responsibility for system safety).

The two outputs of this step are the appropriately represented as-observed model (found at [114]), and the report accompanying the as-observed model (this section).

It was noted in Chapter 2.5 that a full ethnographic study was not feasible within the constraints of a single PhD programme, and interviews were undertaken instead with representatives of JB61834. This does change the assessment from work as-observed to 'work as disclosed' as a proxy for work as observed [150]. The implications and potential limitations of adopting this approach are discussed in Chapter 6.

To model the as-observed element of JB61834's software safety lifecycle, interviews with two of its representatives were undertaken:

- KY40540: A Product Safety Engineer throughout the life of the project, who now acts as the internal Independent Safety Assessor and Advisor for the project.

- SJ84999: A former lead software engineer throughout the project, who is now the Software Team Lead for the project. SJ84999 has over 30-years of experience on the XX projects - undertaking a multitude of software roles during that time.

The interviews were carried out remotely, and recorded and transcribed using the project's own encrypted conferencing software. As the recordings of the interviews occasionally had discussions in which the identity of the project and the individuals concerned could be revealed, the recordings are not provided in this thesis in order to maintain anonymity. A redacted transcript of one of the interviews is provided at [127] (redacted to prevent the disclosure of any information which could reveal the identity of the participants or the project). The transcript recording facility of the company's own conference software of the first interview (with KY40540) did not function correctly (which was not under the control of the author). A precis of the information gained from these interviews is now provided, and an extract of the model showing the allocation of safety targets is provided in Figure 5.5.

KY40540

KY40540 provided an overarching description of software safety practice within the project, and this is outlined below:

- Levels of safety required for the avionics software are established at a Platform level

- Standard core safety processes derive these levels of safety into targets for decomposition down to the software-level

  (a) These targets are expressed as safety targets, via top-level system requirements, equipment requirements, and software safety requirements

  (b) Based on historical data, it is assessed whether the software design will be capable of achieving these derived requirements and targets

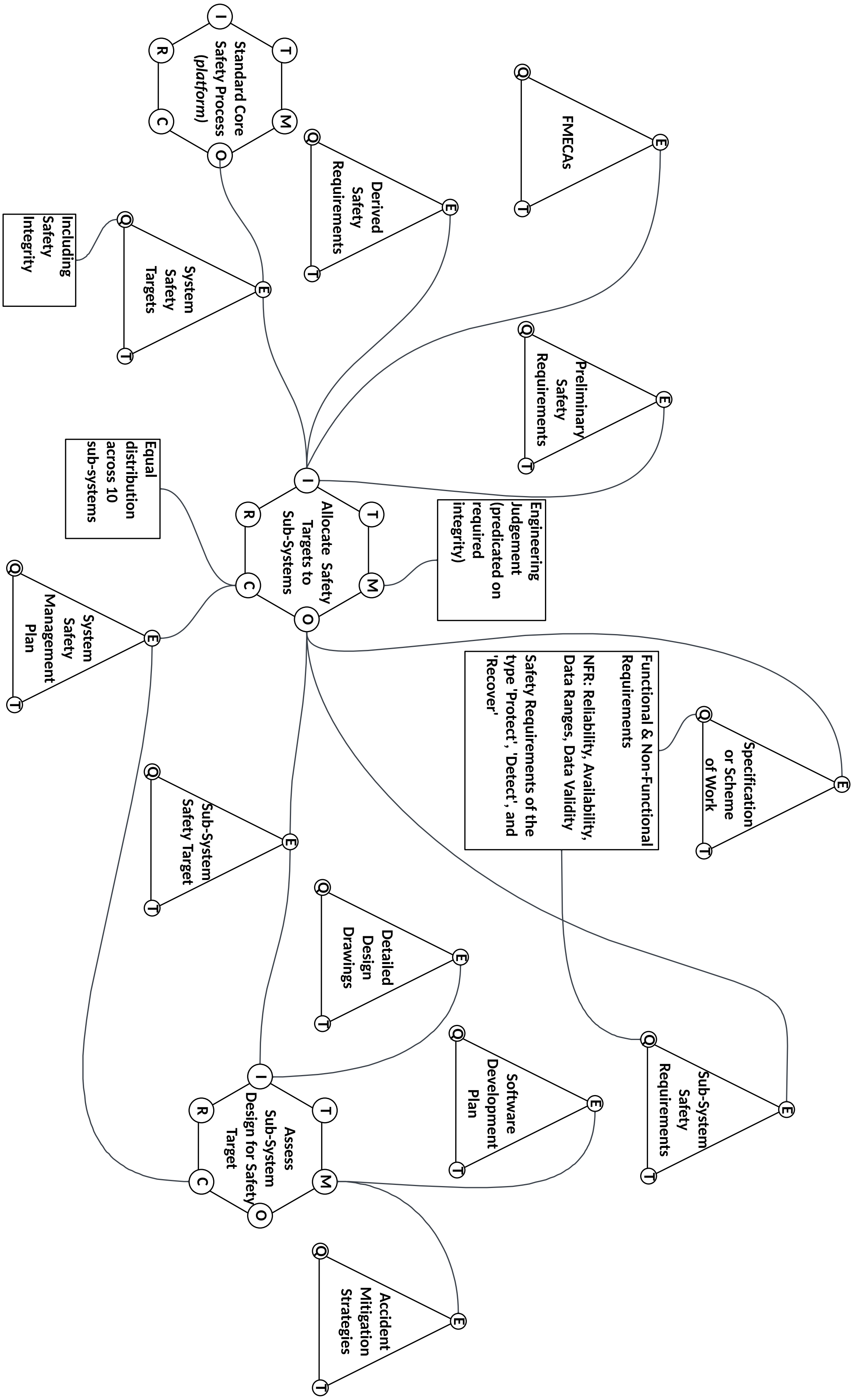

Standard Core Safety Process (*platform*)
I
T
M
O
C
R
FMECAs
Q
E
T
Derived Safety Requirements
Q
E
T
System Safety Targets
Q
E
T
Including Safety Integrity
Preliminary Safety Requirements
Q
E
T
Equal distribution across 10 sub-systems
Allocate Safety Targets to Sub-Systems
I
T
M
O
C
R
Engineering Judgement (predicated on required integrity)
System Safety Management Plan
Q
E
T
Functional & Non-Functional Requirements
NFR: Reliability, Availability, Data Ranges, Data Validity
Safety Requirements of the type 'Protect', 'Detect', and 'Recover'
Specification or Scheme of Work
Q
E
T
Sub-System Safety Target
Q
E
T
Detailed Design Drawings
Q
E
T
Software Development Plan
Q
E
T
Sub-System Safety Requirements
Q
E
T
Assess Sub-System Design for Safety Target
I
T
M
O
C
R
Accident Mitigation Strategies
Q
E
T

Figure 5.5: Extract from JB61834 As-Observed Model

(c) If the targets are not achievable, the design teams must write a justification that the 'best achievable result' has been achieved, but that the target cannot be met owing, for example, to the legacy status of the products

(d) The safety requirements and safety targets are then formalized as contractual agreements.

- Allocation of targets is carried out in accordance with the System Safety Management Plan (SSMP)

  (a) The SSMP requires the safety targets to be equally decomposed over ten systems

  (b) This is stated as being a pure mathematical (division) exercise; which is held to be "straightforward".

- It was stated that difficulties only occur at the next level (of design abstraction)

- If the Design Team feel the allocated target is too onerous for a system, then more work is required by the Design Team

- Any shortfalls in a system's ability to meet safety targets are argued in terms of mitigations (where possible)

- 'Shortfalls' are held to have arisen should the threshold be close to the system's capability, or beyond it

  (a) Mitigations against shortfalls of safety targets occur predominately as 'Accident Mitigation Strategies'

  (b) These accident mitigation strategies have been progressively developed with different customers over the years of the project (which was described as an informal 'framework')

- Safety Arguments are text-based, and argue how the top level claim contributes to an accident as a result of system faults. A top level claim is created in the safety argument which correlates to each Top Level Event in a Fault Tree

- Fault Tree Guidance informs and controls the construction of each Fault Tree. Fault Trees are reviewed as part of the work of creating a safety argument

- The responsibility for managing the risk of Accidents is placed on the Hazard Owner of each contributing hazard

  (a) The Hazard Owner is the respective System Technical Authority

  (b) Hazards are demarcated into 'Physical Hazards' and 'Equipment Hazards'

  (c) Functional Hazards are a type of equipment hazard, and these hazards are the responsibility of the System Technical Authority, and as such, they 'own' the respective Fault Trees and Fault Tree Analyses.

- The Fault Tree Analyses are constructed at the System Level, and represent the “right hand side of the V-lifecycle”

- System Level safety targets are used to elicit preliminary safety requirements (at a component level), which in turn identify derived design safety requirements. Both are allocated at the equipment level

  (a) The derived requirements are ‘top level’, and are captured in tabular form

  (b) The achieved component safety targets are added into the Fault Tree, to calculate whether the System Level safety targets are achieved at the Top Event

  (c) Equipment level safety requirements are either allocated to Mission System equipment owners, or placed in a Specification / Scheme of Work and allocated to suppliers

- Safety engineers may provide additional requirements to include in the specification as appropriate - based on Fault Trees, considering specific failure modes (e.g. invalid / out of range data)

- As the system design transitions from Preliminary Design Review (PDR) to Critical Design Review (CDR), the safety targets in the Fault Tree are replaced with ‘real data’ (historical reliability data, and historical failure rates)

- Reliability data is taken from FMECAs performed by suppliers (assessed for accuracy by the organization’s Reliability Engineers) which is predicated on field data

  (a) This ’real data’ informs the System and Sub-System Hazard Analysis Reports

  (b) System and Sub-System Hazard Analysis Reports are predicated on Fault Tree Analysis, and an argument that the preliminary safety requirements are complete and correct

  (c) Hazard Analysis Reports also look at individual hazards, and uses the initial Fault Trees to decompose requirements down to component-level contributions

  (d) This decomposition is carried out by recourse to system drawings and detailed design drawings (i.e. what the designers know will constitute the final design).

- System failures are assessed by undertaking a cut set analysis of the Fault Trees

  (a) The contribution to system failures from software is assessed by allocating arbitrary failure rates (no probabilistic analysis is attempted), and a subsequent sensitivity analysis

  (b) The means of conducting sensitivity analyses is detailed in the System Safety Processes and Plans, which contain tables of the values to be used.

- The definition of SIL in legacy designs is predicated on the definitions contained within Defence Standard 00-55 at Issue 2

  (a) New customers require argument over the achievement of other military standards (not stated here for reasons of anonymity), or DALs (detailed in the ARP 4754A suite of publications)

  (b) The mapping of SILs to DALs is contained in the Software Development Plan (which also requires a number of safety activities).

- Software safety requirements are identified by a safety tag and are allocated via the XX Database Tool

- Functional and Non-functional requirements are turned into specifications and Statements of Work (SoW), using memoranda to argue over allocation

- Equipment Safety Assessment Reports are compiled to argue that the equipment meets the safety requirements, including an assessment of what the equipment cannot mitigate

- The verification of safety requirements (including acknowledgements where requirements are not met) are argued to be acceptably safe by a mixture of dependencies and assumptions on the assumed operation and operating environment

- Stated assumptions and dependencies are captured by the System Design Teams.

The key findings from the interview are as follows:

1. Safety targets are equally distributed over ten systems, without regard for a system's specific contribution to hazard(s)

   (a) Should the Design Team feel that the target(s) allocated to a system are too onerous, the required 'further work' is not formally captured anywhere.

2. If there are more than ten hazards per sub-system group, the engineers need to set the target at an order of magnitude less. This 'coarse' reduction in targets can be problematic for a legacy programme which is already 'flying'

3. The framework to identify and use accident mitigation strategies (when facing shortfalls in system safety targets) is not formalized into processes or guidance, but is left to the engineers to carry out as they see fit (based on what has worked previously), as the SSMP assumes that the target(s) will always be met

4. Safety arguments are:

   (a) Text-based,

   (b) Predicated on arguing over whole-system operation and Standard Operating Procedures (SOP),

(c) Underpinned by stated assumptions, but these assumptions are not formally managed, and

(d) Predicated on random failure targets, underpinned by independence claims.

5. The decomposition of derived requirements (in mitigation of contributions to hazards) does not inform design decisions. Instead, the decomposition is predicated on what the designers know the architectural solution will comprise

6. Single Point of Failure (SPoF) and Common Cause Factors (CCF) are considered only through an analysis of the 'detailed impact of software factors', and are held to be addressed through the allocation of 'appropriate levels of integrity'

7. Human Factor (HF) considerations are limited to cut set analyses in the Fault Trees (which implies HF are modelled, and modelled completely and correctly), and arguments over independence

8. Design treatments of Human Errors are held to be mitigated by appeal to procedures

9. Numerical analysis (different values applied as part of a sensitivity analysis) are applied to software's contribution to system failures, but the manner by which software contributes to hazards (or even system failures) is neither modelled nor analyzed

10. Should Safety Engineers elicit further safety requirements after undertaking a Fault Tree Analysis, no report is created which would justify their elicitation

11. Confidence in the supply chain meeting software safety requirements is achieved through audits of suppliers. These audits are undertaken by the System Design Teams

12. A safety assessment report considers design reviews which address safety

13. Most suppliers to the organization are not able to argue over integrity, but there is a mapping between SIL and DAL

14. No process exists which states how safety requirements are decomposed and allocated (i.e. what analyses are required). Instead, the Safety Engineers work informally with the Equipment Engineers, and use 'engineering judgement', which is influenced by integrity targets, and the failure modes modelled in the Fault Trees

15. Although processes claim a SHARD analysis was undertaken, it was in fact a HAZOP, and didn't consider the failure modes of software

16. The term 'Safety Case' is not, in fact used by the organization, they are referred to as 'Safety Arguments'

17. Whilst dependencies and assumptions are formally captured, they are 'managed elsewhere'. It is not clear how nor by whom.

The interview then moved to discuss KY40540's role as an internal Independent Safety Auditor (ISA). KY40540 noted (with humour) that theirs was a case of the "poacher turning gamekeeper", and argued the benefits of their role, and their contribution made from having a "wealth of experience" across the teams.

KY40540 noted that they could proactively influence decisions, and give advice where appropriate - especially on how to undertake tasks that may not be well documented. KY40540 was pressed on this matter of how some task instructions were not well documented, and how their own knowledge and benefit of experience would be retained by the project should they leave it. KY40450 noted that, although some product knowledge would be lost, they were confident that process knowledge would be retained as there exists a wealth of knowledge in the project. This runs counter to KY40450's assertion that part of their role was to provide advice on processes that weren't clearly defined. Nor does their response satisfactorily answer the question as to how implicit knowledge can be managed and retained by the project, however.

SJ84999

SJ84999 was interviewed, and the redacted transcript is provided at [127]. The key findings arising from the interview are as follows:

1. As the projects have matured, the organization has gradually moved towards a failure analysis focus as well as the development processes, and have started to introduce a more active element of failure analysis which aims to add more rigour and detail. Naturally, this is problematic when considering legacy equipment

2. Safety requirements are not 'given' to the software team, the software team leads the development of software safety requirements

3. Historically, software safety requirements did not exist, software requirements were 'tagged' with integrity requirements

4. Legacy software was:

   (a) A monolithic build,

   (b) Assumed no partitioning, and

   (c) Everything was built to the highest integrity requirement.

5. Internal efforts to modularize safety cases (NOT referred to as safety arguments) resulted in the introduction of software safety requirements

6. The introduction of an Operating System necessitated a SHARD Analysis

7. Original 'safety requirements' were just a re-expression of existing functional requirements, so were perceived as being a "waste of time". This is perhaps owing to the SHARD analysis having been carried out at an inappropriate level of design abstraction. The SHARD analysis looked at software as a monolithic entity, and only considered failures external to the

software (i.e. a loss of a signal), and NOT how the software itself could contribute to a failure. It did not consider interface failures

8. Despite the SHARD analysis having been carried out, it is not clear where SHARD (or similar analysis) is required by the ARP 4754A suite of documents
9. A mid-life upgrade required software to be copied over to a new piece of equipment, with some 'minor changes'. A review of the software safety requirements revealed they were not fit for purpose, and were not achievable at the software level. These requirements were restricted to considering what should be done, but did not consider what shouldn't (i.e. only considered positive contributions to be made by software, and not the negative). The analysis did not consider what would happen if the software 'failed', nor how it could contribute to system/component failures
10. The Software Design Authority at the time was content to remove these software safety requirements, as they didn't add anything that the existing functional requirements already considered. This was rejected by a customer team, and so the software safety requirements were 're-worked'
11. The original software was monolithic, but the requirement to introduce partitioning revealed the potential for new failure mechanisms across interfaces
12. In the absence of any process or modelling tools, new requirements were elicited by relying on spreadsheets and word-processor documents (and it was felt that this still didn't cover the required information). This absence of process led to a substantial amount of ("too long-winded") documentation to be produced and maintained. This extra documentation was required as the need to elicit further requirements was a 'bolt-on' to the project's normal processes
13. It is asserted that "nothing wrong" was found with the existing software, and that this was due to "great engineers doing the right thing". This goodness of engineering was not recorded formally anywhere
14. It was asserted that if the ARP 4754A suite of documents had been followed originally, then the 'goodness' of the software could have been argued - through following clearly defined objectives for traceability
15. Whilst it is argued that the re-work didn't make the product safer, it did enable its level of safety to be revealed to the customer
16. The original software requirements were expressed as data flows, but the newly-introduced software safety process generated text-based requirements. These new requirements were nothing more than functional descriptions and perhaps pseudo-code, however; and were a re-expression of existing functional requirements
17. Early development of mid life upgrades revealed faults in testing, such as a pitch ladder rolling when the input values weren't changing - i.e. the

software was extrapolating to take into account for system delays. It was felt that more should have been done to elicit faults such as these

18. It was asserted that if they were starting over again (with the designs) they *would* ensure that they elicited formal software safety requirements for instantiation

19. The new processes and analyses for eliciting software safety requirements still do not consider all aspects of software's contribution to a fault / failure / hazard (negative or positive)

20. When the mid-life upgrades required a change in approach to the software safety requirements elicitation process, whilst the process was changed, the process documents were not updated to reflect the new approach required

21. Every mid life upgrade relied on SQEP personnel to get good results anyway - it all relied on people

22. Different customers have different contractual and regulatory obligations, and the organization faces repeated calls to undertake processes which add no value to the product nor its safety

23. Further, newer customers are trying to place on contract processes which the organization does not, and cannot undertake

24. It is not felt that DO 178C is clear as to where the split (and interface) between systems and software is made

25. It is felt that DO 178C doesn't deal well with anything other than a single block of software

26. The Quality Assurance section of DO 178C states that pseudo-code should not be used, but the main section of the Standard does not warn against it

27. People on different projects and teams have differing ideas on how DO 178C is to be interpreted, and as none of them have had the experience of a certification programme, no one can yet agree on how it is to be interpreted

28. The project has not provided a clear definition on how DO 178C is to be interpreted and followed

29. It was asserted that some software currently being written to safety standards has not been written as well as it would have been before the standard was adopted

30. It was asserted that techniques such as 'strongly typed languages' add no value

31. There is an 'uneasy' hierarchical relationship between the system and software teams. The system team assumes that their process ends once they have given the software team the requirements, and are not 'open' to deductive challenges from the software team

32. The Standards the project use assumes a Waterfall development lifecycle, but the requirements do not get flowed down to the software team in time to sufficiently mature the analyses

33. Often the system requirements are allocated to software too late in the design process to make meaningful changes. The software team cannot wait for all requirements however (as nothing would ever be designed)

34. It is asserted that if you get too prescriptive on requirements and test specifications, it prevents people from undertaking critical and intelligent thinking

35. With the introduction of new processes, the software design activity is now out of sync with the software coding activity

36. Derived failure targets are based on the size of the fleet, which distorts the allocation of targets

37. The project is often 'forced' to follow regulatory processes required by their customers' regulators. These processes are held to add no benefit (and may therefore be an example of Administrative Safety [133])

38. Any successes in their project are held to be reliant on the people, and not the processes (nor the requirements flowed down to them).

The two outputs of this step are an appropriately represented as-observed model, and a report accompanying the as-observed model (this sub-section).

The responses from the two interviews resulted in the creation of the as-observed model found at [114]. This model, and the responses from the interviews inform the comparisons with as-required practice (both Open and Closed) and with as-desired practice.

5.1.5 Compare Organizational Practice with Safety Practice As-Desired 5

For this step to proceed, two modelling elements of the framework instantiation process must have already been completed:

1. The as-required (Closed) model of software safety practice (completed at Step 3)

2. The as-desired model of software safety practice (completed at Step 1).

The aim of this step is to create a representation of how as-required (Closed) practice conforms with each objective / criteria of as-desired safety practice in turn.

The process to understand software safety practice considers the following criteria when assessing the completeness and correctness of an element of practice in terms of its compliance with as-desired practice:

- **Internal Completeness and Consistency:** are the activities correct and pertinent commensurate with achieving as-desired practice? Do the right amount of activities exist; and does each activity have the correct amount of supporting activities to ensure it can be completed to the required level of compliance?

- **Consideration of Attributes:** is the information stated for the attributes the correct information (i.e. Inputs, Outputs, Time, Techniques and Methods, Controls, and Resources); and is the correct amount of information given for the attributes for the as-desired practice to be met?

The process to understand software safety practice then considers the following criteria when assessing the levels of compliance between the model of as-required practice and software safety practice as-desired:

- **Sufficiency:** are there the correct amount of artefacts to enable successful completion of all activities, and are the artefacts the correct ones? Does every activity produce an artefact; and does each activity have the correct amount and type of artefacts (as inputs) to comply with the model of as-desired practice?

- **Consideration of Attributes:** is the information stated for the attributes the correct information (i.e. Time, Quality Criteria and Existence) to denote when they need to be produced or used? Are the correct amount of quality attributes considered for each article, and are they the correct quality attributes for as-desired practice to be complied with?

The two required outputs of this step are the appropriately represented model of as-required (Closed) practice compliance, and the report accompanying the model of as-required (Closed) practice compliance (this section). JB61834's lifeycle was assessed for compliance against Principle 1 of the 4 + 1 Principles using this criteria. The full assessment against Principle 1 is contained in Annex C.5, and the main findings are summarized and presented here, before assessing what may be inferred by them.

Precis of Assessment

The overall observation in relation to whether JB61834 meets Principle 1 of the as-desired criteria is that it can be **inferred** that the required information is generated; with one exception. The exception to this claim of compliance relates to the required attributes of software safety requirements. In this regard, organizational practice does not explicitly state **how** the required information will be generated. No single point of truth can be identified for any criterion of Principle 1, in fact.

Many of the documents (and diagrams within) describe what must be done, and these are often presented as options available to the analyst/designer following the process artefacts. What is not stipulated is **how** (nor when) activities are to be carried out. The manner in which activities are to be carried out may be reliant on implicit or tacit knowledge at the as-observed level.

The main findings are summarized as follows:

- No single artefact, nor group of disparate artefacts contains a clear definition of the software within the system under analysis, and it is only possible to assume by inference that this required information will be available.

- The Hazard Log plays a pivotal role in identifying the system hazards which software contributes to, but whilst the data can be argued to flow into the Hazard Log, there is no explicit lifecycle of activities that can trace this data **from** the Hazard Log to the right activity, at the right time, to the required quality, by a suitably qualified person.

- The Hazard Log is the 'single point of truth' as a repository of data that combines to identify the specific failure modes by which software contributes to identified hazards, yet there is a disconnect between the system-level activities and those undertaken at the software-level.

- One can only have a low degree of confidence that the information required to enable the elicitation of software safety requirements in mitigation of the identified contributions to system hazards is created. There are weaknesses in identifying the system hazards however, and when this is combined with the instances of activities that have no explicit input, and artefacts which are produced but do not act as an input to any identified subsequent activity, it is only possible to infer the data will be created. It is not possible to state who would create the data, nor by what means.

- No argument can be made as to the required aspects of the software safety requirements. The criterion requires that the software safety requirements be atomic, unambiguous, defined in sufficient detail, and verifiable, but these required attributes are not considered by the artefacts provided.

A common theme running through much of the text in the supplied artefacts is prose written in both the past and future tenses - a mixture of describing historical design or process decisions, and what is intended to be placed in certain plans or other documentary artefacts. This is perhaps due to the length of time the project has been running for. The systems involved have been successfully deployed in aircraft already, and the artefacts provided reflect a mixture of documents produced for the initial design, for modifications, and for a mid-life upgrade. This mid-life upgrade shifted the design policy to one of modular design with open architectures.

This mix of legacy (yet still extant) processes and procedures with more recent processes that reflect the design policy shift, results in a lack of clarity for one not already familiar with the project and its history. Examples include:

- Development plans that assert what must be done, but not how, nor by whom

- Aspects of the design lifecycle (such as Configuration Management) that have sections in multiple documents concurrently

- A plethora of 'should' statements, without advice on how to decide whether or not to

- Plans that describe the different types of testing, but fall short of arguing the circumstances or criteria for which type should be deployed. This is exacerbated by a clear description as to the weakness of some techniques and methods (but not their merits)
- Documents that highlight what has changed, and what will be placed in other documents at an unstated point in the future.

At this stage there is no *definitive* evidence that any of the deficiencies found are certain impediments to the achievement of Principle 1 of software safety practice as-desired. It may be left reasonably to those charged with the implementation of software safety process as-observed. The risk here however, is that without a robust knowledge and information management system, preventing impediments to the achievement of as-desired practice will rely on retaining personnel within the organization who are suitably experienced (and qualified) in the specifics of the project and its history.

Does as-observed software safety practice recover the weaknesses found in the as-required assessment? Consider each potential impediment in turn:

- No single artefact, nor group of disparate artefacts contains a clear definition of the software within the system under analysis, and it is only possible to assume this required information will be available. **Both models of practice are weak in this regard, and this must now be considered a deficiency which requires follow-up work with the project. (RQ3)**
- The Hazard Log:

  (a) Plays a pivotal role in identifying the system hazards which software contributes to.

  (b) Data can be argued to flow into the Hazard Log, but there is no explicit lifecycle of activities that can trace this data from the Hazard Log to the right activity, at the right time, to the required quality, by a suitably qualified person.

  (c) Is the 'single point of truth' as a repository of data needed to identify the specific failure modes by which software contributes to identified hazards, yet there is a disconnect between the system-level activities and those undertaken at the software-level.

  (d) **Is the 'CRADLE database' in the as-observed model, but this is only cited as an input to the activity 'Allocate Safety Requirements to Equipment'. This must now be considered a deficiency which requires follow-up work with the project. (RQ3)**
- There is a low degree of confidence that the information required to elicit software safety requirements in mitigation of the identified contributions to system hazards is created. There are weaknesses in identifying system hazards that when combined with instances of activities that have no explicit input, and artefacts which are produced but do not act as an input to any identified subsequent activity, mean it is only possible to infer the data will be created. It is not possible to state who would create the data, nor by what

means. **As noted above, this must now be considered a deficiency which requires follow-up work with the project. (RQ3)**

- The criterion to assess relevant aspects requires that software safety requirements be atomic, unambiguous, defined in sufficient detail, and verifiable is not met, as these required attributes are not considered by the artefacts provided.**This must now be considered a deficiency which requires follow-up work with the organization. (RQ3)**

5.1.6 Compare the Open Standard with Safety Practice As-Desired 6

For this step to proceed, two modelling elements of the process must have already been completed:

1. The as-required (Open) model of software safety practice
2. The as-desired model of software safety practice.

The process to understand software safety practice considers the following criteria when assessing the completeness and correctness of an element of practice in terms of its compliance with as-desired practice:

- **Internal Completeness and Consistency:** are the activities correct and pertinent commensurate with achieving as-desired practice? Do the right amount of activities exist; and does each activity have the correct amount of supporting activities to ensure it can be completed to the required level of compliance?
- **Consideration of Attributes:** is the information stated for the attributes the correct information (i.e. Inputs, Outputs, Time, Techniques and Methods, Controls, and Resources); and is the correct amount of information given for the attributes for the as-desired practice to be met?

The process to understand software safety practice then considers the following criteria when assessing the levels of compliance between the model of as-required practice and software safety practice as-desired:

- **Sufficiency:** is there the correct amount of artefacts to enable successful completion of all activities, and are the artefacts the correct ones? Does every activity produce an artefact; and does each activity have the correct amount and type of artefacts (as inputs) to comply with the model of as-desired practice?
- **Consideration of Attributes:** is the information stated for the attributes the correct information (i.e. Time, Quality Criteria and Existence) to denote when they need to be produced or used? Are the correct amount of quality attributes considered for each artefact, and are they the correct quality attributes for as-desired practice to be complied with?

The two outputs of this step are a report accompanying the model of as-required (Open) practice compliance (this section), and an appropriately represented model of as-required (Open) practice compliance. The criteria of as-desired software safety practice is highlighted by the use of bold font in the remainder of this section.

The Open Standard used by JB61834 is ARP 4754A [147], and the model of this Open Standard was completed as part of Step 2. The as-desired model was established as a set of measurable criteria at Step 1. The model and full evaluation of the levels of conformance of ARP 4754A with Principle 1 of the as-desired practice is at Appendix B.7.

The main findings are as summarized as follows:

- There is no definitive artefact which is produced by the ARP 4754A lifecycle that would **clearly identify the software within the system**. There are a number of potential artefacts in which it could be expected reasonably for this information to reside, but there are potential issues with these artefacts (see Section 5.1.5).

- There are six artefacts from the ARP 4754A set of processes which could reasonably contain information on the **operating context of the system in which the software will reside**. There are potential issues with these artefacts however (see Section 5.1.5).

- To provide a **clear description of the system in which the software will reside**, the ARP lifecycle creates two artefacts which could fulfill this criterion. Neither of these artefacts might be provided before the design has been fully instantiated however, and they only contain a section that gives a brief system overview.

- No Hazard Log, nor any form of Hazard List is created which would **identify the system hazards to which software may contribute**. A trace from the Platform-level down through the design levels of abstraction to the Software-level can be made, so it can be assumed that the software safety activities and analyses will identify potential hazardous states for which mitigating safety requirements shall be elicited and managed. Assuming that sister publications govern safety activities, it is reasonable to accept that system hazards will be identified at the software boundary.

- To **describe the specific failure modes by which software contributes to the identified system hazards** there is no clear link between the system hazards and specific failure modes, and the software requirements process does not describe a specific methodology by which software requirements are to be elicited.

- The claim for the **elicitation of software safety requirements which specify the required behaviour for each identified contribution to each system hazard** cannot be made.

- In the absence of a robust software safety requirements process, the criterion to state **all software safety requirements in an atomic, unambiguous manner which is defined in sufficient detail and verifiable** is not met.

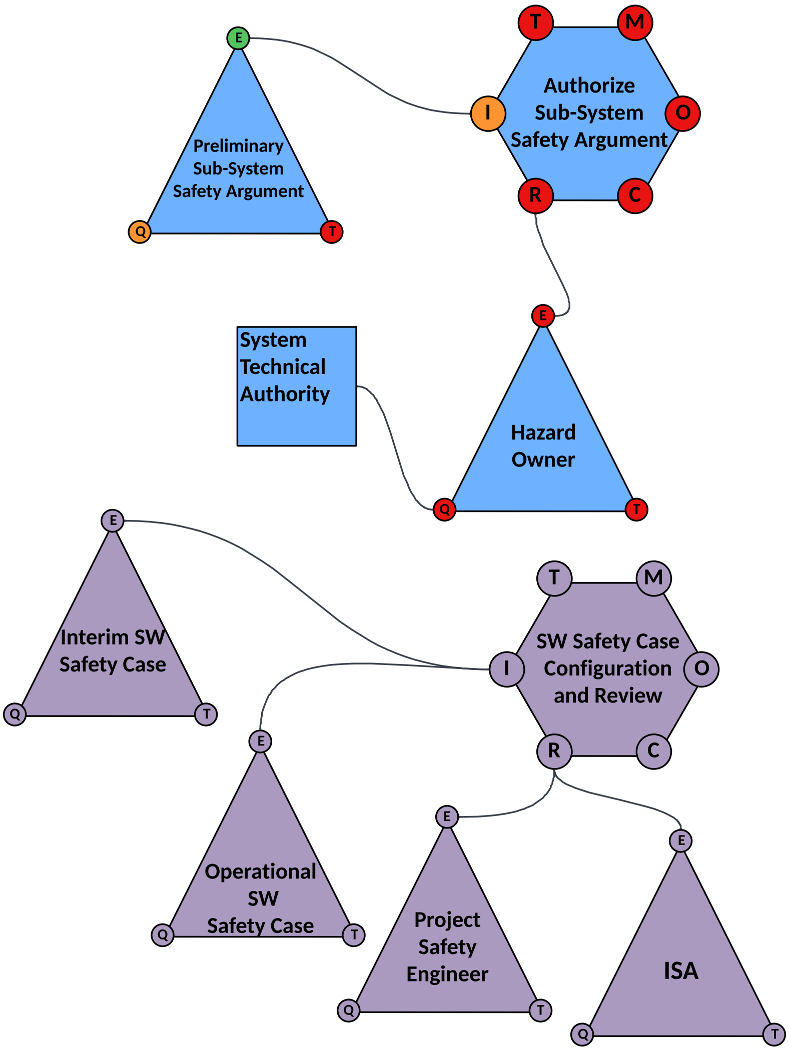

E
Preliminary Sub-System Safety Argument
Q
T
T
M
I
Authorize Sub-System Safety Argument
O
R
C
System Technical Authority
E
Hazard Owner
Q
T
E
Interim SW Safety Case
Q
T
T
M
I
SW Safety Case Configuration and Review
O
R
C
E
Operational SW Safety Case
Q
T
E
Project Safety Engineer
Q
T
E
ISA
Q
T

Figure 5.6: Extract from JB61834 Step 7 Model

5.1.7 Compare As Observed Practice with As Required (Closed) Practice 7

For this step, two modelling elements of the process must have already been completed:

1. The as-required (Closed) model of software safety practice
2. The as-observed model of safety practice.

The two outputs of this step are a report accompanying the comparison of how as-observed practice compares with as-required (Closed) practice (this section), and an appropriately represented model of how as-observed practice compares with as-required (Closed) practice, which is at [115]. An extract from the produced model showing a comparison of Safety Case activities is provided at Figure 5.6

Both original models of practice only considered the activities required to comply with Principle 1, and this is the extent of the comparison between the two models. It is noted that whilst the differences revealed by this comparison step require further investigation (RQ3), the differences are not explicitly linked to a compliance argument with Principle 1 (as this is quite weak for both models anyway).

The as-observed model of JB61834 cannot make any claim of compliance with the criteria of Principle 1, as no activity would explicitly identify the software within a system. This lack of process prevented a direct comparison of this criteria between the two models. The lack of information in this regard may be due to the focus of the respondents more than a weakness of as-observed software practice per se. This potential weakness should be investigated with JB61834, however (RQ3). It was possible that information on the software within a system (Principle 1 criteria) could be contained within, or referenced from the produced safety cases. The process for the delivery of safety cases between the two elements of practice was also compared as part of this step, therefore. The management of safety cases was not part of the original models of software safety practice.

Comparison of the two models for creating safety cases revealed terminological differences; the most obvious discrepancy being the reference to 'software safety cases' (as-required (Closed)) and 'system / subsystem' safety cases (as-observed) respectively. By recourse to the interview transcripts, this terminological difference would appear to emanate from the perspectives (and maybe colloquialisms) of the respondents, and is not deemed a significant difference. Follow-up research with the project should be carried out to confirm whether this is true (RQ3).

The as-observed model doesn't appear to consider the same maturity lifecycle of safety case management to that of the as-required (Closed) model (i.e. initial - preliminary - operational), and only considers the 'preliminary' safety case. However, as this preliminary safety case is subsequently subject to authorization, this difference is deemed to be due to the potential limitations and nuances of modelling verbal responses.

The responsible person who authorizes the safety cases is different between the two lifecycle models. The 'Hazard Owner' is responsible for safety cases in the as-observed model, and the 'Project Safety Engineer' (with input from the ISA) for the as-required (Closed) model respectively. In addition, it is not explicitly stated which activity in the as-required (Closed) lifecycle fulfils the act of 'authorization'. The as-required (Closed) lifecycle matures the safety cases iteratively (not shown for brevity at [115]) from initial safety case through to operational - and an analyst *could* assume that 'configure and review' is synonymous with the act of authorization. A simile cannot be argued between the roles of 'Hazard Owner' and 'Project Safety Engineer' however, as it has already been established in the model of as-required (Closed) practice that the hazard owner is the 'System Technical Authority' and not the 'Project Safety Engineer'. As it is perhaps more prudent for the 'Hazard Owner' to authorize safety cases, this may be an instance of documented processes being out of date, and follow-up work with JB61834 should be carried out to determine the reasons for these discrepancies (RQ3).

Whilst there are issues with terminology between the two models of practice, a comparison has been made. The resultant analysis would suggest that there are differences between the two models of practice which requires further investigation with the project.

5.1.8 Compare As Required (Closed) Practice with As Required (Open) Practice 8

For this step, two modelling elements of the process must have already been completed:

1. The as-required (Closed) model of software safety practice
2. The as-required (Open) model of software safety practice.

The two outputs of this step are a report accompanying the comparison of the two models of as-required practice (this section), and an appropriately represented model of how as-required (Closed) practice compares with as-required (Open) practice, which is at [116].

The model of the ARP 4754A lifecycle (the as-required (Open) model) created at Step 2 is divided into distinct levels of design abstraction:

- Platform
- System
- Item:
  (a) Mechanical
  (b) Hardware
  (c) Software.

JB61834, as avionics and mission system manufacturers, do not have a lifecycle of activities which considers the platform-level of design abstraction, and make assumptions that the platform-level analyses are already undertaken (with data provided to the item-level activities).

It is not possible to directly overlay the lifecycles of both ARP 4754A and JB61834 as a direct comparison (even at the Item-level of design abstraction and lower), and there are two reasons for this. The first is that the terminology and syntax of the two lifecycles are only loosely coupled (owing to the differing levels of design abstractions considered). This is especially true of the term ‘system’, which is a level of abstraction in ARP 4754A, but a product in JB61834. The second reason is that JB61834 has a mixture of legacy, new, and updated processes (as a result of the mid-life upgrades).

This required some additional modelling before an evaluation could commence, and what has been modelled and assessed is a comparison of the two lifecycles at the following levels of abstraction:

- System-level Hazard Management
- System-level Safety Requirements Management
- Software Safety Requirements Management.

An example of this additional modelling can be seen in the extract at Figure 5.7 which shows safety assessment activities at the System-level. Each level of abstraction is now considered in turn, after some generalized observations.

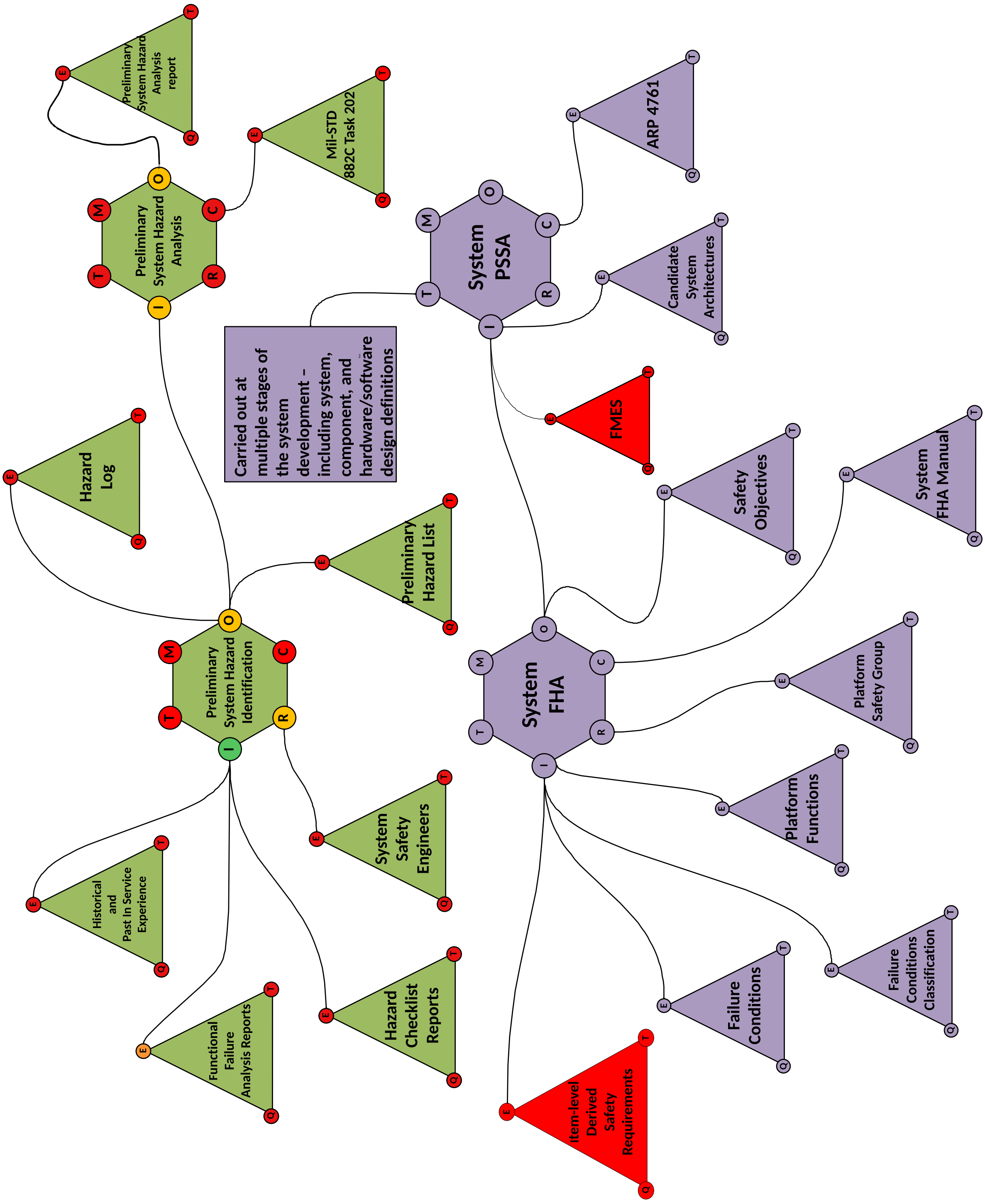

Preliminary System Hazard Analysis report
Mil-STD 882C Task 202
Preliminary System Hazard Analysis
ARP 4761
System PSSA
Candidate System Architectures
Carried out at multiple stages of the system development - including system, component, and hardware/software design definitions
Hazard Log
FMES
Safety Objectives
System FHA Manual
Preliminary Hazard List
Preliminary System Hazard Identification
System FHA
Platform Safety Group
Platform Functions
System Safety Engineers
Historical and Past In Service Experience
Functional Failure Analysis Reports
Hazard Checklist Reports
Item-level Derived Safety Requirements
Failure Conditions
Failure Conditions Classification
M
T
I
R
C
O
E
Q

Figure 5.7: Extract from JB61834 Step 8 Model

General

After completing Step 2, a number of potential deficiencies were identified with the ARP 4754A lifecycle with respect to complying with the as-desired criteria. These potential deficiencies in the lifecycle were held to be 'potential' as it may be reasonable to expect them to be recovered by organizational practice (i.e. as-required (Closed), or perhaps at the as-observed level of practice. Considering the abstraction levels compared in this step, these deficiencies are found to be NOT recovered by as-required (Closed) practice, however.

It can be seen in the comparison model at [116] that some of the aspects in the JB61834 model are coloured 'red'. This is not necessarily an indication of the sufficiency of that aspect, nor whether it is deemed worse than the ARP model's corresponding aspect. A red-coloured aspect signifies only that it has no level of agreement with the corresponding aspect in the activity / artefact of the ARP 4754A model.

In the ARP 4754A as-required (Open) practice model produced at Step 2, activities are often linked directly to each other (and not via produced / consumed artefacts). The model at [116] uses artefacts as inputs / outputs as it is assumed that activities such as a 'System FHA' will produce a 'System FHA Report'. Whilst not part of the original model created at Step 2, this use of artefacts has been undertaken for reasons of internal consistency, and the readability of the model produced by this step. It is also important to note that although some activities have iterative relationships with produced artefacts (i.e. artefacts which are both inputs to and outputs from an activity), only one direction is shown for reasons of brevity and readability of the model. This lack of iterative updates in the model is argued to be acceptable in a model used for comparison purposes.

At a system-engineering / design level, the JB61834 lifecycle majors only on software safety requirements, and does not consider the integration with system engineering activities. This is not necessarily a deficiency of the project, and may be due to the focus of the artefacts supplied for assessment.

System-Level Hazard Management

Whilst the 'Hazard Log' is the single point of truth for JB61834, and is the link between the levels of design abstraction, in ARP 4754A the system-, item- analyses feed directly into the design activities of the lower tier, and no recourse is made to a single Hazard Log.

System-Level Safety Requirements Management

Before results are discussed, it should be noted that some observations drawn out of the comparison are predicated on an inability to overlay the two models directly. The key findings are highlighted using bold text as follows:

- As well as the differences in abstraction levels between the two standards, the ARP does not further decompose 'system' analysis activities into 'sub-system' activities (whilst JB61834 does)

- JB61834 doesn't consider 'item-level' design activities and moves from system- to software-level. This is not necessarily a deficiency, as the 'system' in

JB61834's case may be a single black box (with elements therein) – and so may be an issue of terminology with respect to the levels of design abstraction

- Analysis of the ARP revealed that there is no output from the activity 'System FHA' in terms of creating a requirements specification (in mitigation of the identified hazards). This may be left reasonably to the project (as JB61834 do produce such an artefact). From an ARP perspective this is entirely reliant on the Safety Management System of the project itself following ARP 4761 (or indeed by relying on the regulatory / certifying body), however

- **Neither the ARP, nor JB61834 considers the resources required to carry out the System Safety Analyses. This cannot be left to the 'as observed' level to determine**

- ARP 4754A does not explicitly require an output from the System Safety Analyses, but this is required by the JB61834 lifecycle, which produces an analysis report - **although the required quality attributes of the report are not considered by the JB61834 lifecycle**

- Whilst the ARP lifecycle does not suggest a 'Control' attribute (aspect) by which the System Safety Analyses are carried out (likely relying on sister publications such as ARP 4761 to be followed concurrently), JB61834 uses a referenced Military Standard

- **Neither the ARP, nor JB61834 consider the time or phase by which the system safety analyses should be completed by. This cannot be left to the 'as observed' level to determine**

- **At a sub-system level of design abstraction, neither the ARP, nor JB61834 consider the resources required to carry out the required safety analysis. This cannot be left to the 'as observed' level to determine**

- The ARP does not define the resource (nor quality attributes thereof) which carry out the activity FHA, and although this is considered by the JB61834 lifecycle **the required quality attributes of the 'System Safety Engineers' are not considered**.

Software Safety Requirements Management

Results for JB61834 show that:

- Their lifecycle majors on safety activities at the software-level. These are used as the nearest comparison to the ARP

- The ativity 'Identify mitigations for SW Failure' may improve on the ARP activity 'SW Requirements Process' (as its nearest direct comparison). This potential improvement is because it considers more of the requisite attributes of an activity. Follow up work is required to ensure this *would* be an improvement to the standard (RQ3)

- Software Design activities (specifically the links to and from the safety activities) are very weak, and differ from the ARP to an extent such that no direct comparison is possible

- 'Code Review' activities are shown 'isolated' from any other activity. This is deliberate as it has no inputs. Although not capable of being assessed further, its placement on the comparison model is shown aligned with ARP activity 'SW Conformity Review' (as the nearest equivalent activity).

5.1.9 Compare As Observed Practice with As Desired Practice 9

Along with Step 10, this is a conditional step which may not necessarily have an output. The tasks for Steps 9 and 10 are identical, only the focus of the assessment, and the rationale behind any identified differences will differ.

For this step, two modelling elements of the framework instantiation process must have already been completed:

1. The model of how as-observed practice compares with as-required (Closed) practice

2. The as-desired model of software safety practice.

The output from this step is a report outlining any differences between as-required practice and as-observed (Closed) practice which specifically aims to overcome deficiencies in the as-required practice in order to comply with software safety practice as desired (this section).

Whilst the comparison for this step revealed no explicit instances of as-observed software safety practice moving away from as-required software safety practice (to recover perceived shortfalls in meeting software safety practice as-desired), responses from the interview respondents reveal two implicit instances. Both instances should be investigated with the project through further targeted research (RQ3).

The first instance relates to the use of formal software safety requirements for instantiation into a design. The careful management of software safety requirements is an essential aspect of meeting the as-desired criteria, for which both as-required (Closed) practice and as-observed practice are deficient. There is an acknowledgement from the respondents, however that they would ensure that they elicit formal software safety requirements for instantiation if they were "starting over again", and this could represent an informal 'recovery' act if this was not also mandated in project artefacts.

The second instance concerns a change in process made at the as-observed level which was not subsequently updated in the as-required software safety lifecycle process documents. When the mid-life upgrades required a change in approach to the safety requirements elicitation process, whilst the process was changed, the process documents were not updated to reflect the new approach. This may represent an example of recovering a deficiency in 'real time'.

5.1.10 Compare As Observed Practice with As Required (Open) Practice 10

Along with Step 9, this is a conditional step which may not necessarily have an output. The tasks for Steps 9 and 10 are identical, only the rationale behind any identified differences will differ.

For this Step, the model of how as-observed practice compares with as-required (Closed) practice must already have been completed.

The output from this step is a report outlining any instances of software safety practitioners undertaking working practice required by an Open Standard not associated with their project. For these instances to be of concern, the working practices highlighted must be additional and / or different to aspects of as-required software safety practice associated with their project.

Whilst difficulties with complying with Open Standards have been uncovered (including different suppliers making appeal to different Standards from JB61834), no instances of differences owing to practice perceived to be in accordance with an Open Standard other than one associated with as-required practice were revealed as part of this step.

This concludes the illustrative example of application of the framework, process and representation of software safety practice. The example demonstrates the potential efficacy of the framework and process through an illustrative example. It shows that the process has internal consistency, and can produce results. The evaluation section will consider validity issues further.

5.2 Further Illustrative Examples

We now provide details of two further illustrative examples of models created by the process to understand and assess software safety practice. Both examples arose during the empirical data gathering of this PhD programme. These two models demonstrate further the usability and usefulness of our novel framework and process to understand software safety practice.

The first example model is another model of as-required (Open) software safety practice. This example of as-required (Open) practice is in the form of a draft Open Standard which is aimed at assuring the safety of 'Health Software' (ISO/TC 215 N 2750 – IEC/CD 62304.3 [15]). The model of this standard is contained at [112], and an extract showing software safety activities is provided at Figure 5.8.

The other example model is another model of as-required (Closed) practice and emanates from the automotive industry. The project which provided their process artefacts is referred to as VF3800 for reasons of anonymity. VF3800 are designers of electric drive technology for the automotive industry, and kindly supplied their safety requirements management lifecycle artefacts. The model of their as-required (Closed) practice can be found at [110].

Initial modelling of the VF3800 practice made no judgements on the efficacy nor completeness of the lifecycle, and the only colour coding used was to identify inferred activities and artefacts (see the extract at Figure 5.9), and a lack of clarity concerning optionality / multiplicity.

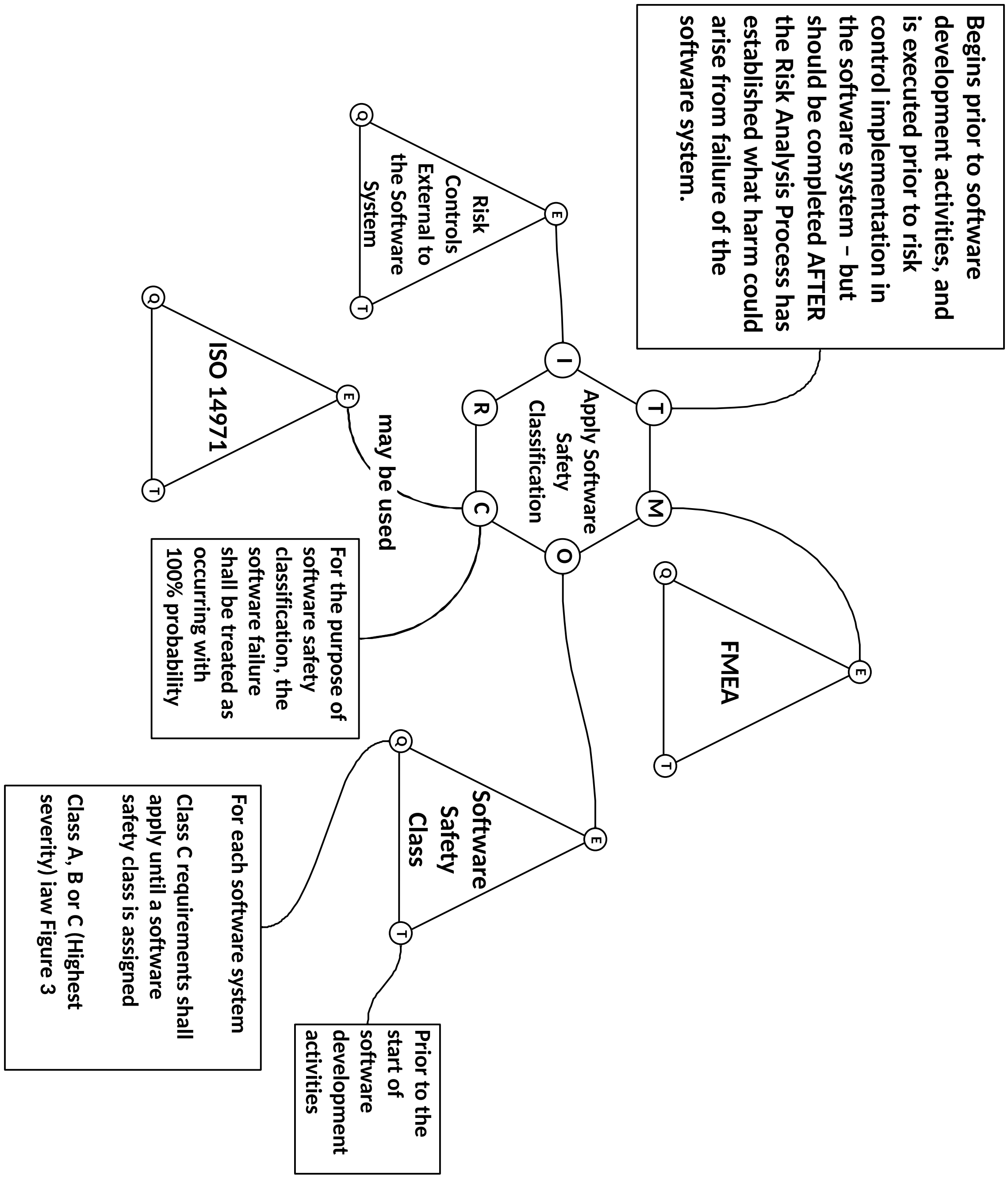

Begins prior to software development activities, and is executed prior to risk control implementation in the software system – but should be completed AFTER the Risk Analysis Process has established what harm could arise from failure of the software system.
Risk Controls External to the Software System
E
Q
T
ISO 14971
E
Q
T
Apply Software Safety Classification
I
T
M
O
C
R
may be used
FMEA
Q
E
T
For the purpose of software safety classification, the software failure shall be treated as occurring with 100% probability
Software Safety Class
Q
E
T
For each software system
Class C requirements shall apply until a software safety class is assigned
Class A, B or C (Highest severity) iaw Figure 3
Prior to the start of software development activities

Figure 5.8: Extract from ISO 62304.3 Model

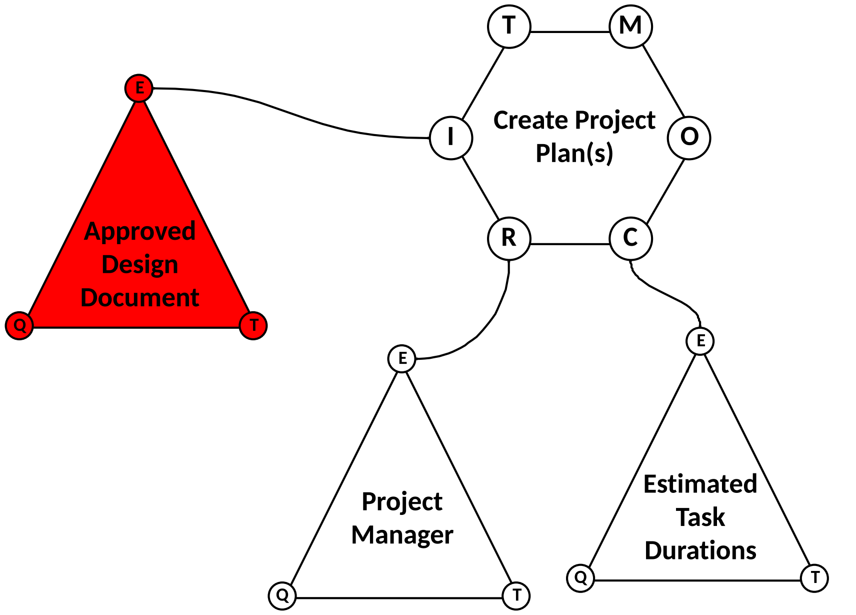


Figure 5.9: Extract from VF3800 As-Required (Open) Model

The project provided three process documents that were extracted from their electronic lifecycle management tool:

1. Design System Architecture
2. Elicit Stakeholder Requirements
3. Specify Requirements.

There were no referenced documents in the artefacts provided, and unfortunately, requests for clarification and the provision of further artefacts were unanswered. The company confirmed they were overhauling their processes, and that the artefacts supplied were an accurate representation of their processes in their current state (as of July 2021), however.

It is clear (from [110]) that the artefacts provided cannot represent the entirety of the company software safety process lifecycle, and requests were made for further data. Despite persistent attempts at contact with the project; with requests for further artefacts, this yielded no results until a point by which the participant in the project gave notice of their intent to leave their organization. Regrettably, there was no response from the participant's successor in the project. As such, no further modelling or analysis of the VF3800 software safety assurance lifecycle was undertaken.

Whilst no empirically significant assessment can be made of the organization's processes and inter-relationships between the elements, the model at [110] further highlights the utility and effectiveness of the framework and process.

5.3 Discussion

This chapter has provided the empirical data from applying the framework and process to understand software safety engineering practice. he data provided by the application of the framework supports the two research questions.

Research Question 1 seeks to answer the question of how an organization can understand its software safety practice. The application of the framework and process has created models of the organization's software safety practice, and this has enabled an understanding of all elements of their practice and the interrelationships of these elements of practice.

Research Question 2 seeks to answer the question of how an organization can assess its software safety practice. The application of the framework and process has revealed internal inconsistencies with the organization's documented processes; inter-team tensions; disagreements between elements of practice; and has uncovered impediments to achieving software safety practice as-desired. This data may not have been revealed to the organization had the framework and process not been followed.

No generalizations are made from the data provided, and it is not possible to identify any themes from a single application of the framework and processes. We do not know whether any themes will emerge from future work (see Chapter 7), as any disagreements, discrepancies, or impediments to achieving as-desired practice for software safety may be localized and pertinent to the organization, sector, and application.

Chapter 6 now considers the extent to which the research objectives have been met - by evaluating the framework and process itself.

Chapter 6

Evaluation of the Proposed Process

This chapter aims to establish how far the research undertaken has fulfilled the research objective and research questions identified in Chapter 2.6.

Research Objective One is to provide a process by which an organization can understand and assess the disparate elements that constitute their software safety practice. This objective is met by answering two research questions: how can an organization understand its software safety practice; and how can an organization assess its software safety practice?

These research questions have been answered by the creation of the framework and process for understanding software safety practice. This chapter aims to present a compelling argument and associated evidence as to the 'goodness' of the framework and associated process.

The framework and process' ability to allow an organization to understand and assess software safety practice is evaluated using five success criteria. The success criteria is derived from the framework design decisions articulated in Chapter 3, and the non-functional requirements used to select the graphical notation (Chapter 4). Namely, the process:

1. Is complete

2. Is easy to use

3. Represents all elements of software safety practice in a consistent manner

4. Is effective, and

5. Is applicable for use in any safety-critical industry

We will elaborate on what these criteria mean explicitly in the following sections.

We provide an argument to demonstrate how the research aims and objectives have been met by this thesis. This is shown in Figure 6.1 using Goal Structuring Notation (GSN). Each criterion is represented as a 'Goal', and each of these goals is now considered in turn. The labels in parentheses correspond to the labels in the associated GSN diagram (i.e those shown in Figures 6.2 to 6.7).

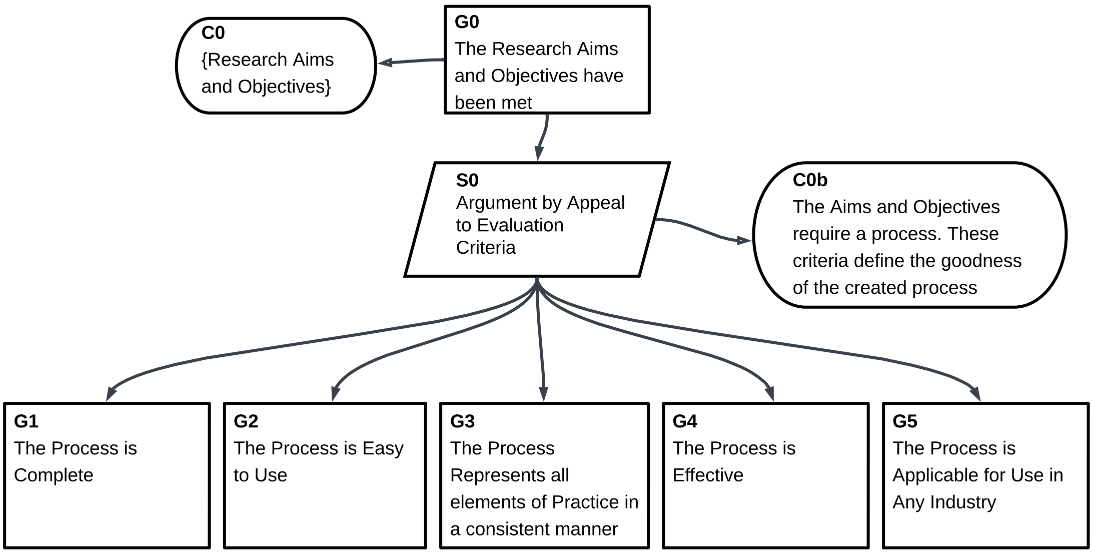

C0
{Research Aims and Objectives}
G0
The Research Aims and Objectives have been met
S0
Argument by Appeal to Evaluation Criteria
C0b
The Aims and Objectives require a process. These criteria define the goodness of the created process
G1
The Process is Complete
G2
The Process is Easy to Use
G3
The Process Represents all elements of Practice in a consistent manner
G4
The Process is Effective
G5
The Process is Applicable for Use in Any Industry

Figure 6.1: Thesis Evaluation Criteria

6.1 G1: The Process is Complete

Goal G1 is instantiated by the argument structure at Figure 6.2. The goal that 'The Process is Complete' (G1) is made in the context of what defines 'completeness'. In this case, the process is argued to be complete if it covers all elements of software safety practice (C1). The need for completeness is justified as impediments to best practice can occur anywhere in the software safety lifecycle (J1), and it is argued in Chapter 3 *how* this 10-step framework and process is complete (J1b).

Goal G1 is divided into an appeal to evaluation by independent experts (G1.1) and a strategy which appeals to the framework itself (S1).

6.1.1 G1.1 Independent Experts have Found the Process Complete

This goal argues that independent experts have reviewed the process to instantiate the framework, and have provided a scored assessment of the completeness of the process ('Evaluation Question EQ1' at Sn1.1). The scoring of the assessment follows a structured evaluation session - the details of which are at Annex D.3.

The evaluation question EQ1 asks independent experts (whose careers have spanned multiple domains and technologies (J1.13)) to annotate their levels of agreement with the statement "The process considers all elements that together constitute software safety practice (the 10 'steps' which assert the elements of software safety practice; the models of practice created; and the assessment of the models of practice)". The scoring mechanism used was in the form of a Likert scale as follows (Context C1.11):

- Fully Disagree (1)
- Somewhat Disagree (2)

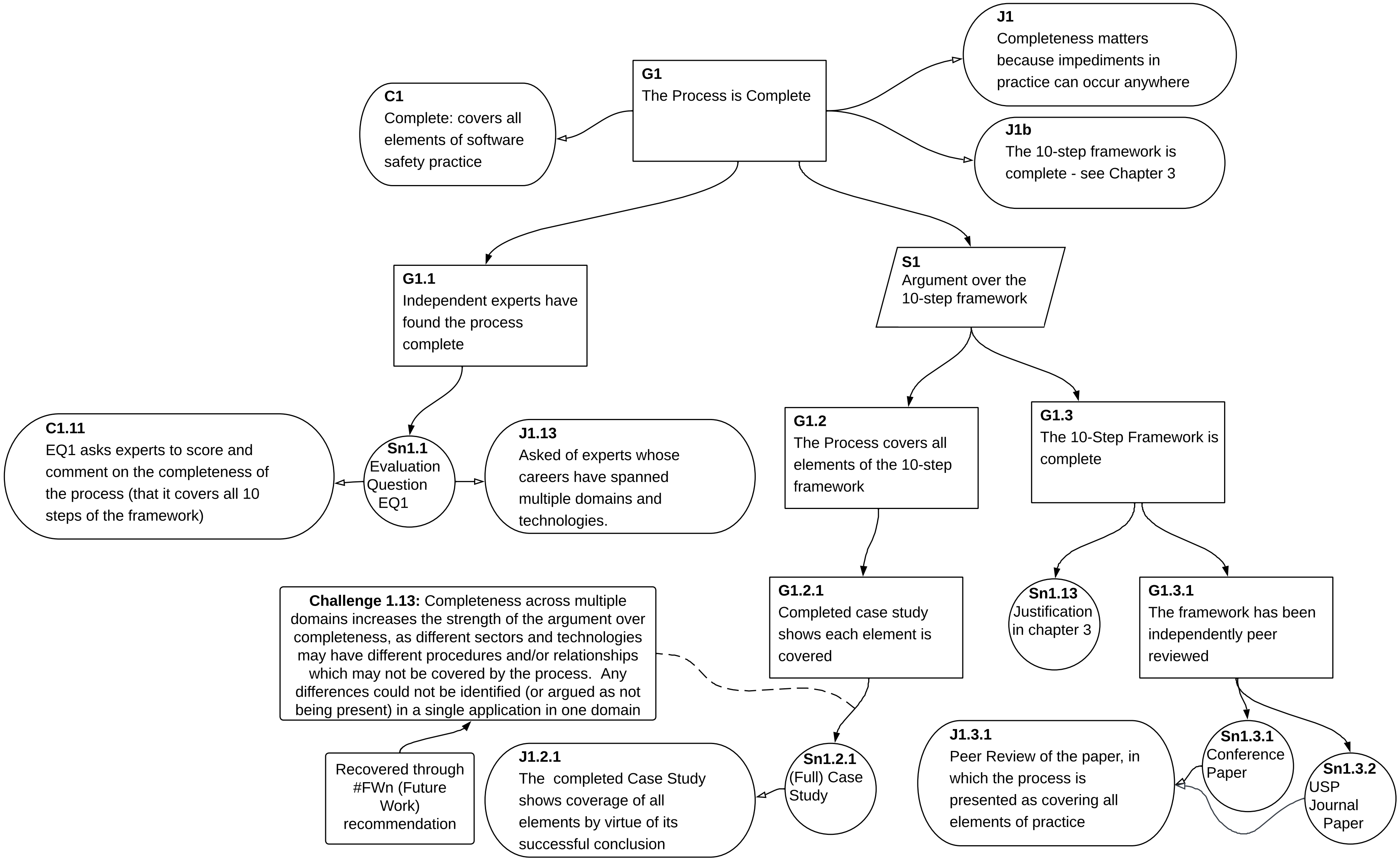

G1
The Process is Complete
C1
Complete: covers all elements of software safety practice
J1
Completeness matters because impediments in practice can occur anywhere
J1b
The 10-step framework is complete - see Chapter 3
G1.1
Independent experts have found the process complete
S1
Argument over the 10-step framework
G1.2
The Process covers all elements of the 10-step framework
G1.3
The 10-Step Framework is complete
C1.11
EQ1 asks experts to score and comment on the completeness of the process (that it covers all 10 steps of the framework)
Sn1.1
Evaluation Question EQ1
J1.13
Asked of experts whose careers have spanned multiple domains and technologies.
G1.2.1
Completed case study shows each element is covered
Sn1.13
Justification in chapter 3
G1.3.1
The framework has been independently peer reviewed
Challenge 1.13: Completeness across multiple domains increases the strength of the argument over completeness, as different sectors and technologies may have different procedures and/or relationships which may not be covered by the process. Any differences could not be identified (or argued as not being present) in a single application in one domain
Recovered through #FWn (Future Work) recommendation
J1.2.1
The completed Case Study shows coverage of all elements by virtue of its successful conclusion
Sn1.2.1
(Full) Case Study
J1.3.1
Peer Review of the paper, in which the process is presented as covering all elements of practice
Sn1.3.1
Conference Paper
Sn1.3.2
USP Journal Paper

Figure 6.2: Goal G1 - The Process is Complete

- Neither Agree / Disagree (3)
- Somewhat Agree (4)
- Fully Agree (5)

No claim is made that this use of the Likert scale to assess the question's responses will generate 'known statistical characteristics' [108], and the general weaknesses of the Likert scale are widely-known (such as not being reproducible; the ability to derive equal overall scores using different question sets; and the vagaries associated with the interpretation of 'neutral scores' [108]).

These potential weaknesses are further compounded in this case by the low number of respondents - but it would not be feasible in the time constraints of a single PhD programme to generate enough questionnaire responses (which are only requested on completion of an evaluation session) to argue statistical significance. It is argued that this potential weakness is offset by the use of experts with a demonstrable record of expertise in multiple domains and applications, however. Further, no generalizations are made from the question's responses as they serve merely as a score against the perceived utility of the framework and process attributes directly.

Sn1.1

Sn1.1 is the output of Evaluation Question EQ1. Evaluation Question 1 was scored by the anonymous respondents as follows:

AY8697
AY8697 'somewhat disagreed' with the notion that 'the process considers all elements that together constitute software safety practice', providing the following qualifying statement:

> *"Subsidiary processes, such as quality, configuration management, etc., are not typically defined in standards or desired practice but are implicitly assumed, yet software safety is implicitly dependent upon them."*

SH27236
SH27236 'somewhat agreed' with the notion that 'the process considers all elements that together constitute software safety engineering practice', providing the following qualifying statement:

> *"It is not clear that the 'as desired' would fully capture all relevant aspects – I guess it comes down to what is understood by 'practice' and what is in scope of 'desired'. Activities and Artefacts could address aspects such as competence, supply-chain management, COTS etc, but it is not clear to me that the process inherently captures this without a reference model for the 'as desired'."*

HH75783
HH75783 'somewhat agreed' with the notion that 'the process considers all elements that together constitute software safety engineering practice', providing

the following qualifying statement:

> *"For this I considered: 1) Tool qual 2) tabular SW quality audit of non-safety OTS sw 3) goal based assessment 4) CI/CD. I think they all work, 2 may be challenging but I think shortfall management, limitations on use, justifications for non-compliance all covered. Step 8 assumes there is a project lifecycle (this may be very poorly defined for some OTS devices, I could tell some stories)."*

It *could* be that the feedback from AY8697 is not an indictment on the framework and process' ability to consider all elements that together constitute software safety practice, but instead a comment that relates to a perceived weaknesses of extant as-required practice. The reasoning behind this assertion is borne out by the comments from the other two respondents - whose comments also seem to express difficulties for an analyst to infer the intent of documented practice. It could also be a failure of EQ1 to clearly ask the right question, but this is argued not to be the case as the feedback from two of the respondents suggest the process *is* capable. Whilst not statistically significant, it provides increased confidence that Goal G1.1 is met.

The knowledge and experience of the analyst is fundamental to the successful application of this process - not least in the ability of an analyst to observe what may be *missing* from an element of practice. This is evident from the feedback received.

The argument of what constitutes a SQEP analyst was declared out of scope in Section 3.2. The need to define the competence and competencies of the analyst undertaking the process to instantiate the framework is mitigated by recourse to future work, see Recommendation 1.

6.1.2 Strategy S1 Argument Over the 10-Step Framework

The strategy to appeal to the framework and 10-step process is realized by the Goals G1.2 and G1.3, and these sub-goals are now considered in turn.

Goal G1.2 The Process Covers all Elements of the 10-Step Framework

This goal is instantiated by arguing that the "Completed Case study Shows each Element is Covered". This Goal is achieved by the completion of the illustrative example detailed in Chapter 5 (Sn1.2.1). The use of this solution is justified as Chapter 5 shows coverage of all elements of software safety practice by virtue of its successful conclusion (J1.2.1).

As a challenge (Challenge 1.13 in Figure 6.2), it is argued that 'completeness' across multiple domains increases the strength of the argument over completeness, as different sectors and technologies may have different procedures and/or relationships which may not be covered by the process. Any differences could not be identified (or argued as not being present) in a single application in one domain. This is addressed by a recommendation for future work (see Recommendation 3).

Goal G1.3 The 10-Step Framework is Complete

This goal is instantiated by appeal to a solution node and a further sub-goal. The solution node (Sn1.13) is the justifications made in Chapter 3, specifically "All existing elements of software safety practice can be mapped onto these three elements, and whilst it is argued this is necessary, it cannot yet be argued whether this is complete - although further instantiations of the process will reveal the levels of confidence in completeness."

Goal G1.3.1 claims that "The Framework has been Independently Peer Reviewed", and is instantiated by appealing to the published Conference Paper at [130] (Sn1.3.1) and a peer-reviewed Journal Paper at [128] (Sn1.3.2). These two solution nodes are justified as being sufficient as the claims over completeness made therein have been subjected to expert and independent peer review.

6.2 G2: The Process is Easy to Use

Goal G2 is instantiated by the argument structure at Figure 6.3 and is made in the context that 'easy' refers to the framework's implementation by a software safety practitioner with at least five years of experience (C2).

This goal is justified in its use as the process will not be adopted by projects if it is NOT easy to use (J2). Goal G2 is instantiated by appealing to individual criteria for 'ease' (S2) and an assumption that the aggregation of these individual criteria equates to an ease of use (A2). The criteria for being easy to use are:

1. The process instructions are written in a manner that allows them to be followed
2. The modelling symbology used in the process is readily understandable
3. The modelling symbology used in the process is readily usable
4. No formal modelling skills are required for the process to be followed
5. The process can be instantiated by anyone with recourse to portable office applications
6. Independent experts with no prior knowledge of the process were able to complete the Case Study.

Each individual criterion is argued by the sub-goals G2.1 to G2.6 inclusive. Each of these goals are now considered in turn.

6.2.1 Goal G2.1 The Process Instructions are Written in a Manner that Allows them to be Followed

This goal is instantiated by asking experts to to score and comment on their ability to follow each step of the instructions (Sn2.11 and C2.11). The evaluation question EQ6 asks independent experts to annotate their levels of agreement with the statement "The modelling process instructions are easy to follow (you could follow each step)" using the same Likert Scale as introduced for Goal G1.1.

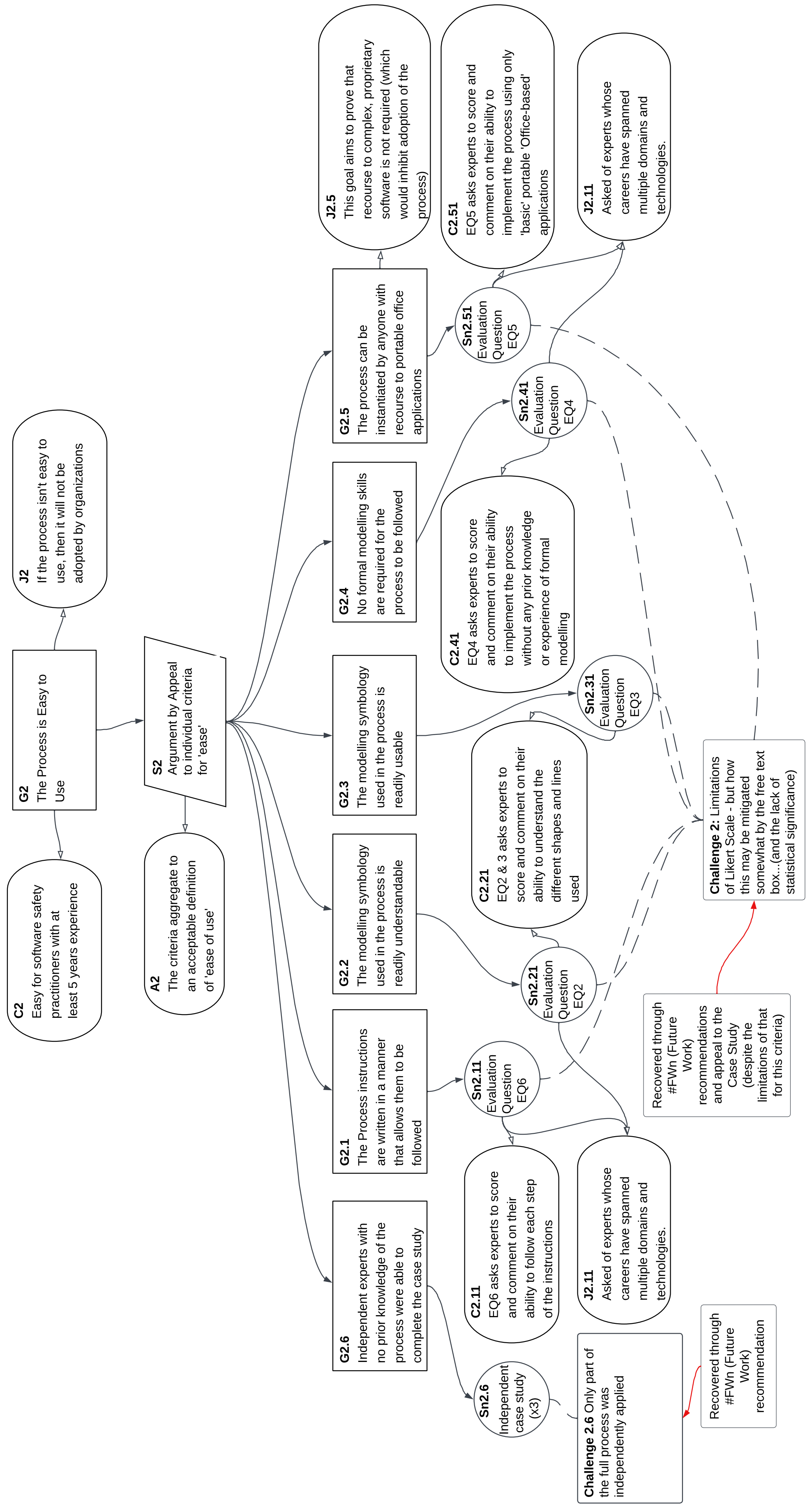

G2
The Process is Easy to Use
C2
Easy for software safety practitioners with at least 5 years experience
J2
If the process isn't easy to use, then it will not be adopted by organizations
S2
Argument by Appeal to individual criteria for 'ease'
A2
The criteria aggregate to an acceptable definition of 'ease of use'
G2.5
The process can be instantiated by anyone with recourse to portable office applications
J2.5
This goal aims to prove that recourse to complex, proprietary software is not required (which would inhibit adoption of the process)
G2.4
No formal modelling skills are required for the process to be followed
G2.3
The modelling symbology used in the process is readily usable
G2.2
The modelling symbology used in the process is readily understandable
G2.1
The Process instructions are written in a manner that allows them to be followed
G2.6
Independent experts with no prior knowledge of the process were able to complete the case study
Sn2.51
Evaluation Question EQ5
C2.51
EQ5 asks experts to score and comment on their ability to implement the process using only 'basic' portable 'Office-based' applications
J2.11
Asked of experts whose careers have spanned multiple domains and technologies.
Sn2.41
Evaluation Question EQ4
C2.41
EQ4 asks experts to score and comment on their ability to implement the process without any prior knowledge or experience of formal modelling
Sn2.31
Evaluation Question EQ3
C2.21
EQ2 & 3 asks experts to score and comment on their ability to understand the different shapes and lines used
Sn2.21
Evaluation Question EQ2
Sn2.11
Evaluation Question EQ6
C2.11
EQ6 asks experts to score and comment on their ability to follow each step of the instructions
J2.11
Asked of experts whose careers have spanned multiple domains and technologies.
Sn2.6
Independent case study (x3)
Challenge 2.6 Only part of the full process was independently applied
Recovered through #FWn (Future Work) recommendation
Challenge 2: Limitations of Likert Scale - but how this may be mitigated somewhat by the free text box...(and the lack of statistical significance)
Recovered through #FWn (Future Work) recommendations and appeal to the Case Study (despite the limitations of that for this criteria)

Figure 6.3: Goal G2 - The Process is Easy to Use

Sn2.11
Sn2.11 is the output of Evaluation Question EQ6. Evaluation Question 6 was scored by the respondents as follows:

AY8697
AY8697 'somewhat agreed' with the notion that 'the modelling process instructions are easy to follow', providing the following qualifying statement:

> *"Standard bits were very easy, but nuances were far more difficult to work out the best approach, e.g. how selection of activities can be modified via the Safety Management Plan, which would impact on how the generic process is modelled."*

SH27236
SH27236 'neither agreed nor disagreed' with the notion that 'the modelling process instructions are easy to follow', providing the following qualifying statement:

> *"Only part of Step 2 has been followed so far... too early to tell. It seems like the scope of application is to software based systems, which is much broader than software safety engineering. I think it would be helpful to be clearer in the scope – what is included and what is not."*

HH75783
HH75783 'fully agreed' with the notion that 'the modelling process instructions are easy to follow', providing the following qualifying statement:

> *"Yes, but a couple of small examples in the process instructions would be helpful."*

AY8697's comment on the difficulties in establishing the required activities in an as-required (Closed) process (i.e. the Safety Management Plan) is acknowledged. However, we noted in Chapter 3.1.4 that the process to translate as-desired practice into as-required practice is not in scope for the framework.

The suggestion to provide examples is of course valid, but an evaluation design decision was to deliberately *not* provide anything but the templates and process instructions for the purpose of independent evaluation. The tutorial included examples (see [121]) and an organization would have access to all examples shown in this thesis when applying the framework and process.

The responses were either positive or neutral. The neutral (neither agree nor disagree) response was relating to the fact that the respondent hadn't completed the full process. This is acknowledged, and somewhat predictable as the design decision taken was to limit the amount of time and effort we could reasonably expect potential respondents to commit to. As such it is argued that this evidence node supports a claim that Goal 2.1 has been met.

6.2.2 Goal G2.2 The Modelling Symbology Used in the Process is Readily Understandable

This goal is instantiated by asking experts to to score and comment on their ability to understand the different shapes and lines used in the adapted version of FRAM (Sn2.21 and C2.21). The evaluation question EQ2 asks independent experts to annotate their levels of agreement with the statement “The modelling symbology is easy to understand (you knew what the different shapes and lines represented)" using the same Likert Scale as introduced for Goal G1.1.

Evaluation Question 2 was asked on completion of the second evaluation session. The completed questionnaires are at Appendix G.

Sn2.21

Sn2.21 is the output of Evaluation Question EQ2. Evaluation Question 2 was scored by the respondents as follows:

SH27236
SH27236 'somewhat agreed' with the notion that 'the modelling symbology is easy to understand', providing the following qualifying statement:

"I needed the reference key to remind me – would probably be less required with frequent use, but would likely need a 'refresh' after gaps in use."

AY8697
AY8697 'somewhat agreed' with the notion that 'the modelling symbology is easy to understand', provding the following qualifying statement:

"A key to the attributes would have been useful."

HH75783
HH75783 'fully agreed' with the notion that 'the modelling symbology is easy to understand', providing the following qualifying statement:

"I understood the symbols and concepts. I needed a key as a reminder for the exercises but once more familiar with the notation am sure would be fine."

The comments and scores are largely positive, and it is clear that the respondents needed to make frequent reference to the supplied key to the symbology used. With hindsight, the fact that the key to attributes was actually provided to the respondents should have been made much clearer. This does not detract from a claim that evidence Sn2.21 supports Goal G2.2, however.

6.2.3 Goal G2.3 The Modelling Symbology Used in the Process is Readily Usable

This goal is instantiated by asking experts to score and comment on their ability to *use* the different shapes and lines required (Sn2.31 and C2.31). The evaluation question EQ3 asks independent experts to annotate their levels of agreement with the statement “The modelling symbology is easy to use (you could easily use the different shapes and lines to construct a model)” using the same Likert Scale as introduced for Goal G1.1.

Sn2.31

Sn2.31 is the output of Evaluation Question EQ3. Evaluation Question 3 was scored by the respondents as follows:

SH27236
SH27236 'neither agreed nor disagreed' with the notion that 'The modelling symbology is easy to use', providing the following qualifying statement:

"Wrt EQ3: I found the scope of the models being compared were not the same, and used different terminology – even where the terms used were the same (or similar) it was not clear that they meant the same thing."

AY8697
AY8697 'somewhat agreed' with the notion that 'The modelling symbology is easy to use' providing the following qualifying statement:

"A key to the attributes would have been useful."

HH75783
HH75783 'fully agreed' with the notion that 'The modelling symbology is easy to use', providing the following qualifying statement:

"I understood the symbols and concepts. I needed a key as a reminder for the exercises but once more familiar with the notation am sure would be fine."

The comments on having a key to the attributes have been discussed for EQ2, but attention now turns to the comment from SH27236. It is asserted that their comment is an indictment on the relationships between the disparate elements of software safety practice, rather than a critique concerning the ease of use of the modelling symbology. Whilst the issues noted may present complications when carrying out the framework and process, it is a 'finding' discovered by the design intent of the framework and process process, and a potential issue that the organization should conduct further research on. It is therefore claimed that evidence node Sn2.31 supports Goal G2.3.

6.2.4 Goal G2.4 No Formal Modelling Skills are Required for the Process to be Followed

This goal is instantiated by asking experts to to score and comment on their ability to implement the process without any prior knowledge or experience of formal modelling (Sn2.41 and C2.41). The evaluation question EQ4 asks independent experts to annotate their levels of agreement with the statement “The process to model software safety practice can be carried out without any prior knowledge of formal modelling (i.e. no training in model-based systems engineering was required)" using the same Likert Scale as introduced for Goal G1.1.

Sn2.41

Sn2.41 is the output of Evaluation Question EQ4. Evaluation Question 4 was scored by the respondents as follows:

SH27236
SH27236 'fully agreed' with the notion that 'the process to model software safety practice can be carried out without any prior knowledge of formal modelling', providing the following qualifying statement:

"The 'editorial' aspects of applying colours uses similar skills, but the judgement of equivalence is a different skillset."

AY8697
AY8697 'somewhat agreed' with the notion that 'the process to model software safety practice can be carried out without any prior knowledge of formal modelling'.

HH75783
HH75783 'neither agreed nor disagreed' with the notion that 'the process to model software safety practice can be carried out without any prior knowledge of formal modelling'. They also made the following qualifying statement:

"Reflecting on this I'm not sure I can answer EQ4 with confidence because I am familiar with these MBSE concepts?"

A potential weakness of this piece of evidence stems from the possibility that one or more of the independent experts may have former modelling knowledge. It is argued that this threat to validity is mitigated by both asking the expert whether *anyone* could instantiate the process in the absence of formal modelling skills (i.e. not whether they are personally capable), and that the question relates to the use of this specific graphical representation (and not formal modelling in general). Noting that HH75783 was already familiar with the concepts of model based systems engineering, these responses are still largely positive.

SH27236 notes that the ability to judge equivalence is an entirely different skillset to that of creating a graphical representation, and is argued to be mitigated by recourse to Recommendation 1. It is therefore claimed that evidence node Sn2.41 supports Goal G2.4.

6.2.5 Goal G2.5 The Process can be Instantiated by Anyone with Recourse to Portable Office Applications

This goal is instantiated by asking experts to to score and comment on their ability to implement the process using only 'basic' portable 'Office-based' applications

(Sn2.51 and C2.51). The evaluation question EQ5 asks independent experts to annotate their levels of agreement with the statement "The process can be instantiated by anyone with access to standard 'Office' applications (such as Visio, Lucid Chart, Word, Pages, Google Docs etc.)" using the same Likert Scale as introduced for Goal G1.1. The use of this goal is justified as it aims to prove that recourse to complex, proprietary software is not required (which could inhibit adoption of the framework and process - see the framework design decision made in Chapter 3.1) (J2.5).

Sn2.51

Sn2.51 is the output from Evaluation Question EQ5. Evaluation Question 5 was scored by the anonymous respondents as follows:

SH27236
SH27236 'somewhat agreed' with the notion that 'the process can be instantiated by anyone with access to standard 'Office' applications'.

AY8697
AY8697 'neither agreed nor disagreed' with the notion that 'the process can be instantiated by anyone with access to standard 'Office' applications', providing the following qualifying statement:

> *"Visio and Lucid charts are not part of our standard office tools. Most people would probably try to use it in word or powerpoint, which would make it more difficult, I think."*

HH75783
HH75783 'fully agreed' with the notion that 'the process can be instantiated by anyone with access to standard 'Office' applications'.

The response from AY8697 is noted, and it is accepted that an organization would be limited to the tools provided. However, if an organization is committing substantial resource to understand and assess its software safety practice, then it is not unreasonable to assume that the project would also commit to a modest investment associated with software license costs. It is therefore claimed that evidence node Sn2.51 supports Goal G2.5.

6.2.6 Goal G2.6 Independent Experts with no Prior Knowledge of the Process were Able to Complete the Case Study

This goal is instantiated by an independent Case Study which was undertaken separately by independent experts in functional safety. After a tutorial [121], each expert was given process instructions [120], an excerpt from an Open Standard [119], and a template of the symbols with which a FRAM model could be compiled (sent via email after the tutorial). The three models created by the independent experts are found at [122], and it is therefore claimed that evidence node Sn2.6 fully supports Goal G2.6.

A 'compliance' step (i.e. in addition to this comparison step) could also have been carried out in support of Goal G2.6 (i.e. asking the independent experts to undertake Steps 5 or 6 of the process to understand software safety practice), and this is a potential limitation of the evaluation. In mitigation, it is argued that the process steps for comparisons, and the process steps for compliance checks are sufficiently similar for the purposes of independent evaluation, because:

- Both kinds of steps require the creation of two models
- Both kinds of steps require the comparison of two models
- Both kinds of steps involve a critical assessment
- Both kinds of steps require one of the models to be annotated with a pre-defined colour-coding scheme.

Further mitigation is also claimed by recourse to future work to apply the framework and process in full, and to more projects (see Recommendation 4).

6.3 G3: The Process Provides a Way to Represent all Elements of Practice in a Consistent Manner

Goal G3 is instantiated by the argument structure at Figure 6.4 and is made in the context that consistent is defined as "done in the same manner" (C3). The use of this goal is justified as the different elements of software safety practice must be modelled in a consistent manner to facilitate meaningful comparisons and evaluations (J3).

Goal G3 is instantiated by appealing to three criteria for 'consistency' (S3):

1. The process creates models of each element of practice using consistent terminology
2. The model of each element of software safety practice is created using identical modelling symbology
3. Independent people produced consistent model representations.

Each individual criterion is argued by the sub-goals G3.1 to G3.3. inclusive. Each of these sub-goals are now considered in turn.

6.3.1 Goal G3.1 The Process Creates Models of Each Element of Practice Using Consistent Terminology

This goal is instantiated by both asking experts to score and comment on the consistency of the terminology used in the process (Sn3.1), and by appeal to the outputs of Steps 1, 2, 3 and 4 in Chapter 5 (Sn3.1b). The evaluation question EQ12 asks independent experts to annotate their levels of agreement with the statement "The process uses consistent terminology when considering each different element that constitutes software safety practice" using the same Likert Scale as introduced for Goal G1.1. Goal G3.1 is claimed in the context of the terms used for each element of the process and the models (C3.1). The use of this goal is justified as the different models must be created in a manner that allows for 'like-for-like' comparison and assessment (J3.2)

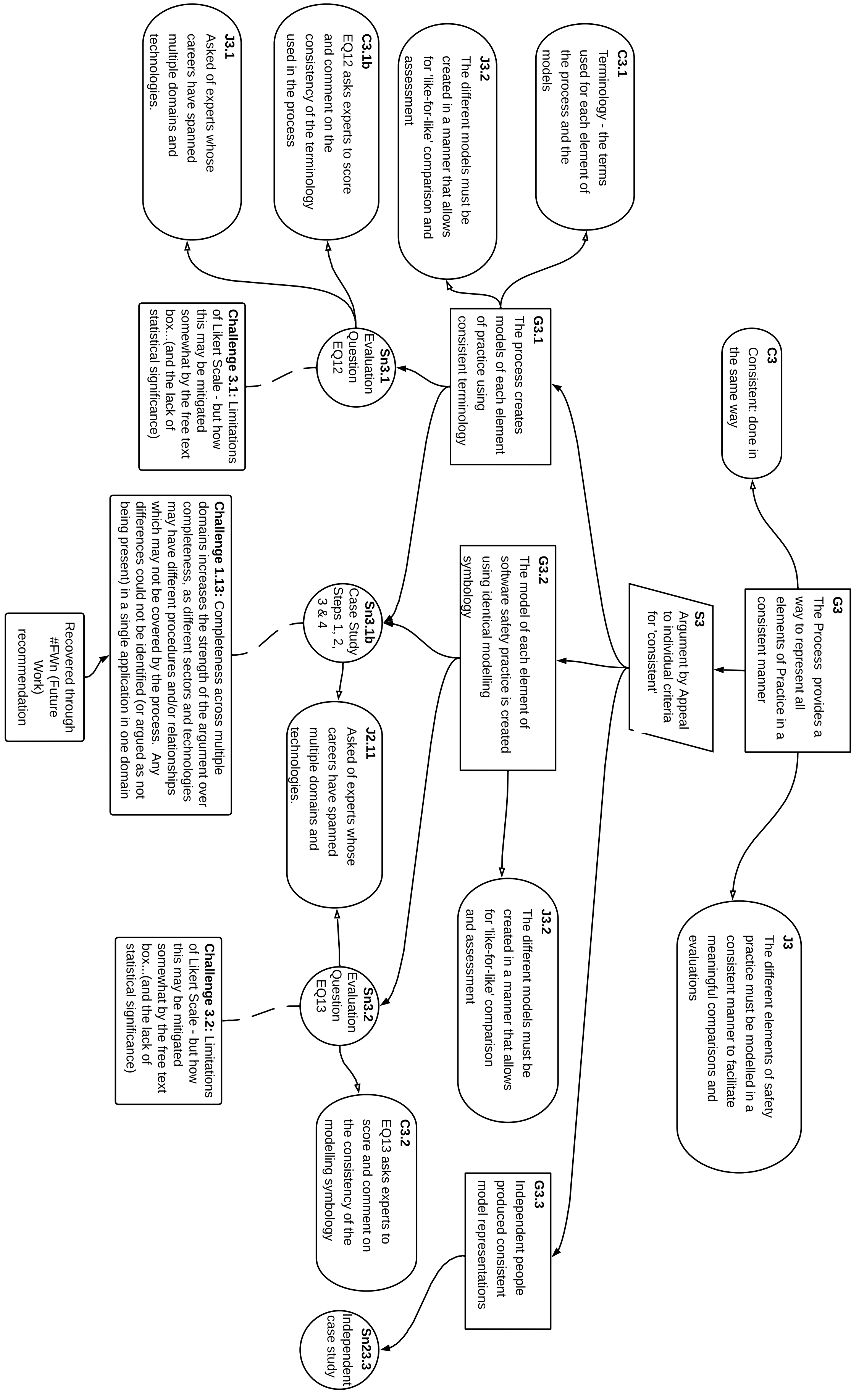

G3
The Process provides a way to represent all elements of Practice in a consistent manner
C3
Consistent: done in the same way
J3
The different elements of safety practice must be modelled in a consistent manner to facilitate meaningful comparisons and evaluations
S3
Argument by Appeal to individual criteria for 'consistent'
G3.1
The process creates models of each element of practice using consistent terminology
G3.2
The model of each element of software safety practice is created using identical modelling symbology
G3.3
Independent people produced consistent model representations
C3.1
Terminology - the terms used for each element of the process and the models
J3.2
The different models must be created in a manner that allows for 'like-for-like' comparison and assessment
C3.1b
EQ12 asks experts to score and comment on the consistency of the terminology used in the process
J3.1
Asked of experts whose careers have spanned multiple domains and technologies.
Sn3.1
Evaluation Question EQ12
Challenge 3.1: Limitations of Likert Scale - but how this may be mitigated somewhat by the free text box...(and the lack of statistical significance)
Sn3.1b
Case Study Steps 1, 2, 3 & 4
Challenge 1.13: Completeness across multiple domains increases the strength of the argument over completeness, as different sectors and technologies may have different procedures and/or relationships which may not be covered by the process. Any differences could not be identified (or argued as not being present) in a single application in one domain
Recovered through #FWn (Future Work) recommendation
J2.11
Asked of experts whose careers have spanned multiple domains and technologies.
J3.2
The different models must be created in a manner that allows for 'like-for-like' comparison and assessment
Sn3.2
Evaluation Question EQ13
Challenge 3.2: Limitations of Likert Scale - but how this may be mitigated somewhat by the free text box...(and the lack of statistical significance)
C3.2
EQ13 asks experts to score and comment on the consistency of the modelling symbology
Sn23.3
Independent case study

Figure 6.4: Goal G3 - The Process Provides a Way to Represent all Elements of Practice in a Consistent Manner

Sn3.1

Sn3.1 is the output of Evaluation Question EQ12, and it partially instantiates Goal G3.1 along with Sn3.1b. Evaluation Question 12 was scored by the anonymized respondents as follows:

SH27236
SH27236 'somewhat agreed' with the notion that 'the process uses consistent terminology when considering each different element that constitutes software safety practice', providing the following qualifying statement:

> *"I found it useful to model at different levels of granularity, This can lead to a hierarchy of 'practice' models being applied across a hierarchy of product architectures and at differing levels of abstraction. Which works best in a given situation will likely depend on context and intent of the exercise to 'understand system safety engineering practice' and is likely to be a learned skill from experience. Your research may offer an insight into the validity of this observation."*

AY8697
AY8697 'fully agreed' with the notion that 'the process uses consistent terminology when considering each different element that constitutes software safety engineering practice', providing the following qualifying statement:

> *"The interpretation of nuance in standards or observed practice will lead some people to model things in different ways to others. I don't think that can ever be fully resolved, but this type of standardisation goes a long way towards helping with that!"*

HH75783
HH75783 'fully agreed' with the notion that 'the process uses consistent terminology when considering each different element that constitutes software safety engineering practice', providing the following qualifying statement:

> *"I had no difficulty understanding the concepts and terminology."*

The responses and the comments are positive (notwithstanding the comments from SH27236 which again emphasizes the importance of a skilled analyst for the implementation of this framework and process (see Recommendation 1)). It is therefore argued that Sn3.1 supports the claim that Goal G3.1 is met.

Sn3.1b

Sn3.1b addresses the outputs of the process Steps 1 to 4 described in Chapter 5:

- STEP 1: produced a set of criteria which, when met will ensure the as-desired criteria would be met. The criteria was taken from the 4 + 1 Principles - which are argued to be *"constant across domains and across projects, and can be regarded as the immutable core of any software safety justification"* [49]. The representation of as-desired practice as a set of criteria is therefore argued to be capable of consistent application across domains, using a consistent terminology in the form of the criteria.

- STEP 2: created a model of an Open Standard, and the model uses consistent terminology - *activities* which produce and/or consume *artefacts*, supported by free text boxes where an artefact is not required.

- STEP 3: created a model of a Closed Standard, and the model uses consistent terminology - *activities* which produce and/or consume *artefacts*, supported by free text boxes where an artefact is not required.

- STEP 4: created a model of practice as-observed, and the model uses consistent terminology - *activities* which produce and/or consume *artefacts*, supported by free text boxes where an artefact is not required.

It is therefore argued that Sn3.1 and Sn3.1b combine to support the claim that Goal G3.1 has been met.

6.3.2 Goal G3.2 The Model of Each Element of Software Safety Practice is Created Using Identical Modelling Symbology

This goal is instantiated by asking experts to score and comment on consistency of the modelling symbology (Sn3.2), and by undertaking and analyzing the outputs of Steps 1, 2, 3 and 4 in Chapter 5 (Sn3.1b). The evaluation question EQ13 asks independent experts to annotate their levels of agreement with the statement "The process creates models whose symbology is consistent across all elements of software safety engineering practice" using the same Likert Scale as introduced for Goal G1.1

The goal is claimed in the context of the consistency of the modelling symbology (C3.2).

Sn3.1b

Sn3.1b is the outputs of the process Steps 1 to 4 described in Chapter 5:

- STEP 1: produced a set of criteria, and as such no argument over consistency of modelling symbology can be argued. It is argued that this is NOT a limitation, as Steps 2 to 4 provide sufficient evidence in support of the efficacy of this solution node.

- STEP 2: created a model of an Open Standard, and the model used consistent symbology - a *hexagon* to represent activities; a *triangle* to represent artefacts; a *rectangle* for free text, and *single, curved lines* to link symbols together.

- STEP 3: created a model of a Closed Standard, and the model used consistent symbology - a *hexagon* to represent activities; a *triangle* to represent artefacts; a *rectangle* for free text, and *single, curved lines* to link symbols together.

- STEP 4: created a model of practice as-observed, and the model used consistent symbology - a *hexagon* to represent activities; a *triangle* to represent artefacts; a *rectangle* for free text, and *single, curved lines* to link symbols together.

Sn3.2

Evidence node Sn3.2 combines with Sn3.1 to meet Goal G3.2. Sn3.2 is the output of Evaluation Question EQ13 which asks independent experts to annotate their levels of agreement with the statement "The process creates models whose symbology is consistent across all elements of software safety engineering practice" using the same Likert Scale as introduced for Goal G1.1. Evaluation Question EQ 13 was scored by the respondents as follows:

SH27236
SH27236 'neither agreed nor disagreed' with the notion that 'the process creates models whose symbology is consistent across all elements of software safety practice', providing the qualifying statement as stated for Question EQ 12.

AY8697
AY8697 'fully agreed' with the notion that 'the process creates models whose symbology is consistent across all elements of software safety practice", providing the qualifying statement as stated for Question EQ 12.

HH75783
HH75783 'fully agreed' with the notion that 'the process creates models whose symbology is consistent across all elements of software safety practice."

These positive responses received therefore contribute to meeting Goal G3.2. It is therefore argued that Sn3.2 and Sn3.1b combine to support the claim that Goal G3.2 has been met.

6.3.3 Goal G3.3 Independent People Produced Consistent Model Representations

This goal is instantiated by direct appeal to the partial Case Study carried out by experts (Sn3.3). The term 'consistent' in this case refers to the equivalence of the symbology and terminology produced by using the adapted FRAM schema. It is NOT claiming a 'repeatability' of output from the independent experts used in this evaluation.

Three independent system safety experts were asked to create a model on two separate occasions, and this modelling was carried out independently and in isolation. For each of the two modelling tasks, the modelling activity was preceded by a short tutorial given to the independent experts. The completed models are found at [122]. Each model is now considered in turn.

Model One

The first modelling task required the independent experts to create a model that represents a snapshot of as-required (Open) software safety practice.

All three models used the same correct symbology and terminology, and the 'consistent' claim is fully supported. There are some differences observed across the models however, and these are illustrated in Table 6.1. In Table 6.1, amber font denotes limited agreement, and red font denotes either no agreement or incorrect modelling (as will now be discussed).

Table 6.1: Evaluation Session One Modelling Outputs Comparison

Activity	**Aspect**	**AY8697**	**HH57583**	**SH27236**
Safety Assessment	**Input**	Design Information	-	-
	Time	When info is available	-	-
	Method	(MooN) from 7 other activities	-	-
	Output	Safety Assessment Report	-	-
	Control	-	-	-
	Resource	-	-	-
AFHA	**Input**	Aircraft functions	Aircraft functions	-
	Time	-	-	-
	Method	ARP 4754B	ARP4761	ARP 4761
	Output	Failure conditions, effects and classifications	Failure conditions, effects and classifications; Assumptions	Assumptions; Potential hazards; Failure conditions, effects, and severity; Safety objectives
	Control	-	-	-
	Resource	-	-	-
PASA	**Input**	Aircraft architecture; Failure conditions, effects and classifications; Proposed requirements	Failure conditions, effects and classifications; Proposed requirements	-

Table 6.1: Evaluation Session One Modelling Outputs Comparison

Activity	**Aspect**	**AY8697**	**HH57583**	**SH27236**
	Time	-	-	Before ASA

	Method	ARP 4754B	ARP 4761;
			Methods for CCA;
			Other safety analysis methods
				ARP4761;
				Qualitative/ Quantitative safety analysis methods

	Output	Proposed aircraft safety requirements;
		System failure conditions, effects, classifications and safety requirements;
		Failure Conditions and Safety Objectives
			Assumptions;
			FDAL, safety requirements, CCA and architecture assignments
				Proposed assumptions;
				Proposed safety requirements;
				independence requirements;
				FDAL

	Control	-	-	-

	Resource	-	-	Failure Conditions;
				Safety objectives;
				Proposed architecture

SFHA	**Input**	System Functions; System failure conditions, effects, classifications and safety requirements	Failure conditions and classifications; FDAL safety requirements, CCA and architecture assignments	Crew awareness; Flight phase environmental conditions; Operational considerations
	Time	-	-	Beginning of system development process; change to system functions
	Method	ARP 4754B	ARP4761	ARP 4761

Table 6.1: Evaluation Session One Modelling Outputs Comparison

Activity	Aspect	AY8697	HH57583	SH27236
	Output	Failure conditions, effects and classifications	Assumptions, Safety Objectives, Failure Conditions for functions	Failure conditions; Functional Hazards
	Control	-	-	-
	Resource	-	-	-
PSSA	**Input**	System architecture; Failure conditions, effects and classifications; Proposed Requirements	Failure conditions for functions; Safety objectives	-
	Time	-	-	Early phases of design
	Method	ARP 4754B	ARP 4761; Other safety analysis methods	ARP 4761
	Output	Failure conditions, effects, classifications, safety requirements; Failure conditions and safety objectives; Proposed item safety requirements	System safety requirements including IDAL; FDAL, safety requirements, CCA and architecture assignments	Proposed assumptions; Proposed safety requirements; independence requirements; FDAL; IDAL
	Control	-	-	-
	Resource	-	-	Proposed architecture; Safety objectives; Failure conditions

Table 6.1: Evaluation Session One Modelling Outputs Comparison

Activity	Aspect	AY8697	HH57583	SH27236
SSA	**Input**	Implementation; Failure conditions, effects, classifications, safety requirements; Results of CCA	System safety requirements including IDAL; Safety Objectives; FDAL, safety requirements, CCA and architecture assignments	-
	Time	-	-	-
	Method	ARP 4754B	ARP4761	-
	Output	Results	Exposure times to latent failures; Analysis results	-
	Control	-	-	-
	Resource	-	Implemented System	-
ASA	**Input**	System level integration and verification evidence; System failure conditions, effects, classifications and safety requirements, Results of CCA	Analysis results; Safety objectives; FDAL, safety requirements, CCA and architecture assignments; Failure conditions and classifications	-
	Time	-	-	-
	Method	ARP 4754B	ARP4761	-
	Output	Results	-	-
	Control	-	-	-
	Resource	-	Complete aircraft	-

Table 6.1: Evaluation Session One Modelling Outputs Comparison

Activity	Aspect	AY8697	HH57583	SH27236
Common Cause Considerations	**Input**	Aircraft architecture; System failure conditions, effects, classifications and safety requirements; Failure conditions, effects, classifications, safety requirements (PSSA); Failure conditions and safety objective (from safety objectives)	-	-
	Time	-	-	-
	Method	ARP 4761	-	-
	Output	Proposed requirements; Results	-	-
	Control	-	-	-
	Resource	-	-	-

None of the differences identified in the comparison of the three models created a challenge to the claim that the process creates consistent models. With few exceptions, the differences between the models can be categorized either as errors in modelling, or differences in the levels of detail considered by the respondents in their respective models.

No respondent's model considered aspects which control an activity, nor the resources consumed by an activity (notwithstanding the comments below), and this is consistent with the extract provided to the three respondents.

In terms of errors, it is noted that AY8697's model shows that 'ARP 47564B' is an artefact which supplies the 'Method' for the various safety analyses (see the extract of the model at Figure 6.5). The other two respondents' models show that the artefact is in fact ARP 4761 (which is correct). It is argued that this is a form of typographical error, as the extract given is taken from ARP 4754B, and as such it would not reference itself. It can be seen from Figure 5.2 that ARP 4761 forms part of the ARP 4754 suite of standards.

The other observed error is that SH27236 has linked artefacts to the 'Resource' aspect of safety activities (and not the 'Input' aspect). This error can easily be remedied by recourse to the process steps (or perhaps more training of the an-

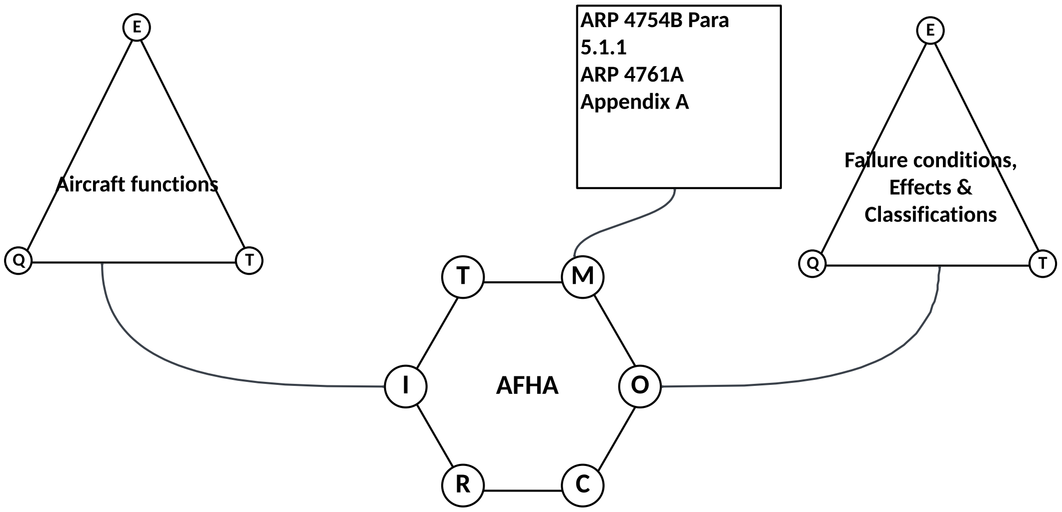


Figure 6.5: Extract from AY8697 Model One

alyst).

Differences in the levels of details between the models also revealed inconsistencies - notably SH27236 considered the activities at a much more detailed level of granularity (perhaps owing to their own experience). AY8697 also created some extra activities (compared to the other two respondents) but this again does not detract from the claim that the process creates consistent models.

No model is entirely the same, and this is perhaps more to do with the vagaries of Open Standards (as we have discussed in Chapter 2) than the framework and process to understand software safety practice. To strengthen the claim that the framework and process to understand software safety practice produces consistent models, each model is now considered in turn.

AY8697

AY8697 produced a model which was centred around the activity “Safety Assessment”. It shows seven safety analysis activities linked to the activity “Safety Assessment” as ‘Controls’ in a MooN relationship. These analysis activities should have been linked to the ‘Method’ activity but this may not be clear enough in the process instructions (see Recommendation 6).

A red-coloured linking line has been used to connect an artefact with an activity, and whilst this is not ontologically correct (in accordance with the process steps), the use of a ‘?’ suggests this is to signify uncertainty in what is required by the ARP.

HH75783

HH75783 created a model with five safety analysis activities which develop in detail and abstraction in a chronological (left to right) manner. The model ‘starts’ with “AFHA” and culminates with the “ASA”. HH75783 noted on the model that there are many feedback loops in reality - but that they were not shown for clarity and brevity.

SH27236
SH27236 also considered each safety analysis activity in turn (although some are in apparent isolation). They applied a great deal of assessment into the consumed and produced artefacts (more than the other two respondents), and also considered a number of other activities implied by the ARP.

Model Two

The second modelling task required the independent experts to undertake a comparison of as-required (Closed) software safety practice with as-required (Open) software safety practice. Both models were provided to the respondents as an 'overlay' so they did not need to undertake any modelling as a precursor to this evaluation. Evaluating the attribute of consistency here is relevant only in the context of attributing the colour-coding required by this process step. Other than instances discussed below, the three respondents used the colour-coding to annotate the levels of agreement in a manner which supports the claim that the process produces 'consistent' models.

All three respondents found a direct comparison between the model of Open and Closed practice challenging for the same reasons we articulated in Chapter 5 (i.e. the ARP's use of different terminology and syntax, and the fact that Company X's processes are a mixture of legacy, new, and updated processes). The three respondents 'handled' this in subtly different ways, as we now discuss by considering each respondent's model in turn.

AY8697
Although AY8697 used the correct colour coding to denote the levels of agreement between the two models of practice, they coloured the entire artefact/activity rather than considering each aspect of an activity/artefact in turn. This may be a misunderstanding of the process required, but it is argued that (when considering the other two respondent's models) this is owing to the difficulty in making direct comparisons between the two models of practice. This once again emphasizes the importance of having a SQEP analyst undertake this process (see Recommendation 1). AY8697 also annotated their model with notes to identify the activities and artefacts which were potentially comparable, or partially comparable.

HH75783
HH75783 used the correct colour coding to denote the levels of agreement between the two models of practice, and annotated the colour coding scheme to both the entire artefacts / activities, and to individual aspects as well. It is argued that the use of this colour-coding scheme is also because of the difficulties in making a direct comparison. For the instances where a confident comparison could be made, individual aspects were appropriately coloured. When confident assertions over a comparison could not be made, the entire activity / artefact has been coloured. HH75783 also added useful notes to highlight where they suspected deficiencies, or inefficiencies in both models of practice (see the extract of the model at Figure 6.6). Such data will prove invaluable for Research Questions 3 and 4.

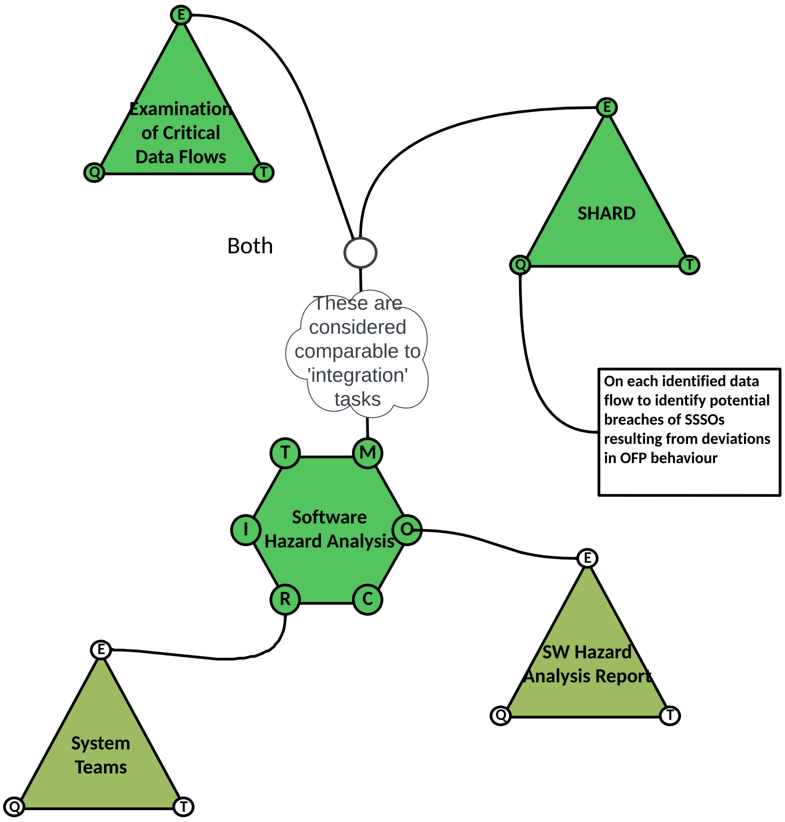

E
Examination of Critical Data Flows
Q
T
E
SHARD
Q
T
Both
These are considered comparable to 'integration' tasks
On each identified data flow to identify potential breaches of SSSOs resulting from deviations in OFP behaviour
T
M
I
Software Hazard Analysis
O
R
C
E
SW Hazard Analysis Report
Q
T
E
System Teams
Q
T

Figure 6.6: Extract from HH75783 Model One

SH27236
SH27236 used the correct colour coding scheme to denote the levels of agreement between the two models, but again struggled with making a direct comparison between the two models. This was confirmed in the 'return email' and, as can be seen from the model, SH27236 only annotated the model where they were confident in a direct comparison. SH27236 used the correct colour coding scheme - applying it only to the aspects of activities/artefacts.

It is therefore claimed that the evidence from the three respondents (Sn3.3) supports Goal G3.3 (independent people produced consistent model representations).

6.4 G4: The Process is Effective at Enabling an Organization to Identify Impediments to Best Practice

Goal G4 is instantiated by the argument structure at Figure 6.7 and is made in the context that 'effective' equates to providing meaningful data relating to potential impediments to achieving best practice (C4). The use of this Goal is justified as, for a project to invest time on employing the process, the process must deliver meaningful data on potential impediments to achieving best practice (J4).

Goal G4 is instantiated by appealing to individual criteria for 'effectiveness'

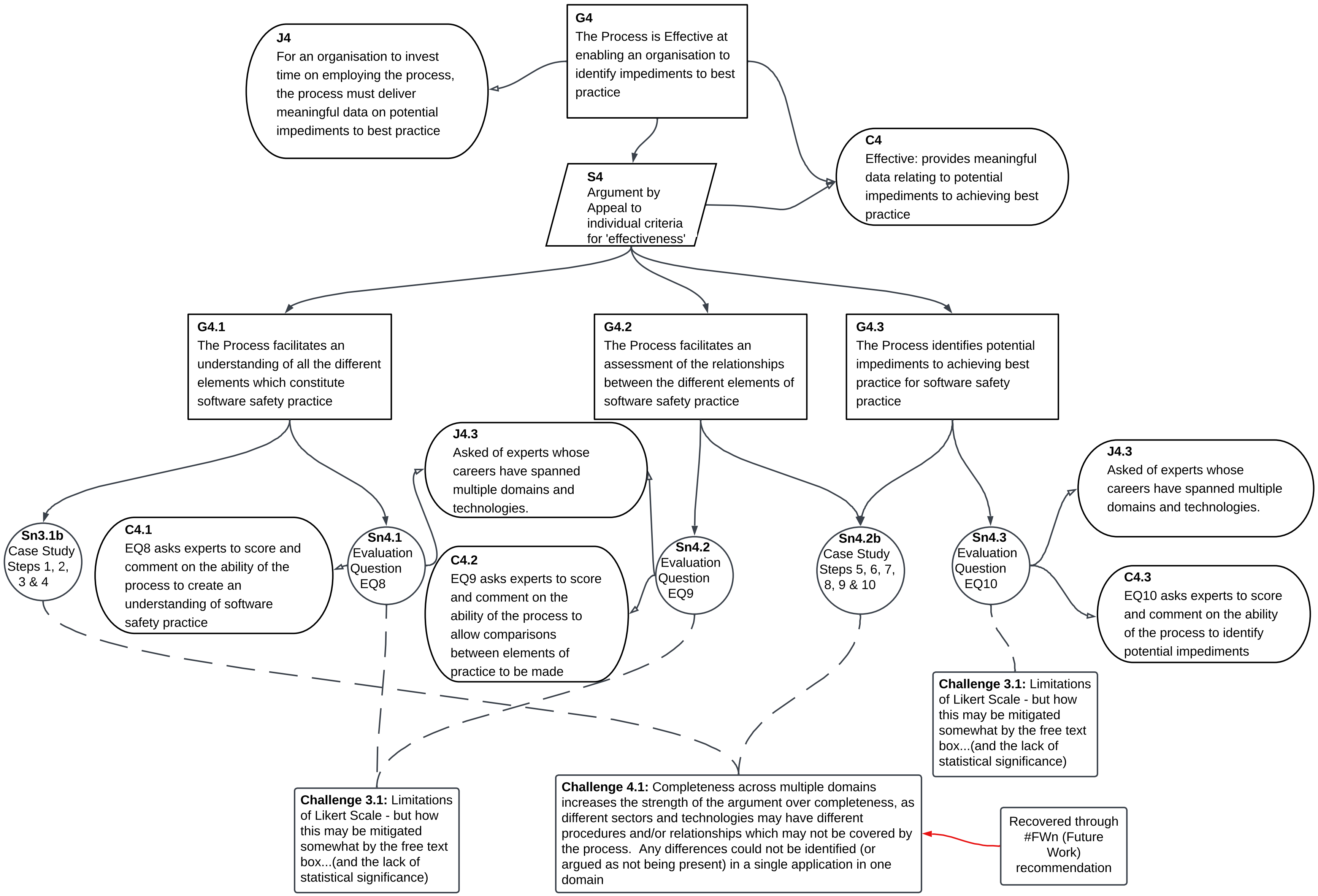

G4
The Process is Effective at enabling an organisation to identify impediments to best practice
J4
For an organisation to invest time on employing the process, the process must deliver meaningful data on potential impediments to best practice
C4
Effective: provides meaningful data relating to potential impediments to achieving best practice
S4
Argument by Appeal to individual criteria for 'effectiveness'
G4.1
The Process facilitates an understanding of all the different elements which constitute software safety practice
G4.2
The Process facilitates an assessment of the relationships between the different elements of software safety practice
G4.3
The Process identifies potential impediments to achieving best practice for software safety practice
J4.3
Asked of experts whose careers have spanned multiple domains and technologies.
Sn3.1b
Case Study Steps 1, 2, 3 & 4
C4.1
EQ8 asks experts to score and comment on the ability of the process to create an understanding of software safety practice
Sn4.1
Evaluation Question EQ8
C4.2
EQ9 asks experts to score and comment on the ability of the process to allow comparisons between elements of practice to be made
Sn4.2
Evaluation Question EQ9
Sn4.2b
Case Study Steps 5, 6, 7, 8, 9 & 10
Sn4.3
Evaluation Question EQ10
J4.3
Asked of experts whose careers have spanned multiple domains and technologies.
C4.3
EQ10 asks experts to score and comment on the ability of the process to identify potential impediments
Challenge 3.1: Limitations of Likert Scale - but how this may be mitigated somewhat by the free text box...(and the lack of statistical significance)
Challenge 3.1: Limitations of Likert Scale - but how this may be mitigated somewhat by the free text box...(and the lack of statistical significance)
Challenge 4.1: Completeness across multiple domains increases the strength of the argument over completeness, as different sectors and technologies may have different procedures and/or relationships which may not be covered by the process. Any differences could not be identified (or argued as not being present) in a single application in one domain
Recovered through #FWn (Future Work) recommendation

Figure 6.7: Goal G4 - The Process is Effective at Enabling an Organisation to Identify Impediments to Best Practice

(S4). The criteria for effectiveness are:

1. The process facilitates an understanding of all the different elements which constitute software safety practice
2. The process facilitates an assessment of the relationships between the different elements of software safety practice
3. The process identifies potential impediments to achieving best practice for software safety practice.

Each individual criterion is argued by the sub-goals G4.1 to G4.3. inclusive. Each goal is now considered in turn.

6.4.1 Goal G4.1 The Process Facilitates an Understanding of all the Different Elements which Constitute Software Safety Practice

This Goal is instantiated by asking experts to score and comment on the ability of the process to create an understanding of software safety practice (Sn4.1 and C4.1), and by the outputs of Steps 1, 2, 3 and 4 in Chapter 5 (Sn3.1b). Evaluation question EQ8 asks independent experts to annotate their levels of agreement with the statement “Using the modelling process allows me to understand all elements of software safety practice” using the same Likert Scale as introduced for Goal G1.1.

Sn4.1

Sn4.1 is one half of the supporting evidence in support of claiming Goal G4.1 is met. Sn 4.1 is the output of Evaluation Question EQ8. Evaluation Question EQ8 was scored by the anonymous respondents as follows:

SH27236
SH27236 ‘somewhat disagreed’ with the notion that ‘using the modelling process allows me to understand all elements of software safety practice’, providing the following qualifying statement:

“My main ‘disagreement’ is with the ‘all’ part of the statement. There are benefits in making explicit an interpretation of written processes and standards. I take ‘practice’ to mean what is actually performed in an instantiation of those processes to realise satisfaction of a standard. I can see no step in the evaluation that deals with completeness.”

AY8697
AY8697 ‘somewhat agreed’ with the notion that ‘using the modelling process allows me to understand all elements of software safety practice’, providing the following qualifying statement:

“The two ‘as required’ models are pitched at different levels. The standard is at relatively high level, but the closed model is much more detailed. For example, recording

outcomes in a database is far too low level for the 'open' model. Hence the implied completeness of 'all elements' is difficult to ascertain."

HH75783
HH75783 'fully agreed' with the notion that 'using the modelling process allows me to understand all elements of software safety engineering practice', providing the following qualifying statement:

> *"The modelling helps me understand how individual processes works. Interpretation is needed where there are mismatches in levels of abstraction."*

The negative and neutral responses received are argued to be a limitation of the way on which the evaluation questions and evaluation instructions were phrased. It should have been made clearer to the respondents that they were asked to "also consider applications and technologies not covered by the artefacts we provided you with (i.e. from experience throughout your career), and don't restrict your response to just the artefacts sent to you."

It is claimed that the qualifying statements and the results from the third respondent enable a limited claim to be made that Sn4.1 supports Goal G4.1

Sn3.1b

Sn3.1b is the other half of the evidence which combines to satisfy Goal G4.1. Sn3.1b represents the outputs of the process steps one to four described in Chapter 5:

- STEP 1: produced a model of as-desired software safety practice in the form of a set of criteria which all (other) elements of software safety practice must meet
- STEP 2: produced a model of as-required (Open) software safety practice.
- STEP 3: produced a model of as-required (Closed) practice
- STEP 4: produced a model of as-observed software safety practice.

Whilst it can be argued that all existing elements of software safety practice can be mapped onto these three elements, it cannot *yet* be argued whether these three elements represent the full range of software safety practice. Further instantiations of the process will reveal the levels of confidence in completeness, (see Recommendation 3).

6.4.2 Goal G4.2 The Process Facilitates an Assessment of the Relationships Between the Different Elements of Software Safety Practice

This goal is instantiated by both asking experts to score and comment on the ability of the process to allow comparisons between elements of practice to be made (Sn4.2 and C4.2), and by undertaking and analyzing the outputs of Steps 5, 6, 7, 8, 9 and 10 in Chapter 5 (Sn4.2b). The evaluation question EQ9 asks independent experts to annotate their levels of agreement with the statement "Using the modelling process allows me to assess all aspects of software safety practice (through comparisons between the elements of practice and their relationships)" using the same Likert Scale as introduced for Goal G1.1.

Sn4.2

Sn4.2 is one half of the supporting evidence in support of claiming Goal G4.2. is met. Sn 4.2 is the output of evaluation question EQ9. Evaluation question EQ9 was scored by the anonymous respondents as follows:

SH27236
SH27236 'somewhat disagreed' with the notion that 'using the modelling process allows me to assess all aspects of software safety practice.' They also made the qualifying statement discussed under Goal G4.1.

AY8697
AY8697 'somewhat agreed' with the notion that 'using the modelling process allows me to assess all aspects of software safety practice.' They also made the qualifying statement discussed under Goal G4.1.

HH75783
HH75783 'somewhat agreed' with the notion that 'using the modelling process allows me to assess all aspects of software safety practice.' They also made the qualifying statement discussed under Goal G4.1.

Following the discussions in Goal G4.1, and in combination with the neutrally positive responses from respondents, it is argued that Sn4.2 meets its contribution to Goal G4.2.

Sn4.2b

Sn4.2b is the other half of the evidence which combines to satisfy Goal G4.2. Sn4.2b represents the outputs of process steps five to ten in Chapter 5:

- STEP 5: produced a comparison (compliance check) between software safety practice as-required (Closed) and software safety practice as-desired.
- STEP 6: produced a comparison (compliance check) between software safety practice as-required (Open) and software safety practice as-desired.
- STEP 7: produced a comparison between software safety practice as-observed and software safety practice as-required (Closed).
- STEP 8: produced a comparison between software safety practice as-required (Open) and software safety practice as-required (Closed).
- STEP 9: assessed whether the model of software safety-practice as-observed revealed activities which were additional and/or different from software safety practice as-required.

- STEP 10: assessed whether the model of software safety practice as-observed revealed activities which would appeal to an open standard which was not influencing / informing software safety practice as-required (Closed).

It is therefore argued that Sn4.2 and Sn4.2b combine to show Goal G4.2 has been met.

6.4.3 Goal G4.3 The Process Identifies Potential Impediments to Achieving Best Practice for Software Safety Practice

This Goal is instantiated by asking experts to score and comment on the ability of the process to identify potential impediments (Sn4.3 and C4.3), and by undertaking and analyzing the outputs of Steps 5, 6, 7, 8, 9 and 10 in Chapter 5 (Sn4.2b). The evaluation question EQ10 asks independent experts to annotate their levels of agreement with the statement “The process to understand software safety practice will help to identify potential impediments to achieving best practice for software safety” using the same Likert Scale as introduced for Goal G1.1.

Sn4.2b

Goal G4.3 is partially constituted by the outputs of process steps five to ten in Chapter 5:

- STEP 5: produced a comparison (compliance check) between software safety practice as-required (Closed) and software safety practice as-desired. The revealed potential deficiencies are discussed in Chapter 5.1.5.

- STEP 6: produced a comparison (compliance check) between software safety practice as-required (Open) and software safety practice as-desired. The revealed potential deficiencies are discussed in Chapter 5.1.6.

- STEP 7: produced a comparison between software safety practice as-observed and software safety practice as-required (Closed). The potential discrepancies are discussed in Chapter 5.1.7.

- STEP 8: produced a comparison between software safety practice as-required (Open) and software safety practice as-required (Closed). The potential discrepancies are discussed in Chapter 5.1.8.

- STEP 9: assessed whether the model of software safety-practice as-observed revealed activities which were additional and/or different from software safety practice as-required. The observations made as a result of this step are contained in Chapter 5.1.9.

- STEP 10: assessed whether the model of software safety practice as-observed revealed activities which would appeal to an open standard which was not influencing / informing software safety practice as-required (Closed). The observation made as a result of this step is contained in Chapter 5.1.10.

Sn4.3

Sn4.3 is the other half of the evidence which combines to satisfy Goal G4.3. Sn4.3 is the output of evaluation question EQ10. Evaluation Question EQ10 was scored by the respondents as follows:

AY8697
AY8697 'fully agreed' with the notion that 'the process to understand software safety practice will help to identify potential impediments to achieving best practice for software safety'. They also made the qualifying statement discussed under Goal 5.

HH75783
HH75783 'somewhat agreed' with the notion that 'the process to understand software safety practice will help to identify potential impediments to achieving best practice for software safety'.

SH27236
HH75783 'somewhat agreed' with the notion that 'the process to understand software safety practice will help to identify potential impediments to achieving best practice for software safety'.

It is therefore argued that Sn4.2b and Sn4.3 combine to show Goal G4.3 has been met.

6.5 Goal G5 The Process is Applicable for Use in any Industry

Goal G5 is instantiated by the argument structure in Figure 6.8. Goal G5 is instantiated by direct appeal to two evidence nodes - Sn5 and Sn5b.

Sn5

Sn5 is one half of the supporting evidence in support of claiming Goal G5 is met. Sn5 is the output of Evaluation Question EQ11 which asks independent experts to annotate their levels of agreement with the statement "The process to understand software safety engineering practice can be used for any industry and any technological application" using the same Likert Scale as introduced for Goal G1.1. Evaluation Question EQ11 was scored by the respondents as follows:

AY8697
AY8697 'neither agreed nor disagreed' with the notion that 'the process to understand software safety engineering practice can be used for any industry and any technological application', providing the following qualifying statement:

"Theoretically, I believe it could be applied to all industries and applications, however, the nature of some industries, where process is low priority, would make it very difficult to apply. I think its fair to say that it should be able to be used effectively for all

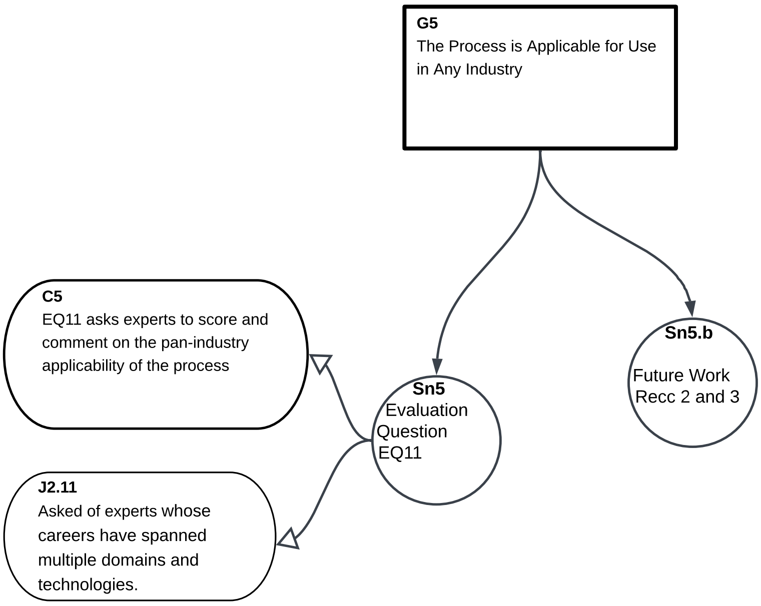

G5
The Process is Applicable for Use in Any Industry
C5
EQ11 asks experts to score and comment on the pan-industry applicability of the process
J2.11
Asked of experts whose careers have spanned multiple domains and technologies.
Sn5
Evaluation Question EQ11
Sn5.b
Future Work Recc 2 and 3

Figure 6.8: Goal G1 - The Process is Applicable for Use in Any Industry

safety involved industries!"

HH75783
HH75783 'somewhat agreed' with the notion that 'the process to understand software safety engineering practice can be used for any industry and any technological application', providing the following qualifying statement:

"As noted in detailed comments, I'm not sure if there's an assumption about the sw developer originally being in the SC domain or using an open standard for their development."

SH27236
SH27236 'somewhat agreed' with the notion that 'the process to understand software safety engineering practice can be used for any industry and any technological application'.

These responses are neutrally positive, and the respondents have also provided some interesting comments. Taking AY8697's observation that the nature of practice in some industries may prove problematic for this framework and process, the final part of their comment suggests it *should* be applied within those industries. This suggests the issues lie within the industry and not with the framework and process itself. Any organization must of course wish to understand its software safety practice in the first place, and no 'external process' can be forced upon a non-receptive audience.

HH75783's comment on the possibility that there is an assumption that the 'sw developer' (taken to mean the project rather than an individual) is 'originally in the SC domain' is interesting, but the framework and process is not exclusively designed for industries operating in any specific industry. The framework and process has been designed to contribute to the improvement of software safety practice in any industry where software may contribute to system hazards (either directly in a causal chain, or by preventing the mitigation of a hazard). This is discussed in Section 1.2, but was not made explicitly clear to the respondents. This doesn't answer the second part of the comment relating to whether a project uses an Open Standard or not, however. It is hard to envisage a project where software contributes to a hazard, and where some form of Open Standard would not be used during the design and build phase (as Open Standards are not limited to specific safety concerns). Should such a project exist, then they would simply not be able to follow the entire process (i.e. Steps 2, 6, 8, and 10 would be 'null'). It is argued that this does not detract from the ability to use the framework and process in any industry (notwithstanding the outcomes from 'future work' which are considered in Chapter 7).

Sn5b

Sn5 is the other half of the supporting evidence for claiming Goal G5 is met. The future work evidence will not be available within the timescale of this PhD, naturally.

It is argued that despite the inevitable lag in gathering the data from future instantiations, there is confidence in the combination of evidence nodes Sn5 and Sn5b to support a claim that Goal G5 is met. Other than completing multiple studies simultaneously with the instantiation of the framework and process (see Chapter 5) there is no other (current) means of arguing the claim of applicability further.

6.6 Summary

We have argued that the argument outlined at Figure 6.1 is a compelling defence that the Research Objective has been met, because, despite the noted potential limitations:

1. The process is complete. Supported by:
   (a) Evaluation by independent experts
   (b) Argument over the 10-step process,
   (c) An illustrative example of the process being instantiated, and
   (d) Publication of Conference and Journal papers.
2. The process is easy to use. Supported by:
   (a) Evaluation by independent experts, and
   (b) An independent Case Study.
3. The process represents all elements of software safety practice in a consistent manner. Supported by:
   (a) Evaluation by independent experts,

   (b) An illustrative example of the process being instantiated, and
   (c) An independent Case Study.

4. The process is effective. Supported by:
   (a) Evaluation by independent experts
   (b) An illustrative example of the process being instantiated, and
   (c) An independent Case Study.

5. The process is applicable for use in any industry. Supported by:
   (a) Evaluation by independent experts
   (b) Appeal to Future Work recommendations.

6.7 Empirical Research Evaluation

"An inherent problem with a scientific approach to safety is that it appears to be difficult to devise unambiguous experiments to determine whether safety programmes, interventions, concepts, or optimisations actually work (and there is an insurmountable set of practical and ethical problems involved in the study of interventions)" [157].

As a science, there are steps we can take to understand the current state of safety practice, however. This chapter has argued over the 'goodness' of our novel framework and process to understand existing software safety practice. Having evaluated the framework and process for 'goodness', attention now turns to an evaluation of the empirical research methodology. We now consider in turn the use of graphical representation, and the modelling process itself.

6.7.1 Graphical Representation

FRAM was selected as the most appropriate modelling tool because it was the best fit for both the functional and non-functional requirements (see Chapter 4). The motivation to use an open-source software package had an unforeseen consequence, however. Two publicly-available software applications were used to build the models of software safety practice during the period of research - Microsoft's Visio, and then Lucid's Lucidchart.

As the models of software safety practice grew in size, a text-based word search was often used to locate relevant activities and artefacts for the purpose of comparison. The text-based search functions of both software programmes were found to be lacking, however. Words or phrases which were visible in a current screen view would not be found by the two applications' 'search' function, which eroded confidence in the thoroughness of the search returns.

Once identified, this issue was mitigated by a manual search which was time-consuming. This is a potential impediment to the utility and widespread adoption of the framework and process in its current guise, but it is argued to be mitigated by appeal to future work - see Recommendation 2.

6.7.2 Modelling Process

The data gathered during the empirical research revealed a plethora of artefacts which - although listed in the reference section of the provided processes and plans - where not supplied by the participating project. The decision to include these as 'Referenced Artefacts' (using an inverted colour scheme) was discussed in Chapter 4.5, but the positioning of these artefacts were based only on the knowledge and experience of the author.

Whilst we retain confidence in the benefits and relationships of the 'Referenced Artefacts', there is a risk that the models of as-required (Closed) practice may not be complete and correct. This is mitigated by appeal to the fact that the participating organization (see Chapter 5) was supplied the models of their practice, and confirmation that the models were accurate representations was received. An additional step to ask the participating organization to supply referenced documents could serve to mitigate the risk of having incomplete models, however - see Recommendation 7.

In Chapter 3.1 it was noted that, when transforming elements of software safety practice into a graphical representation, it could be challenged that all such representations are simply a form of 'Work as Imagined' [58]. Whilst this is a potential weakness of the framework and process, if we are to be able to understand and assess software safety practice then we need to transform each element of practice into comparable models that describe each element as accurately as possible, and this is a necessary compromise.

In the feedback after Session 1 of the independent evaluation, SH27236 noted that they could *"see benefits from the modelling"*, but that they could *"see challenges in understanding the objective of modelling, the fidelity required – and the degree of completeness needed to achieve a 'safety' focused outcome."* This is a valid point, and an aphorism often attributed to George Box suggests that "all models are wrong, some are useful". Any model of practice will only be useful if the objective of the modelling is clear, and the fidelity is appropriate for the desired outcome.

SH27236 also observed that they had *"worked with some large generic process models designed to be tailored to the specific context (and most safety standards and inhouse management systems are created in this mould). There is a real skill in capturing these in a way that conveys clearly the intent whilst also enabling the flexibility – I'm not sure that any of the cases I've seen have got it correct – but some are useful...I think the success of the approach will come down to the practical relevance of the findings that can be extracted from the 'understanding' that the approach enables."* Although perhaps not made explicitly clear to the respondents, the process requires the analyst to consider the required attributes (aspects) of each activity and artefact. The colour coding scheme also notes that 'green' can signify an appropriate level of detail is considered, OR that no consideration is required for that aspect. The decision made by the analyst in this regard is reliant on their skills and experience however (see Recommendation 1).

Two or more people may create different models of software safety practice from the same textual artefact (indeed this was apparent in the models created by the independent experts who participated in the evaluation. As noted by SH27236, some Open Standards are goal-based (and sometimes contain vague language (see Appendix A) for example). This leads to different interpretations of what practice should constitute. That two or more people could produce different interpretations of the same process artefact is a critique of the efficacy of the Open Standard, however - and not of the framework and process for understanding software safety practice.

The limitations of Open Standards are discussed further in Chapter 6.8.2 when considering data gleaned from the early stages of the empirical research.

6.8 Analyzing the Empirical Data

This section now evaluates the findings from each step of the process, and considers whether the models produced by each step were sufficient to facilitate analysis.

6.8.1 Software Safety Practice As Desired

Step 1 of the process required a model of software safety practice to be created which was both tangible and measurable. The relating representation had to be either a graphical representation of as-desired practice, or a set of measurable criteria which could then be used to assess the other elements of software safety practice for compliance.

The produced output was a set of criteria which was predicated on the 4+1 Principles for software safety assurance [53], [49]. Whilst these principles are well-established in both academia and in practice, the unique contribution of this thesis is the creation of a set of tangible, and measurable criteria against which other elements of software safety practice can be measured.

6.8.2 Software Safety Practice As Required (Open)

The initial design decision was to create a model of 'best practice' predicated and built upon three Open Standards. Although this initial direction revealed some interesting data, it no longer forms part of the framework and process to understand software safety practice. It nevertheless provides a valuable contribution to knowledge regarding the efficacy and usefulness of current Open Standards, and an insight into how standards committees' intent is articulated in textual form.

Recognized good practice for software safety assurance is currently expressed in the form of functional safety lifecycles depicted by Open Standards such as BS EN 61508 [13] and ARP 4754A [147], and the initial objective for the first part of the empirical research was to create a 'metamodel' or 'supermodel' of available standards. To meet this objective, a design decision was taken to model a representation of 'best practice' using the denoted activities and methods therein described.

The design decision taken to meet this objective was to select three Open Standards ([13], [147], and [14]), and taking each standard in turn model the lifecycle both as depicted pictorially (typically a flow chart or 'V'model), and as conveyed by the accompanying and supporting text. The decision to undertake the modelling in two distinct steps was predicated on an observation that the main text of the standards do not necessarily match the simplistic overview portrayed by the visual representation – and often contradicts and / or confuses it (which may in itself suggest an impediment).

Each standard was to be compared to the wider state of academic literature, enabling the identification of any potential shortfalls, vagaries, or disagreements between the standards and the literature – with each subsequent standard improving on the shortfalls, and mitigating the vagaries of the last.

On completion, further recourse to academic literature and personal experience would be made to eliminate any residual shortfalls, and clarify any remaining vagaries. This 'supermodel' of best practice would then be offered for review and feedback as a lifecycle representation that may be considered a referenceable benchmark in support of the empirical research.

As the data produced by Step 2 highlights, the processes and practices required of Open Standards cannot defensibly be asserted to constitute best practice for software safety, and an alternative strategy had to be adopted as the model of practice was already unwieldy (for which there are many different reasons). For example, even though ARP 4754A relies heavily on sister publications such as ARP 4761 for safety activities, and child publications such as DO178C and DO254 for the assurance of software and complex electronic hardware respectively, its model of required practice still resulted in a large monolithic model which had to be divided into multiple layers of abstraction. Even with the division of the model, it was still unwieldy, and undertaking focused searches was cumbersome (see Recommendation 2).

Despite the monolithic nature of the ARP 4754A model, the process required to create it revealed interesting data. The full list of findings is found at Annex A, and some findings relate to the following potential issues which otherwise may not have been revealed had the framework and process not been followed:

- The standard majors on the establishment of an overarching intent, but is lacking in any detailed guidance or mandated / recommended activities
- Throughout the standard, the stipulation of what constitutes 'recommended practice' is avoided – offering what can only be described as 'helpful advice', but without making any stipulations on whether / how it would be beneficial to follow it
- In many cases, a list of recommendations is offered with a caveat that they may not, in fact, constitute best practice
- Although the standard offers a brief insight in the constituent parts of each methodology, it stops short of discussing which methods / techniques are more suited / appropriate.

Even though the created model is somewhat sparse in detail (i.e. the attributes of an activity are not considered fully), and not always internally consistent, this may be owing to an incorrect interpretation by the author. However, because the standard is written in a manner which is open to interpretation a subjective interpretation which manifests as a graphical representation cannot, in fact, be 'wrong'. "All models are wrong, some are useful."

Step 2 of the process (Chapter 5) produced the model of software safety practice required by ARP4754A - for which the accompanying model is at [111], and the accompanying report is at Appendix A. As the report highlights, there are issues with the standard, and there exists the potential for readers to arrive at different interpretations. However, everything within the ARP was able to be modelled using the notation described in Chapter 4. The model revealed potential impediments to achieving software safety best practice, but these may have been left reasonably to as-observed practice.

6.8.3 Software Safety Practice As Required (Closed)

The model of software safety practice as-required (Closed) was predicated on the eight artefacts supplied by JB61834. The model of this element of practice was shared with the organization to confirm its accuracy. JB61834 confirmed the model was an accurate representation of their practice.

JB61834 also informed us that they intended to use the model to inform an upcoming Training Needs Analysis. This is an additional benefit of the framework and process which was not anticipated at the onset of this research.

6.8.4 Software Safety Practice As Observed

As noted in Chapter 5, an ethnographic study was not carried out, and the model of this element of practice is more accurately described as 'Software Safety Practice As Disclosed' (see Recommendation 5).

The transcripts from the two respondents have been redacted, as some aspects of the recorded conversations may have revealed the organization and / or the respondent. The redaction does not detriment the quality of the data as suitable non-specific terminology has replaced the redacted aspects.

The process to create the model of as-disclosed practice (as a proxy for as-observed practice) revealed interesting data, including some interesting potential issues, but initial caution and care must be taken with the data until such a time that further, follow-up investigations have taken place with the organization (RQ3 and RQ4). Caution is required as the data gathered is only as-disclosed by the respondents, and we cannot yet know whether the data is subjective, opinion-based, biased, or motivated by other non-altruistic means.

Biases may exist in the data gathered from the respondents. The presence of such biases would represent an interesting finding in its own right, and would merit further follow-up investigations regardless. The full findings are discussed in Chapter 5, with the transcripts contained at [127], and some data findings include the following potential issues which otherwise may not have been revealed had our novel framework and process not been followed:

- Some activities inferred by the respondents are not found in the process artefacts supplied by the organization
- Decisions (and the rationale behind them) which impacted how requirements were derived and apportioned were not always recorded formally
- Discrepancies between as-required and as-observed practice appear to exist
- The management of safety requirements is not always robust, and is sometimes performed informally

- Quantifiable targets are apportioned by mathematically simple means
- If an existing item is incapable of meeting safety targets, the targets are adjusted and reapportioned elsewhere on a purely mathematical basis
- There exists a disconnect between customer requirements and supplier capability
- There exists a knowledge management risk - (one of corporate knowledge retention) as some important knowledge appears to be held by a single individual
- The documentary artefacts provided were either too sparse or too full
- It is asserted by respondents that successful delivery of safe products is achieved by people, and not process
- Engineers are not confident in the appropriateness of, nor the guidance within the organization's associated as-required (Open) standards
- Inter-team tensions exist between the system engineering and software engineering teams.

6.8.5 Comparison between Software Safety Practice As Required (Closed) and Software Safety Practice As Desired

The annotated model which denotes the comparison between software safety practice as-required (Closed) and software safety practice as-desired was carried out using Step 5 of the process (see Chapter 5), and revealed further interesting data. The creation of this model was made possible by the creation of the as-desired criteria - against which any other element of software safety practice can be assessed for compliance.

Only the criteria for the first principle of as-desired software safety practice was assessed for this step, and this was for three reasons. The first reason is the limitation of time in a single PhD programme - as the modelling and comparison is time- and resource-intensive. Linked to the rationale over time, the second reason is efficiency. Whilst carrying out compliance checks for all principles would reveal more data, the proof of concept is not improved by the creation of more models and their data. The third and final reason is one of efficacy. As the model of as-required (closed) practice had already revealed potential deficiencies (which may be recovered by practice as-observed), it was already discovered that software safety practice as-required (Closed) would not be capable of meeting the as-desired criteria.

The outputs and associated findings from this step are found in Chapter 5.1.5, with the full assessment contained in Annex C.5. Some findings include the following potential issues which may otherwise not have been revealed had our novel framework and process not been followed:

- With a single exception, the first principle can be *inferred* to be met by as-required (Closed) practice, but the process artefacts do not state explicitly *how* the required information will be generated

- Process artefacts present options for which activities could be carried out, but stop short of defining a selection criteria. None of the required attributes of any outputs are stated either

- There exists a disconnect between system- and software-level activities

- Potential deficiencies in compliance with the Principle 1 criteria have been identified

- The required quality attributes of software safety requirements cannot be argued to have been met by following the as-required (Closed) activities alone

- The process steps for the activities contain a mixture of tenses (past, present, and future) as they consider processes, design decisions, and intentions. This is likely due to the length of time the project has been running for

- Activities are described in terms of what must be done, but not always *how*

- Descriptions of different testing methods are provided, along with perceived advantages and disadvantages of each. What isn't provided is a basis of selection

- Continued (albeit limited) compliance with Principle 1 is currently reliant on the retention of a small number of key individuals who hold pertinent knowledge implicitly.

6.8.6 Comparison between Software Safety Practice As Required (Open) and Software Safety Practice As Desired

The annotated model which denotes the comparison between software safety practice as-required (Open) and software safety practice as-desired was carried out using Step 6 of the process (see Chapter 5), and revealed further interesting data. The creation of this model was made possible by the creation of the as-desired criteria - against which any other element of software safety practice may be assessed for compliance.

Only the criteria for the first Principle was assessed for this step for the same three reasons argued over in the consideration of the comparison between as-required (Closed) practice, and practice as-desired.

The outputs and associated findings from this step are found in Chapter 5.1.6, with the full assessment contained in Annex B.7. Findings include the following potential issues which may otherwise not have been revealed had our novel framework and process not been followed:

- There is no definitive, identifiable artefact created by the as-required (Open) practice which would clearly identify the software contained within a system

- Any one of six artefacts required to be produced by the as-required (Open) practice *may* contain the context of the system in which the software will reside - but there are potential issues identified with each artefact

- Although two artefacts are produced which will contain a clear description of the system in which the software will reside, these artefacts are not produced until late in the development lifecycle.

- The as-required (Open) lifecycle will not produce a single 'point of truth' in which the system hazards are identified, contained, and managed

- It must be assumed that sister publications are used to identify the system hazards at the software boundary

- There is no clear link between system hazards and specific failure modes

- No claim can be made as to the specification of software safety requirements elicited in mitigation of system hazards

- No process exists for the careful management of software safety requirements.

It must be noted however, that any potential impediment *may* be reasonably be left to practice as-required (Closed) practice or as-observed.

6.8.7 Comparison between Software Safety Practice As Observed and Software Safety Practice As Required (Open)

The annotated model which denotes the comparison between software safety practice as-observed and software safety practice as-required (Open) was carried out using Step 7 of the process (see Chapter 5), and revealed further interesting data.

Only the criteria for the first Principle was assessed for this step for the same three reasons argued over in the consideration of the comparison between as-required (Closed) practice, and practice as-desired.

The outputs and associated findings from this step are found in Chapter 5.1.7. Findings include the following potential issues which may otherwise not have been revealed had our novel framework and process not been followed:

- The data revealed potential issues with both elements of practice, but these cannot be confirmed without recourse to follow-up investigations with the project

- The data required for assessing compliance with Principle 1 was so sparse that further, extensive searches where undertaken. This is not a weakness of the process, however. This extensive search could have reasonably been left to future follow-up work, but was instead undertaken with the aim of proving the efficacy and utility of Step 7

- There is a mismatch between the two elements of practice with regard to the management of Safety Cases

- There is a potential mismatch between the two models of practice in terms of the abstraction-level at which activities and artefacts are discussed. This may be owing to colloquial terms used by the respondents, however.

- Assertions as to who the hazard owner is differs between the two models of practice

- Step 5 of the process revealed potential issues which may have been reasonably left to the as-observed element of software safety practice to recover. None of the issues were recovered however, and so must now be considered as issues which require further follow-up work

- Once activities and artefacts required for compliance with as-desired practice had been separated from the monolithic models, a number of activities and artefacts remained 'unused' from a safety perspective. Even after aspects such as certification are considered, activities and artefacts remain 'unused', and these are perhaps superfluous, or even examples of safety clutter [136]. This is found for all comparison steps, and the ability of the framework and process to reveal such issues is indicative of the usefulness of it.

6.8.8 Comparison Between Software Safety Practice As Required (Open) and Software Safety Practice As Required (Closed)

The annotated model which denotes the comparison between software safety practice as-required (Open) and software safety practice as-required (Closed) was carried out using Step 8 of the process (see Chapter 5), and revealed further interesting data.

The outputs and associated findings from this step are found in Chapter 5.1.8. Findings include the following potential issues which may otherwise not have been revealed had our novel framework and process not been followed:

- JB61834 do not have a Platform-level element of their practice in their process artefacts, but make assumptions about the completeness and correctness of the safety analyses undertaken at the Platform-level, and the resulting safety requirements derived by the System-level processes

- It is not possible to overlay both models of practice as a direct comparison. There are two reasons for this. The first reason is the terminology and syntax used are divergent. The second reason is that JB61834's lifeycle artefacts are constituted by a mixture of both legacy and updated processes. This is not a weakness of the process, however, as the activities and artefacts in both elements of practice are unlikely to be identically-named, and the levels of abstraction will often differ. Whilst the extra modelling activities required to facilitate this comparison is labour-intensive, it is a necessary part of the process for understanding and assessing software safety practice

- After Step 2 a number of potential issues with the ARP's ability to meet the as-desired criteria where identified. These potential issues could be left reasonably to organizational practice. These potential issues are not recovered by the as-required (Closed) practice, however

- Instances of no-agreement between the two elements of practice were identified

- Whilst the JB61834 lifecycle has the Hazard Log as a singe point of truth for safety data, the ARP makes no recourse to a Hazard Log, nor similar artefact

- Neither element of practice considers the resources required to perform safety activities (nor their required quality attributes). This cannot be left to as-observed practice to determine

- Neither element of practice considers the time or phase by which the system safety analyses should be completed by. This cannot be left to as-observed practice to determine

- It is also acknowledged that any potential issues may be incurred by the inability to precisely overlay the models of both elements of practice. This is mitigated by appeal to follow-up work to confirm their existence.

It is acknowledged that the issues and potential issues identified by this step *could* be due to the full range of as-required (Closed) process artefacts not being delivered. This is mitigated by both the fact that the model of as-required (Closed) practice was reviewed by the organization for completeness and correctness; and that the existence of any further process artefacts should be revealed as part of any follow-up work. It is also acknowledged that any potential issues may be incurred by the inability to precisely overlay the models of both elements of practice. This is mitigated by appeal to follow-up work with the organization to confirm their existence (neither of which is in scope for this PhD).

After the first evaluation session, HH75783 queried whether all organizations would use Open Standards to influence or inform the development of a project's development - citing an occasion in the nuclear industry where this was absent. Their comment was in reference to a 'smart device' of low integrity requirements ('below SIL 1') which was procured from a 'small company'. The assertion was that such sub-contracted organisations may not have a significant suite of practices (as-required (Closed)). We have discussed whether some projects may not have such practice in Chapter 6.5, and the issue of safety assuring individual components is not precluded by this framework and process (in fact it supports it fully). HH75783 in fact concluded by noting that *"I think what you propose works fine in this situation."*

AY8697 found this comparison step *"much more difficult than the initial modelling task"*, noting that:

> *"trying to articulate an equivalence between a number of tasks in the closed standard to parts of one or more activity in the open model was tricky. That is as much a challenge of the two processes rather than a challenge of the modelling, but you could almost do with rearranging the models to group together comparable activities visually, perhaps directly under each other, during the comparison. I wonder if that might make it easier? Wish I'd tried that now!"*

It is acknowledged that this evaluation step was more challenging - and we discuss this very issue in Chapter 5. It is reassuring that the respondent acknowledges this is more owing to the nuances of the models rather than the framework and process itself.

HH75783 noted that:

"It was hard to separate criticising (or praising!) individual processes from assessing differences between them. I'm not very familiar with the ARP which may have slowed me down. I've done a lot of comparisons of internal sw processes to standards. There are usually strengths and weaknesses and sometimes those don't matter in context. For example, WCET analysis not done but the system isn't time critical. Or lacking separate requirements document from a manual/design diagrams when the sw has existed for a long time. The latter can be okay for generating tests in some circumstances. In step 8 task 4 – reasons for why they are different, does the process thinking about whether it matters that they are different? (I think this is different than comparing the as observed but you may well have covered it)."

This comment both emphasizes the potential issues with Open Standards which we have discussed, and also notes the importance of understanding the 'discipline' in the process steps. The steps have been designed to only require an analyst to (in this case) note (rather than posit the reasons for) any differences observed - leaving the discovery of the rationale to RQ3 and RQ4. It is argued that this is commentary in support of the framework and process rather than a critique necessitating a form of corrective action.

SH27236 made similar comments to HH75783, noting:

"the evaluation doesn't seem to handle well the differing purposes, abstraction levels (hierarchy and conceptual) or non-like-for-like models. Standards may be goal based or prescriptive, and may be a step on the path between legislation/regulation (which are inherently very abstract and non-specific) to tasking at the coal face. Standards tend to set out to avoid constraint on the detail of implementation – deliberately so to permit innovation, and to allow application to a broad spectrum of real world problems. I think it is inevitable that there will be disconnects between the four nodes of the diamond, because they are doing different things. The modelling seems to have some benefits in understanding self-consistency and coherence of any one of the nodes, but the relationship between them is more complex that a traffic light rating can communicate".

The purpose of Step 8 (within the constraints of Research Questions 1 and 2) is to identify levels of agreement in order that further investigations can occur as part of Research Questions 3 and 4; and in the context of the outputs of all process steps. This doesn't detract from the observations made by SH27236, who highlights nuances with various Open Standards (as we have discussed in Chapters 2 and 5).

The concern raised over comparing non 'like-for like' models is valid. For example, a goal-based Open Standard could not be readily overlaid with a model of organizational practice which had been established as a set of activities in a safety lifecycle. This can be remedied by inserting optional steps in the process to define and use a set of criteria for goal-based (Open) standards - see Recommendation 8.

6.8.9 Comparison of Software Safety Practice As Observed with Software Safety Practice As Desired

Although no ethnographic study was carried out, responses from the as-disclosed interviews did reveal two instances which *may* be examples of as-observed practice deviating from as-required practice in an attempt to comply with as-desired practice. It is possible that without carrying out this process to understand and assess software safety practice, that these potential instances may not have been revealed.

The first potential instance concerned an assertion from the respondents that existing processes for the management of requirements is not robust, and would attempt to formalise the process if they were 'starting over'. The second potential instance concerned the local changes made to the software safety elicitation process as a result of mid-life upgrades - which was not reflected in an update in the as-required process artefacts.

Further follow up work is required to confirm these two potential instances, and it is acknowledged that further explicit instances may have been revealed, had a full ethnographic study been carried out. See Recommendation 5.

6.8.10 Comparison of Software Safety Practice As Observed with Software Safety Practice As Required (Open)

Whilst application of the process to understand software safety practice have revealed difficulties and issues with complying with Open Standards (including suppliers making appeal to different Open Standards), no instances of as-observed working to a different Open Standard were identified.

Instances *may* have been revealed if a full ethnographic study had been carried out (see Recommendation 6), but it is argued that the lack of identified instances is not an indication on the utility of the process, merely that nothing was found on this occasion.

The data produced by applying the framework and process to understand and assess software safety practice meets the thesis aims and the research objective. Although some threats to the validity of data have been identified (and discussed next), these threats have been successfully mitigated by appeal to both claims made in this chapter and to recommendations for future work (see Chapter 7).

6.8.11 Validity of Data

The illustrative example in Chapter 5 has demonstrated the efficacy and usefulness of the framework and process to understand and assess software safety practice, and has revealed useful data for the organization under assessment. The usefulness of this data relates to identified and potential impediments to the achievement of good practice for the organization. Further research is required by the organization to determine whether and how one or more elements of their software safety practice needs to change.

Notwithstanding the successful application of the framework and process to understand and assess software safety practice, it has only been applied to one organization and project in one sector. The claim of successful application and claims over ease of use, completeness, effectiveness, and pan-industry applicability would have been stronger if an independent researcher had applied the framework and process concurrently to another project in a different sector. This was neither a realistic nor viable option for a single PhD programme, but it is argued that this threat to validity is mitigated by the independent evaluation, and by appeal to future work at Recommendation 3.

Threats to the validity of data gathered during the illustrative example in Chapter 5 may also exist and these are now discussed - along with a discussion on limitations of the process.

The strategy for the empirical research involved the creation (adaptation) of a graphical representation that facilitated an understanding and subsequent assessment of software safety practice. Whilst not predicated on a formal ontology with robust claims over its modelling capabilities, the chosen graphical representation was selected not only to facilitate the modelling of the aspects of software safety practice we had identified, but also to promote widespread use and understanding.

There may be more appropriate modelling tools or ontologies that exist or are in the process of being created, but we remain unaware of their existence. Whilst we argue that modified FRAM best suited the needs of the empirical research, we do not mandate its use however, and encourage others to use graphical representations and modelling tool(s) that best suits their needs.

The modelling aspects of the empirical research was undertaken using a non-judgemental perspective with the aim of representing process as written and / or described. One can never argue reasonably over the subjective interpretation of textual or oral descriptions, however.

To mitigate this threat to validity, we could have employed an independent researcher to carry out the exact modelling and analysis in tandem, before comparing and contrasting the results. Such an approach would more than double the effort of the research activities (which would not be commensurate with the timeframe of a single PhD programme), and would perhaps only serve to compare the subjective personas of the independent researchers. As such we argue this threat to validity is mitigated by the independent evaluation, and the publication of our results in peer-reviewed journals.

Empirical data relating to work as-observed (and its relationship with work as-required and as-desired) was gathered through interviews with representatives of the organization which submitted their software safety process artefacts for analysis. The representatives were a software engineer with system safety responsibilities, and a safety engineer with the role of assessing software's contribution to system safety.

Ideally, the evaluation of work as-observed would have been borne out of a comprehensive ethnographic study that independently and objectively observed software safety practice over an extended period of time (and with a team of impartial assessors). Whilst we recommend this is undertaken (Recommendation 5), it was not possible within the remit of a single PhD research programme.

Extracting data from interviews poses risks to the validity of data (and hence any inferences made from it), as it is entirely possible that external factors may have influenced the responses from the respondents. Such factors include those that may be self-imposed on the interviewee (a reluctance to self-incriminate, potentially sensitive responses, or any biases held by the respondent).

External influencing factors may also have influenced the responses from the respondents. These could range from restrictions associated with commercially sensitive data, Intellectual Property Rights concerns, though to formal constraints placed on the interviews by the organization (which we are unaware of).

In mitigation, we took care to avoid any leading questions, and questions that could have hinted at the identification of the organization (whose identity we have taken all reasonable steps to protect throughout this research).

We could also have embarked on a formal Pilot Study of the interviews ahead of the research into work as-observed, but this would have required either another demand for time from the projects (in the form of extra interviews) who had supplied their process for scrutiny (which was not palatable); or a fabricated interview with a colleague. It is argued that this would have yielded little benefit - as the questions where not 'closed' (and no claim at arguing statistical significance is made anyway). A Pilot Study does allow one to test (and adjust) a series of questions, in order that any biases can be eliminated, but as our interviews emanated from a single initiating request, it was assessed that there was no benefits to be gained from a Pilot Study.

Notwithstanding these argued mitigations and rationale, it would be invaluable to compare our analysis of this empirical data with that realised by a full ethnographic study. Such a study would also be hugely beneficial in accurately identifying any issues with software safety practice at large (see Recommendation 5).

6.9 Coda

The modelling process is labour-intensive. The modelling phases of the research took almost 2-years (part time) to complete. However, all elements of software safety practice must be modelled, understood, and assessed if we are to truly improve software safety practice. Failure to do so will likely manifest in further years of effort wasted 'fiddling with the edges' of practice, with little or no tangible improvements in software safety practice. Further work is needed to make the modelling process as efficient as possible, however.

The graphical notation selection process pointed to FRAM as the best fit for both the needs of the process, and the non-functional requirements we established to promote maximal use across industry sectors and technologies. The trade-off between maximal use and the detriments manifest in a publicly-available tool for creating models which has probably been skewed in favour of widespread use over a reliable search function. Whilst the framework and process could be abstracted up to general safety engineering, its focus in this thesis is firmly on software safety practice.

Comparing a proposal with existing frameworks and processes is a common way to evaluate the efficiency of a proposal. Unfortunately, we are not aware of the existence of a framework and associated process to understand and assess any form of safety (or indeed any engineering) practice. The proposed framework and instantiation of the framework and process was focussed on software safety practice of the project. Independent evaluation involved very experienced software safety assurance practitioners and researchers. It is accepted that a full ethnographic study has not been completed, and there are good reasons why this would be an appropriate next step. However, there are many reasons why an organisation would not wish to participate in an ethnographic study. We discovered this when trying to convince even one organisation to participate. Perhaps these reasons are some or all of the following:

- (Perceived) cost
- Concerns about Intellectual Property Rights
- Security concerns
- Disenchantment with safety science and safety research
- Not wishing to 'air one's dirty linen in public'
- Perceived and real gaps (and attitude issues) between safety science and safety practice
- Ethnography is not always appropriate. This may be true for any cognitive processes; and
- Observing any form of safety activity moderates behaviour.

We encourage organizations to instantiate this framework and process, as:

- For a field of research to move forward, each new project or paper must strive to change what has come before – adding, synthesising, testing, tearing down or making anew [135]
- There is systemic pressure placed on researchers to ground their work in untested models, reductionist categories, and proxy measurements, rather than on direct observation and sophisticated analysis of real people doing real work in real organisations [135]. This reduces the uptake of such research by industry and perpetuates a "credibility gap"
- "The work of...designers and software developers is complex" [135]. Applying the proposed process may allow an organisation to draw broader conclusions about how it supports design workers in creating safe designs. That is the remit of safety science [135]
- "Describing" encompasses much more than raw data collection. Safety science researchers have addressed the use of descriptive research. In [134], [135], [136] they indicate that descriptive research encompasses:

  Making direct and indirect observations of the thing being studied. This needs a wide range of data collection methods;

  Analyzing and modelling the thing being studied; and

  Assessing and evaluating the thing being studied.

Safety science descriptive research [134], [135], [136] indicates that:

- Most empirical data problems in safety science (i.e. generation of, and access to) arise from an inability by researchers to directly observe the phenomena that they are interested in measuring
- If data generated by researchers is merely a re-representation of work-as-imagined, we can never really see the problems that we are trying to describe
- A researcher observing work should look at the relationships between practices, how these practices are produced and re-produced, what their underlying assumptions and meanings are, and what this might imply in terms of workplace tensions and power relations. None of this is possible, however, without observing what is going on
- One model of safety work in organizations proposed that safety activities fulfil broader political and social needs, in addition to the reduction of safety risk. If this is true then the nature of these needs and whether they are being met must be uncovered and understood by organisations
- Repeated observations of front-line activities enable safety professionals to identify operational changes and probe the potential for normalization of deviance.

In the development and evaluation of the proposal in this thesis we have employed these safety science descriptive research findings This chapter has contributed by undertaking an evaluation of the application of the proposed framework and process to understand and assess software safety practice. The combination of evidence provided, and recommendations for future work combine to argue the ’goodness’ of the framework and process. It is argued that the framework and process is complete, straight forward to use, and represents all elements of software safety practice consistently. It is effective, and applicable for use in any industry.

Chapter 7

Conclusions and Recommendations for Future Work

7.1 Conclusions

The thesis now concludes with a commentary on what the work has achieved, and by making recommendations for future work. These recommendations are not weaknesses of the thesis, but are argued to be a natural step in advancing the research which started with this PhD programme. They may also be considered as steps to recovering gaps in the current state of research into how software safety practice can be improved. The recommendations represent a call to action for all safety science and safety engineering researchers - including the author.

The thesis aim stated in Chapter 1 was to provide a framework through which an organization can gain an understanding of the elements which constitute their software safety practice, and a process by which that organization can therefore understand and assess its software safety practice. The aim has been met by fulfilling the research objective, which has in turn, been met through answering the two research questions. The research objective and research questions were derived after identifying the gaps in the literature highlighted in Chapter 2. The research objective has been met by answering the following two research questions:

- **RQ1** How can an organization understand its software safety practice?
- **RQ2** How can an organization assess its software safety practice?

7.1.1 The Framework and Process

An organization can understand and assess its software safety practice by using our framework to establish the elements which constitute their software safety practice. The framework establishes the relationships between the elements of the organization's software safety practice. Our novel process provides the step-by-step instructions to guide an organization through the activities required to understand and assess their software safety practice. Following our novel process will enable an organization to identify potential, and appropriately-distal impediments to the achievement of best practice for software safety.

The successful implementation of the process is dependent on an organization utilizing a skilled analyst who possesses the necessary skills and experience. The competence and competencies required of an organization's analyst to successfully implement the process has not been considered by this thesis, however. See Recommendation 1.

Completion of our novel process will elicit data which is presented in a format which supports the organization in undertaking further work. This further work is necessary to complete the process in its entirety, and is carried out in three stages. The first stage of work is to carry out investigations to confirm the presence of any potential impediments which have been elicited during the process to understand and assess software safety practice. The second stage of work required is to perform mitigation research to identify the mechanisms by which any confirmed impediments to best practice for software safety can be eliminated. Such mechanisms must first be assessed for any unintended consequences in other elements of software safety practice. The third stage of work is to repeat relevant steps of the process to understand and assess software safety practice. This is important, as an organization must confirm the impediments have been removed, and must update the models of practice so they can be maintained effectively. The process to undertake these final stages are provided in this thesis, but have not been evaluated as part of this empirical research.

Whilst these latter three stages have not been implemented in the illustrative example in this thesis, we can take confidence in their efficacy and utility from the independent evaluation. Notwithstanding this confidence held, it is important to implement fully the final three stages of the process. This work will meet Research Objective 2 (provide a process by which a project can identify potential impediments to achieving best practice for software safety practice, in a manner that gives confidence that any potential impediments are appropriately-distal, and enables effective remedies to be derived), and will be fulfilled by answering the remaining two research questions:

- **RQ3** How can an organization identify true impediments (i.e. appropriately-distal) to achieving best practice for its software safety practice?
- **RQ4** How can an organization derive effective mitigations for the identified impediments to software safety engineering best practice?

Answering Research Questions 3 and 4 will be fulfilled by completing Future Work Activity 1.

7.1.2 Process Enablers

Having argued that a graphical notation would be a key enabler of the process to understand and assess software safety practice, we carried out research into the available graphical and modelling notations and tools which could be used (with or without adaptation) for the framework and process. The selected notation was an adapted version of FRAM, and this was selected as it provided the best fit for both the needs of the process, and the non-functional requirements we established to promote maximal use across industry.

The use of our adapted version of FRAM is not without its challenges, however. In Chapter 6.7.1, it is noted that a sub-optimal text-based search function in the software programmes resulted in reduced confidence in the totality of the models created as a result of the comparison and compliance checks. See Recommendation 2.

7.1.3 Illustrative Example

An illustrative example has been completed which implements fully the framework and process to understand and assess software safety practice. The data produced by the process has revealed areas of disagreement between elements of software safety practice; levels of non-compliance with as-desired criteria; practice carried out which is not in accordance with the latest documentary artefacts; and potential deviations from the organizational lifecycle of activities. This data is argued to support the claims as to the usefulness and efficacy of the process to understand software safety practice.

7.1.4 Thesis Evaluation

We have evaluated the framework and process, and have used Goal Structuring Notation to argue by inference how the disparate evidence sources contribute to successfully arguing that the framework and process to understand and assess software safety practice is:

- Complete
- Easy to use
- Represents all elements of software safety practice in a consistent manner
- Effective, and
- Applicable for use in any industry.

The evaluation of the thesis has revealed some areas of potential weakness, but it is argued that these can be mitigated by appeal to further recommendations for future work. These further recommendations for future work are considered under themes, and their consideration completes the thesis.

7.1.5 Pan-industry Applicability

In Section 6.1.2, Challenge 1.13 notes that 'completeness' across multiple domains increases the strength of the argument over completeness, as different sectors and technologies may have different procedures and/or relationships which may not be covered by the process. They also note that any differences could not be identified (or argued as not being present) in a single application in one domain. See Recommendation 3.

7.1.6 Independent Evaluation

In Section 6.2.6 it was noted that the independent evaluation may be limited by not asking the respondents to carry out a process step which required levels of compliance to be assessed (i.e. Steps 5 and 6 of the process). See Recommendation 4.

7.1.7 Software Safety Practice As-Observed

In Chapter 6, and throughout the thesis it is noted that as-observed practice has not been assessed explicitly, as an ethnographic study has not been carried out. See Recommendation 5.

7.1.8 Process Instructions

In Chapter 6, it is noted that the process instructions may not have been as clear, nor as explicit as possible (leading to small deviations in the intended outputs from the modelling steps). See Recommendation 6.

In Chapter 6.7.2 it is noted that a risk to the completeness of the completed models exists if 'referenced' artefacts are not received and assessed for their utility. See Recommendation 7.

7.1.9 Comparing Software Safety Practice As-Required

In Chapter 6.8.8 it was noted that the framework and process would not cope well when comparing organizational practice with an Open Standard which was goal-based (as the two models could not be simply overlayed and compared). See

Recommendation 8.

The thesis now concludes by listing the recommendations and asserting the future work activity for the author.

7.2 Recommendations

7.2.1 Recommendation 1

In Section 3.2 it was stated that the argument of what constitutes a SQEP individual is outside of the scope of this research. The need to establish the skills and experience of an analyst implementing this process is reinforced by the feedback of independent reviewers, however (see Chapter 6). Recommendation 1 is therefore to determine the competence and competencies [129] required of an analyst to successfully implement our framework and process to understand and assess a project's software safety practice.

Competence and competencies should be determined to ensure the analyst possesses *at least* the following:

- Expert[1] knowledge of software safety practice (Chapters 3, and 6.7.2)
- Expert appreciation on what attributes (aspects) are required of software safety activities and supporting / produced artefacts (Chapter 6.7.2)
- Expert knowledge of the different names for software safety activities used throughout and pan-industry (Chapter 6.8.8)

7.2.2 Recommendation 2

Recommendation 2 is to assess whether any proprietary software modelling packages, or formal modelling tools would render a more effective search function in support of the framework and process to understand software safety practice. Any assessment must also take into account the functional and non-functional criteria specified in Chapter 4 to assess whether each criterion remains valid.

7.2.3 Recommendation 3

Recommendation 3 is to apply the framework and process to understand and assess software safety practice to more domains (other than the military air do-

[1] For definitions of expert (Level 3), practitioner (Level 2), and supervised practitioner (Level 1) see [1].

main). This is to increase confidence in the completeness of the framework and process.

7.2.4 Recommendation 4

Recommendation 4 is to apply the framework and process in full to more projects. This will increase the confidence in the ease of use of the framework and process to understand and assess software safety practice.

7.2.5 Recommendation 5

Recommendation 5 is to carry out a full ethnographic study of software safety practice as-observed, and use this model of practice to facilitate all associated comparison and conformance steps (Steps 7, 9, and 10).

7.2.6 Recommendation 6

Recommendation 6 is to review the process instructions for instantiating the framework and process to understand software safety practice. Changes to the process instructions should be tested in a Pilot Study before embarking on Recommendation 3 and Recommendation 4.

7.2.7 Recommendation 7

Recommendation 7 is to add a step in the process to understand software safety practice which suggests the analyst asks the project under assessment to provide any further pertinent artefacts.

7.2.8 Recommendation 8

Recommendation 8 is to add an optional process instruction to use when a goal-based as-required (Closed) element of practice is used. Should a goal-based standard be used, then optional steps should be inserted into the following (current) process steps:

- **Step 2:** insert instructions as to how as-required practice can be established as a set of criteria predicated on the goals denoted in the Open Standard
- **Step 6:** insert instructions as to how the criteria established in the optional instructions for Step 2 can be compared with the as-desired criteria. This

should use the same schema as per Steps 7 and 8 in the extant process instructions

- **Step 8:** insert instructions as to how organizational practice can be compared to a goal-based Open Standard. These optional process instructions should 'mirror' Step 1 (i.e. defining a set of criteria against which a project should demonstrate conformance against). This changes Step 8 from one of comparison to one of conformance.

Step 1 (defining as-desired practice is unaffected), as it is already using criteria derived from a set of principles - and these principles are already 'goals'.

Step 3 (and resulting comparisons and conformance checks associated with it) cannot be changed, as it is not appropriate to have goal-based instructions at an organizational level. Should an organization have goal-based processes established, then a process of converting the goals into activities capable of instantiation by the workforce must be created. These newly-created activities are then established as software safety practice as-required (Closed), and the remainder of steps associated with it remain unaffected.

7.3 Future Work

7.3.1 Future Work Activity 1

Undertake the work to implement Research Questions 3 and 4. Answering these research questions will fulfill the second research objective and confirm that the process is capable of mitigating any identified impediments.

7.3.2 Further Uses of the Framework and Process

The framework and process to understand and assess software safety practice could be abstracted up a level to understand and assess safety practice in general (see [128]).

It may also be useful to safety regulators and safety auditors. Regulators could use the models as a means of assessing compliance with regulations / legislation / standards. In a similar vein, safety auditors could use the framework to do the same as regulators - both assessing the efficacy of practice and judging the safety of a product in real-time (and as a management tool for the life of a project).

The models created by the framework and process could also be tailored to suit the needs of different stakeholders, creating:

- Compliance models for regulators
- Comparison model of both types of as-required practice for projects (i.e. Open and Closed as-required practice)
- All types of models for safety managers and Independent Safety Auditors / Advisers
- All as-observed models for assessment and evaluation of software safety practice

The models could also be used to facilitate investigations and planning into SQEP profiling, and competence and competency management.

Bibliography

[1] *IET Code of Practice: Competence for Safety-Related Practitioners*. The IET, 2017.

[2] Robert Allen, editor. *The New Penguin English Dctionary*. Penguin, 2000.

[3] Scott W. Ambler. *Process Patterns, Building Large-Scale Systems Using Object Technology*. Cambridge University Press, Cambridge, 1998.

[4] Pablo Oliveira Antonio, Mario Trapp, Paulo Barbosa, Edmar.C Gurjao, and Jeferson Rosario. The Safety Requirements Decomposition Pattern. *Proceedings of SAFECOMP 2015*, pages 269 –282, 2015. doi:10.1007/978-3-319-24255-2_20.

[5] Fredrik Asplund, Greg Holland, and Saleh Odeh. Conflict as software levels diversify: Tactical elimination or strategic transformation of practice? *Safety Science*, 126:104682, 2020. doi:10.1016/j.ssci.2020.104682.

[6] Terje Aven. A risk science perspective on the discussion concerning safety i, safety ii and safety iii. *Reliability Engineering & System Safety*, 217:108077, 2022. doi:10.1016/j.ress.2021.108077.

[7] Kristian Beckers, Isabelle Côté, Thomas Frese, Denis Hatebur, and Maritta Heisel. Systematic derivation of functional safety requirements for automotive systems. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8666 LNCS(256980):65–80, 2014. doi:10.1007/978-3-319-10506-2_5.

[8] Geoffrey Biggs, Takeshi Sakamoto, and Tetsuo Kotoku. A profile and tool for modelling safety information with design information in SysML. *Software and Systems Modeling*, 15(1):147–178, 2016. URL: http://dx.doi.org/10.1007/s10270-014-0400-x, doi:10.1007/s10270-014-0400-x.

[9] Mishap Investigation Board. Mars climate orbiter mishap investigation board phase i report november 10, 1999, 1999.

[10] Barry Boehm and Prasanta Bose. A Collaborative Spiral Software Process Model Based on Theory W. *3rd International Conference on the Software Process, ICSP 1994*, pages 59–68, 1994.

[11] Barry W. Boehm. A spiral model of software development and enhancement. *Computer*, 21(5):61–72, 1988. doi:10.1109/2.59.

[12] Cari Borrás. Overexposure of radiation therapy patients in panama: problem recognition and follow-up measures. *Revista Panamericana de Salud Pública*, 20(2-3):173–187, 2006.

[13] BSi. Functional safety of electrical / electronic / programmable electronic safety related systems Parts 1-7. Standard BS EN 61508, 2010.

[14] BSi. Systems and Software Engineering - Vocabulary. standard BS ISO/IEC IEEE 24765:2010, 2010.

[15] BSi. DRAFT INTERNATIONAL STANDARD: Health Software - Software Life Cycle Processes. standard IEC/CD 62304.3, 2019.

[16] Ana Carolina Scanavachi Moreira Campos and Adiel Teiveira de Almeida. Multicriteria framework for selecting a process modelling language. *Enterprise Information Systems*, 10(1):17–32, 2016. doi:http://dx.doi.org/10.1080/17517575.2014.906047.

[17] Carnegie-Mellon-SEI. +SAFE, V1.2 A Safety Extension to CMMI-DEV, V1.2. Technical Report 1.2, 2007. doi:CMU/SEI-2007-TN-006.

[18] Ryan A. Carter, Annie I. Antón, Aldo Dagnino, and Laurie Williams. Evolving beyond requirements creep: A risk-based evolutionary prototyping model. *Proceedings of he IEEE International Conference on Requirements Engineering*, pages 94–101, 2001. doi:10.1109/ISRE.2001.948548.

[19] J. Chudge and D. Fulton. Trust and co-operation in system development: applying responsibility modelling to the problem of changing requirements. *Software Engineering Journal*, 11(3):193, 1996. doi:10.1049/sej.1996.0025.

[20] M Dominic Cooper. Towards a model of safety culture. *Safety science*, 36(2):111–136, 2000.

[21] Thomas H Davenport. Working knowledge: How organizations manage what they know. *NewYork Harvard Business School*, 1998.

[22] JoseLuis de la Vara and RajwinderKaur Panesar-Walawege. SafetyMet: A Metamodel for Safety Standards. 8107:69–86, 2013. doi:10.1007/978-3-642-41533-3_5.

[23] Sidney Dekker. Malicious compliance. *Hindsight 25*, 2017. Last accessed 08 Dec 2023. URL: https://www.eurocontrol.int/sites/default/files/publication/files/hindsight25.pdf.

[24] Ewen Denney, Ganesh Pai, and Iain Whiteside. Model-Driven Development of Safety Architectures. *Proceedings - ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems, MODELS 2017*, pages 156–166, 2017. doi:10.1109/MODELS.2017.27.

[25] p. Douglass, Bruce. Introduction to model-based engineering, 2021. URL: https://www.incose.org/docs/default-source/michigan/what-does-a-good-model-smell-like.pdf?sfvrsn=5c9564c7_4.

[26] Omar El Ariss, Dianxiang Xu, and W. Eric Wong. Integrating safety analysis with functional modeling. *IEEE Transactions on Systems, Man, and Cybernetics Part A:Systems and Humans*, 41(4):610–624, 2011. doi:10.1109/TSMCA.2010.2093889.

[27] Neil A. Ernst, Alexander Borgida, John Mylopoulos, and Ivan J. Jureta. Agile requirements evolution via paraconsistent reasoning. In *International Conference on Advanced Information Systems Engineering*, pages 382–397. Springer, 2012. doi:10.1007/978-3-642-31095-9_25.

[28] Weam M. Farid and Frank J. Mitropoulos. NORMATIC: A visual tool for modeling non-functional requirements in agile processes. *Conference Proceedings - IEEE SOUTHEASTCON*, (978), 2012. doi:10.1109/SECon.2012.6196989.

[29] Jane Fenn and Brian Jepson. Putting Trust into Safety Arguments. *Constituents of Modern System Safety Thinking - Proceedings of the 13th Safety-Critical Systems Symposium*, pages 21 –35, 2005. doi:10.1007/1-84628-130-X_2.

[30] Floyd J. Junior Fowler. *Survey Research Methods - Applied Social Research Methods*. Sage, California, 3rd edition, 2002.

[31] Dominic Furniss, Paul Curzon, and Ann Blandford. Using FRAM beyond safety: a case study to explore how sociotechnical systems can flourish or stall. *Theoretical Issues in Ergonomics Science*, 17(5-6):507–532, 2016. doi:10.1080/1463922X.2016.1155238.

[32] Barbara Gallina, Edin Sefer, and Atle Refsdal. Towards safety risk assessment of socio-technical systems via failure logic analysis. *Proceedings - IEEE 25th International Symposium on Software Reliability Engineering Workshops, ISSREW 2014*, pages 287–292, 2014. doi:10.1109/ISSREW.2014.49.

[33] Fakhredin Ghasemi, Esmaeil Zarei, and Vahid Salehi. Unraveling complexity: Fram applications in sociotechnical systems safety analysis. In *Safety Causation Analysis in Sociotechnical Systems: Advanced Models and Techniques*, pages 213–236. Springer, 2024.

[34] Dimitra Giannakopoulou, Thomas Pressburger, Anastasia Mavridou, Julian Rhein, Johann Schumann, and Nija Shi. Formal requirements elicitation with FRET. *CEUR Workshop Proceedings*, 2584, 2020.

[35] A Ian Glendon and Neville A Stanton. Perspectives on safety culture. *Safety science*, 34(1-3):193–214, 2000.

[36] Patrick Graydon, Ibrahim Habli, Richard Hawkins, Tim Kelly, and John Knight. Arguing conformance. *IEEE Software*, 29(3):50–57, 2012. doi:10.1109/MS.2012.26.

[37] Herbert P Grice. Logic and conversation. In *Speech acts*, pages 41–58. Brill, 1975. doi:10.1163/9789004368811_003.

[38] The Assurance Case Working Group. Goal Structuring Notation Community Standard. Standard, 2018.

[39] Atul Guwande. The checklist manifesto. *New York: Picadur*, 2010.

[40] Torgeir Kolstø Haavik. Debates and politics in safety science. *Reliability Engineering & System Safety*, 210:107547, 2021. doi:10.1016/j.ress.2021.107547.

[41] Ibrahim Habli. *Model-based assurance of safety-critical product lines*. PhD thesis, University of York, 2009.

[42] Ibrahim Habli. Safety standards: Chronic challenges and emerging principles. *Handbook of Safety Principles*, pages 732–746, 2017. doi:10.1002/9781119443070.ch31.

[43] Ibrahim Habli, Richard Hawkins, and Tim Kelly. Software safety: Relating software assurance and software integrity. *International Journal of Critical Computer-Based Systems*, 1(4):364–383, 2010. doi:10.1504/IJCCBS.2010.036605.

[44] Ibrahim Habli and Tim Kelly. A Model-Driven Approach to Assuring Process Reliability. *19th International Symposium on Software Reliability Engineering*, pages 7–16, 2008. doi:10.1109/ISSRE.2008.19.

[45] Kirsten M Hansen, Anders P Ravn, and Victoria Stavridou. From safety analysis to software requirements. *IEEE Transactions on Software Engineering*, 24(7):573–584, 1998.

[46] Martie G. Haselton, Daniel Nettle, and Paul W. Andrews. The Evolution of Cognitive Bias. *The Handbook of Evolutionary Psychology*, pages 724–746, 2015. doi:10.1002/9780470939376.ch25.

[47] John Hatcliff, Alan Wassyng, Tim Kelly, Cyrille Comar, and Paul Jones. Certifiably safe software-dependent systems: challenges and directions. *Future of Software Engineering Proceedings*, pages 182–200, 2014.

[48] Jop Havinga, Sidney Dekker, and Andrew Rae. Everyday work investigations for safety. *Theoretical issues in ergonomics science*, 19(2):213–228, 2018. doi:10.1080/1463922X.2017.1356394.

[49] Richard Hawkins, Ibrahim Habli, and Tim Kelly. The Principles of Software Safety Assurance. *International System Safety Conference*, 2013.

[50] Richard Hawkins, Ibrahim Habli, Tim Kelly, and John McDermid. Assurance cases and prescriptive software safety certification: A comparative study. *Saf. Science*, 59:55–71, 2013. doi:10.1016/j.ssci.2013.04.007.

[51] Richard Hawkins and Tim Kelly. A structured approach to selecting and justifying software safety evidence. 2010.

[52] Richard Hawkins and Tim Kelly. A framework for determining the sufficiency of software safety assurance. *System Safety, incorporating the Cyber Security Conference 2012, 7th IET International Conference on*, pages 1–6, 2012. doi:10.1049/cp.2012.1529.

[53] Richard Hawkins and Tim Kelly. A Software Safety Argument Pattern Catalogue. page 32, 2013. URL: http://www.cs.york.ac.uk/ftpdir/reports/2013/YCS/482/YCS-2013-482.pdf.

[54] Richard D Hawkins. *Using Safety Contracts in the Development of Safety Critical Object-Oriented Systems*. PhD thesis, 2006.

[55] Petra Heck and Andy Zaidman. A framework for quality assessment of just-in-time requirements: the case of open source feature requests. *Requirements Engineering*, 22(4):453–473, 2017. arXiv:1408.1293, doi:10.1007/s00766-016-0247-5.

[56] Mats PE Heimdahl. Safety and software intensive systems: Challenges old and new. In *Future of Software Engineering (FOSE'07)*, pages 137–152. IEEE, 2007.

[57] Erik Hollnagel. *FRAM: The Functional Resonance Analysis Method*. Ashgate, Dorchester, 2012.

[58] Erik Hollnagel. Why is work-as-imagined different from work-as-done? In *Resilient health care, Volume 2*, pages 279–294. CRC Press, 2017. doi:10.1201/9781315605739.

[59] Erik Hollnagel. *Safety–I and safety–II: the past and future of safety management*. CRC press, 2018. doi:10.1201/9781315607511.

[60] Erik Hollnagel. The FRAM Manual, 2018.

[61] Erik Hollnagel, David D Woods, and Nancy Leveson. *Resilience engineering: Concepts and precepts*. Ashgate Publishing, Ltd., 2006.

[62] Andrew Hopkins. Studying organisational cultures and their effects on safety. *Safety science*, 44(10):875–889, 2006.

[63] IEC. Signalling and security apparatus for railways / Reliabilty, availability, maintainability and safety. standard IEC 62278:2002, 3.41, 2020.

[64] ISO. Electrical and magnetic devices / Operating conditions and testing. standard SO/CEI Guide 2 (14.1, 2001.

[65] ISO. Environmental standardization for electrical and electronic products and systems / General terms relating to environmental protection and management. standard ISO 14040:2006, 2006.

[66] ISO/IEC. Systems and software engineering — System life cycle processes. standard ISO/IEC 15288:2008, 2008.

[67] ISO/IEC. Internet of Things (IoT). Standard ISO/IEC 20924:2018, 2015.

[68] Yan Jia, John Alexander McDermid, Nathan Hughes, Mark-Alexander Sujan, Tom Lawton, and Ibrahim Habli. The need for the human-centred explanation for ml-based clinical decision support systems. In *2023 IEEE 11th International Conference on Healthcare Informatics (ICHI)*. IEEE, 2023. doi:10.1109/ICHI57859.2023.00064.

[69] Zador Daniel Kelemen, Rob Kusters, Jos Trienekens, and Katalin Balla. Selecting a Process Modeling Language for Process Based Unification of Multiple Standards and Models. pages 1–14, 2013.

[70] Tim Kelly. *Arguing Safety – A Systematic Approach to Managing Safety Cases*. PhD thesis, University of York, 1998. doi:10.1007/s00779-007-0163-2.

[71] Tim Kelly. Software Certification : Where is Confidence Won and Lost? In *Addressing Systems Safety Challenges*, pages 1–13, 2014.

[72] Gene Kim, Patrick Debois, John Willis, and Jez Humble. *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution Press, 2016.

[73] Karen Klockner and Yvonne Toft. Railway accidents and incidents: Complex socio-technical system accident modelling comes of age. *Safety Science*, 110(November 2017):59–66, 2018. doi:10.1016/j.ssci.2017.11.022.

[74] Mathieu Lavallée and Pierre N Robillard. Why good developers write bad code: An observational case study of the impacts of organizational factors on software quality. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 677–687. IEEE, 2015.

[75] Jean-Christophe Le Coze. *Safety science research: evolution, challenges and new directions*. CRC Press, 2019.

[76] Chris Leong, Tim Kelly, and Rob Alexander. Incorporating epistemic uncertainty into the safety assurance of socio-technical systems. *Electronic Proceedings in Theoretical Computer Science, EPTCS*, 259:56–71, 2017. doi:10.4204/EPTCS.259.7.

[77] Nancy Leveson. A new accident model for engineering safer systems. *Safety Science*, 42(4):237–270, 2004. doi:10.1016/S0925-7535(03)00047-X.

[78] Nancy Leveson, Mirna Daouk, Nicolas Dulac, and Karen Marais. A systems theoretic approach to safety engineering. *Dept. of Aeronautics and Astronautics, Massachusetts Inst. of Technology, Cambridge*, pages 16–17, 2003.

[79] Nancy G Leveson. Safeware: System safety and computers. *ACM New York*, 1995.

[80] Nancy G Leveson. A systems-theoretic approach to safety in software-intensive systems. *IEEE Transactions on Dependable and Secure computing*, 1(1):66–86, 2004.

[81] Nancy G Leveson. *Engineering a safer world: Systems thinking applied to safety*. MIT Press, 2011.

[82] Nancy G Leveson and John P Thomas. Stpa handbook. *Cambridge, MA, USA*, 2018.

[83] Nancy G Leveson and Kathryn Anne Weiss. Software system safety. In *Safety Design for Space Systems*, pages 791–823. Elsevier, 2009.

[84] N.G. Leveson and C.S. Turner. An investigation of the therac-25 accidents. *Computer*, 26(7):18–41, 1993. doi:10.1109/MC.1993.274940.

[85] Jacques-Louis Lions et al. Flight 501 failure. *Report by the Inquiry Board*, 190, 1996.

[86] Raanan Lipshitz, Gary Klein, Judith Orasanu, and Eduardo Salas. Taking stock of naturalistic decision making. *Journal of behavioral decision making*, 14(5):331–352, 2001.

[87] Martin H. Lloyd and Paul J. Reeve. IEC 61508 and IEC 61511 assessments - some lessons learned. *4th IET International Conference on Systems Safety 2009. Incorporating the SaRS Annual Conference*, pages 2A1–2A1, 2009. doi:10.1049/cp.2009.1540.

[88] Russell Lock, Tim Storer, and Ian Sommerville. Responsibility modelling for risk analysis. 2009.

[89] Robyn R. Lutz. Analyzing software safety requirements errors in safety critical systems. *Proceedings of the IEEE International Symposium on Requirements Engineering*, pages 126–133, 1993. doi:10.1109/ISRE.1993.324825.

[90] Rebecca Marschan-Piekkari and Catherine Welch, editors. *Handbook of Qualitative Research Methods for International Business*. Edward Elgar, Cheltenham, 2004.

[91] Tomás Martınez-Ruiz, Félix Garcıa, Mario Piattini, and Jürgen Münch. Modelling software process variability: an empirical study. *IET software*, 5(2):172–187, 2011. doi:10.1049/iet-sen.2010.0020.

[92] Pierre Mauborgne, Samuel Deniaud, Eric Levrat, Eric Bonjour, Jean Pierre Micaëlli, and Dominique Loise. Operational and System Hazard Analysis in a Safe Systems Requirement Engineering Process - Application to automotive industry. *Safety Science*, 87:256–268, 2016. doi:10.1016/j.ssci.2016.04.011.

[93] Alois Mayr, Reinhold Plösch, and Matthias Saft. Towards an operational safety standard for software: Modelling IEC 61508 part 3. *Proceedings - 18th IEEE International Conference and Workshops on Engineering of Computer-Based Systems, ECBS 2011*, pages 97–104, 2011. doi:10.1109/ECBS.2011.8.

[94] John A McDermid. Software safety: where's the evidence? In *SCS*, volume 1, pages 1–6, 2001.

[95] John A. McDermid and David J. Pumfrey. Software Safety: Why is there no Consensus? *Proceeds of the International System Safety Conference*, 2001.

[96] Catherine Menon. Technical Report on Defining Software Safety Requirements. Technical Report 1, Software Systems Engineering Initiative Management, York, 2009. doi:SSEI-TR-000046.

[97] Catherine Menon and Tim Kelly. Eliciting software safety requirements in complex systems. *Systems Conference, 2010 4th Annual IEEE*, pages 616–621, 2010. doi:10.1109/SYSTEMS.2010.5482343.

[98] Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artificial intelligence*, 267:1–38, 2019. doi:10.1016/j.artint.2018.07.007.

[99] Ramin Mojdehbakhsh, Satish Subramanian, Ramakrishna Vishnuvajjala, W-T Tsai, and Lynn Elliott. A process for software requirements safety analysis. In *Proceedings of 1994 IEEE International Symposium on Software Reliability Engineering*, pages 45–54. IEEE, 1994.

[100] Jessica Morley, Lisa Murphy, Abhishek Mishra, Indra Joshi, Kassandra Karpathakis, et al. Governing data and artificial intelligence for health care: Developing an international understanding. *JMIR formative research*, 6(1):e31623, 2022. doi:10.2196/31623.

[101] Simona Motogna, Diana Cristea, Diana-Florina Sotropa, and Arthur-Jozsef Molnar. Uncovering bad practices in junior developer projects using static analysis and formal concept analysis. In *ENASE*, pages 752–759, 2024.

[102] John Murdoch, Graham Clark, Antony Powell, and Paul Caseley. Measuring safety: applying psm to the system safety domain. pages 47–55, 2003.

[103] Sireen Kamal Najjar and Khalid T. Al-Sarayreh. Capability Maturity Model of Software Requirements Process and Integration (SRPCMMI). *Proceedings of the International Conference on Intelligent Information Processing, Security and Advanced Communication - IPAC '15*, pages 1–5, 2015. doi:10.1145/2816839.2816856.

[104] Ministry of Defence. ASEMS. URL: https://www.asems.mod.uk/.

[105] Ministry of Defence. Safety Management Requirements for Defence Systems Part 1: Requirements. Standard Defence Standard 00-56 Part 1, Ministry of Defence, 2017.

[106] Object Management Group (OMG). Structures . Technical report, 2018.

[107] SPEM OMG and O Notation. Software systems process engineering metamodel specification. *OMG Std., Rev*, 2, 2008.

[108] Abraham Naftali Oppenheim. *Questionnaire design, interviewing and attitude measurement*. Bloomsbury Publishing, 2000.

[109] Karen O'Reilly. *Ethnographic Methods*. Routledge, Abingdon, 2nd edition, 2012.

[110] Matt Osborne. Vf3800 as required (closed). URL: https://github.com/thefuriousengineer/Empirical-Research/blob/Empirical-Research/VF38000_As_Described.pdf,year={2021}.

[111] Matt Osborne. Arp4754a - as desired principle 1, 2021. URL: ttps://github.com/thefuriousengineer/Empirical-Research/blob/Empirical-Research/ARP4754A_AsDesiredPrinciple1_v01.pdf.

[112] Matt Osborne. Iso 62304 as required (open), 2021. URL: https://github.com/thefuriousengineer/Empirical-Research/blob/Empirical-Research/ISO62304_As%20Described.pdf.

[113] Matt Osborne. Jb61834 - as required (closed), 2021. URL: https://github.com/thefuriousengineer/Empirical-Research/blob/Empirical-Research/JB61834_AsDescribed_Clean.pdf.

[114] Matt Osborne. Jb61834 - as observed, 2023. URL: https://github.com/thefuriousengineer/Empirical-Research/blob/Empirical-Research/JB61834%20As%20Observed.pdf.

[115] Matt Osborne. Jb61834 - step 7, 2023. URL: https://github.com/thefuriousengineer/Empirical-Research/blob/Empirical-Research/Step%207.pdf.

[116] Matt Osborne. Jb61834 - step 8, 2023. URL: https://github.com/thefuriousengineer/Empirical-Research/blob/Empirical-Research/JB61834_Step%208.pdf.

[117] Matt Osborne. Ay8597 questionnaire - session one, 2024. URL: https://github.com/thefuriousengineer/Independent-Evaluation/blob/main/AY8697_EQs_SessionOne.pdf.

[118] Matt Osborne. Ay8597 questionnaire - session two, 2024. URL: https://github.com/thefuriousengineer/Independent-Evaluation/blob/main/AY8697_EQs_SessionTwo.pdf.

[119] Matt Osborne. Evaluation session one handout, 2024. URL: https://github.com/thefuriousengineer/Independent-Evaluation/blob/main/Session%20One%20Handout.pdf.

[120] Matt Osborne. Evaluation session one process instructions, 2024. URL: https://github.com/thefuriousengineer/Independent-Evaluation/blob/main/Session%20One%20Instructions.pdf.

[121] Matt Osborne. Evaluation session one tutorial, 2024. URL: https://github.com/thefuriousengineer/Independent-Evaluation/blob/main/Evaluation%20Session%20One.pdf.

[122] Matt Osborne. Evaluation session repository, 2024. URL: https://github.com/thefuriousengineer/Independent-Evaluation.

[123] Matt Osborne. Hh75783 questionnaire - session one, 2024. URL: https://github.com/thefuriousengineer/Independent-Evaluation/blob/main/HH75783_EQs_SessionOne.pdf.

[124] Matt Osborne. Hh75783 questionnaire - session two, 2024. URL: https://github.com/thefuriousengineer/Independent-Evaluation/blob/main/HH75783_EQs_SessionTwo.pdf.

[125] Matt Osborne. Sh27236 questionnaire - session one, 2024. URL: https://github.com/thefuriousengineer/Independent-Evaluation/blob/main/SH27236_EQs_SessionOne.pdf.

[126] Matt Osborne. Sh27236 questionnaire - session two, 2024. URL: https://github.com/thefuriousengineer/Independent-Evaluation/blob/main/SH27236_EQs_SessionTwo.pdf.

[127] Matt Osborne. Sj 84999 interview transcript, 2024. URL: https://github.com/thefuriousengineer/Empirical-Research/blob/Empirical-Research/SJ84999%20Interview%20Transcript_redacted.pdf.

[128] Matt Osborne, Richard Hawkins, Mark Nicholson, and Rob Alexander. Understanding safety engineering practice: Comparing safety engineering practice as desired, as required, and as observed. *Safety science*, 172:106424, 2024. doi:10.1016/j.ssci.2024.106424.

[129] Matt Osborne and Mark Nicholson. Skills for assuring the safe adoption of emerging technology. 2023.

[130] Matt Osborne, Mark Nicholson, and Richard Hawkins. Empirical evaluation of the impediments to an "as desired" model of software safety assurance. *Systems and Covid-19*, 2021.

[131] João Pimentel, Jaelson Castro, Emanuel Santos, and Anthony Finkelstein. Towards requirements and architecture co-evolution. *Lecture Notes in Business Information Processing*, 112 LNBIP, 2012. doi:10.1007/978-3-642-31069-0_14.

[132] David J. Provan, Sidney W.A. Dekker, and Andrew J. Rae. Bureaucracy, influence and beliefs: A literature review of the factors shaping the role of a safety professional. *Safety science*, 98:98–112, 2017. doi:10.1016/j.ssci.2017.06.006.

[133] David J. Provan, Andrew J. Rae, and Sidney W.A. Dekker. An ethnography of the safety professional's dilemma: Safety work or the safety of work? *Safety science*, 117:276–289, 2019. doi:10.1016/j.ssci.2019.04.024.

[134] David J. Provan, David D. Woods, Sidney W.A. Dekker, and Andrew J. Rae. Safety ii professionals: How resilience engineering can transform safety practice. *Reliability Engineering & System Safety*, 195:106740, 2020. doi:10.1016/j.ress.2019.106740.

[135] Andrew Rae, David Provan, Hossam Aboelssaad, and Rob Alexander. A manifesto for reality-based safety science. *Safety science*, 126:104654, 2020. doi:10.1016/j.ssci.2020.104654.

[136] Andrew J Rae, David J Provan, David Emanuel Weber, and Sidney WA Dekker. Safety clutter: the accumulation and persistence of 'safety'work that does not contribute to operational safety. *Policy and practice in health and safety*, 16(2):194–211, 2018.

[137] Andrew J. Rae, David E. Weber, and Sidney W.A. Dekker. Work as planned, as done and as desired: A framework for exploring everyday safety-critical practice. In *Inside Hazardous Technological Systems*, pages 115–132. CRC Press, 2021.

[138] Balasubramaniam Ramesh, Lan Cao, and Richard Baskerville. Agile requirements engineering practices and challenges: an empirical study. *Information Systems Journal*, 20(5):449–480, 2010. doi:10.1111/j.1365-2575.2007.00259.x.

[139] Jens Rasmussen. Risk management in a dynamic society: a modelling problem. *Safety science*, 27(2-3):183–213, 1997. doi:10.1016/S0925-7535(97)00052-0.

[140] Felix Redmill. Agile Methods for Developing Safety-related Software ? (September 2014):1–24, 2015.

[141] D.W. Reinhardt and John McDermid. Contracting for Assurance of Military Aviation Software Systems. In *Proceedings of the Australian System Safety Conference (ASSC 2012)*, number Assc, pages 91–105, 2012.

[142] Karlene H. Roberts. Managing high reliability organizations. *California management review*, 32(4):101–113, 1990. doi:10.2307/41166631.

[143] Carl Rollenhagen. Can focus on safety culture become an excuse for not rethinking design of technology? *Safety Science*, 48(2):268–278, 2010.

[144] John Rooksby, Mark Rouncefield, and Ian Sommerville. Testing in the wild: The social and organisational dimensions of real world practice. *Computer Supported Cooperative Work (CSCW)*, 18(5-6):559, 2009. doi:10.1007/s10606-009-9098-7.

[145] RTCA. Software Considerations in Airborne Systems and Equipment Certification. standard RTCA DO-178C, 2011.

[146] SAE Aerospace. Aerospace Recommended Practice Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment. Standard ARP 4761, 1996.

[147] SAE Aerospace. Aerospace Recommended Practice (R) Guidelines for Development of Civil Aircraft and Systems. Standard ARP 4754A, 2010.

[148] Ali Shahrokni and Robert Feldt. Towards a framework for specifying software robustness requirements based on patterns. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 79–84. Springer, 2010. doi:10.1007/978-3-642-14192-8_9.

[149] Hui Shen, Brian Wall, Michal Zaremba, Yuliu Chen, and Jim Browne. Integration of business modelling methods for enterprise information system analysis and user requirements gathering. *Computers in Industry*, 54(3):307–323, 2004. doi:10.1016/j.compind.2003.07.009.

[150] Steven Shorrock. Proxies for Work-as-Done: 3. Work-as-Disclosed, 2020. URL: https://humanisticsystems.com/2020/11/01/proxies-for-work-as-done-3-work-as-disclosed/.

[151] Bridget Somekh and Cathy Lewin. *Research Methods in the Social Sciences*. Sage, London, 2005.

[152] MJ Squair. Issues in the application of software safety standards. *Proceedings of the 10th Australian Workshop on Safety Critical Systems and Software*, 55:13–26, 2006. URL: http://dl.acm.org/citation.cfm?id=1151818.

[153] Tor Stålhane, Thor Mykelbust, and Geir K. Hanssen. Safety Standards and Scrum - A Synopsis of Three Standards. *Nbl.Sintef.No*, 2013. URL: https://nbl.sintef.no/upload/IKT/9013/SafetystandardsandScrum_May2013.pdf.

[154] Margaret-Anne Storey, Neil A Ernst, Courtney Williams, and Eirini Kalliamvakou. The who, what, how of software engineering research: a socio-technical framework. *Empirical Software Engineering*, 25:4097–4129, 2020.

[155] Mark A. Sujan, Dominic Furniss, Janet Anderson, Jeffrey Braithwaite, and Erik Hollnagel. Resilient health care as the basis for teaching patient safety–a safety-ii critique of the world health organisation patient safety curriculum. *Safety Science*, 118:15–21, 2019. doi:10.1016/j.ssci.2019.04.046.

[156] Alistair G. Sutcliffe and Shailey Minocha. Linking business modelling to socio-technical system design. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1626:73–87, 1999. doi:10.1007/3-540-48738-7_7.

[157] Paul Swuste, Jop Groeneweg, Coen Van Gulijk, Walter Zwaard, Saul Lemkowitz, and Yvette Oostendorp. The future of safety science. *Safety science*, 125:104593, 2020. doi:10.1016/j.ssci.2019.104593.

[158] Norris Syed Abdullah, Shazia Sadiq, and Marta Indulska. A compliance management ontology: Developing shared understanding through models. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7328 LNCS:429–444, 2012. doi:10.1007/978-3-642-31095-9_28.

[159] Bastian Tenbergen, Thorsten Weyer, and Klaus Pohl. Hazard Relation Diagrams: a diagrammatic representation to increase validation objectivity of requirements-based hazard mitigations. *Requirements Engineering*, 23(2):291–329, 2018. doi:10.1007/s00766-017-0267-9.

[160] UK Ministry of Defence. Defence Standard 00-055 Part 1 Requirements for Safety of Programmable Elements (PE) in Defence Systems Part 1 : Requirements and Guidance. (4), 2016.

[161] I. B. Utne, P. Hokstad, and J. Vatn. A method for risk modeling of interdependencies in critical infrastructures. *Reliability Engineering and System Safety*, 96(6):671–678, 2011. doi:10.1016/j.ress.2010.12.006.

[162] Jose Luis de la Vara and Rajwinder Kaur Panesar-Walawege. Safetymet: A metamodel for safety standards, 2013. doi:10.1007/978-3-642-41533-3_5.

[163] Jéssyka Vilela, Jaelson Castro, Luiz Eduardo G. Martins, and Tony Gorschek. Integration between requirements engineering and safety analysis: A systematic literature review. *Journal of Systems and Software*, 125:68–92, 2017. doi:10.1016/j.jss.2016.11.031.

[164] Georg Von Krogh, Cristina Rossi-Lamastra, and Stefan Haefliger. Phenomenon-based research in management and organisation science: When is it rigorous and does it matter? *Long range planning*, 45(4):277–298, 2012. doi:10.1016/j.lrp.2012.05.001.

[165] Robert A. Weaver. *The safety of software: Constructing and assuring arguments*. Citeseer, 2003.

[166] Lian Wen, David Tuffley, and R. Geoff Dromey. Formalizing the transition from requirements' change to design change using an evolutionary traceability model. *Innovations in Systems and Software Engineering*, 10(3):181–202, 2014. doi:10.1007/s11334-014-0230-6.

[167] Ron Westrum. A typology of organisational cultures. *BMJ Quality & Safety*, 13(suppl 2):ii22–ii27, 2004.

[168] W. Eric Wong, Vidroha Debroy, Adithya Surampudi, HyeonJeong Kim, and Michael F. Siok. Recent catastrophic accidents: Investigating how software was responsible. In *2010 Fourth International Conference on Secure Software Integration and Reliability Improvement*, pages 14–22, 2010. doi:10.1109/SSIRI.2010.38.

[169] Robert K. Yin. *Case Study Research Design and Methods*. Sage, California, 5th edition, 2014.

[170] Koen Yskout, Riccardo Scandariato, and Wouter Joosen. Change patterns: Co-evolving requirements and architecture. *Software & Systems Modeling*, 13:625–648, 2014. doi:10.1007/s10270-012-0276-6.

[171] Didar Zowghi, Aditya K. Ghose, and Pavlos Peppas. A Framework for Reasoning about Requirement Evolution. *Proceedings of the 4th Pacific Rim International Conference on Artificial Intelligence, Cairns, Australia, 1996*, pages 157–168, 1996. doi:10.1007/3-540-61532-6_14.

Appendix A

ARP 4754A - A Critique and Characterisation

This characterization of ARP 4754A was created as part of the early stages of the empirical research. The aim at the time was to create a meta- or supermodel of recognized good practice in an attempt to create a referenceable benchmark of 'best practice' for software safety. Whilst a completed model was never created, this appendix provides valuable supporting data to the literature review in Chapter 2 as to the limitations of Open Standards in their role as software safety practice as-required (Open).

The ARP relies heavily on sister publications such as ARP 4761 [146] (for Functional Safety Techniques and Measures), and child publications such as DO178C [145] and DO254 for the assurance of software and complex electronic hardware respectively. It regards the activities of standards such as DO-178C as the means by which development mitigates 'design errors'; but limits this unidirectionally as a *"development assurance process from aircraft level down to item level"*.

In directing the reader to the process of allocating requirements from 'system' to 'item', ARP 4754A notes that it is an iterative cycle (where requirements become 'clearer' with each iteration); but falls short of asserting an iterative bi-directional process that links activities, or stating the potential entry/exit points at which the sibling/child standards are entered or exited (including any assurance activities and analyses that must occur in order to do so).

The ARP majors on objectives and an overarching intent but is lacking in any detailed guidance or mandated/recommended analyses. As such it is not explicit in its treatment of software hazard analyses [71] (accepting that it doesn't set out to do this – relying instead on related publications).

Throughout the standard, the stipulation of what constitutes 'recommended practice' is avoided – offering what can only be described as 'helpful advice', without making any stipulations on whether/how it would be beneficial to follow it. This is done under the explicit assumption that a regulator or certifying body will agree the analyses, techniques, or practices with the developer (in a manner that will result in 'certification').

It should be noted that in many cases, a list of recommendations is offered with a caveat that they may not, in fact, constitute best practice. If accepted as an acceptable means of compliance, the terminology of the ARP changes between 'could' and 'may' to 'shall' and 'should'. The caveat that it may not constitute best practice hints at the existence of more suitable practice, however.

The right hand (verification) side of the ARP 4754A V-model again implies a sequential, yet iterative and recursive process that appears strictly chronological in order; and the findings on the inadequacies of the left-hand side of the V-model are echoed for the right. Indeed, the ARP notes that the bottom of the V is the point at which it 'hands over' to DO-178/254. It does highlight the need for "*extra rigor*" [sic] when interfaces span organizational or contractual boundaries but offers no guidance on how such commercial, communication, legal, or contractual complexities can be managed. Such shortcomings in open standards are highlighted, and guidance is provided as to how the shortcomings can be mitigated by Menon [97].

Dealing with verification, the ARP notes that independence may be key (dependent on the DAL) and offers suggested (but high-level) methods (Inspection/Review; Analysis; Test/Demonstration; and Service Experience). Although the ARP offers a brief insight in the constituent parts of each methodology, it stops short of discussing which methods are more suited/appropriate to different assurance levels; what the verification activities should cover; the (code / data / timing / states) coverage required for each methodology; what artefacts / activities are required for each; whether formal tools should / may be employed; what the output of each method should comprise; who is expected to carry them out / independently assess them; or when they should occur in relation to the development lifecycle. It is assumed by ARP 4754A that the specific requirements required for software are dealt with in child publications such as DO-178C.

In terms of safety integrity, the ARP uses the concept of Development Assurance Levels (DALs) to moderate the objectives of the standard according to the criticality of the software under development [97]. It asserts at Section 5.2 (Development Assurance Level Assignment) that the means of eliminating errors (in fact limiting "*the likelihood of development errors*") is by appeal to an assurance of the development process of 'items', and only begins to consider the mitigating/contributing effects of the proposed architecture when assigning Functional Development Assurance Levels (FDAL) at a lower level in the lifecycle process.

Taken from 5.2.3.1 when discussing a single FDAL A development process:

"... the development assurance process needs to provide confidence that the development error(s) will be detected and resolved within the process rather than relying on independence within the architecture".

Such a (prescriptive) process-based assurance process (in isolation of a product-based assurance process, or robust assurance of safety requirements) is not supported by the literature, however [152], [43], [?], [53], [29], [54].

A particular strength of the ARP is found within its discussions on conceptual diversity, by asserting the need to consider functional independence of different functions *"in order to minimize the likelihood of common requirement errors"*. This consideration takes the premise of conceptual diversity (as a means of mitigating common mode / cause failure) a stage further than is found in some open standards, by mitigating not only requirement errors, but also common requirement interpretation errors. It does not offer guidance on how this should be achieved, however – noting only that *"this should be substantiated at all levels of abstraction or requirement decomposition"*.

Requirements Engineering is only loosely considered by the ARP as it majors on traceability only; neglecting the assurance of the traced requirements and offers no guidance on what would constitute trustworthy evidence.

Vague statements abound with regards to the management of safety requirements, such as the section that purports to deal with the 'determination' of safety requirements; noting only that they are derived through assessment(s) "*commensurate with the guidance*" – but stopping short of stipulating the guidance (again relying on its sister publication ARP 4761 to provide this).

The subject of 'Requirements Capture' is discussed at Section 5.3 of the ARP, but again, only at a very high-level of abstraction; noting only that the "*selection of the architecture establishes additional requirements necessary to implement the architecture*" and that each phase of requirements identification and allocation process further and derived requirements will be identified (as well as more granularity of detail for existing requirements).

Safety requirements, specifically, are dealt with in Section 5.3.1.1, noting only that they should be "*determined by conducting a safety assessment consistent with the processes in Section 5.1.*" With very limited advice on the visibility of requirements at the software and electronic hardware design level, no specific details are offered.

Although the derivation of requirements is referenced throughout the ARP, Section 5.3.1.4 specifically discusses 'Derived Requirements' as a category in its own right; using a definition of 'Derived Requirements' here as "*requirements that may not be uniquely related to a higher-level requirement*". This definition (not widely shared with in other Open Standards) is unhelpful as it may be inferred that this can be read across to the wider parts of the standard when referencing the derivation of requirements.

Section 5.3.2 notes that certain safety assessments will derive safety requirements, but again offers only vague guidance on what types of safety assessment are required, and the measures of performance and constraints that can be allocated against such requirements. It offers no assertions as to how these requirements should be managed as they evolve however. Nor does the ARP consider what constitutes a set of reasonable assessment techniques (pertinent to the level of design abstraction/stage in the safety lifecycle) that may be undertaken to elicit such safety requirements.

Section 5.4 of the ARP considers the validation of requirements and presents a list of generic considerations that *"may be helpful"* – leaving the specific format of requirements validation to the developer. This high-level guidance (and the various other shortcomings posited within this Annex) can only be effective and robust when relying on a certification authority or regulator to endorse the subset of developmental activities. Otherwise, the vagaries of the standard are left to the developer to design to a commensurate level of assurance – relying instead on sister publications such as ARP 4761.

Figure 12 of the ARP presents a 'Validation Process Model', which also appears linear and hierarchical in manner – although the accompanying text suggests that (in reality) the process is cyclic and recursive; with confidence in the process increasing with each cycle. It notes that a "*completeness and correctness*" check should be carried out, but without guidance on what constitutes such properties.

Although the ARP discusses the management and validation of requirements, it offers no guidance on HOW assumptions should be validated – nor what constitutes 'trustworthy' evidence of 'supporting data' (related to assumption validation or indeed evidence at large). It is noted that only in the fifth section (5.4.2) does the standard acknowledge the presence of *"bottom-up influences"*.

Section 5.4.4.1 (Templates and Checklists) suggests considerations when assessing whether the requirements have been managed, but whilst it does so, it offers no guidance on how judgements may be formed and assured. In-service considerations for the validation of safety requirements are dealt with in Section 5.4.4.2 and asserts some techniques which may be beneficial in eliciting requirements that may have been missed - but asserts that they may not be 'the best methods' to use. Such a statement calls their efficacy into question. A more useful guide would be to specify what the best (or at least recommended) techniques are – perhaps varying in accordance with FDAL/IDAL (which is only mentioned in abstract at 5.4.5).

Further techniques are asserted in Section 5.4.6 (including a table of recommended techniques in line with the DAL) – but these major on traceability (alone), testing, or some form of independent or peer review and 'vigilance' - all of which are of limited value for justifying the trustworthiness of data.

The lifecycle diagram at Fig 5.3 suggests that software safety requirements are only considered by the time the 'bottom of the V' is reached. Hawkins observed in 2006 that the Preliminary System Safety Assessment (PSSA) is the first point at which software safety requirements are considered [54] – as part of the system requirements which must be allocated to software; which is not articulated in the denoted linear process. Hawkins later affirmed the importance of establishing the specific failure modes of software that may contribute to hazards [49], and the linking of hazards to software safety requirements are also remiss from the ARP's process. Hawkins cautions that – if this linking is not not undertaken – there will exist a *"danger of defining generic software safety requirements, or simply correctness requirements, that can fail to address the specific hazardous failure modes that affect the safety of the system."*

Habli and Kelly urged further caution against not considering software requirements at the system level [44]; asserting that a lack of system-level requirements review by software analysts may give rise to common mode failures.

Conspicuous by its absence is the lack of consideration concerning the entry/exit points in the lifecycle for pre-existing software (or hardware elements).

Appendix B

ARP 4754A and DO178C Assessment Against the As Desired Criteria

In this Annex we present the assessment of ARP 4754A and DO 178C against the criteria that must be met to comply with the first principle of the 'As Desired' model.

Principle 1 requires that the lifecycle process under assessment ensures:

- A clear description of the software in the system will be provided
- The operating context of the system in which the software resides will be described
- A clear description of the system in which the software resides will be provided
- The system hazards to which software may contribute will be identified
- The specific failure modes by which software contributes to the identified system hazards will be described
- The software contribution to the identified system hazards will be acceptably managed through the elicitation of software safety requirements that specify the required behaviour(s); for each identified software contribution, to each system hazard
- All software safety requirements will be atomic, unambiguous, defined in sufficient detail, and verifiable.

. . . and to comply with the 'plus one' Principle, each Principle must ensure:

- The required confidence behind the attainment of each Principle (1 to 4) will be determined

- The most effort in generating evidence will be focussed on the areas with the highest risk from software's contribution to hazards (noting that the areas denoted as requiring the most effort are currently indicated in Open Standards by the notions of integrity/assurance levels)
- For each Principle (1 to 4), the required confidence that each has been met will be reflected in:

  (a)The appropriateness of evidence

  (b) The trustworthiness of each evidential artefact (the rigour in the approaches to be used):

  (i) Independence

  (ii) Resources and required attributes (personnel)

  (iii) Techniques and Methods used

  (iv) Audits and reviews

  (v)Tools

  (c) The type of evidence to be used.
- An understanding of the limitations with each type of evidence being used will be clearly understood.

Taking each criterion in turn, we assess the levels of compliance.

B.1 A Clear Definition of Software Within the System

There is no definitive artefact that is produced by the ARP 4754A lifecycle that would clearly identify the software within the system. Whilst many disparate artefacts may be used to extract the information, it remains unclear and un-defined from a coherent, identifiable source.

Of the artefacts in the model of as-required (Open) software safety practice, it would be reasonably expected for this information to reside in one or more of the following artefacts:

- System Architectures
- System Description and Environment
- Software Design Details
- Software Load Control Records
- (Software) Loading Data

- Software Configuration Management Records
- Software Configuration Index
- Candidate Platform Architecture.

The relationship of these artefacts in the ARP 4754A model of practice are shown in Figure B.1. Not all links are modelled here, as only the contributors to the criteria of Principle 1 are represented.

Whilst the model at Fig B.1 represents the candidate artefacts within which a clear description of the software in a system may reside, it is an unconnected set of artefacts and activities that produces/consumes them. The artefacts 'System Description and Environment', and 'Software Design Details' – which may be reasonably considered the most obvious artefacts that would contain a clear description of the software – are, in fact 'orphan' artefacts (in that there is no identified activity that produces them).

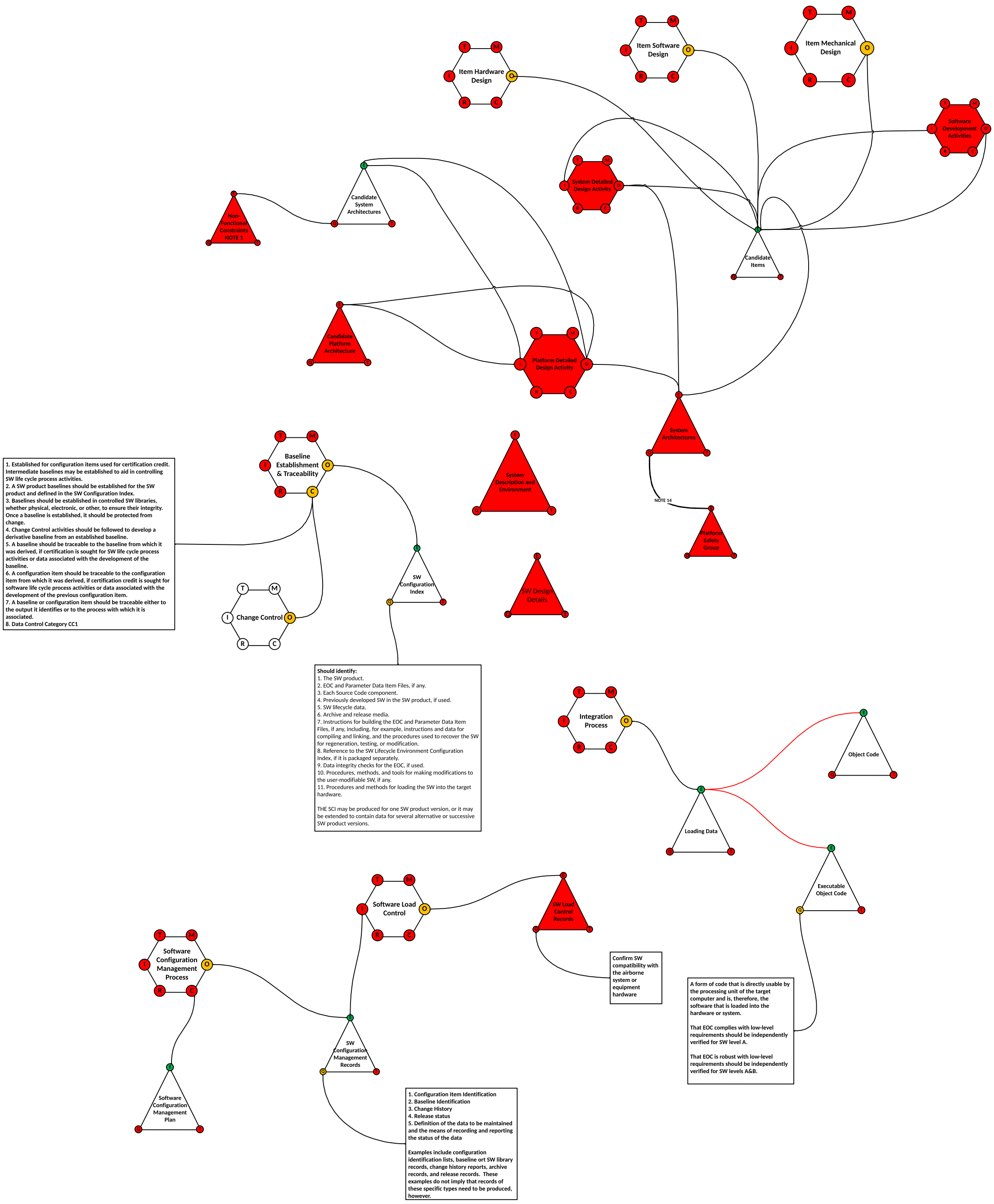
'System Description and Environment' is an inferred artefact created at 'Platform' level as an input to the activities 'Validation Planning' and 'Safety Requirements Validation. When discussing the 'Validation Process Model' at Section 5.4.2, ARP 4754 notes that "*inputs to the validation process may include a description of the system (including the operating environment*". The ARP does not define when this data is created, nor by what activity, however.

'Software Design Details' is also an inferred activity created at 'Platform' level as an input to the activity 'Safety Requirements Verification'. Section 2.5.6 of DO-178C notes that "*Software design details that relate to the system functionality need to be made available to aid system verification*", yet the nature of these details is not expanded on, nor is the activity that produces them identified.

'System Architectures' is an artefact created by the (also inferred) activities 'Platform Detailed Design Activity' and 'System Detailed Design Activity'. Section 4.6 of ARP 4754A notes that "*candidate system architectures are derived from the activity 'System Requirements Identification' which are iteratively and recursively evaluated using the PSA, PSSA, and CCA processes in order to establish their feasibility in meeting the requirements*". At some stage, these candidate architectures must be formally endorsed through design decision(s); yet such an activity is not considered. There is no activity that 'transforms' candidate architectures into baselined architectures, and as an artefact cannot link to another artefact, the linking line between these two artefacts is coloured red.

'Software Load Control Records' is another inferred artefact – assumed to be created by the activity 'Software Load Control'. The only quality attribute considered for this artefact is the conformation of compatibility with the system or equipment hardware, however.

'Software Loading Data' is an artefact created by the activity 'Integration Process', but no consideration of its aspects is made by the lifecycle process.


Item Hardware Design
Item Software Design
Item Mechanical Design
Software Development Activities
System Detailed Design Activity
Candidate System Architectures
Non-Functional Constraints NOTE 1
Candidate Items
Candidate Platform Architecture
Platform Detailed Design Activity
System Architectures
Baseline Establishment & Traceability
System Description and Environment
NOTE 14
Platform Safety Group
SW Configuration Index
SW Design Details
Change Control
Integration Process
Object Code
Loading Data
Executable Object Code
Software Load Control
SW Load Control Records
Software Configuration Management Process
SW Configuration Management Records
Software Configuration Management Plan
T
M
I
O
R
C
E
Q

1. Established for configuration items used for certification credit. Intermediate baselines may be established to aid in controlling SW life cycle process activities.
2. A SW product baselines should be established for the SW product and defined in the SW Configuration Index.
3. Baselines should be established in controlled SW libraries, whether physical, electronic, or other, to ensure their integrity. Once a baseline is established, it should be protected from change.
4. Change Control activities should be followed to develop a derivative baseline from an established baseline.
5. A baseline should be traceable to the baseline from which it was derived, if certification is sought for SW life cycle process activities or data associated with the development of the baseline.
6. A configuration item should be traceable to the configuration item from which it was derived, if certification credit is sought for software life cycle process activities or data associated with the development of the previous configuration item.
7. A baseline or configuration item should be traceable either to the output it identifies or to the process with which it is associated.
8. Data Control Category CC1

Should identify:
1. The SW product.
2. EOC and Parameter Data Item Files, if any.
3. Each Source Code component.
4. Previously developed SW in the SW product, if used.
5. SW lifecycle data.
6. Archive and release media.
7. Instructions for building the EOC and Parameter Data Item Files, if any, including, for example, instructions and data for compiling and linking, and the procedures used to recover the SW for regeneration, testing, or modification.
8. Reference to the SW Lifecycle Environment Configuration Index, if it is packaged separately.
9. Data integrity checks for the EOC, if used.
10. Procedures, methods, and tools for making modifications to the user-modifiable SW, if any.
11. Procedures and methods for loading the SW into the target hardware.

THE SCI may be produced for one SW product version, or it may be extended to contain data for several alternative or successive SW product versions.

Confirm SW compatibility with the airborne system or equipment hardware

A form of code that is directly usable by the processing unit of the target computer and is, therefore, the software that is loaded into the hardware or system.

That EOC complies with low-level requirements should be independently verified for SW level A.

That EOC is robust with low-level requirements should be independently verified for SW levels A&B.

1. Configuration Item Identification
2. Baseline Identification
3. Change History
4. Release status
5. Definition of the data to be maintained and the means of recording and reporting the status of the data

Examples include configuration identification lists, baseline ort SW library records, change history reports, archive records, and release records. These examples do not imply that records of these specific types need to be produced, however.

Figure B.1: ARP 4754A a Clear Definition of Software Within the System

'Software Configuration Management Records' are produced by the activity 'Software Configuration Management Process' and offer partial compliance with the criteria for Principle 1, but it cannot be held as full compliance as it does not specifically concern the software instances in each system.

The 'Software Configuration Index' is created by the activity 'Baseline Establishment and Traceability' and offers partial compliance with the criteria for Principle 1, but it cannot be held as full compliance as it does not specifically concern the software instances in each system.

The 'Candidate Platform Architecture' is an inferred artefact that is iteratively assessed by the inferred activity 'Platform Detailed Design Activity'. No quality criteria is specified for either the activity or the artefact, so no level of compliance can be claimed.

B.2 The Operating Context of the System in Which the Software Resides will be Described

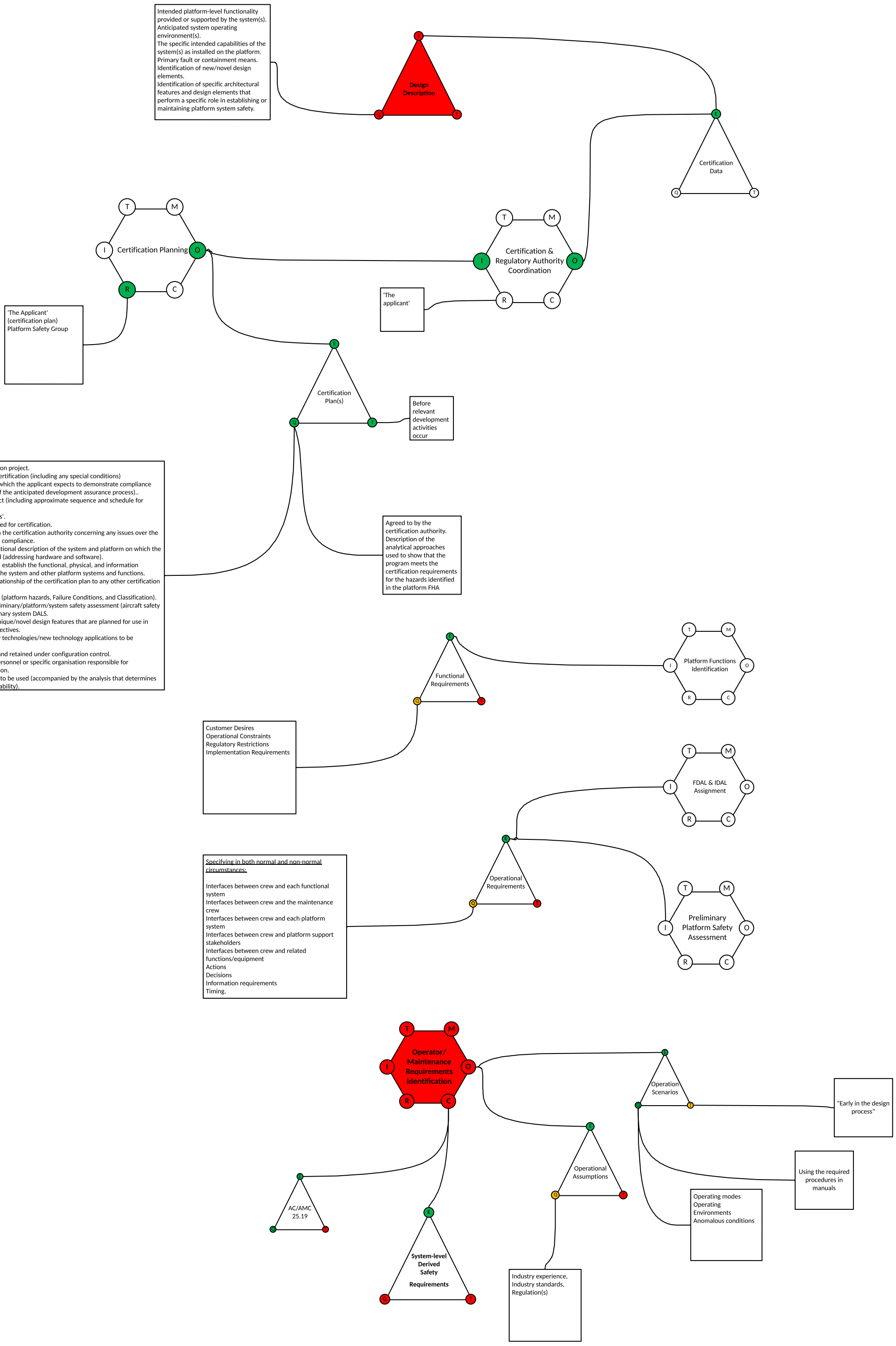
ARP4754A meets this criterion of Principle 1 by ensuring this information is captured in six artefacts as shown in Fig B.2. The six artefacts that could contain this information are one or more of the following:

- Design Description
- Certification Plan(s)
- Functional Requirements
- Operational Requirements
- Operation Scenarios
- Operational Assumptions.

The artefact 'Design Description is an inferred artefact without a producing activity. Table 9 of ARP 4754A lists the sources of information that constitute 'Certification Data', with a 'Design Description' being one of five such sources. Although some of these sources are produced by activities that are explicitly stated by the ARP, there is no activity that transforms a 'Design Description' into 'Certification Data', and as an artefact cannot directly link to another artefact, the linking line between these two artefacts is coloured red.

Despite this shortfall, the ARP does list some quality criteria for a 'Design Description' pertinent to this criterion:

- Intended platform-level functionality provided or supported by the system(s)



Intended platform-level functionality provided or supported by the system(s).
Anticipated system operating environment(s).
The specific intended capabilities of the system(s) as installed on the platform.
Primary fault or containment means.
Identification of new/novel design elements.
Identification of specific architectural features and design elements that perform a specific role in establishing or maintaining platform system safety.

'The Applicant'
(certification plan)
Platform Safety Group

'The applicant'

Before relevant development activities occur

Describe the certification project.
Identify the basis for certification (including any special conditions)
Outline the means by which the applicant expects to demonstrate compliance (including an outline of the anticipated development assurance process)..
Schedule for the project (including approximate sequence and schedule for certification events).
Definition of 'processes'.
Requirements to be used for certification.
Agreed resolution with the certification authority concerning any issues over the means to demonstrate compliance.
A functional and operational description of the system and platform on which the system will be installed (addressing hardware and software).
The description should establish the functional, physical, and information relationship between the system and other platform systems and functions.
A statement on the relationship of the certification plan to any other certification plan(s).
A summary of the FHA (platform hazards, Failure Conditions, and Classification).
A summary of the preliminary/platform/system safety assessment (aircraft safety objectives and preliminary system DALS.
A description of any unique/novel design features that are planned for use in meeting the safety objectives.
Description of the new technologies/new technology applications to be implemented.
Data to be submitted and retained under configuration control.
Identification of the personnel or specific organisation responsible for certification coordination.
How Service History is to be used (accompanied by the analysis that determines the extent of its applicability).

Agreed to by the certification authority.
Description of the analytical approaches used to show that the program meets the certification requirements for the hazards identified in the platform FHA

Customer Desires
Operational Constraints
Regulatory Restrictions
Implementation Requirements

Specifying in both normal and non-normal circumstances:

Interfaces between crew and each functional system
Interfaces between crew and the maintenance crew
Interfaces between crew and each platform system
Interfaces between crew and platform support stakeholders
Interfaces between crew and related functions/equipment
Actions
Decisions
Information requirements
Timing.

"Early in the design process"

Using the required procedures in manuals

Operating modes
Operating
Environments
Anomalous conditions

Industry experience,
Industry standards,
Regulation(s)

Figure B.2: ARP 4754A a Clear Definition of Software Within the System

- Anticipated system operating environment(s).

'Certification Plan(s)' are produced by the activity 'Certification Planning, which is resourced by 'The Applicant' and 'The Platform Safety Group'. Certification Plans are to be produced before relevant development activities occur, and include the following quality aspects pertinent to this criterion:

- A functional and operational description of the system and platform on which the system will be installed (addressing hardware and software)
- The description should establish the functional, physical, and information relationship between the system and other platform systems and functions.

'Functional Requirements' is an orphan artefact (as the ARP does not consider its creation) but is assumed to be customer-generated. Although time is not considered by the ARP, it is reasonably assumed that this is required at the concept stage of development as it is an input to the activity 'Platform Functions Identification'. The 'Functional Requirements' artefact is required to establish:

- Customer desires
- Operational constraints.

'Operational Requirements' is also an orphan artefact (as the ARP does not consider its creation) but is assumed to be customer-generated. Although time is not considered by the ARP, it is reasonably assumed that this is required at the early stages of the lifecycle as it is an input to the activities 'FDAL & IDAL Assignment' and 'Preliminary Platform Safety Assessment'. Pertinent to this criterion, this artefact will establish (in both normal and non-normal circumstances), the interfaces between:

- Crew and each functional system
- Crew and the maintenance crew
- Crew and each platform system
- Crew and platform support stakeholders
- Crew and related functions or equipment.

It will also establish the necessary:

- Actions
- Decisions

- Information requirements
- Timing.

'Operation Scenarios' are created by the inferred activity 'Operator / Maintenance Requirements Identification'. Whilst this is an inferred activity that is not resource-profiled, it is controlled by the artefacts 'AC/AMC 25.19' and 'System-level Derived Safety Requirements'. In support of this criterion, the artefact will provide details on the:

- Operating modes
- Operating Environments
- Anomalous conditions.

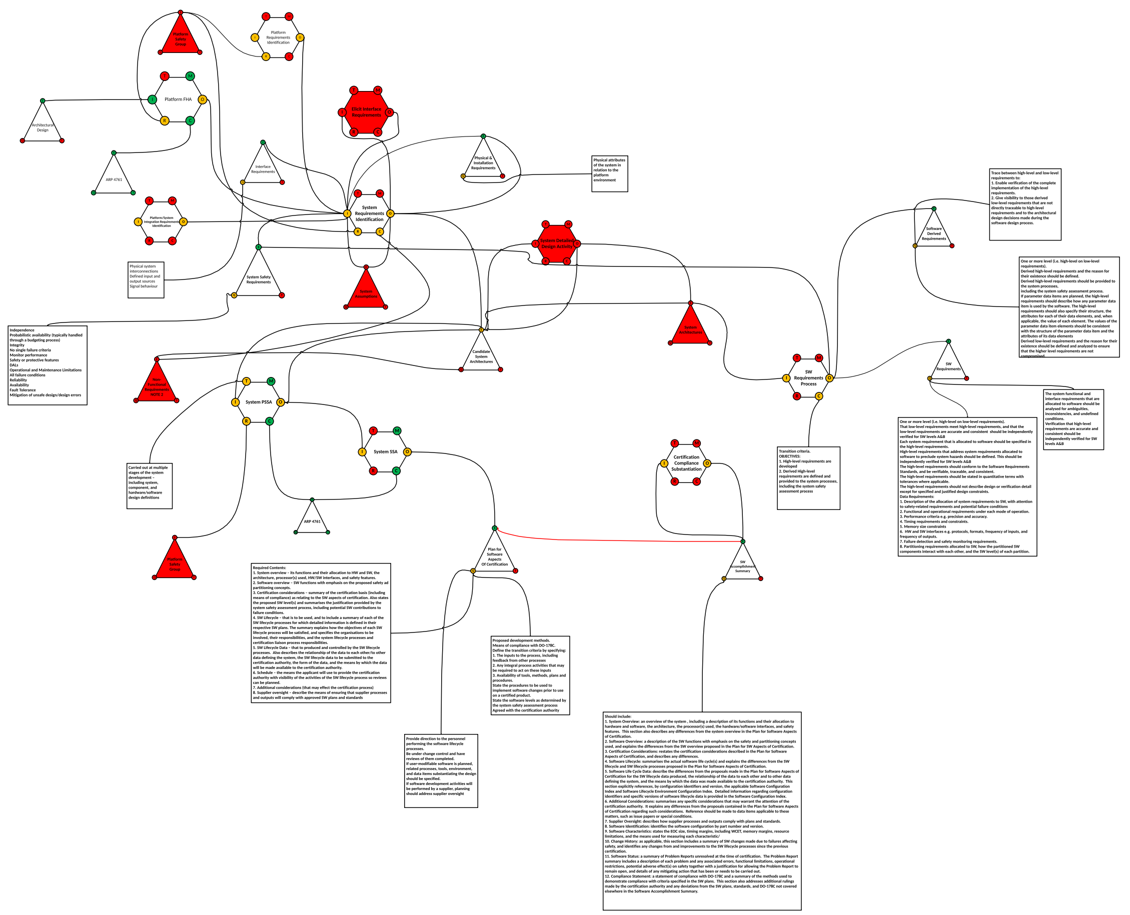
'Operational Assumptions' are created by the inferred activity 'Operator / Maintenance Requirements Identification'. Whilst this is an inferred activity that is not resource-profiled, it is controlled by the artefacts 'AC/AMC 25.19' and 'System-level Derived Safety Requirements'. In support of this criterion, the artefact will provide details on the industry standards and regulations that must be met (which will be pertinent to the intended operating context and intended use).

B.3 A Clear Description of the System in Which the Software Resides will be Provided

At the software boundary, there are only two artefacts in the lifecycle process that will provide a clear description of the system in which the software resides:

- Plan for Software Aspects of Certification
- SW Accomplishment Summary.

Both artefacts only provide this data once the design has been fully instantiated, however – as can be seen in Figure B.3.


Platform Safety Group
Platform Requirements Identification
Platform FHA
Architectural Design
ARP 4761
Elicit Interface Requirements
Interface Requirements
Physical & Installation Requirements
Platform/System Integration Requirements Identification
System Requirements Identification
System Safety Requirements
System Assumptions
System Detailed Design Activity
System Architectures
Candidate System Architectures
Non-Functional Requirements NOTE 2
System PSSA
System SSA
ARP 4761
Platform Safety Group
Software Derived Requirements
SW Requirements Process
SW Requirements
Certification Compliance Substantiation
Plan for Software Aspects Of Certification
SW Accomplishment Summary

Physical attributes of the system in relation to the platform environment

Physical system interconnections
Defined input and output sources
Signal behaviour

Independence
Probabilistic availability (typically handled through a budgeting process)
Integrity
No single failure criteria
Monitor performance
Safety or protective features
DALs
Operational and Maintenance Limitations
All failure conditions
Reliability
Availability
Fault Tolerance
Mitigation of unsafe design/design errors

Carried out at multiple stages of the system development – including system, component, and hardware/software design definitions

Trace between high-level and low-level requirements to:
1. Enable verification of the complete implementation of the high-level requirements.
2. Give visibility to those derived low-level requirements that are not directly traceable to high-level requirements and to the architectural design decisions made during the software design process.

One or more level (i.e. high-level on low-level requirements).
Derived high-level requirements and the reason for their existence should be defined.
Derived high-level requirements should be provided to the system processes,
including the system safety assessment process.
If parameter data items are planned, the high-level requirements should describe how any parameter data item is used by the software. The high-level requirements should also specify their structure, the attributes for each of their data elements, and, when applicable, the value of each element. The values of the parameter data item elements should be consistent with the structure of the parameter data item and the attributes of its data elements
Derived low-level requirements and the reason for their existence should be defined and analyzed to ensure that the higher level requirements are not compromised.

Transition criteria.
OBJECTIVES:
1. High-level requirements are developed
2. Derived High-level requirements are defined and provided to the system processes, including the system safety assessment process

One or more level (i.e. high-level on low-level requirements).
That low-level requirements meet high-level requirements, and that the low-level requirements are accurate and consistent should be independently verified for SW levels A&B
Each system requirement that is allocated to software should be specified in the high-level requirements.
High-level requirements that address system requirements allocated to software to preclude system hazards should be defined. This should be independently verified for SW levels A&B
The high-level requirements should conform to the Software Requirements Standards, and be verifiable, traceable, and consistent.
The high-level requirements should be stated in quantitative terms with tolerances where applicable.
The high-level requirements should not describe design or verification detail except for specified and justified design constraints.
Data Requirements:
1. Description of the allocation of system requirements to SW, with attention to safety-related requirements and potential failure conditions
2. Functional and operational requirements under each mode of operation.
3. Performance criteria e.g. precision and accuracy.
4. Timing requirements and constraints.
5. Memory size constraints
6. HW and SW interfaces e.g. protocols, formats, frequency of inputs, and frequency of outputs.
7. Failure detection and safety monitoring requirements.
8. Partitioning requirements allocated to SW, how the partitioned SW components interact with each other, and the SW level(s) of each partition.

The system functional and interface requirements that are allocated to software should be analysed for ambiguities, inconsistencies, and undefined conditions.
Verification that high-level requirements are accurate and consistent should be independently verified for SW levels A&B

Required Contents:
1. System overview – its functions and their allocation to HW and SW, the architecture, processor(s) used, HW/SW interfaces, and safety features.
2. Software overview – SW functions with emphasis on the proposed safety ad partitioning concepts.
3. Certification considerations – summary of the certification basis (including means of compliance) as relating to the SW aspects of certification. Also states the proposed SW level(s) and summarises the justification provided by the system safety assessment process, including potential SW contributions to failure conditions.
4. SW Lifecycle – that is to be used, and to include a summary of each of the SW lifecycle processes for which detailed information is defined in their respective SW plans. The summary explains how the objectives of each SW lifecycle process will be satisfied, and specifies the organisations to be involved, their responsibilities, and the system lifecycle processes and certification liaison process responsibilities.
5. SW Lifecycle Data – that to produced and controlled by the SW lifecycle processes. Also describes the relationship of the data to each other/to other data defining the system, the SW lifecycle data to be submitted to the certification authority, the form of the data, and the means by which the data will be made available to the certification authority.
6. Schedule – the means the applicant will use to provide the certification authority with visibility of the activities of the SW lifecycle process so reviews can be planned.
7. Additional considerations (that may effect the certification process)
8. Supplier oversight – describe the means of ensuring that supplier processes and outputs will comply with approved SW plans and standards

Proposed development methods.
Means of compliance with DO-178C.
Define the transition criteria by specifying:
1. The inputs to the process, including feedback from other processes
2. Any integral process activities that may be required to act on these inputs
3. Availability of tools, methods, plans and procedures.
State the procedures to be used to implement software changes prior to use on a certified product.
State the software levels as determined by the system safety assessment process
Agreed with the certification authority

Provide direction to the personnel performing the software lifecycle processes.
Be under change control and have reviews of them completed.
If user-modifiable software is planned, related processes, tools, environment, and data items substantiating the design should be specified.
If software development activities will be performed by a supplier, planning should address supplier oversight

Should include:
1. System Overview: an overview of the system , including a description of its functions and their allocation to hardware and software, the architecture, the processor(s) used, the hardware/software interfaces, and safety features. This section also describes any differences from the system overview in the Plan for Software Aspects of Certification.
2. Software Overview: a description of the SW functions with emphasis on the safety and partitioning concepts used, and explains the differences from the SW overview proposed in the Plan for SW Aspects of Certification.
3. Certification Considerations: restates the certification considerations described in the Plan for Software Aspects of Certification, and describes any differences.
4. Software Lifecycle: summarises the actual software life cycle(s) and explains the differences from the SW lifecycle and SW lifecycle processes proposed in the Plan for Software Aspects of Certification.
5. Software Life Cycle Data: describe the differences from the proposals made in the Plan for Software Aspects of Certification for the SW lifecycle data produced, the relationship of the data to each other and to other data defining the system, and the means by which the data was made available to the certification authority. This section explicitly references, by configuration identifiers and version, the applicable Software Configuration Index and Software Lifecycle Environment Configuration Index. Detailed information regarding configuration identifiers and specific versions of software lifecycle data is provided in the Software Configuration Index.
6. Additional Considerations: summarises any specific considerations that may warrant the attention of the certification authority. It explains any differences from the proposals contained in the Plan for Software Aspects of Certification regarding such considerations. Reference should be made to data items applicable to these matters, such as issue papers or special conditions.
7. Supplier Oversight: describes how supplier processes and outputs comply with plans and standards.
8. Software Identification: identifies the software configuration by part number and version.
9. Software Characteristics: states the EOC size, timing margins, including WCET, memory margins, resource limitations, and the means used for measuring each characteristic/
10. Change History: as applicable, this section includes a summary of SW changes made due to failures affecting safety, and identifies any changes from and improvements to the SW lifecycle processes since the previous certification.
11. Software Status: a summary of Problem Reports unresolved at the time of certification. The Problem Report summary includes a description of each problem and any associated errors, functional limitations, operational restrictions, potential adverse effect(s) on safety together with a justification for allowing the Problem Report to remain open, and details of any mitigating action that has been or needs to be carried out.
12. Compliance Statement: a statement of compliance with DO-178C and a summary of the methods used to demonstrate compliance with criteria specified in the SW plans. This section also addresses additional rulings made by the certification authority and any deviations from the SW plans, standards, and DO-178C not covered elsewhere in the Software Accomplishment Summary.

Figure B.3: ARP 4754A Clear Description of the System in which the Software Resides

The 'Plan for Software Aspects of Certification' is informed by the activity 'System SSA' and has the following relevant quality attributes:

- System overview – its functions and their allocation to hardware and software, the architecture, processor(s) used, hardware/software interfaces, and safety features
- State the software levels as determined by the system safety assessment processes.

It also informs the artefact 'SW Accomplishment Summary', but as there is no activity that transforms a 'Plan for Software Aspects of Certification' into a 'Software Accomplishment Summary', and as an artefact cannot link directly to another artefact, the linking line between these two artefacts is coloured red.

As well as being informed by the 'Plan for Software Aspects of Certification', the artefact 'Software Accomplishment Summary' is created by, and iteratively informs the activity 'Certification Compliance Substantiation'. It has the following quality attributes specified for the system overview:

- An overview of the system, including a description of its functions and their allocation to hardware and software
- The processor(s) used
- The hardware/software interfaces
- Safety features.

It also describes the differences from the system overview in the Plan for Software Aspects of Certification, yet it is not clear why there should be any differences between these two artefacts (especially by the time a certification / accomplishment stage is reached).

As these two artefacts are produced towards the end of the design lifecycle, and only contain a section that gives a system overview, full compliance with this criterion cannot be claimed without recourse to activities and artefacts undertaken and produced at higher levels of abstraction.

The system-level activity 'System SSA' (which creates the 'Plan for Software Aspects of Certification', also informs/creates (relevant to this criteria):

- Item Software Design (activity with iterative links)
- Item Hardware Design (activity with iterative links)
- SW Levels (artefact).

Other than the safety engineering activities that contribute to the ability to create a (P)SSA, there are no inputs to the activity 'System SSA' that could impart a clear description of the system to the lower levels of design abstraction. We must therefore look to other safety engineering activities:

- System PSSA
- System FTA (an inferred activity)
- System CCA.

The System PSSA is further informed by more safety engineering activities, the relevant activities to this criterion being:

- System FHA
- System Requirements Identification.

The System PSSA also takes 'Candidate System Architectures' as an iterative input linked to the artefact 'System Architectures' (an inferred activity that is not linked by a consuming activity). 'Candidate System Architectures' is created by iterative links to:

- Preliminary Platform Safety Assessment
- Platform Safety Assessment
- System Requirements Identification
- FDAL and IDAL Assignment
- Platform CCA.

These are all linked through to the activity 'System Requirements Identification'. System Requirements Identification is informed by:

- System Safety Requirements (artefact)
- Elicit Interface Requirements (inferred activity with iterative links)
- Interface Requirements (artefact)
- System Assumptions (inferred artefact with iterative links)
- Platform/System Integration Requirements Identification (activity)
- Platform FHA (activity)
- Platform Requirements Identification (activity)
- Physical and Installation Requirements (artefact with iterative links).

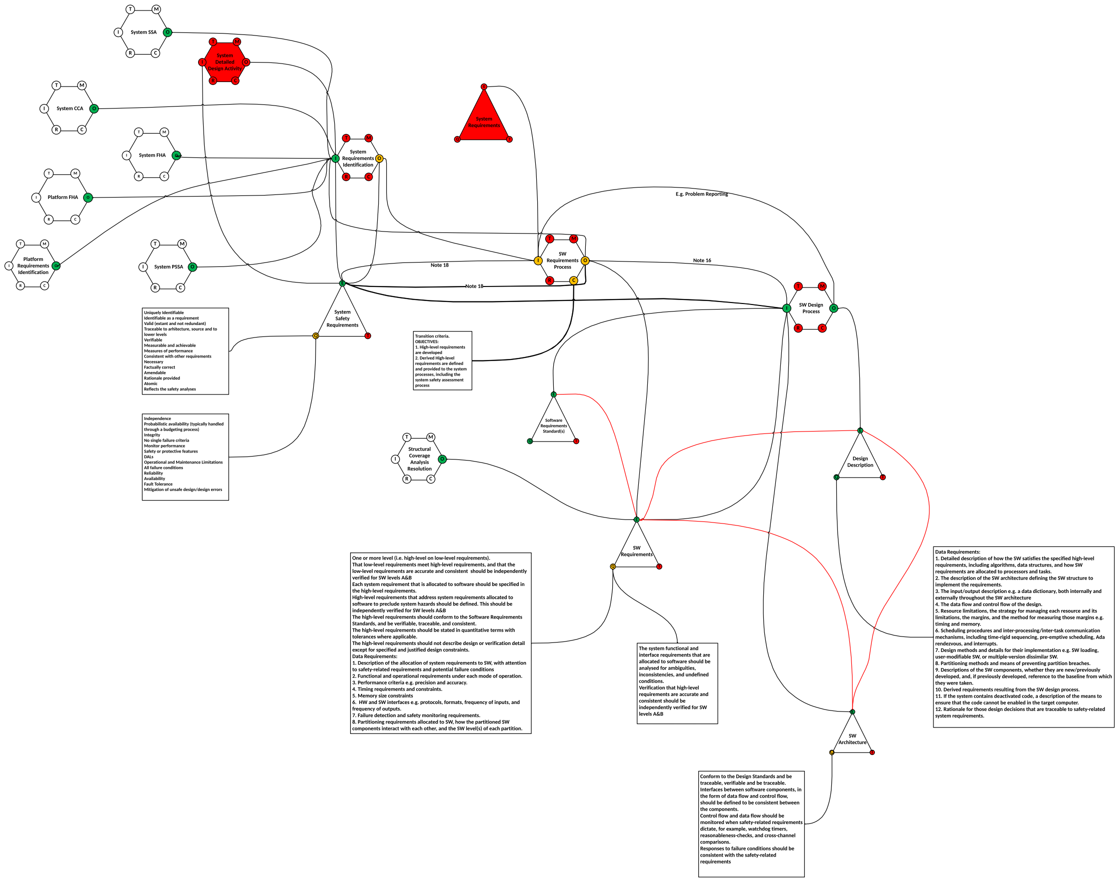
Other than 'System Requirements Identification', the activities reveal no further (relevant) trace, however. The activity 'SW Requirements Process' has inputs from the (inferred) artefact 'System Architectures', which is 'informed' by the artefact 'Candidate System Architectures'; and the inferred activity 'System Detailed Design Activity - with all potentially relevant data to this criterion passed through the artefact 'Candidate System Architectures'.

As a deductive link to the activity 'Software Requirements Process' can be traced back to all the expected activities/artefacts that would contribute to imparting a clear definition of the system in which the software will reside, one can assume that such a description will be known at the software boundary. However, as this is not explicitly made, only partial compliance with this criterion can be claimed.

B.4 The System Hazard to Which Software may Contribute will be Identified

An artefact entitled 'X Hazards' or even 'Hazard Log' does not exist in the ARP 4754A lifecycle process; rather the artefacts 'Platform/System/Item/Software Requirements' exist (in mitigation of the identified hazards, presumably). A trace from the Platform-level down through the design levels of abstraction to the Software level can be made. Following this trace, it can be assumed (but not explicitly proven) that the safety engineering activities and analyses identified potential hazardous states for which mitigating safety requirements are elicited and managed.

The contribution made by software to system hazards will ultimately be managed and mitigated through software safety requirements, so we start a deductive trace from this point. This is illustrated in Figure B.4.



Uniquely Identifiable
Identifiable as a requirement
Valid (extant and not redundant)
Traceable to arhitecture, source and to lower levels
Verifiable
Measurable and achievable
Measures of performance
Consistent with other requirements
Necessary
Factually correct
Amendable
Rationale provided
Atomic
Reflects the safety analyses

Independence
Probabilistic availability (typically handled through a budgeting process)
Integrity
No single failure criteria
Monitor performance
Safety or protective features
DALs
Operational and Maintenance Limitations
All failure conditions
Reliability
Availability
Fault Tolerance
Mitigation of unsafe design/design errors

Transition criteria.
OBJECTIVES:
1. High-level requirements are developed
2. Derived High-level requirements are defined and provided to the system processes, including the system safety assessment process

One or more level (i.e. high-level on low-level requirements).
That low-level requirements meet high-level requirements, and that the low-level requirements are accurate and consistent should be independently verified for SW levels A&B
Each system requirement that is allocated to software should be specified in the high-level requirements.
High-level requirements that address system requirements allocated to software to preclude system hazards should be defined. This should be independently verified for SW levels A&B
The high-level requirements should conform to the Software Requirements Standards, and be verifiable, traceable, and consistent.
The high-level requirements should be stated in quantitative terms with tolerances where applicable.
The high-level requirements should not describe design or verification detail except for specified and justified design constraints.
Data Requirements:
1. Description of the allocation of system requirements to SW, with attention to safety-related requirements and potential failure conditions
2. Functional and operational requirements under each mode of operation.
3. Performance criteria e.g. precision and accuracy.
4. Timing requirements and constraints.
5. Memory size constraints
6. HW and SW interfaces e.g. protocols, formats, frequency of inputs, and frequency of outputs.
7. Failure detection and safety monitoring requirements.
8. Partitioning requirements allocated to SW, how the partitioned SW components interact with each other, and the SW level(s) of each partition.

The system functional and interface requirements that are allocated to software should be analysed for ambiguities, inconsistencies, and undefined conditions.
Verification that high-level requirements are accurate and consistent should be independently verified for SW levels A&B

Data Requirements:
1. Detailed description of how the SW satisfies the specified high-level requirements, including algorithms, data structures, and how SW requirements are allocated to processors and tasks.
2. The description of the SW architecture defining the SW structure to implement the requirements.
3. The input/output description e.g. a data dictionary, both internally and externally throughout the SW architecture
4. The data flow and control flow of the design.
5. Resource limitations, the strategy for managing each resource and its limitations, the margins, and the method for measuring those margins e.g. timing and memory.
6. Scheduling procedures and inter-processing/inter-task communication mechanisms, including time-rigid sequencing, pre-emptive scheduling, Ada rendezvous, and interrupts.
7. Design methods and details for their implementation e.g. SW loading, user-modifiable SW, or multiple-version dissimilar SW.
8. Partitioning methods and means of preventing partition breaches.
9. Descriptions of the SW components, whether they are new/previously developed, and, if previously developed, reference to the baseline from which they were taken.
10. Derived requirements resulting from the SW design process.
11. If the system contains deactivated code, a description of the means to ensure that the code cannot be enabled in the target computer.
12. Rationale for those design decisions that are traceable to safety-related system requirements.

Conform to the Design Standards and be traceable, verifiable and be traceable.
Interfaces between software components, in the form of data flow and control flow, should be defined to be consistent between the components.
Control flow and data flow should be monitored when safety-related requirements dictate, for example, watchdog timers, reasonableness-checks, and cross-channel comparisons.
Responses to failure conditions should be consistent with the safety-related requirements

Figure B.4: ARP 4754A System Hazards to which Software may Contribute will be Identified

The artefact ‘SW Requirements’ (in the absence of a specific software safety requirements artefact) can be expected reasonably to be the single point of truth for software safety requirements that link deductively to system hazards, and inductively to the software design. This artefact is created by the activity ‘SW Requirements Process’, is updated during the activity ‘Structural Coverage Analysis Resolution’, and is an input to the activity ‘SW Design Process’. It is influenced by the artefact ‘Software Requirements Standard(s)’, and informs two other artefacts. As no activity is defined for these, the connecting lines are coloured red):

- Design Description

- SW Architecture.

Various quality attributes are defined for ‘SW Requirements’ (modelled as aspects), with the following being relevant to this criterion:

- High-level requirements allocated to software to preclude system hazards should be defined. This should be independently verified for software levels A and B
- Description of the allocation of system requirements to software, with attention to safety-related requirements and potential failure conditions
- Failure detection and safety monitoring requirements.

Although safety requirements are considered here, it is not robust enough to argue a link to the hazards to which software contributes. To identify the hazards to which the software contributes, we must look to the activity that produces the ‘SW Requirements’ – ‘SW Requirements Process’. This activity has the following controlling aspects:

- Define the transition criteria (not established by the lifecycle)
- Ensure high-level requirements are developed (an objective)
- Ensure the high-level requirements are defined and provided to the system processes, including the system safety assessment process (an objective).

It is informed by (has inputs from):

- System Safety Requirements (artefact) as an iterative link (Note 18)
- System Requirements Identification (activity)
- System Requirements (an assumed artefact)
- SW Design Process (an iterative link between the 2 activities).

The ‘SW Requirements Process’ activity does not have a specific methodology that defines how software requirements are to be elicited, nor does it have a time constraint for completion, nor an indication of the resource(s) required to carry it out.

To deductively trace to system hazards, we must look to the inputs to the activity ‘SW Requirements Process’. ‘System Requirements’ is an assumed artefact that is an input to the activity ‘SW Requirements Process’ (and others), and although the ARP makes frequent reference to ‘system requirements’ as a quality attribute required of many activities and artefacts, it does not positively assert an activity that produces them.

As an example, Section 2.2.2 notes "*The software life cycle processes analyze the system requirements allocated to software as part of the software requirements process*". It further states "*If such an analysis identifies any system requirements as inadequate or incorrect, the software life cycle processes should capture the issues and refer them to the system processes for resolution*" – but fails to specify what this 'analysis' is, or what such 'system processes' may be.

'System Safety Requirements' are produced by, and have iterative links with the activities:

- System Requirements Identification
- SW Requirements Process, and
- System Detailed Design Activity.

The trace to system hazard data must logically emanate from one of these activities. Each activity is assessed in turn, therefore.

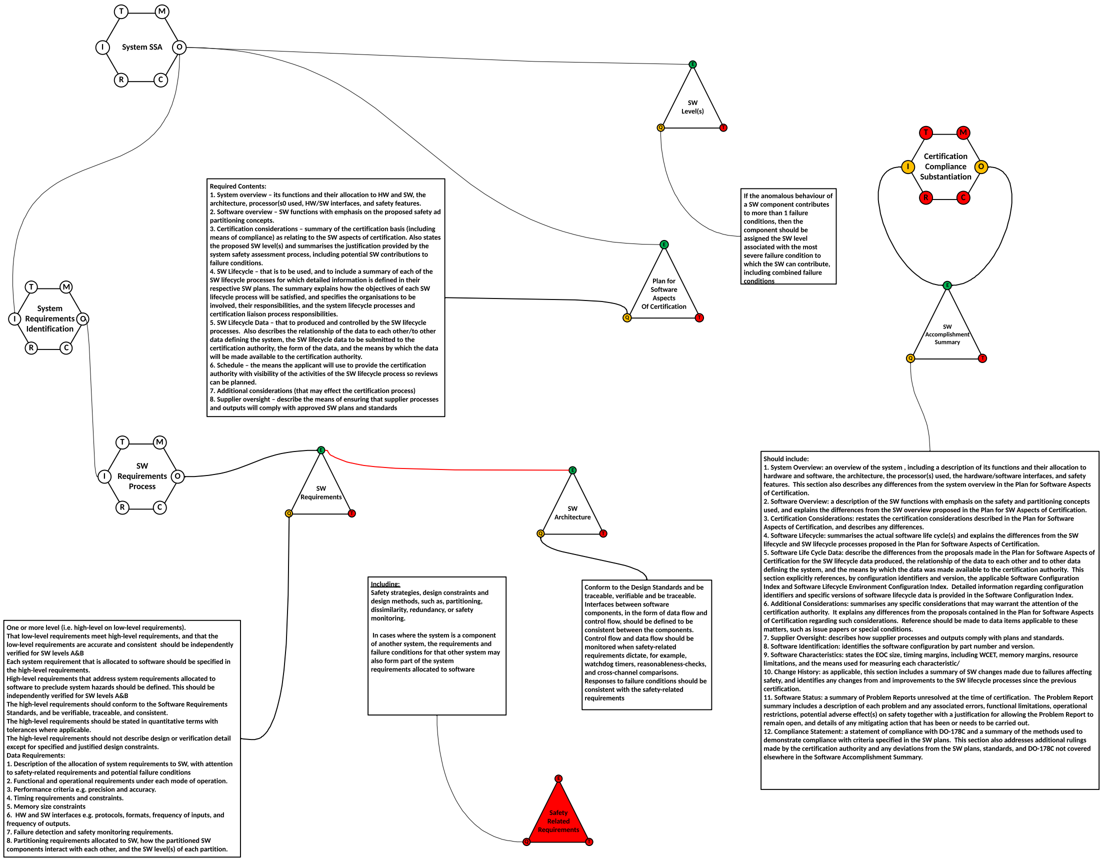
The inputs to 'System Requirements Identification' include:

- System SSA (activity)
- System Detailed Design Activity (an assumed activity)
- System CCA (activity)
- System FHA (activity)
- System PSSA (activity)
- Platform FHA (activity), and
- Platform Requirements Identification.

On the assumption that sister publications to ARP 4754A (such as ARP 4761) govern these safety activities, it is reasonable to accept that the identification of the system hazards (and corresponding mitigating safety requirements) are identified at the software boundary. As this is not explicitly stated (and requires assumptions based on a deductive trace), this criterion is only partially satisfied.

B.5 The Specific Failure Modes by Which Software Contributes to the Identified System Hazards will be Described

The modelling of this criterion is illustrated in Figure B.5, and whilst a trace from the system-level safety activities can be made to software requirements artefacts, there is no clear link between the system hazards and specific software failure modes.



Required Contents:
1. System overview – its functions and their allocation to HW and SW, the architecture, processor(s0 used, HW/SW interfaces, and safety features.
2. Software overview – SW functions with emphasis on the proposed safety ad partitioning concepts.
3. Certification considerations – summary of the certification basis (including means of compliance) as relating to the SW aspects of certification. Also states the proposed SW level(s) and summarises the justification provided by the system safety assessment process, including potential SW contributions to failure conditions.
4. SW Lifecycle – that is to be used, and to include a summary of each of the SW lifecycle processes for which detailed information is defined in their respective SW plans. The summary explains how the objectives of each SW lifecycle process will be satisfied, and specifies the organisations to be involved, their responsibilities, and the system lifecycle processes and certification liaison process responsibilities.
5. SW Lifecycle Data – that to produced and controlled by the SW lifecycle processes. Also describes the relationship of the data to each other/to other data defining the system, the SW lifecycle data to be submitted to the certification authority, the form of the data, and the means by which the data will be made available to the certification authority.
6. Schedule – the means the applicant will use to provide the certification authority with visibility of the activities of the SW lifecycle process so reviews can be planned.
7. Additional considerations (that may effect the certification process)
8. Supplier oversight – describe the means of ensuring that supplier processes and outputs will comply with approved SW plans and standards

If the anomalous behaviour of a SW component contributes to more than 1 failure conditions, then the component should be assigned the SW level associated with the most severe failure condition to which the SW can contribute, including combined failure conditions

Should include:
1. System Overview: an overview of the system , including a description of its functions and their allocation to hardware and software, the architecture, the processor(s) used, the hardware/software interfaces, and safety features. This section also describes any differences from the system overview in the Plan for Software Aspects of Certification.
2. Software Overview: a description of the SW functions with emphasis on the safety and partitioning concepts used, and explains the differences from the SW overview proposed in the Plan for SW Aspects of Certification.
3. Certification Considerations: restates the certification considerations described in the Plan for Software Aspects of Certification, and describes any differences.
4. Software Lifecycle: summarises the actual software life cycle(s) and explains the differences from the SW lifecycle and SW lifecycle processes proposed in the Plan for Software Aspects of Certification.
5. Software Life Cycle Data: describe the differences from the proposals made in the Plan for Software Aspects of Certification for the SW lifecycle data produced, the relationship of the data to each other and to other data defining the system, and the means by which the data was made available to the certification authority. This section explicitly references, by configuration identifiers and version, the applicable Software Configuration Index and Software Lifecycle Environment Configuration Index. Detailed information regarding configuration identifiers and specific versions of software lifecycle data is provided in the Software Configuration Index.
6. Additional Considerations: summarises any specific considerations that may warrant the attention of the certification authority. It explains any differences from the proposals contained in the Plan for Software Aspects of Certification regarding such considerations. Reference should be made to data items applicable to these matters, such as issue papers or special conditions.
7. Supplier Oversight: describes how supplier processes and outputs comply with plans and standards.
8. Software Identification: identifies the software configuration by part number and version.
9. Software Characteristics: states the EOC size, timing margins, including WCET, memory margins, resource limitations, and the means used for measuring each characteristic/
10. Change History: as applicable, this section includes a summary of SW changes made due to failures affecting safety, and identifies any changes from and improvements to the SW lifecycle processes since the previous certification.
11. Software Status: a summary of Problem Reports unresolved at the time of certification. The Problem Report summary includes a description of each problem and any associated errors, functional limitations, operational restrictions, potential adverse effect(s) on safety together with a justification for allowing the Problem Report to remain open, and details of any mitigating action that has been or needs to be carried out.
12. Compliance Statement: a statement of compliance with DO-178C and a summary of the methods used to demonstrate compliance with criteria specified in the SW plans. This section also addresses additional rulings made by the certification authority and any deviations from the SW plans, standards, and DO-178C not covered elsewhere in the Software Accomplishment Summary.

Including:
Safety strategies, design constraints and design methods, such as, partitioning, dissimilarity, redundancy, or safety monitoring.

In cases where the system is a component of another system, the requirements and failure conditions for that other system may also form part of the system requirements allocated to software

Conform to the Design Standards and be traceable, verifiable and be traceable.
Interfaces between software components, in the form of data flow and control flow, should be defined to be consistent between the components.
Control flow and data flow should be monitored when safety-related requirements dictate, for example, watchdog timers, reasonableness-checks, and cross-channel comparisons.
Responses to failure conditions should be consistent with the safety-related requirements

One or more level (i.e. high-level on low-level requirements).
That low-level requirements meet high-level requirements, and that the low-level requirements are accurate and consistent should be independently verified for SW levels A&B
Each system requirement that is allocated to software should be specified in the high-level requirements.
High-level requirements that address system requirements allocated to software to preclude system hazards should be defined. This should be independently verified for SW levels A&B
The high-level requirements should conform to the Software Requirements Standards, and be verifiable, traceable, and consistent.
The high-level requirements should be stated in quantitative terms with tolerances where applicable.
The high-level requirements should not describe design or verification detail except for specified and justified design constraints.
Data Requirements:
1. Description of the allocation of system requirements to SW, with attention to safety-related requirements and potential failure conditions
2. Functional and operational requirements under each mode of operation.
3. Performance criteria e.g. precision and accuracy.
4. Timing requirements and constraints.
5. Memory size constraints
6. HW and SW interfaces e.g. protocols, formats, frequency of inputs, and frequency of outputs.
7. Failure detection and safety monitoring requirements.
8. Partitioning requirements allocated to SW, how the partitioned SW components interact with each other, and the SW level(s) of each partition.

Figure B.5: ARP 4754A specific Software Failure Modes

It is important to note that although there is an artefact entitled 'Safety Related Requirements' this is an inferred artefact that has no activity to produce it, yet it must serve as an input to other activities for them to be fulfilled.

It is already known that an artefact containing specific software safety requirements is not produced by this lifecycle, and our assessment therefore focussed initially on the artefact 'SW Requirements'.

'SW Requirements' are produced by the activity 'SW Requirements Process' and have the following quality attributes:

- Each system requirement that is allocated to software should be specified in the high-level requirements
- High level requirements that address system requirements allocated to software to preclude system hazards should be identified. This should be independently verified for Software Levels A and B
- A description of the allocation of system requirements to software, with attention to safety-related requirements and potential failure conditions

- Failure detection and safety monitoring requirements.

Whilst this may be argued to assert the system hazards that software contributes to (including performance criteria such as precision and accuracy), it falls short of describing the specific failure modes by which the software contributes to system hazards.

The artefact 'SW Requirements' is linked to the artefact 'SW Architecture', but as there is no activity that links these two artefacts, the connecting line is coloured red. 'SW Architecture' has the following relevant quality attributes:

- Control flow and data flow should be monitored when safety-related requirements dictate, for example, watchdog timers, reasonable-checks, and cross-channel comparisons
- Responses to failure conditions should be consistent with the safety-related requirements.

These attributes suggest a form of mitigation to specific failure modes (at the software boundary), but it is not clear how these failure modes and the system hazards they contribute to are identified. As such we must look to whether relevant hazard and failure data can be traced to the architecture (via the artefact 'SW Requirements').

'SW Requirements' are produced by the activity 'SW Requirements Process', which is informed by the activity 'System Requirements Identification' – which itself is informed by the activity 'System SSA'. Whilst this provides assurance that the system hazard data will flow deductively to the software boundary, the activities or analyses that identify the specific software failure modes are not defined.

'System SSA' produces the artefact 'SW Levels', but the only pertinent quality attribute considers the requirement for Software Levels – should the component contribute to more than one failure mode. This assumes the failure modes will be identified, but not how.

The only other artefact in the lifecycle process that considers this criterion is the 'SW Accomplishment Summary', which requires:

- A system overview that includes safety features
- A software overview. . . with emphasis on the safety and partitioning concepts used
- Software characteristics. . . including Worst Case Execution Time.

However, this is produced towards the end of design lifecycle as an assurance and certification artefact.

As the means of identifying the software failure modes are not explicitly stated (and requires assumptions based on a deductive trace), this criterion is only partially satisfied.

B.6 The Software Contribution to the Identified System Hazards will be Acceptably Managed Through the Elicitation of Software Safety Requirements Which Specify the Required Behaviour(s); for each Identified Software Contribution, to each System Hazard

We have already assessed that software safety requirements are not specifically created by this lifecycle. Whilst some aspects could be inferred, the elicitation of software safety requirements cannot be assured with any degree of confidence. ARP 4754A is therefore non-compliant with this criterion.

B.7 All Software Safety Requirements will be Atomic, Unambiguous, Defined in Sufficient Detail, and Verifiable

In the absence of any software safety requirements, this criterion cannot be held to be met.

Appendix C

JB61834 Assessment Against Principle 1

The as-required (Closed) practice of the JB61834 project was assessed against the first principle of the as-desired model, and the findings of this assessment are presented in this Appendix.

Principle 1 requires that the lifecycle process under assessment ensures that:

- A clear description of the software in the system will be provided
- The operating context of the system in which the software resides will be described
- A clear description of the system in which the software resides will be provided
- The system hazards to which software may contribute will be identified
- The specific failure modes by which software contributes to the identified system hazards will be described
- The software contribution to the identified system hazards will be acceptably managed through the elicitation of software safety requirements that specify the required behaviour(s); for each identified software contribution, to each system hazard
- All software safety requirements will be atomic, unambiguous, defined in sufficient detail, and verifiable.

Further, to comply with the 'plus one' Principle, each Principle must ensure:

- The required confidence behind the attainment of each Principle (1 to 4) will be determined

- The most effort in generating evidence will be focussed on the areas with the highest risk from software's contribution to hazards (noting that the areas denoted as requiring the most effort are currently indicated in Open Standards by the notions of integrity/assurance levels)
- For each Principle (1 to 4), the required confidence that each has been met will be reflected in:

  (a) The appropriateness of evidence

  (b) The trustworthiness of each evidential artefact (the rigour in the approaches to be used):

    (i) Independence

    (ii) Resources and required attributes (personnel)

    (iii) Techniques and Methods used

    (iv) Audits and reviews

    (v) Tools

  (c) The type of evidence to be used.
- An understanding of the limitations with each type of evidence being used will be clearly understood.

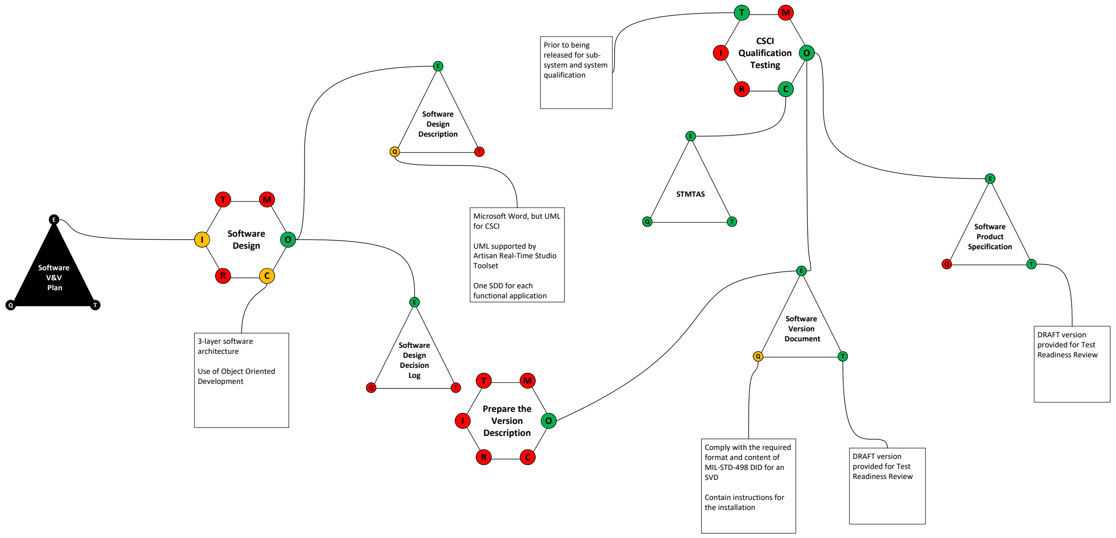
Taking each criterion in turn, the levels of compliance of the JB61834 lifecycle is assessed. The complete representation of JB61834's lifecycle is not included here owing to its size. The full model of JB61834's assessment against the as-desired criteria can be found at [114].

C.1 A Clear Definition of Software in the System

As illustrated in Figure C.1, there are four artefacts which could be reasonably expected to contain a clear definition of software within the system. This description could reside in any or all of the following:

1. Software Design Description
2. Software Design Decision Log
3. Software Version Document
4. Software Product Specification.

The required quality attributes for these artefacts however, are either not stated (with the exception of 'Time' which is stated for the artefacts Software Product Specification, and Software Design Decision Log), or only stipulate the


Software V&V Plan
E
Q
T
Software Design
T
M
I
O
R
C
3-layer software architecture

Use of Object Oriented Development
Software Design Description
E
Q
T
Microsoft Word, but UML for CSCI

UML supported by Artisan Real-Time Studio Toolset

One SDD for each functional application
Software Design Decision Log
E
Q
T
Prepare the Version Description
T
M
I
O
R
C
Prior to being released for sub-system and system qualification
CSCI Qualification Testing
T
M
I
O
R
C
STMTAS
E
Q
T
Software Version Document
E
Q
T
Comply with the required format and content of MIL-STD-498 DID for an SVD

Contain instructions for the installation
DRAFT version provided for Test Readiness Review
Software Product Specification
E
Q
T
DRAFT version provided for Test Readiness Review

Figure C.1: JB61834 Software Description

required format (for the artefacts Software Version Document, and Software Design Description).

The three activities that produce the four artefacts which could reasonably be expected to contain a clear definition of software within the system are:

- For the Software Decision Log - 'Software Design'
- For the Software Design Description - 'Software Design'
- For the Software Version Document - 'Prepare the Version Description' and 'CSCI Qualification Testing'
- For the Software Product Specification - not stated; although 'CSCI Qualification Testing' updates the specification on completion.

Taking each activity in turn, the aspects that denote the required quality criteria are assessed.

Software Design

1. **Inputs:** there is a single input to this activity which is a document (Software V&V Plan). This artefact alone would not allow the software design process to proceed, as there are no requirement specifications on which to undertake a design activity.
2. **Time:** when this activity is to be completed by/start is not stated.
3. **Method:** no technique or method by which the activity is undertaken is stated.

4. **Output:** the activity produces the required artefacts.
5. **Control:** this is partially satisfied by requiring a 3-layered architecture and the use of Object Oriented Design
6. **Resource:** no indication is made as to the consumed resources, nor personnel required.

CSCI Qualification Testing

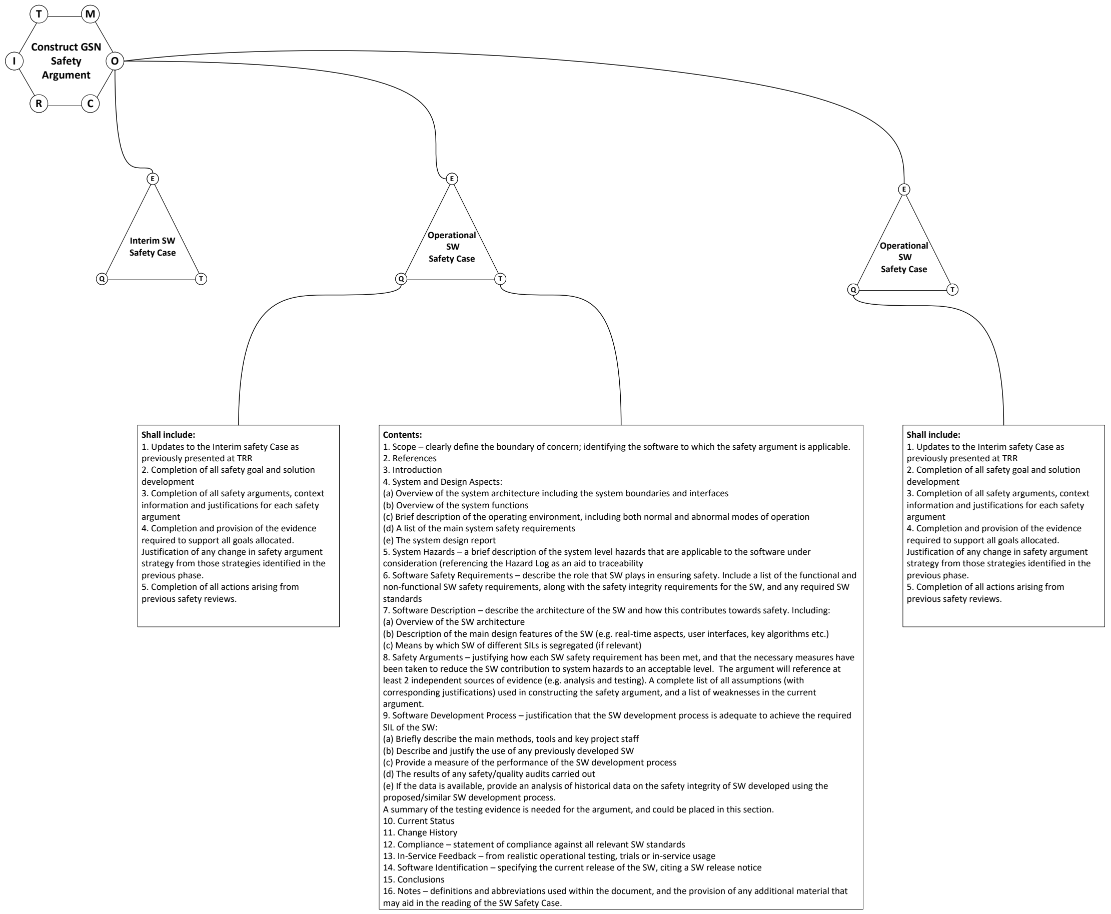
1. **Inputs:** none stated.
2. **Time:** given as "prior to being released for sub-system and system qualification".
3. **Method:** no technique or method by which the activity is undertaken is stated.
4. **Output:** the activity produces the required artefacts.
5. **Control:** satisfied by the STMTAS.
6. **Resource:** no indication is made as to the consumed resources, nor personnel required.

Prepare the Version Description

1. **Inputs:** none stated.
2. **Time:** when this activity is to be completed by/start is not stated.
3. **Method:** no technique or method by which the activity is undertaken is stated.
4. **Output:** the activity produces the required artefacts.
5. **Control:** not controlling aspects are stated.
6. **Resource:** no indication is made as to the consumed resources, nor personnel required.

Whilst it is possible to identify activities that produce these artefacts, it cannot be shown compellingly how they are produced, by whom, nor to what standard. In addition, the aspects of the produced artefacts are not considered sufficiently to argue over the first criterion of Principle 1 being met.

There is an activity entitled ‘Software Development’in the JB61834 lifecycle, and although similar in title to ‘designing’ the software, it also lacks any defined inputs.


T
M
I
Construct GSN Safety Argument
O
R
C
E
Interim SW Safety Case
Q
T
E
Operational SW Safety Case
Q
T
E
Operational SW Safety Case
Q
T

Shall include:
1. Updates to the Interim safety Case as previously presented at TRR
2. Completion of all safety goal and solution development
3. Completion of all safety arguments, context information and justifications for each safety argument
4. Completion and provision of the evidence required to support all goals allocated.
Justification of any change in safety argument strategy from those strategies identified in the previous phase.
5. Completion of all actions arising from previous safety reviews.

Contents:
1. Scope – clearly define the boundary of concern; identifying the software to which the safety argument is applicable.
2. References
3. Introduction
4. System and Design Aspects:
(a) Overview of the system architecture including the system boundaries and interfaces
(b) Overview of the system functions
(c) Brief description of the operating environment, including both normal and abnormal modes of operation
(d) A list of the main system safety requirements
(e) The system design report
5. System Hazards – a brief description of the system level hazards that are applicable to the software under consideration (referencing the Hazard Log as an aid to traceability
6. Software Safety Requirements – describe the role that SW plays in ensuring safety. Include a list of the functional and non-functional SW safety requirements, along with the safety integrity requirements for the SW, and any required SW standards
7. Software Description – describe the architecture of the SW and how this contributes towards safety. Including:
(a) Overview of the SW architecture
(b) Description of the main design features of the SW (e.g. real-time aspects, user interfaces, key algorithms etc.)
(c) Means by which SW of different SILs is segregated (if relevant)
8. Safety Arguments – justifying how each SW safety requirement has been met, and that the necessary measures have been taken to reduce the SW contribution to system hazards to an acceptable level. The argument will reference at least 2 independent sources of evidence (e.g. analysis and testing). A complete list of all assumptions (with corresponding justifications) used in constructing the safety argument, and a list of weaknesses in the current argument.
9. Software Development Process – justification that the SW development process is adequate to achieve the required SIL of the SW:
(a) Briefly describe the main methods, tools and key project staff
(b) Describe and justify the use of any previously developed SW
(c) Provide a measure of the performance of the SW development process
(d) The results of any safety/quality audits carried out
(e) If the data is available, provide an analysis of historical data on the safety integrity of SW developed using the proposed/similar SW development process.
A summary of the testing evidence is needed for the argument, and could be placed in this section.
10. Current Status
11. Change History
12. Compliance – statement of compliance against all relevant SW standards
13. In-Service Feedback – from realistic operational testing, trials or in-service usage
14. Software Identification – specifying the current release of the SW, citing a SW release notice
15. Conclusions
16. Notes – definitions and abbreviations used within the document, and the provision of any additional material that may aid in the reading of the SW Safety Case.

Shall include:
1. Updates to the Interim safety Case as previously presented at TRR
2. Completion of all safety goal and solution development
3. Completion of all safety arguments, context information and justifications for each safety argument
4. Completion and provision of the evidence required to support all goals allocated.
Justification of any change in safety argument strategy from those strategies identified in the previous phase.
5. Completion of all actions arising from previous safety reviews.

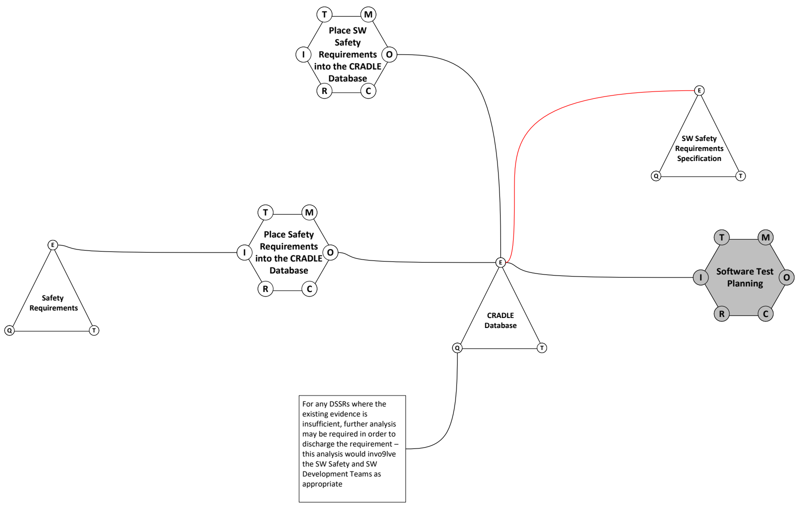
Figure C.2: JB61834 Software Safety Cases

Examination of the Software Safety lifecycle of JB61834 identifies three Safety Cases that are created, and these are shown (in an abridged form) in Figure C.2. As a safety case should contain or reference a clear definition of software within a system, these safety cases were assessed. The safety cases are:

1. Preliminary Software Safety Case
2. Interim Software Safety Case
3. Operational Software Safety Case.

Analysis of the lifecycle activities that produce the safety cases reveals that for these safety cases to be completed, the following attributes are required:

- Scope - clearly identifying the software to which the safety argument is applicable
- Software Safety Requirements - describe the role that software plays in ensuring safety. Include a list of the functional and non-functional SW safety requirements, along with the safety integrity requirements for the software, and any required software standards


T
M
Place SW Safety Requirements into the CRADLE Database
I
O
R
C
E
SW Safety Requirements Specification
Q
T
T
M
Place Safety Requirements into the CRADLE Database
I
O
R
C
E
Safety Requirements
Q
T
E
CRADLE Database
Q
T
T
M
Software Test Planning
I
O
R
C

For any DSSRs where the existing evidence is insufficient, further analysis may be required in order to discharge the requirement – this analysis would invo9lve the SW Safety and SW Development Teams as appropriate

Figure C.3: JB61834 Requirements Database

- Software Description – describe the architecture of the SW and how this contributes towards safety. Including:

  (a) Overview of the software architecture

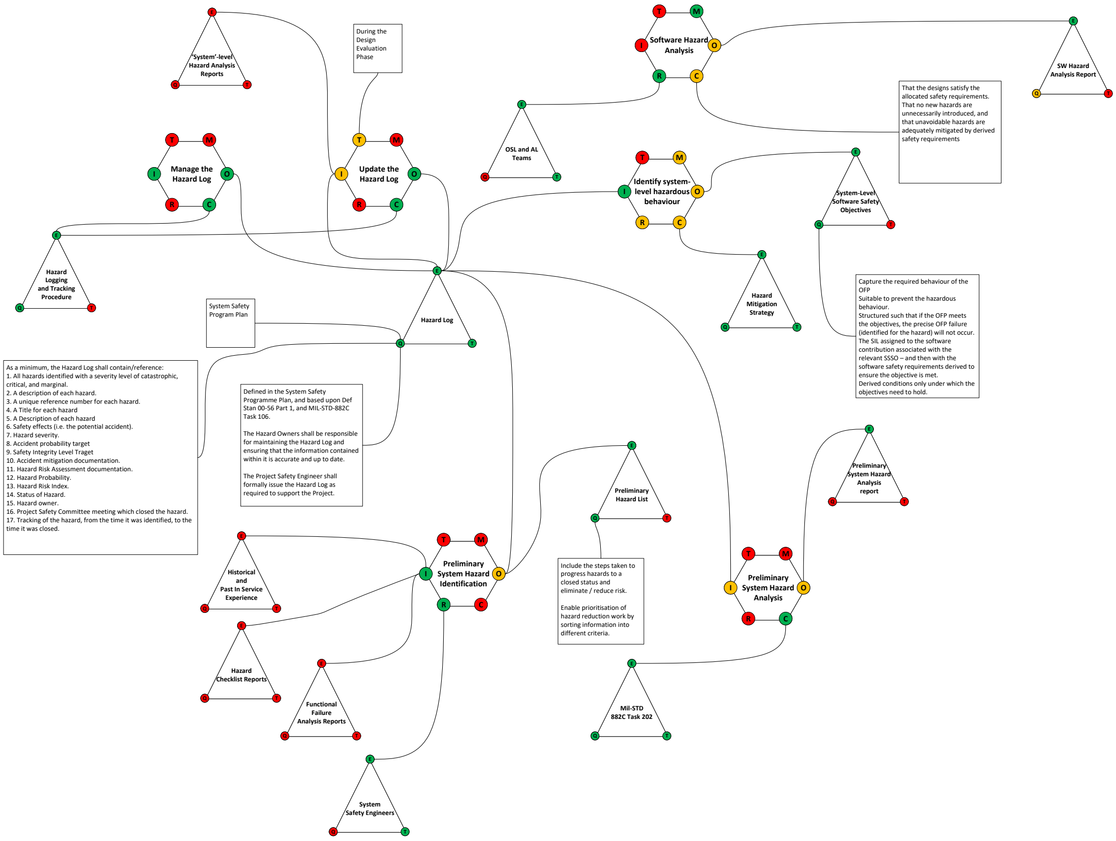
  (b) Description of the main design features of the software (e.g. real-time aspects, user interfaces, key algorithms etc.)

  (c) Means by which SW of different SILs is segregated (if relevant)

- Safety Arguments – justifying how each software safety requirement has been met, and that the necessary measures have been taken to reduce the software contribution to system hazards to an acceptable level

Although a trace of activities and artefacts cannot be etablished for the data to be provided within the safety cases, in order for the software safety cases to have been satisfactorily completed and accepted, such data must have been provided. We can also determine that there is a requirements database ('CRADLE'), whose inputs are shown in Figure C.3. Figure C.3 shows that platform and software safety requirements are placed in the database (and presumably the system safety requirements too), and that software safety requirements are derived from it. As there is no consuming activity that produces the software safety requirements specification, the linking line between these two artefacts is coloured red.

Whilst the JB61834 as-required (Closed) model does not establish explicit activities required to produce a clear definition of the software in the system(s), it is possible to assume that this information is created - largely relying on the required quality attributes defined by the aspects of the software safety cases. There cannot be a high confidence in this owing to the lack of clearly defined activities, however. As such this first criterion is partially met, with a low degree of confidence.


'System'-level Hazard Analysis Reports
Manage the Hazard Log
Update the Hazard Log
Software Hazard Analysis
SW Hazard Analysis Report
OSL and AL Teams
Identify system-level hazardous behaviour
System-Level Software Safety Objectives
Hazard Logging and Tracking Procedure
Hazard Log
Hazard Mitigation Strategy
Preliminary Hazard List
Preliminary System Hazard Analysis report
Historical and Past In Service Experience
Preliminary System Hazard Identification
Preliminary System Hazard Analysis
Hazard Checklist Reports
Functional Failure Analysis Reports
Mil-STD 882C Task 202
System Safety Engineers
T
M
I
O
R
C
E
Q

During the Design Evaluation Phase

That the designs satisfy the allocated safety requirements. That no new hazards are unnecessarily introduced, and that unavoidable hazards are adequately mitigated by derived safety requirements

System Safety Program Plan

Capture the required behaviour of the OFP
Suitable to prevent the hazardous behaviour.
Structured such that if the OFP meets the objectives, the precise OFP failure (identified for the hazard) will not occur.
The SIL assigned to the software contribution associated with the relevant SSSO – and then with the software safety requirements derived to ensure the objective is met.
Derived conditions only under which the objectives need to hold.

As a minimum, the Hazard Log shall contain/reference:
1. All hazards identified with a severity level of catastrophic, critical, and marginal.
2. A description of each hazard.
3. A unique reference number for each hazard.
4. A Title for each hazard
5. A Description of each hazard
6. Safety effects (i.e. the potential accident).
7. Hazard severity.
8. Accident probability target
9. Safety Integrity Level Traget
10. Accident mitigation documentation.
11. Hazard Risk Assessment documentation.
12. Hazard Probability.
13. Hazard Risk Index.
14. Status of Hazard.
15. Hazard owner.
16. Project Safety Committee meeting which closed the hazard.
17. Tracking of the hazard, from the time it was identified, to the time it was closed.

Defined in the System Safety Programme Plan, and based upon Def Stan 00-56 Part 1, and MIL-STD-882C Task 106.

The Hazard Owners shall be responsible for maintaining the Hazard Log and ensuring that the information contained within it is accurate and up to date.

The Project Safety Engineer shall formally issue the Hazard Log as required to support the Project.

Include the steps taken to progress hazards to a closed status and eliminate / reduce risk.

Enable prioritisation of hazard reduction work by sorting information into different criteria.

Figure C.4: JB61834 System Hazards

C.2 The System Hazards to Which Software may Contribute will be Identified

As illustrated in Figure C.4, there are two artefacts at the software level of design which could reasonably be expected to contain data concerning the system hazards to which software contributes:

1. System-Level Software Safety Objectives
2. SW Hazard Analysis Report.

Whilst it can be established that these two artefacts are produced by explicitly-stated activities ('Identify System-level Hazardous Behaviour' and 'Software Hazard Analysis' respectively), a traceable link between these activities and those undertaken at the system level cannot be established (with the exception of the artefact 'Hazard Log' being an input to the software activity 'Identify System-level Hazardous Behaviour').

The system-level artefacts which cannot be traced to the software-level, but which contain system-level hazard data are:

- Preliminary Hazard List, and

- Preliminary System Hazard Analysis Report.

To meet this criterion therefore, the Hazard Log is the single repository that must bridge the gap between system hazards and the contribution made by software. Examination of the quality attributes of the Hazard Log reveals that no specific software attributes, nor any requirements for the specification of a causal failure path are stated, however.

Whilst is is reasonable to infer that the activity ‘Identify System-level Hazardous Behaviour’ will extract the system-level data and assess the causal path and reveal software’s contribution, the activity does not feed back to the Hazard Log, and there are deficiencies regarding the aspects of this activity. The deficiencies are seen by considering the aspects of the activity:

- **Time:** no consideration as to when the activity is to start/complete
- **Method:** Whilst it is noted (not shown in Figure C.4) that the Fault Trees are to be analysed to identify ’behaviour which may contribute to a system hazard’, this only partially satisfies this attribute (as system behaviour is but one contribution)
- **Outputs:** Only the ‘System-level Software Safety Objectives’ are noted to be created by this activity, and having identified such behaviour, feedback to the platform-level would be expected
- **Control:** Although the activity will be influenced by the strategy, this artefact is not sufficient in isolation (as a mitigation strategy will not control the identification of behaviours)
- **Resource:** Only a Pilot (not shown) is noted as being required for this activity. It is asserted that at least software safety and design engineers will be required.

Although there can be a medium-level of confidence that system-level hazards will be identified, it can only be inferred that the Hazard Log will be the singly repository for establishing the software contribution to these hazards. As this is not compellingly demonstrated, this criterion cannot be supported.

C.3 The Specific Failure Modes by Which Software Contributes to the Identified System Hazards will be Described

As illustrated in Figure C.5, there is no single artefact which contains a dedicated list of the failure modes by which software contributes to the identified system

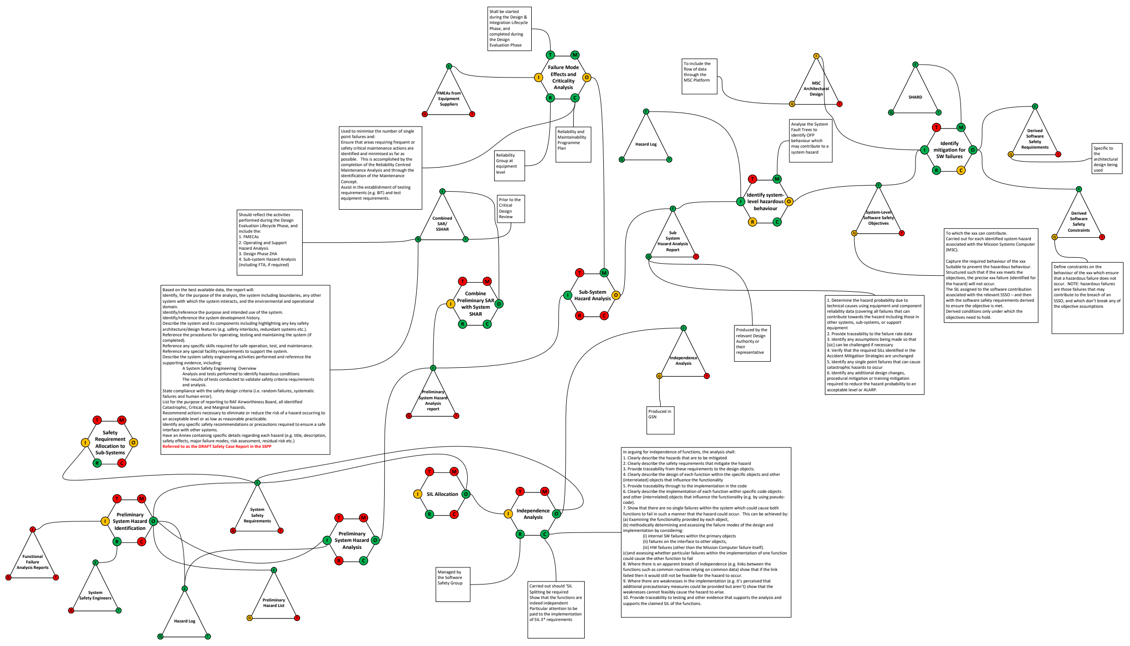

Shall be started during the Design & Integration Lifecycle Phase, and completed during the Design Evaluation Phase
Failure Mode Effects and Criticality Analysis
FMEAs from Equipment Suppliers
To include the flow of data through the MSC Platform
MSC Architectural Design
SHARD
Identify mitigation for SW failures
Derived Software Safety Requirements
Specific to the architectural design being used
Used to minimise the number of single point failures and:
Ensure that areas requiring frequent or safety critical maintenance actions are identified and minimised as far as possible. This is accomplished by the completion of the Reliability Centred Maintenance Analysis and through the identification of the Maintenance Concept.
Assist in the establishment of testing requirements (e.g. BIT) and test equipment requirements.
Reliability and Maintainability Programme Plan
Reliability Group at equipment level
Hazard Log
Analyse the System Fault Trees to identify OFP behaviour which may contribute to a system hazard
Identify system-level hazardous behaviour
System-Level Software Safety Objectives
Derived Software Safety Constraints
Should reflect the activities performed during the Design Evaluation Lifecycle Phase, and include the:
1. FMECAs
2. Operating and Support Hazard Analysis
3. Design Phase ZHA
4. Sub-system Hazard Analysis (including FTA, if required)
Combined SAR/ SSHAR
Prior to the Critical Design Review
Sub System Hazard Analysis Report
To which the xxx can contribute.
Carried out for each identified system hazard associated with the Mission Systems Computer (MSC).
Capture the required behaviour of the xxx Suitable to prevent the hazardous behaviour.
Structured such that if the xxx meets the objectives, the precise xxx failure (identified for the hazard) will not occur.
The SIL assigned to the software contribution associated with the relevant SSSO – and then with the software safety requirements derived to ensure the objective is met.
Derived conditions only under which the objectives need to hold.
Define constraints on the behaviour of the xxx which ensure that a hazardous failure does not occur. NOTE: hazardous failures are those failures that may contribute to the breach of an SSSO, and which don't break any of the objective assumptions
Based on the best available data, the report will:
Identify, for the purpose of the analysis, the system including boundaries, any other system with which the system interacts, and the environmental and operational domain.
Identify/reference the purpose and intended use of the system.
Identify/reference the system development history.
Describe the system and its components including highlighting any key safety architecture/design features (e.g. safety interlocks, redundant systems etc.).
Reference the procedures for operating, testing and maintaining the system (if completed).
Reference any specific skills required for safe operation, test, and maintenance.
Reference any special facility requirements to support the system.
Describe the system safety engineering activities performed and reference the supporting evidence, including:
A System Safety Engineering Overview
Analysis and tests performed to identify hazardous conditions
The results of tests conducted to validate safety criteria requirements and analysis.
State compliance with the safety design criteria (i.e. random failures, systematic failures and human error).
List for the purpose of reporting to RAF Airworthiness Board, all identified Catastrophic, Critical, and Marginal hazards.
Recommend actions necessary to eliminate or reduce the risk of a hazard occurring to an acceptable level or as low as reasonable practicable.
Identify any specific safety recommendations or precautions required to ensure a safe interface with other systems.
Have an Annex containing specific details regarding each hazard (e.g. title, description, safety effects, major failure modes, risk assessment, residual risk etc.)
Referred to as the DRAFT Safety Case Report in the SSPP
Combine Preliminary SAR with System SHAR
Sub-System Hazard Analysis
Produced by the relevant Design Authority or their representative
1. Determine the hazard probability due to technical causes using equipment and component reliability data (covering all failures that can contribute towards the hazard including those in other systems, sub-systems, or support equipment
2. Provide traceability to the failure rate data
3. Identify any assumptions being made so that [sic] can be challenged if necessary
4. Verify that the required SILs identified in the Accident Mitigation Strategies are unchanged
5. Identify any single point failures that can cause catastrophic hazards to occur
6. Identify any additional design changes, procedural mitigation or training mitigation required to reduce the hazard probability to an acceptable level or ALARP.
Independence Analysis
Preliminary System Hazard Analysis report
Produced in GSN
Safety Requirement Allocation to Sub-Systems
In arguing for independence of functions, the analysis shall:
1. Clearly describe the hazards that are to be mitigated
2. Clearly describe the safety requirements that mitigate the hazard
3. Provide traceability from these requirements to the design objects.
4. Clearly describe the design of each function within the specific objects and other (interrelated) objects that influence the functionality
5. Provide traceability through to the implementation in the code
6. Clearly describe the implementation of each function within specific code objects and other (interrelated) objects that influence the functionality (e.g. by using pseudo-code).
7. Show that there are no single failures within the system which could cause both functions to fail in such a manner that the hazard could occur. This can be achieved by:
(a) Examining the functionality provided by each object,
(b) methodically determining and assessing the failure modes of the design and implementation by considering:
(i) internal SW failures within the primary objects
(ii) failures on the interface to other objects,
(iii) HW failures (other than the Mission Computer failure itself).
(c)and assessing whether particular failures within the implementation of one function could cause the other function to fail
8. Where there is an apparent breach of independence (e.g. links between the functions such as common routines relying on common data) show that if the link failed then it would still not be feasible for the hazard to occur.
9. Where there are weaknesses in the implementation (e.g. it's perceived that additional precautionary measures could be provided but aren't) show that the weaknesses cannot feasibly cause the hazard to arise.
10. Provide traceability to testing and other evidence that supports the analysis and supports the claimed SIL of the functions.
SIL Allocation
Independence Analysis
Preliminary System Hazard Identification
System Safety Requirements
Preliminary System Hazard Analysis
Functional Failure Analysis Reports
System Safety Engineers
Hazard Log
Preliminary Hazard List
Managed by the Software Safety Group
Carried out should 'SIL Splitting be required
Show that the functions are indeed independent
Particular attention to be paid to the implementation of SIL 3* requirements

Figure C.5: JB61834 Software Failure Contribution to Hazards

hazards. It is possible to trace some of this required data through to the artefact 'System-Level Software Safety Objectives' - an artefact that informs the activity 'Identify Mitigation for SW Failures', however.

Figure C.5 is not the complete model of this criterion, as to show all artefacts, activities and the links between them would be too large to distill down to a readable figure for this Appendix. The colour coding used does represent the strength of aspects from analysis of the full model with regard to this criterion, however.

According to the project artefacts, the artefact 'System-level Software Safety Objectives' must:

- Have objectives for each identified system hazard associated with the product
- Capture the product's required behaviours - suitable to prevent the hazardous behaviour, and
- Assert the derived conditions only under which the objectives need to hold.

The activity which produces the artefact 'System-level Software Safety Objectives' is 'Identify System-level Hazardous Behaviour'. Considering whether there is sufficient data to undertake this activity, it can be seen from Figure C.5 that both the architectural design and the 'Sub-system Hazard Analysis Report' are sufficient inputs for this activity to proceed.

The 'Sub-System hazard Analysis Report' (inter alia):

- Determines the hazard probabilities due to technical causes using equipment and component reliability data
- Provides traceability to the failure rate data
- Identifies any assumptions
- Identifies any single points of failure.

The 'Sub-system Hazard Analysis Report' is created by the activity 'Sub-System Hazardous Analysis', which:

- Is carried out by undertaking a FMECA
- Has the 'Preliminary System hazard Analysis Report' as an input, which is created by an activity which has the required inputs to inform it ('Preliminary System Hazard Analysis').

The 'Preliminary System Hazard Analysis is created by the activity 'Preliminary System Hazard Analysis', which has inputs from the 'HAZID' and the 'Hazard Log'.

It is possible to trace pertinent hazard data from the 'Platform' level through the layers of design abstraction to the software boundary, and this involves numerous activities which use the 'Hazard Log' as a source of frequently updated hazard data. There are two instances of data that would also be required to meet this criterion, but which cannot be traced to the artefact 'System-level Software Safety Objectives', however.

The first instance concerns independence analysis. The activity 'Independence Analysis' produces an artefact of the same name, and as a control, when arguing for the independence of (software) functions the activity and associated artefact must (inter alia):

- Clearly describe the hazards to be mitigated
- Clearly describe the safety requirements to mitigate the hazards
- Show that there are no single points of failure within the system that could cause functions to fail in such a manner that the hazard(s) could occur.

The activity also iteratively informs the 'System Safety Requirements', and would contain vital data regarding how software could contribute to system hazards (a key data source for eliciting specific software failure modes). However, other than an inferred link to the 'Hazard Log' through other activities that contribute to the 'System Safety Requirements', no explicit trace can be made from either the 'System Safety Requirements' or the 'Independence Analysis' to the 'System-level Software Safety Objectives'.

The second instance concerns the combination of Safety Assessment reports. There is an activity to 'Combine the Preliminary SAR with the System SAR', and the output of the preliminary system hazard analysis is a defined input to this, along with the 'Preliminary SAR', and the 'Sub-System Hazard Analysis'.

The artefact 'Combined SAR/SSHAR' has attributes that state it must (inter alia):

- Include the FMECAs
- Describe the system and its components, including highlighting any key safety architecture/design features
- Recommend any actions necessary to eliminate or reduce the risk of a hazard occurring to an acceptable level, or as low as reasonably practicable
- Identify any specific safety recommendations or precautions required to ensure a safe interface with other systems.

No explicit trace can be made to the 'System-level Software Safety Objectives', however. It is only the iterative inputs to and outputs from the 'Hazard Log' that are capable of eliciting the specific failure modes which software contributes to. It could be argued that this specific data will be elicited from the FTAs used to analyse system safety, or from the FMECAs, yet the former does not specifically argue for this criterion, and the latter is informed by supplier data only, and is not performed by the safety team.

As such only partial compliance can be argued for this criteria, with a medium-level of confidence.

C.4 The Software Contribution to the Identified System Hazards will be Acceptably Managed Through the Elicitation of Software Safety Requirements that Specify the Required Behaviours; for each Identified Software Contribution, for Each System Hazard

It is key to note at the start of this analysis the partial compliance with the previous criteria, which will already hinder the ability of the lifecycle to meet this criterion.

Figure C.6 is not the complete model of this criterion, as to show all artefacts, activities and the links between them would be too large to distill down to a readable figure for this Appendix. The colour coding used does represent the

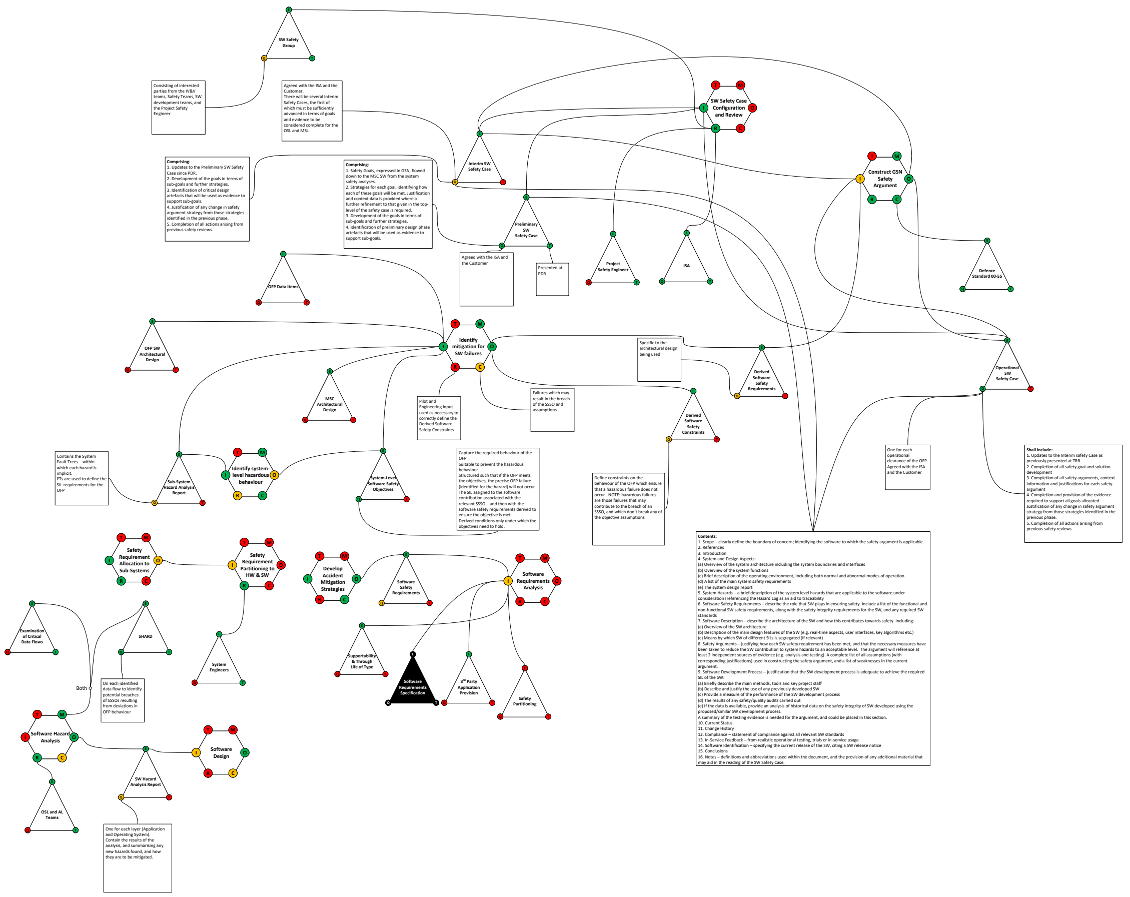

SW Safety Group
Consisting of interested parties from the IV&V teams, Safety Teams, SW development teams, and the Project Safety Engineer
Agreed with the ISA and the Customer. There will be several Interim Safety Cases, the first of which must be sufficiently advanced in terms of goals and evidence to be considered complete for the OSL and MSL.
SW Safety Case Configuration and Review
Interim SW Safety Case
Comprising: 1. Updates to the Preliminary SW Safety Case since PDR. 2. Development of the goals in terms of sub-goals and further strategies. 3. Identification of critical design artefacts that will be used as evidence to support sub-goals. 4. Justification of any change in safety argument strategy from those strategies identified in the previous phase. 5. Completion of all actions arising from previous safety reviews.
Comprising: 1. Safety Goals, expressed in GSN, flowed down to the MSC SW from the system safety analyses. 2. Strategies for each goal, identifying how each of these goals will be met. Justification and context data is provided where a further refinement to that given in the top-level of the safety case is required. 3. Development of the goals in terms of sub-goals and further strategies. 4. Identification of preliminary design phase artefacts that will be used as evidence to support sub-goals.
Construct GSN Safety Argument
Preliminary SW Safety Case
Agreed with the ISA and the Customer
Presented at PDR
Project Safety Engineer
ISA
Defence Standard 00-55
OFP Data Items
OFP SW Architectural Design
Identify mitigation for SW failures
Specific to the architectural design being used
Derived Software Safety Requirements
Operational SW Safety Case
MSC Architectural Design
Pilot and Engineering input used as necessary to correctly define the Derived Software Safety Constraints
Failures which may result in the breach of the SSSO and assumptions
Derived Software Safety Constraints
Contains the System Fault Trees – within which each hazard is implicit. FTs are used to define the SIL requirements for the OFP
Sub-System Hazard Analysis Report
Identify system-level hazardous behaviour
System-Level Software Safety Objectives
Capture the required behaviour of the OFP Suitable to prevent the hazardous behaviour. Structured such that if the OFP meets the objectives, the precise OFP failure (identified for the hazard) will not occur. The SIL assigned to the software contribution associated with the relevant SSSO – and then with the software safety requirements derived to ensure the objective is met. Derived conditions only under which the objectives need to hold.
Define constraints on the behaviour of the OFP which ensure that a hazardous failure does not occur. NOTE: hazardous failures are those failures that may contribute to the breach of an SSSO, and which don't break any of the objective assumptions
One for each operational clearance of the OFP Agreed with the ISA and the Customer
Shall include: 1. Updates to the Interim safety Case as previously presented at TRR 2. Completion of all safety goal and solution development 3. Completion of all safety arguments, context information and justifications for each safety argument 4. Completion and provision of the evidence required to support all goals allocated. Justification of any change in safety argument strategy from those strategies identified in the previous phase. 5. Completion of all actions arising from previous safety reviews.
Contents: 1. Scope – clearly define the boundary of concern; identifying the software to which the safety argument is applicable. 2. References 3. Introduction 4. System and Design Aspects: (a) Overview of the system architecture including the system boundaries and interfaces (b) Overview of the system functions (c) Brief description of the operating environment, including both normal and abnormal modes of operation (d) A list of the main system safety requirements (e) The system design report 5. System Hazards – a brief description of the system level hazards that are applicable to the software under consideration (referencing the Hazard Log as an aid to traceability 6. Software Safety Requirements – describe the role that SW plays in ensuring safety. Include a list of the functional and non-functional SW safety requirements, along with the safety integrity requirements for the SW, and any required SW standards 7. Software Description – describe the architecture of the SW and how this contributes towards safety. Including: (a) Overview of the SW architecture (b) Description of the main design features of the SW (e.g. real-time aspects, user interfaces, key algorithms etc.) (c) Means by which SW of different SILs is segregated (if relevant) 8. Safety Arguments – justifying how each SW safety requirement has been met, and that the necessary measures have been taken to reduce the SW contribution to system hazards to an acceptable level. The argument will reference at least 2 independent sources of evidence (e.g. analysis and testing). A complete list of all assumptions (with corresponding justifications) used in constructing the safety argument, and a list of weaknesses in the current argument. 9. Software Development Process – justification that the SW development process is adequate to achieve the required SIL of the SW: (a) Briefly describe the main methods, tools and key project staff (b) Describe and justify the use of any previously developed SW (c) Provide a measure of the performance of the SW development process (d) The results of any safety/quality audits carried out (e) If the data is available, provide an analysis of historical data on the safety integrity of SW developed using the proposed/similar SW development process. A summary of the testing evidence is needed for the argument, and could be placed in this section. 10. Current Status 11. Change History 12. Compliance – statement of compliance against all relevant SW standards 13. In-Service Feedback – from realistic operational testing, trials or in-service usage 14. Software Identification – specifying the current release of the SW, citing a SW release notice 15. Conclusions 16. Notes – definitions and abbreviations used within the document, and the provision of any additional material that may aid in the reading of the SW Safety Case.
Safety Requirement Allocation to Sub-Systems
Safety Requirement Partitioning to HW & SW
Develop Accident Mitigation Strategies
Software Safety Requirements
Software Requirements Analysis
Examination of Critical Data Flows
SHARD
System Engineers
Supportability & Through Life of Type
Software Requirements Specification
3rd Party Application Provision
Safety Partitioning
Both
On each identified data flow to identify potential breaches of SSSOs resulting from deviations in OFP behaviour
Software Hazard Analysis
SW Hazard Analysis Report
Software Design
OSL and AL Teams
One for each layer (Application and Operating System). Contain the results of the analysis, and summarising any new hazards found, and how they are to be mitigated.

Figure C.6: JB61834 Software Contribution to System Hazards

strength of aspects from analysis of the full model with regard to this criterion, however.

The first step in assessing compliance with this criterion is to identify all hazards to which software contributes. It is already known however, that there is only a medium level of confidence that this can be claimed - relying on the 'Hazard Log' artefact as a single point of truth for this data.

Despite the medium confidence held, a trace can be made (as illustrated in Figure C.6) to assess the efficacy of the lifecycle in enabling the complete identification of the ways software contributes to these hazards (and thereafter the elicitation of safety requirements that contribute to mitigating the hazards through specified software behaviour(s)).

The artefact 'Hazard Log' (although not annotated in Figure C.6) is an input to the activity 'Identify System-level Hazardous Behaviour', which in turn produces the artefact 'System-Level Software Safety Objectives'.

'System-level Software Safety Objectives' have the following required quality attributes:

- Capture the required behaviour of the (software)

- Suitable to prevent the hazardous behaviour
- Structured such that if the (software) meets the objectives, the precise failure (identified for the hazard) will not occur.

It further notes that any assigned SIL (associated with the contribution) should be assigned to the relevant software safety requirements derived to ensure the objective is met. There are two shortfalls with this, however. The first shortfall is that there is no link between the activity 'SIL Allocation' (a system-level activity) and the software safety requirements. Secondly, there is no direct link between the 'System-level Software Safety Objectives' and the 'Software Safety Requirements'.

Although the 'System-level Software Safety Objectives' are an input to the activity 'Identify Mitigation for SW Failures', which in turn produces 'Derived Software Safety Constraints' and 'Derived Software Safety Requirements', these two artefacts do not inform any further engineering nor design activity (only feeding in to the requirements database and safety arguments respectively).

'Software Safety Requirements' can only be traced as an output from the platform-level activity 'Develop Accident Mitigation Strategies' which implies that the level of granularity of these requirements will be at too high a level for considering specific behaviours and failures of the software. Other identified artefacts that could contain such data are:

- Derived Software Safety Requirements
- Derived Software Safety Constraints
- Sub-System Hazard Analysis Report
- Software Hazard Analysis Report.

Taking each of these in turn, we already know that the derived software safety requirements and constraints do not inform any further design/engineering activity, and the required attributes are also weak. The 'Derived Software Safety Requirements must only be "specific to the architectural design being used", and the required attributes for the 'Derived Software Safety Constraints' note only that they should constrain the behaviour in a manner that ensures a hazardous failure does not occur (not what these hazardous failures are)

The required attributes for the artefact 'Sub-System Hazard Analysis Report' stipulate that the report must contain the system fault trees, within which "each hazard is implicit", and these fault trees are used to define the SIL requirements.

The required attributes for the artefact 'Software Hazard Analysis Report' stipulate that any new hazards that are found should be summarized, along with how they are to be mitigated. Although this appears positive, the only consumer of this product is the activity 'Software Design', and the producing activity ('Software Hazard Analysis') has no denoted inputs to it (despite fully articulating how the analysis is to be carried out).

The lifecycle does have a robust process for the creation of Software Safety Cases whose maturity is increased in line with the increasing maturity of design, and so these were also assessed for this criterion. Artefacts such as safety cases contain a wealth of data required to support any safety argument, and can be expected reasonably to contain specific details on software's behaviour, contribution to system hazards, and evidence that safety requirements have been both elicited and instantiated in mitigation of dangerous behaviour(s).

The lifecycle requires three versions of a Software Safety Case (listed in increasing levels of maturity):

- Preliminary Software Safety Case
- Interim Software Safety Case
- Operational Software Safety Case (not shown in Figure C.6 as it is not relevant to this criterion).

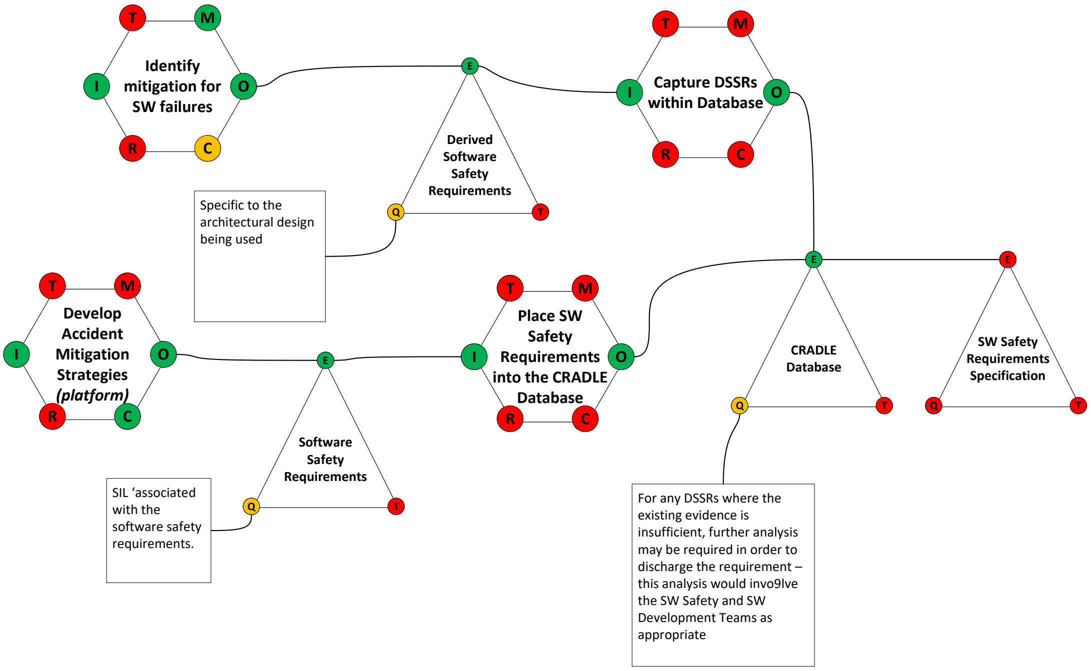
But these offer no evidence required of this criterion, with their aspects being concerned only with the nature of the GSN structure and updates from previous versions.

Whilst it can established, with high confidence, that the lifecycle produces sufficient artefacts from which the manner that software contributes to the mitigation of identified hazards can be identified, there is nothing within the model that identifies the specific behaviours of software that would result in the manifestation of these hazards. Neither can there be any confidence, from the as described lifecycle alone, that the data contained in various artefacts will be made available in the right format, at the right time, to the right people, for the necessary activities. Therefore, no claim can be made against the satisfaction of this criterion.

C.5 All Software Safety Requirements Will be Atomic, Unambiguous, Defined in Sufficient Detail, and Verifiable

Having already analysed how software safety requirements are established by the JB61834 lifecycle, attention now turns to the specific quality attributes of the software safety requirements themselves. The supporting activities and artefacts appropriate to this criterion are illustrated in Figure C.7.

There are three artefacts to be assessed here, and each are considered in turn:


T
M
Identify mitigation for SW failures
I
O
R
C
E
Derived Software Safety Requirements
Q
T
T
M
Capture DSSRs within Database
I
O
R
C
T
M
Develop Accident Mitigation Strategies *(platform)*
I
O
R
C
E
Software Safety Requirements
Q
T
T
M
Place SW Safety Requirements into the CRADLE Database
I
O
R
C
E
CRADLE Database
Q
T
E
SW Safety Requirements Specification
Q
T

Specific to the architectural design being used

SIL 'associated with the software safety requirements.

For any DSSRs where the existing evidence is insufficient, further analysis may be required in order to discharge the requirement – this analysis would invo9lve the SW Safety and SW Development Teams as appropriate

Figure C.7: JB61834 Software Safety Requirements

- Derived Software Safety Requirements
- Software Safety Requirements
- SW Safety Requirements Specification.

'Derived Software Safety Requirements' are required only to be "specific to the architectural desgin being used". 'Software Safety Requirements' are raw requirements, and are required only to have a measure of performance expressed against them in terms of SIL. The software safety requirements are placed into the CRADLE database, but the activity which does this has no considerations made for time, method, resources, nor control. A formal 'SW Safety Requirements Specification' is extracted from the requirements database, but no activity is stipulated to do so (hence the red line connecting the database to the specification), and the specification has no consideration as to any of the required quality attribute.

The 'Software Requirements Specification' artefact was also assessed to determine whether this criterion could be argued to have been met by the attributes required of it. However, this is an inferred artefact created to fulfil internal consistency and no attributes are considered by the as-required practice.

As such, this criteria cannot be claimed.

Appendix D

Evaluation Session One

The first Evaluation Session was held with the following anonymous participants as an online tutorial and evaluation session held on the following dates:

1. AY8697 on 07 June 2024
2. SH27236 on 11 June 2024
3. HH75783 on 13 June 2024.

The suitability of each participant in this evaluation is argued by appeal to their experience claimed in the Questionnaire (see Section D.2).

Evaluation Session One comprised a short tutorial, and the presentation for Session One contained the following topics:

1. Introduction to the FRAM notation
2. Introduction to the adapted version of FRAM used in this research
3. Tutorial on the Framework and Process to understand software safety practice
4. A Practical Session (see below)
5. An opportunity to ask any questions.

The Evaluation Session presentation pack and script, along with the provided handout and process instructions is found at [122].

D.1 Session One Practical Session

The participants were each given digital templates of the modified FRAM symbols and the participants' understanding of their use and meaning was confirmed verbally with each participant.

Confident in the participants understanding of the modified FRAM symbology, the participants were then given Section 5.1 of ARP 4754B [1] [119].

Participants were then invited to create a partial model of as-required (Open) practice using only Section 5.1 of ARP 4754B. Participants were told they could use the text of Section 5.1, the diagram of the process detailed by Section 5.1 OR a combination of the two.

There was no time limit for the task explicitly, but the evaluation session was planned to last for two hours (deemed by the author to be sufficient time for the tutorial and practical exercise, and yet short enough so as not to demand an onerous commitment from the respondents (thereby promoting engagement)).

On completion of the task, the participants were asked to complete a questionnaire, which is contained in the next section. The completed questionnaires from Session One - along with the completed models from the three sessions are found at [122], and the findings are discussed in Chapter 6. The Questionnaire is now provided.

D.2 Session One Questionnaire

Having completed this first evaluation session, you are kindly invited to respond to the following questions.

It would be beneficial for us to be able to argue over your expertise in the field of software safety practice. To that end we would be grateful if you could list the attributes of your experience as a software safety practitioner and indicate in parentheses afterwards the number of years experience you have. For example:

1. Software Safety Engineer (5 years)
2. Principle Software Safety Engineer (3 years)
3. Safety Manager (3 years)
4. Independent Safety Assessor (2 years).

[1] It was positively confirmed that each participant had paid access to this IPR-protected document through their employment and / or participation in national / international Standards' Committees

Please use the text box below and list all attributes you believe are relevant. There is no word count limit, but please complete this digitally using an appropriate word-processing software package (such as Microsoft Word), so that we can ensure all comments are fully legible.

Table D.1: Session One Evaluation Questions

Statement	**Fully Disagree 1**	**Somewhat Disagree 2**	**Neither Agree/ Disagree 3**	**Somewhat Agree 4**	**Fully Agree 5**
Completeness: Reflecting on your understanding of the process to understand software safety engineering practice, we would like your opinion on the following statements. In considering your response, we ask that you ALSO consider applications and technologies NOT covered by the artefacts provided (i.e. from experience throughout your career), and don't restrict your response to JUST the artefacts sent to you					
EQ1: The process considers all elements that together constitute software safety engineering practice (the 10 'steps')					
EQ6: The modelling process instructions are easy to follow (you could follow each step)					
Effectiveness: Having applied part of the process to understand software safety engineering practice through modelling and assessment of practice, we are interested in your thoughts on the usefulness of this process. How much do you agree with the following statements? In considering your response, we ask that you ALSO consider applications and technologies NOT covered by the artefacts we provided you with (i.e. from experience throughout your career), and don't restrict your response to JUST the artefacts sent to you					
EQ10: The process to understand software safety practice will help to identify potential impediments to achieving best practice for software safety					
(Pan-industry Applicability) EQ11: The process to understand software safety practice can be used for any industry and any technological application					

Table D.1 continued from previous page

Statement	**Fully Disagree 1**	**Somewhat Disagree 2**	**Neither Agree/ Disagree 3**	**Somewhat Agree 4**	**Fully Agree 5**
Consistency: Having applied part of the process to understand software safety engineering practice, we are interested in your thoughts on the consistency of the outputs which the process creates. How much do you agree with the following two statements?					
EQ12: The process uses consistent terminology when considering each different element that constitutes software safety practice					
EQ13: The process creates models whose symbology is consistent across all elements of software safety practice					

If you have any additional comments on the process, or on this specific evaluation you are invited to make them in the box below. There is no word count limit, but please complete this digitally using an appropriate word-processing software package (such as Microsoft Word), so that we can ensure all comments are fully legible.

This appendix now provides the details on the experience stated by the three independent experts. The anonymity of the respondents has been maintained.

D.3 Session One Questionnaire Responses

The three independent experts who took part in the evaluation sessions are as follows:

1. AY8697 - a System Safety Engineer for thirty years; working for a company which "considers system safety and software safety as two different disciplines". AY8697 has worked as an 'industrial researcher' for twenty-five years and over their career have had 'sporadic involvement' with Safety Management and Independent Safety Assessments.
2. SH27236 - an Independent Safety Consultant with thirty-five years experience in developing, managing, assuring, and certifying safety critical software-based systems.
3. HH75783 a 'general software safety practitioner with twenty-five years' experience in both academia and industry). The experience includes time as an independent safety assessor and auditor (including software) for six and a half years, and a software verification tool R&D manager for one and a half years. They have worked in defence (land, aviation, maritime), civil aviation, nuclear, rail, and automotive industries.

AY8697 provided the completed questionnaire at [117].

SH27236 provided the completed questionnaire at [125].

HH75783 provided the completed questionnaire at [123].

Appendix E

Evaluation Session Two

The second Evaluation Session was held with the same anonymous participants from Session One, and an online tutorial and evaluation session held on the following dates:

1. SH27236 on 12 June 2024
2. AY8697 on 14 June 2024
3. HH75783 on 18 June 2024.

The suitability of each participant in this evaluation is argued by appeal to their experience claimed in the completed questionnaire (see Section D.2).

This Evaluation Session required the participants to complete a short Case Study which involved the comparison of two different models of software safety practice. The participants were presented with two models of as-required practice created as part of this Thesis - the as-required (Closed) model of practice (Step 3 in Chapter 5), and the as-required (Open) model of practice (Step 2 in Chapter 5).

Having received a tutorial in the use of colour coding for use in our adapted version of FRAM, the participants were then asked to complete Step 8 of the process to understand and assess software safety practice. On completion of the task, the participants were asked to complete a questionnaire, which is contained in the next section.

The session instructions, and completed questionnaires from Session Two - along with the completed models are found at [122], and the findings are discussed in Chapter 6.

E.1 Session Two Questionnaire

As the same participants were used for both evaluation sessions, the participants were not asked to provide details of their experience again.

The questionnaire provided to the participants is now provided.

Having completed this first evaluation session, you are kindly invited to respond to the following questions.

Statement	Fully Disagree 1	Somewhat Disagree 2	Neither Agree/ Disagree 3	Somewhat Agree 4	Fully Agree 5
Ease of Use: Reflecting on your experience with using the symbology that was supplied to you for the purpose of creating / assessing a model, we would like your opinion on how much you agree with the following two statements					
EQ2: The modelling symbology is easy to understand (you knew what the different shapes and lines represented)					
EQ3: The modelling symbology is easy to use (you could easily use / interpret the different shapes and lines to construct / assess a model					
Ease of Use: Reflecting on your experience of following the steps in the process to model and assess software safety engineering practice, we would like your opinion on how much you agree with the following three statements.					
EQ4: The process to model software safety engineering practice can be carried out without any prior knowledge of formal modelling (i.e. no training in model-based systems engineering was required)					

Table E.1 continued from previous page

Statement	Fully Disagree 1	Somewhat Disagree 2	Neither Agree/ Disagree 3	Somewhat Agree 4	Fully Agree 5
EQ5: The process can be instantiated by anyone with access to standard 'Office' applications (such as Visio, Lucid Chart, Word, Pages, Google Docs etc.)					
EQ6: The modelling process instructions are easy to follow (you could follow each step)					
Reflecting on your experience of following the steps in the process to assess software safety engineering practice, we would like your opinion on how much you agree with the following statement					
EQ7: The process instructions to assess software safety engineering practice (the way in which comparisons are made) are easy to follow					
Effectiveness: Having applied part of the process to understand software safety engineering practice through the modelling and assessment of practice, we are interested in your thoughts on the overall usefulness of this process. How much do you agree with the following four statements? In considering your response, we ask that you also consider applications and technologies NOT covered by the artefacts we provided you with (i.e. from experience throughout your career), and don't restrict your response to JUST the artefacts sent to you					
EQ8: Using the modelling process allows me to understand all elements of software safety engineering practice					

Table E.1 continued from previous page

Statement	Fully Disagree 1	Somewhat Disagree 2	Neither Agree/ Disagree 3	Somewhat Agree 4	Fully Agree 5
EQ9: Using the modelling process allows me to assess all aspects of software safety engineering practice (through comparisons between the elements of practice and their relationships)					

Table E.1: Session Two Evaluation Questions

If you have any additional comments on the process, or on this specific evaluation you are invited to make them in the box below. There is no word count limit, but please complete this digitally using an appropriate word-processing software package (such as Microsoft Word), so that we can ensure all comments are fully legible.

SH27236 provided the completed questionnaire at [126].

AY8697 provided the completed questionnaire at [118].

HH75783 provided the completed questionnaire at [124].