Improved Methods for Review of Software Assurance Standards using Def Stan 00-055 as a Case Study

James Inge

Kellogg College
University of Oxford

March 2019

A dissertation submitted in partial fulfilment of the requirements for the MSc in Software Engineering

Abstract

Defence Standard 00-055 is the UK Ministry of Defence's contracting standard for safe software. In the 1990s, it mandated specific practices that contractors saw as too onerous and prescriptive, such as formal design and specification methods. By 2005 it was technically outdated and declared obsolescent, but now it has been resurrected. The new Issue 4 tries to avoid the original criticisms by being less prescriptive. But is it still a good document? How can it be improved? This dissertation explores how to answer these questions.

Software safety standards have a vital role in delivering safe products, services and systems. In the defence industry, software failures can lead to significant loss of life. This makes it especially important that such standards are well understood by their users. Yet, they are often verbose, lengthy documents written by committees; hard for the uninitiated to immediately digest and understand, and awkward to implement as written. Anecdotal feedback suggests Def Stan 00-055 Issue 4 may suffer from these problems, implying that its review process was not entirely effective. The author will shortly be involved in organizing the standard's next periodic review. His experience of reviewing similar documents consists mainly of being asked to read a text and provide comments, without any detailed guidance or methodology being provided. This seems a dull chore, and perhaps not the best way to detect flaws and potential improvements in a standard. Our motivation is to find a better standards review method that will lead to better safety standards and safer systems.

In this project, we develop a guidance framework to help make future standards reviews more effective. We build on the concept that software safety assurance standards are themselves artefacts of the software engineering process: part of the decomposition of organizational goals into software requirements and designs. With this idea, we examine the software engineering and software safety literature to identify potential review methods that might work for standards. To evaluate these methods, we experiment by applying them to Def Stan 00-055 to compare their effectiveness and practicality. While some of the methods, such as argument modelling, have previously been applied to reviews of other standards, Def Stan 00-055 has not previously been modelled. Novel aspects of this project include using assurance framework meta-models in standards reviews, and making a comparison of the effectiveness of model-based and other review methods.

We find that methods inspired by software engineering can indeed improve the quality of software assurance standards like Def Stan 00-055. The guidance framework describes how to do this using a combination of methods. We evaluate the usefulness of the guidance by seeking feedback from its potential users, finding that both the guidance and the experimental results are likely to be useful in the upcoming review of Def Stan 00-055. The guidance also appears broadly applicable to review of other assurance standards beyond the software safety field, and has implications for the preparation of standards.

Acknowledgements

I am very grateful to the Ministry of Defence for sponsoring me via the Defence Equipment & Support upskilling fund and allowing me the time away from the office to study. Without this, I would not have had the opportunity to take the Software Engineering MSc in Oxford. I would also like to thank my colleagues in the MOD and elsewhere who provided feedback on the guidance framework produced through this project.

I am indebted to my supervisor, Peter Bloodsworth, for his time and encouragement talking through my work; to Shirley Sardar and all the other staff of the Software Engineering Programme for guiding me over the administrative hurdles of the course; and to all the lecturers for the high quality of their teaching.

I am also grateful to Adelard LLC and Change Vision Inc. for providing free student licences for the Assurance and Safety Case Environment (ASCE) and Astah UML respectively; also to the PolarSys Eclipse Foundation working group, for making the open source AMASS toolset available.

Finally, and most importantly, I wouldn't have been able to complete the course without the support of my loving wife and family. They have encouraged me along the way and put up with my frequent absences, either in Oxford or just upstairs in my study working on this dissertation.

Declaration. The author confirms that: this dissertation does not contain material previously submitted for another degree or academic award; and the work presented here is the author's own, except where otherwise stated.

This version was reprinted in July 2019 with minor typographic corrections suggested by the Assessors.

Opinions expressed in this document are those of the author, and do not necessarily represent those of his employer, the Ministry of Defence.

Contents

1 Introduction

1.1 Context and Motivation

In order to equip the United Kingdom (UK) armed forces, the Ministry of Defence (MOD) acquires a great variety of military systems. Many of these depend on software for at least part of their functionality [1]. Because of their military role, they are frequently bespoke or customized designs and often at the cutting edge of technology. As these systems are often deployed in situations where their role has an impact on safety, the MOD has a keen interest in ensuring that the software enabling them is safe. This need was underlined by high-profile software-related incidents like the Patriot missile defence system failure in 1991 [2] and the Ariane 501 launch failure in 1996 [3]. More recently, in 2015, logic errors were implicated in crashes of Watchkeeper military drones [4] and software loading problems are thought to have caused the loss of an A400M military transport plane [5].

In the past, the MOD has been criticized for its procurement of software, with issues in software development a notable cause of delays to some of its major projects [1, 6]. A particular example was the procurement of the Chinook Mk 3 helicopter. Eight Mk 3s were delivered to specification by Boeing in 2001 at a cost of some £259 million. As the avionics software for their bespoke digital 'glass cockpit' could not be certified to meet UK military airworthiness standards, they could not be used in operations until 2009 [7]. The problem was not that the software was known to be unsafe, but that it was not known to be safe. The MOD could not demonstrate its safety as it had not contracted for Boeing to provide either sufficient evidence of safety analysis, or access to source code that the MOD could analyse itself [8]. The issue was resolved by first reverting the Mk 3 Chinooks to an earlier, proven design standard at a cost of over £90 million, then later upgrading them to a different type of glass cockpit. The project for this upgrade was itself delayed a further nine months due to software development issues [6]. This example illustrates that to deliver military capability to the required time, cost and performance, the MOD needs software that does not just work, or even just work safely: it needs to be demonstrably safe. Achieving this does not happen by accident. When the MOD sets out to acquire new software, it needs to communicate its safety requirements to the supplier as part of the contract. To do this in a repeatable way, the MOD needs to use a standard.

Interim Defence Standard 00-55—The Procurement of Safety Critical Software in Defence Equipment was published in 1991 [9], then updated to a full standard in 1997 [10]. Unfortunately, both these original versions of Defence Standard (Def Stan) 00-55 were seen as too onerous and prescriptive, in some cases pushing beyond the state of the art [11–13]. This made contractors wary of accepting the standard as a contractual requirement, and increased the price of contracts where it was used. While Def Stan 00-55 was influential in driving the debate on the use of formal methods in software engineering, it did not achieve widespread acceptance by defence contractors. By the early 2000s it

was seen as becoming dated, moving away from best practice and MOD policy [14], with some contractors claiming that it constrained them from using more effective, modern techniques [13]. In 2005, the standard was officially designated obsolescent [15]. Despite this, it remained a popular reference on safety critical software [16], and attempts to fill its role through other guidance were not wholly successful [17].

In 2014, the MOD published its new contracting standard for safe software: *Interim Defence Standard 00-55 Issue 3—Requirements for Safety of Programmable Elements (PE) in Defence Systems* [15]. Following a consultation period, in 2016 it was updated to a full standard: Def Stan 00-055 Issue 4 [18]. Issues 3 and 4 set requirements that are much less prescriptive and more goal-based than Issues 1 and 2, setting out general objectives to be met rather than mandating particular techniques. The new objective-setting approach should make Def Stan 00-055 Issue 4 more flexible and easier to use, by enabling developers to choose appropriate tools and development standards for the context of the system they are delivering. Issue 4 was subject to review by a broad range of stakeholders, who were able to draw on experience from use of Interim Issue 3. Despite this, the author has heard anecdotal feedback that it is seen as complicated and difficult to understand. This would make it hard for MOD staff to use for setting requirements and hard for contractors to understand how to demonstrate compliance.

1.2 Problem

The author has been employed by the MOD for some years in a variety of roles connected with system safety engineering. He has been involved with the review of standards and policy documents both as a sponsor, an editor, an author, and as a contributor to reviews led by other business areas. He has also used these documents himself in the procurement and support of software and hardware systems, and provided advice and guidance on their application to other MOD staff. Although he was not involved in the production of the current issues of these standards, he has recently joined the team that sponsors Defence Standards 00-055 and 00-056. As Chair of the Safety Standards Review Committee (SSRC), he now has responsibility for organizing their review. We are now approaching the point when preparation for the next iteration of Def Stan 00-055 should be starting. This gives the motivation for this project: to better understand the current issues with the standard, and to improve the review process so that such problems can be avoided in future.

In the author's experience, formality in reviews of standards and similar documents often extends only to having a process of official committees and meetings that leads to endorsement of a new version. They tend not to be formal in the sense of actually examining the standard in a structured manner, or using formal methods to exploit the structure and semantics of the standard itself as part of the review. Most often, reviewers are simply presented with a draft document and asked to respond with comments. Standards can be lengthy, and reading through and making meaningful comments can be time-consuming for reviewers. In software engineering, reviews are recognised as an effective way of improving software quality, and a variety of more structured methods are available to help verify and validate development artefacts. Inspired by the software engineering methods, we seek a more effective method of review for the next iteration of Def Stan 00-055. We aim to produce guidance to focus reviewers' efforts where their expertise can have best effect, to ensure that the published standard has few errors and can readily be put to use.

1.3 Approach

In this project we test the premise that Def Stan 00-055 is itself an artefact of the software engineering process and hence amenable to software engineering methods. As a standard that sets assurance requirements for software, it is part of the decomposition from high-level organizational goals to low-level software requirements. On this basis, we may be able to investigate its quality by applying similar methodologies and techniques to those that software engineers use to review artefacts like software specifications, designs or code. We wish to determine which methods will be most effective for this purpose.

We begin by examining the software engineering literature to identify what might be suitable for this task, drawing on methods from the fields of software quality, requirements engineering and software safety assurance. To test these methods, we first need a baseline for comparison, which we create by carrying out a traditional 'naïve' review of the standard. We then experiment by applying some of the techniques and methods we have identified to the current edition of Def Stan 00-055. Through this we investigate how traditional styles of review can be improved by using software engineering ideas and how newer, more structured methods can be applied. Based on the experience of using these methods, we propose a guidance framework for carrying out future reviews of Def Stan 00-055 and other similar standards. We also use the experimental results to comment on the quality of Def Stan 00-055. We discuss whether the latest issue appears to be a good document in its own right and likely to be effective in helping the MOD deliver demonstrably safe software. Finally, we comment on opportunities for potential improvements, both to Def Stan 00-055 and to the standards review process.

1.4 Evaluation

The goal of the project is to improve the review of software safety standards, in the hope that this will result in better standards, and thus safer systems that use the resulting software. Ideally then, to evaluate the project's success, we would quantify the effect of our new guidance on safety. Unfortunately, this is not feasible within the scope of an MSc project. Given that 'safety' is the freedom of risk from harm and accidents are hopefully rare events, it is difficult to measure how safe something is in absolute terms, especially over a short period of time. Judging the guidance's impact on safety would require comparing the safety performance of standards that had been reviewed and updated according to the guidance with those that had not. The defence projects that use Def Stan 00-055 are typically long-term, meaning that such a study could only be done over a period of years. It would also be difficult to control the study to understand the impact of the guidance and the standard compared to other influences on safety.

Instead of trying to measure safety impact, we will focus on evaluating the guidance itself and its recommended methods. We start by evaluating the effectiveness of individual review methods by comparing the number, type and relative importance of the issues found by each method against those from the baseline naïve review. If our premise is correct, we hope to show that review methods that have proven effective in software engineering will be more effective in standards reviews than traditional methods. We also compare the practicality of using each method, and use these interim findings to steer the development of a standards review guidance framework. Finally, we evaluate the framework by seeking feedback from stakeholders and subject matter experts. We gather their views on the usability of the guidance and the significance of the findings

made by applying it to Def Stan 00-055.

1.5 Structure of this dissertation

- **Chapter 1** (this chapter) introduces the context and motivation for this project, the problem it addresses, the desired outcome and the premise behind the approach taken.
- **Chapter 2** gives a background to Def Stan 00-055, the software safety standard used as the running case study throughout the dissertation; explaining why it is necessary and why we are interested in reviewing it. It then examines the literature to identify potential methods and techniques from software engineering that could be applied to review it robustly.
- **Chapter 3** explains the selection of review methods and their application to Def Stan 00-055, and summarizes the results.
- **Chapter 4** evaluates what these results mean for the standards review process, discusses the sources of error, and proposes a guidance framework for practical reviews of assurance standards.
- **Chapter 5** reflects on the work that has been produced, using feedback from stakeholders to assess the usefulness of the guidance. From this it makes recommendations for the further development of the guidance framework and Def Stan 00-055, and broader research in the area.
- **Chapter 6** concludes the dissertation by summarizing the key outcomes.

2 Background and literature review

We start this chapter by discussing background Def Stan 00-055's background to understand why it is important, how it fits into MOD policy, and the issues found with it in the past. We then review the literature to investigate potential methods from software engineering that we can use to improve the review of the standard. We look at the typical conduct of reviews in the standardization process, and how software engineering practices could provide benefits, before discussing the type of criteria to be used in the reviews. As Def Stan 00-055 is a requirement-setting standard, we draw heavily on the requirements engineering field. As this is an individual project, we focus on methods that can assist a solo reviewer, rather than addressing the social sciences aspects of requirements elicitation and review as group activities. We then review the software safety assurance literature, to look for more structured ways of examining the adequacy of a standard. Promising methods identified are taken forward to Chapter 3 to apply to Def Stan 00-055.

2.1 Def Stan 00-055: the MOD's software safety standard

2.1.1 The MOD acquisition framework for software

The MOD acquires a very broad range of software. Some may be supplied as 'pure' applications (e.g. a mission planning system) or information services, but more commonly the software will be embedded in a hardware system. In some cases the role of software is obvious. In others (such as in the controller of an electronically actuated valve or fuse), it may be hidden in a device traditionally thought of as purely mechanical or electronic. As well as the code itself, the MOD also acquires through-life maintenance and support services for software and the hardware it runs on. The number and variety of software-enabled and software-dependent systems it acquires means that the MOD cannot rely purely on the expertise of specialists to set software requirements and manage projects. It has therefore generated a framework of policies, instructions and guidance to assist non-specialist staff in procuring software.

The "primary bearer of all policy and guidance governing Defence's Project Delivery and Commercial Functions as well as all other current business areas" is the Knowledge in Defence (KiD) website[1] [19]. In this context, 'acquisition' refers to both initial procurement and acquisition of support, and applies to all types of products, services and systems for Defence use, not just software. Three main parts of the KiD relate to software:

- The 'Software Acquisition Management and Software Support' sub-site gives general guidance, focussing on supportability requirements.
- The 'Managing Quality' sub-site mandates software suppliers to hold an appropriately certified Quality Management System. It gives guidance on appropriate

[1]The KiD was previously known as the Acquisition System Guidance.

quality standards, supplier selection, contracting for software quality, supplier monitoring and treatment of off-the-shelf software [20].

- The 'Safety and Environmental Protection' sub-site introduces various guidance documents and standards related to safety-related or safety-critical software. In particular, it introduces Def Stan 00-055, the focus of this report.

2.1.2 Def Stan 00-055 and its goals

Although nominally aimed at both MOD and its industry partners and made available externally via an internet portal, most of the material on the KiD is actually written to guide internal MOD project teams. However, the MOD produces little software in-house. Instead, it contracts out this task to a range of suppliers. As Mattern argues [21], when a government department commissions software-intensive systems, software assurance activities must be adequately addressed in the Request for Proposal. Without formal requirements in the request (commonly known an Invitation To Tender (ITT) in the UK), contractors proposing such activities against a "phantom requirement" would be at a commercial disadvantage. This means that the MOD needs to translate its internal guidance into terms that can be included in the ITT as part of a requirements document, statement of work, or some other contractually binding form. This could be done on a case-by-case basis for each contract. However, it is more efficient to gather these requirements together into a standard that can be reused for each project.

The MOD recognized the potential impact of software in safety-critical systems and wished to be able to contract out its production but retain some control over its quality [13]. As a result, in 1991 it introduced Interim Defence Standards 00-55 and 00-56. 00-56 (*Hazard Analysis and Safety Classification of the Computer and Programmable Electronic System Elements of Defence Equipment*) explained how to assess the risk from a programmable system [22]. 00-55 (*The Procurement of Safety Critical Software in Defence Equipment*) set out the measures necessary to assure safety in critical software [9].

These standards are known as 'assurance standards'. They do not set functional requirements for the software or the systems that run it. Instead, they attempt to give a customer assurance that the software supplied to them will be adequate with respect to some particular attribute. In the case of Def Stan 00-055 and Def Stan 00-056, this attribute is safety. Other similar assurance standards exist for attributes such as quality, security, environmental management or reliability.

Why was it necessary for MOD to create and own these safety assurance standards itself? The MOD has a policy to "use civil standards wherever practicable and military standards only where necessary" [23]. In selecting standards, it uses a hierarchy that prioritizes international standards over national, with the MOD's own Defence Standards taking the lowest priority [23]. The United States (US) Department of Defense (DOD) took a similar approach in the 1990s, reducing military standards and adopting civil alternatives [24]. Ideally, the MOD would favour using a civil standard with broad international acceptance over the expense of creating and maintaining its own standard. Several candidate standards exist, including the basic functional safety standard IEC 61508 [25] and its various domain-specific interpretations; or DO-178 [26], a software certification standard commonly used in avionics[2]. However, these standards are aimed at organizations that develop systems, rather than customer organizations that procure

[2]For standards issued by RTCA, we use the standard number with revision letter to refer to specific versions (e.g. DO-178C), or without the letter to refer generically to the series (e.g. DO-178).

Number	Principle
1	PE Safety Requirements shall be defined to address the PE contribution to system hazards.
2	The intent of the PE Safety Requirements shall be maintained throughout requirements decomposition.
3	PE Safety Requirements shall be satisfied.
4	Hazardous behaviour of the PE shall be identified and mitigated—addressed by failure modes and supported by designing for safety.
5	The confidence established in addressing the other PE safety principles shall be commensurate to the contribution of the PE contribution to system risk and will be addressed by Design Integrity requirements.

Table 2.1: Def Stan 00-056 programmable element safety requirement principles, from [34].

them. While the MOD recognizes them as good practice, they do not include the so-called 'military delta' [27]. This term includes the MOD's requirements for oversight of the software development process, and provision to cater for the military imperative to be able to take higher levels of safety risk in particular circumstances. The DOD's standard practice for system safety, Mil-Std-882 [28], does include these type of customer requirements. It is unsuitable for the UK MOD though, because it fits into the US legal system and does not address UK health and safety concerns [24].

At the highest level, the MOD's policy is to "minimise work-related fatalities, injuries, [and] ill-health" and "reduce health and safety risks so that they are As Low As Reasonably Practicable (ALARP)" [29]. These policy requirements are driven by UK health and safety law [30]. The MOD's approach to demonstrating that it has satisfied the policy is to produce a 'safety case'. This is defined as "a structured argument, supported by a body of evidence that provides a compelling, comprehensible and valid case that a system is safe for a given application in a given operating environment" [31]. This approach is captured in Def Stan 00-056, the MOD's standard for general safety management requirements. Its guidance includes an annex on integrity and open standards, which defines five safety requirement principles for software and complex electronics. These are shown in Table 2.1. These principles do not follow directly from the higher-level policy, but have been taken from academic research into common features of safety assurance standards and other good practice [32, 33].

Def Stan 00-055 has been written to support the 00-056 approach. It places requirements on contractors such that products, systems and services acquired by the MOD that include software or other programmable elements[3] can be used safely; and such that the MOD gains the evidence it needs to build its safety case. Its explicit strategy to achieve this is to set objectives that meet the five principles (Table 2.1), and subsidiary requirements that enable these objectives to be met [18, § 6]. The hope is that this will cause PE safety requirements to be set that will result in software that does not contribute to unsafe behaviour. Using the standard should also give the MOD the assurances and

[3]The scope of Def Stan 00-055 extends beyond pure software to data and complex user-defined hardware such as field programmable gate arrays and application-specific integrated circuits, referred to by the general term Programmable Elements (PE). As this project is concerned with applying software engineering methods, while keeping this broader scope in mind, we will use the term 'software' as shorthand to refer to the products delivered in accordance with Def Stan 00-055.

evidence it needs to build its safety case. Beyond these goals, the authoring team aimed to create a level playing field for potential suppliers, give transparency about required deliverables and ensure that the resulting products were maintainable. They also wanted to allow suppliers to use familiar techniques to achieve this [16].

When called up in a contract, Def Stan 00-055 forms part of the specification for software acquired by the MOD. It is part of the decomposition of generic high-level safety requirements (derived from law and government policy), into more specific requirements on software and its development process. Using Ould's [35] terminology, these are not part of the user's requirement for the software's behaviour, or the design of a specific system. Instead, they identify desirable attributes and properties of software at a generic level. Capturing these requirements in a standard, instead of embedding them directly in an ITT or system specification, allows good practice to be captured centrally and re-used, reducing the workload for individual teams.

2.1.3 Previous criticism of Def Stan 00-055

At the time of its introduction, Def Stan 00-55 was considered among the most rigorous of software safety standards, in terms of its engineering practice requirements [36]. It attempted to drive forward the software engineering discipline to produce more robust software, in a large part through a strong emphasis on formal (mathematically-based) methods. This provoked much debate about the commercial practicality of implementing these methods on a broad scale [11, 37]. Commentators such as Ould [12, 38] believed it would take several years before industry was able to consistently achieve some of its requirements.

Although updated in 1997, by 2004 Def Stan 00-55 had become dated and its prescriptive mandatory requirements were seen as significantly limiting development and system safety assurance of software [39]. Some people saw it as too expensive to train developers to produce software using the formal methods it required [11, 40]. Others maintained it prevented use of more modern techniques. For example, it was seen as precluding automatic conversion from formal requirements to code (potentially reducing defect rate), as the auto-generated code could not be walked through the way the standard required [13]. Issue 2 of Def Stan 00-55 and 00-56 were both withdrawn and superseded by Def Stan 00-56 Issue 3. Now titled *Safety Management Requirements for Defence Systems*, Def Stan 00-56 took a new, goal-based approach. It was intended to apply generically to software and software-dependent systems alongside other types of system [41]. Def Stan 00-55 became officially obsolescent: still available for guidance, but no longer to be used for new MOD contracts [42].

Unfortunately, Def Stan 00-56 Issues 3 and 4 were not fully successful in replacing Def Stan 00-55. While there had been much objection to Def Stan 00-55, it remained one of the most popular standards on the Defence Standardization (DStan) website despite its obsolescence [43]. After several attempts to provide stand-alone guidance on how to satisfy Def Stan 00-56 for safety-critical software, the MOD decided to resurrect Def Stan 00-55. Interim Issue 3 was published in January 2014 [15]. After a period of formal consultation, in April 2016 it was uplifted to Issue 4 (with an extra '0' in its number[4]) [18].

Little work has been published criticizing or evaluating Def Stan 00-055 Issue 4. However, we know that Issue 3 was initially drafted by a small authoring team with MOD,

[4]In 2016, DStan started to standardize their numbering system to the format *XX-YYY*, so Def Stan 00-55 became 00-055. In this report, we use the new numbers to refer generically to a series of DStan standards, or to the current issue; and the old number where necessary to refer to specific earlier issues.

industry and academic participation [27]. It was then subject to review by a larger Software Safety Working Group, before being reviewed by the MOD's Safety Standards Review Committee (SSRC). The SSRC also has participation from trade bodies such as ADS Group (representing businesses in the aerospace, defence, security and space sectors; including major defence suppliers) and the Independent Safety Assurance Working Group (representing the professional engineering institutions and safety assurance consultancies). Issue 3 was then published as an Interim standard with the aim of gathering stakeholder comment based on its use over a one-year period [44]. This feedback should have been taken into account in the final publication of Issue 4. With this degree of participation in the review process that led to the release of Issue 4, one would expect that it had reached a relatively high level of maturity. Anecdotally however, consultants and software suppliers the author has spoken to still have concerns that the standard is complicated and difficult to work with.

Def Stan 00-055 aims to ensure that the software delivered to and operated by the MOD embodies the quality of safety. Ould [35] cautions that quality is 'fitness for purpose' and varies from system to system, implying that it is difficult to produce a prescriptive standard that can be applied generically. As Def Stan 00-055 has evolved, its scope has widened. Issue 1 was concerned with 'safety-critical' software, Issue 2 expanded the scope to 'safety-related software', while the titles of Issues 3 and 4 refer to all 'programmable elements'. In reaction to this broadening of scope and the earlier criticism of its prescriptive nature, the standard has become more goal-based. While this will make it more generally applicable, it raises several questions: Do its requirements now give sufficient direction to deliver safe software? Is it effective in meeting its aim?

2.2 Approaches to evaluation of standards

Wong et al. [45] evaluated five standards used in software safety, scoring each standard on a scale of one to five against twelve evaluation criteria. The criteria question how thoroughly each standard covers various topics that the authors consider important (such as quality assurance and complexity management); whether techniques they expect to be required are included (such as cost-benefit analysis and use of integrity levels); and other factors such as whether the standard is easy to use and under active maintenance. They found that some of the standards performed better against some criteria, and some against others, and suggested that projects should select their standards carefully to suit their needs. This analysis seems somewhat unsatisfactory and, while their paper provides a narrative to justify each criterion, it does not explain how they chose the set as a whole. Of the five standards Wong et al. evaluated, three are general system safety standards, rather than being software-specific (Def Stan 00-56 [46], the Federal Aviation Administration (FAA) System Safety Handbook [47] and Mil-Std-882D [28]). One is not safety-specific (DO-178B [26], which addresses certification considerations). Only NASA-STD-8719.13B [48] is a dedicated software safety standard.

Their results may be explained by considering that each document Wong et al. evaluated has a different purpose. Def Stan 00-56 and Mil-Std-882D are contracting standards, used by government agencies to set requirements for safety management. DO-178B is used by developers to help provide assurance information about the software development process to support certification of airborne software, but assumes that safety requirements are set under a separate standard. The NASA standard deals with setting safety requirements, but assumes that software assurance and detailed implementation are covered

by different documents. The FAA Handbook is a risk management guidance document rather than a requirement-setting standard, and draws on both DO-178B and an earlier version of Mil-Std-882 for examples. It is understandable that their scores will differ when assessed against arbitrary criteria, as they are intended to cover different parts of the overall problem of assuring safety. Each document fits in a different place in a different framework of standards and governance. They do not need to discuss each relevant topic in depth when these topics are adequately covered elsewhere in their framework, in separate documents.

While software developers who have a free choice of safety standard may welcome some abstract criteria to aid their selection, standards developers need a different sort of criteria. They need to understand whether their particular standard is good for its intended purpose. Wong et al.'s work tells us that standards should be easy to use and have a good coverage of the topics deemed relevant to their scope. However, it does not give us any clear guidance on how to evaluate a given standard on its own.

Graydon and Holloway [49] have also investigated the evaluation of software safety standards, motivated by the lack of evidence for their efficacy. They argue that there is little evidence to show that either the standards or the 'recipes' used to comply with them actually work. Without this, the apparent correlation reported between use of safety standards and lack of accidents could just be down to developers taking care when working with critical systems. Further, the authors claim that there is rarely a testable hypothesis of what it means for a software safety standard to 'work'. In order to evaluate such a standard properly, one must first gain a clear understanding of what the standard is supposed to achieve and what the evaluation is expected to test, then plan accordingly.

The software engineering community often advocates various methods of Verification and Validation (V&V) to ensure the quality of software code [35, 50]. However, various authors argue that the usefulness of V&V techniques extends beyond code to other artefacts used in the software development process [35, 51, 52]. Sansone and Rocca-Serra [53] go further, arguing that some types of standard should be treated as digital entities in their own right. Taking these concepts, we will investigate how use of methods from software engineering, systems engineering and software safety can assist in evaluating and improving the quality of standards.

2.2.1 Review approaches

The author's experience of reviewing standards consists mainly of what one might call 'naïve' reviews: reviewers are simply given a text and asked to read through and make comments. The comments are collated by an editor or editorial committee, their resolution determined, and the document amended accordingly. In some cases, a meeting or workshop may be held to resolve the comments, while in others this may be left to individual editors. In many cases, a form will be issued to guide respondents to provide their comments in a certain format. As an interim standard issued for consultation, Def Stan 00-055 Issue 3 included a comments form asking for page and clause references, comments and proposed solutions [15]. The review template used by the British Standards Institute is similar, but also asks for a rationale for each comment and a classification of its type as general, technical or editorial [54]. The structure of these forms and the presentation of the document under review tends to lead to a particular style of comment. Reviewers read the document sequentially and comment on specific sentences, paragraphs or figures as they come to them. Typically, the comments relate to the wording of a particular part of the text; it is less usual to receive comments that relate to inconsistencies or interrela-

tions between different parts of the standard. This experience does not seem uncommon, with other authors also bemoaning the quality of ad hoc standard review processes and seeking more rigorous methods [55, 56].

As review of software safety standards appears a similar activity to review of software code or specifications, in this report we look to the literature in these areas for inspiration on how to improve standards reviews. Unfortunately, the classic literature on software safety does not enlighten us much on review techniques. In *Safeware* [57], Leveson seems to assume that reviews of various descriptions are carried out. Unfortunately, she does not provide any detail and focuses more on accident theory than the practice of verifying software engineering artefacts. In *Safety-Critical Computer Systems* [58], Storey mentions reviews, inspections and design walkthroughs as types of static testing or analysis, but again does not detail how these are generally carried out. However, he provides a useful case study illustrating how formal modelling was used retrospectively to increase confidence in the software for a nuclear power plant. This is discussed further in section 2.3. The avionic software certification standard DO-178B distinguishes between software analyses, which "provide repeatable evidence of correctness" and reviews, "which provide a qualitative assessment of correctness" [26, § 6.3]. This implies that analyses are more algorithmic approaches, which could potentially be automated, while reviews apply human expertise and opinion. "Repeatable evidence of correctness" sounds attractive, but we are dealing with standards documents that are written in natural language and have no clear correctness criterion. Hence, we shall look mainly at software review techniques—although with a view to improving their repeatability and favouring evidence over opinion.

Turning to more general software engineering literature, we find more detail about the conduct of reviews. Patton [50] identifies three general types of review. In peer reviews, a developer's peers gather to provide informal feedback on the work being reviewed. In walkthroughs, the author presents their work to the reviewers, who question and challenge it. Inspections are a more structured, formal variant of review, where the reviewers ('inspectors') are assigned specific roles or viewpoints from which to provide comment. Fagan [59], credited with the development of inspections, also recommends that rather than just "direct[ing] people to find errors", the inspectors need to be prompted and coached to identify them. He suggests using checklists of clues for finding high-cost or high-frequency issues. Humphrey [51] introduces personal reviews, which follow similar principles but are carried out by an individual, before exposure to others.

The reviews experienced by the author seem to be a blend of personal reviews and peer reviews. The majority of the work is done by individual reviewers acting independently, without the benefit of being able to ask the author for clarification. However, the resolution of the comments is often more like a peer review, with meetings held to discuss how to respond to them. In some cases, a walkthrough-like approach is used to present a particular aspect of a standard to elicit more focussed stakeholder comment. Here, we can see a potential difference between standards reviews and code reviews. Fagan recommended four people as an effective size for a code inspection team [59]. As standards are intended for re-use on multiple projects, they are likely to have a much broader range of stakeholders (and hence potential reviewers) than typical software code. This makes it unwieldy to carry out reviews as a group activity as is normally done for code reviews. With large numbers of reviewers commenting, it becomes impractical to hold meetings to discuss each comment in detail. Editorial committees are often formed of a subset of key stakeholders, and comments triaged before meetings. This means that many of the

reviewers are not present when their comments are reviewed, losing the opportunity to clarify their views. Walkthrough-style workshops can provide an opportunity to engage with more stakeholders concurrently. They can be steered by the facilitator to focus on more important areas of the standard, but can be limited by how much material can be covered in the available time.

Ould [35] argues that V&V techniques should be reliable (i.e. able to expose all faults) and economical (using minimal effort). Knight and Myers [52] have similar aims, seeking a review tool that provides repeatability while making the best use of human resources. They approach this problem by breaking the review task into smaller 'phases' and using more computer support. In contrast, Humphrey [51] finds it more efficient to work with print outs for code reviews. He suggests that the small viewport of a screen makes it difficult to examine large-scale program properties such as structure or relationships. He also holds that working on paper makes it easier to review emergent properties such as security or performance, but it is less obvious why working offline should be helpful in this respect. Ould [35] maintains that the potential for V&V arises from formalism. He argues that the more structure and formality that is involved in creating a product, the more well-defined its meaning, and the easier it is to check. This agrees with Fagan [59], who held that compared to less structured walkthroughs, they were a "*formal, efficient* and *economical* method of finding errors", providing improved productivity through lower rework costs. In Ould's view [35], structured walkthroughs suffer from focusing on what is there, and overlook omissions. In inspections, this is addressed by using a specific coordinator role to avoid any focus on trivia, and checklists to prompt for omissions. Gathering statistics from previous inspections allows these checklists to be updated and improved [51, 59].

In practice, these approaches are not mutually exclusive, and we can combine the best parts of each of them. Patton [50] identifies four elements essential to carrying out formal reviews: identifying problems, prior preparation, producing tangible written outputs, and following fixed rules. Identifying problems contrasts with attacking people, or getting drawn into solving the problems during the review. Following rules is necessary for repeatability and to manage expectations. If we take identifying problems and producing tangible outputs as hygiene factors that should be basic objectives of any review, we can focus this project on addressing the need for preparation and repeatability. We will seek to develop guidance that builds on the good practice outlined above. This will help the sponsor of a standard set both the rules for the method by which a software safety standard might be reviewed and the conformance criteria to judge it by. The next question is then, what criteria should be built into these rules?

2.2.2 Quality criteria for standards

One criterion might be that standards should meet a 'standard for standards'. Ould [35] advocates performing validation and verification on early development items such as specifications, recommending comparing them against a relevant standard. Similarly, Patton [50] maintains that "code that follows set standards and guidelines is easier to read, understand and maintain". One could argue that the same could be true for a standard. Indeed, the MOD did publish a standard for Defence Standards, *Def Stan 00-00—Standards for Defence*. Part 2 focused on *Management and Production of Defence Standards*, including requirements and guidance for the formatting and structure of standards. It also included some specific content issues such as the need to consider the environmental impact of the standard [60]. Def Stan 00-00 was withdrawn in 2012

in favour of the International Organisation for Standardization (ISO)/International Electrotechnical Commission (IEC) Directives [61] and some much briefer guidance on fonts and heading styles [62].

Def Stan 00-00 did include a facility for draft standards to be published with 'Interim' status to stimulate comments, as a type of peer review before full publication. There was also a requirement for standards to be periodically reviewed for currency, with the aim of determining whether they could be confirmed as fit for purpose, needed amendment, or could be withdrawn or made obsolescent. However, no particular method or process was given for carrying out these reviews [60]. Similarly, the ISO/IEC Directives require standards to be reviewed by various groups (e.g. committees, general public) as part of the development and maintenance process, but do not suggest how this should be undertaken. Helpfully though, they do include a checklist for writers and editors [61, Annex A]. One might expect this to have been applied if the new DStan guidance has been followed.

Patton [50] argues that good documentation improves software usability and reliability and lowers support costs. Again, one might argue that for standards, similar factors apply. Standards that are hard to read and understand are less usable. If there is ambiguity in what they ask for, they will be less reliable in producing the desired outcome. Badly written standards may directly increase software development costs if suppliers factor in a larger risk budget to address the challenges of understanding and meeting them. They can also increase the overhead costs of software procurement and maintenance of the standard, by provoking clarification questions that have to be resolved.

Patton maintains that "from a software quality and testing perspective style doesn't matter" and a difference in style "isn't a bug" [50]. However, in a standard, style may be a more important factor, as it contributes to the readability, and hence usability of the document. It is perhaps telling that when DStan withdrew Def Stan 00-00, the drafting guidance that they retained [62] related mainly to presentational style.

Patton and Ould provide guidance on reviewing specifications that appears also to be applicable to standards, as they are a type of high-level specification. Patton [50] recommends first taking a high-level view to search for fundamental problems, oversights and omissions; before looking at lower-level issues. For the high-level review, he recommends the reviewer gain inspiration by putting themselves in the customer's place, researching relevant standards and guidelines, and looking at problems with similar products to identify read-across. At the detailed level, he suggests checklists of desirable attributes (e.g. completeness, consistency) and problematic terminology (e.g. 'usually', 'etc.'). Similarly, Ould [35] recommends checking specifications for internal consistency, and comparing them against any superior specification to ensure that there is completeness and behavioural equivalence in fulfilling higher-level requirements. His list of positive attributes for design (feasibility, economy, robustness and flexibility) also appears relevant to standards or specifications [35].

As standards set requirements, guidance from the requirements engineering field also suggests quality criteria for standards. Hull et al. state that "the criteria for writing a good requirement are exactly those criteria against which the requirement should be reviewed" [63]. They address requirements both individually and as a collection, with an emphasis on attributes that affect how the requirement can be shown to be satisfied. Individual requirements must be *atomic* and *unique*ly identifiable, so that they can be referenced. It must be both *feasible* and *legal* to satisfy them, and to allow this, they must be written in a way that is *clear* and *precise*. Requirements must be *verifiable*, so that one

can tell whether they have been satisfied; and they must be sufficiently *abstract* that they do not prescribe a particular implementation. As a collection, requirements should form a *complete* set; be *non-redundant* and *consistent* with each other; and be well-*structured* and *modular*, so that like requirements are grouped together. Each requirement should either be *satisfied* by a lower-level requirement, or *qualified* by some means of checking the requirement has been met [63].

All the criteria discussed above appear just as relevant to the requirements set by a software safety assurance standard as to other types of requirement. As the standard will form part of a contractual arrangement, the supplier will need to be able to demonstrate that each requirement has been met. It is helpful to write each individual requirement in a way that facilitates this, and to structure the whole standard to make it easy to understand and avoid redundant effort. It increases the V&V potential of the standard if the structure allows the contractor's efforts to be linked to satisfaction of particular requirements and traced through to their contribution to the safety of the overall system.

Wilkinson [64] discusses problems with contractual system safety requirements, identifying similar issues around vague and confusing terminology. He also identifies several potential types of issue relating to referenced-out requirements. Some contractual clauses can expand to "too many requirements", for example citing multiple specifications where one would do or giving a directive to comply with "all relevant legislation and standards". In other cases, requirements cite outdated specifications, or none at all, increasing the risk that a contractually compliant response will not meet the intent of the contract.

DO-178B sets seven objectives for high-level software requirement reviews: they should show whether requirements are compliant with system requirements; correct in their algorithms; compatible with the target computer; accurate and consistent; traceable; verifiable; and conformant with standards [26, § 6.3.1]. The last four objectives seem appropriate for software safety standards reviews without modification and align with the recommendations discussed above. We can interpret the others as follows:

Compliance with requirements: The requirements set by the standard should be sufficient to ensure that the aim of the standard is met.
Algorithm aspects: Processes and practices set out by the standard should be sensible, effective and efficient in meeting their aims.
Compatibility: Practices described by the standard should be compatible with the organizations and capabilities of the actors involved, such that they are able to fulfil obligations placed on them by the standard.

2.2.3 Guidance for reviews

The material discussed above seems to be largely consistent and non-contentious, and could potentially be incorporated in guidance for standards reviewers. First though, we wish to understand how effective these methods will be when applied to standards rather than software. Unfortunately, as peer reviews, walkthroughs and inspections are social methods, it is not feasible to compare their effectiveness within the scope of this MSc project. However, the guidance on these methods appears common sense, and is an improvement on the existing lack of available guidance for standards reviews. It therefore seems worth discussing in the framework at Appendix D anyway. The material about checklists and quality criteria for requirements also appears helpful. As these methods can be used without needing multiple reviewers, we can test them out on Def Stan 00-055 to check how effective they are and see what can be learnt from their practical application.

This experimentation is described in Chapter 3.

2.3 Approaches to modelling standards

Reviewing the text appears to be an effective generic V&V technique that can be applied to standards, but these reviews are formal only in terms of their process, not the treatment of their underpinning semantics. Reviews of text-based documents are also not especially effective in uncovering high-level issues with strategy or design [35, 51]. Some formality can be introduced through using inspection techniques with rules, criteria and checklists, but to gain greater V&V potential we somehow need to exploit the underlying structure of the standard. Ould recommends using diagrams and abstraction to help identify high-level faults in strategy [35]. Similarly, Humphrey finds designs easier to review when using a condensed design format such as pseudocode or a mathematical notation [51]. This all points to reviewing some kind of abstract version of the standard, suggesting that a modelling approach might be useful.

Storey [58] gives a case study from the Darlington nuclear power plant, where mathematical modelling was used as a review technique to increase the regulator's confidence in the plant's control software. The software's requirements were translated into a formal mathematical model, function tables generated from an examination of the code, then the two compared to produce 'proofs' that the functions correctly implemented the requirements. This work was expensive to do retrospectively (it would have been cheaper to do concurrently with the initial development), and did not reveal many faults. However, Storey reports that important safety-related faults were found and led to modifications to the code. The exercise was found cost-effective compared to reworking the system to use a design that could have been more easily verified. It is not clear that software safety assurance standards are as amenable to mathematical representation as the reactor control software. However, there are other modelling techniques available to us from elsewhere in software assurance that may be more helpful. In particular, we turn to the software safety assurance field itself, since it is concerned with judging the adequacy of work products.

2.3.1 Argument modelling

One method commonly used to construct a safety case for software is to build an argument structure. This shows the claims being made about the software and links them to supporting evidence, forming a software safety argument [65]. Making the argument explicit is intended to help the author explain their safety case and let reviewers identify problems in its logic. Where a text-based safety case does not already use an argument structure, a reviewer can try to reverse-engineer one to identify flaws or omissions in its implicit arguments. Similar arguments can also be made about security and other software properties. In general these are known as assurance arguments.

The assurance argument approach can be also applied to standards. One can construct an argument to represent how the standard's aims are to be met, then use this structure as a drafting framework. The argument structure can be reviewed for completeness and consistency, then the standard can be verified against it. The author used this method successfully in 2011 to draft a new issue of Joint Service Publication (JSP) 430, an MOD maritime safety policy document [66]. Even when this method of drafting has not been used, a standard can be modelled retrospectively as an argument structure to facilitate review. This approach does not provide validation of whether the standard actually works

(evidence from its practical use would be required). Despite this, argument models can help reviewers verify that the high-level goals of a software safety standard have been properly decomposed into requirements placed on the software. They can also check that meeting these low-level requirements will plausibly satisfy the overall goal. When using this approach, the need to set a top-level goal forces modellers to address Graydon and Holloway's point head-on: deciding "what it means for a standard to 'work' " [49].

Previous research into modelling the argument structures underlying software standards has focused on the DO-178B software certification standard [26] and the Common Criteria for security [67]. Ankrum and Kromholz [68] modelled these standards to investigate whether different assurance standards shared common structures, and whether the standard's structure implied a particular assurance case structure (finding neither generally true). Galloway et al. [69] reverse-engineered part of DO-178B's safety argument to argue that an alternative analysis technique was at least as convincing as the one recommended by the standard. To do this, they had to deduce what goals DO-178B's writers were attempting to achieve, so that they could argue that their proposal was equivalent. Argument structures are clearly useful models for demonstrating (and allowing challenge of) the goals that are expected to be satisfied by following a standard. They can also help identify omissions or ambiguities. Graydon and Kelly [55] go further, suggesting that such models can demonstrate whether conformance with a standard will be sufficient to demonstrate properties such as safety or security. This could help show whether the standard was effective.

Many graphical and symbolic notations are available for representing argument structures, with different styles favoured in different fields [70, 71]. Two similar notations have found widespread use in software safety arguments [55]: Adelard Safety Case Development methodology (ASCAD) and Goal Structuring Notation (GSN). These represent arguments as tree structures, with a cascade of sub-arguments supporting a single top-level claim or goal. ASCAD [72] has three types of node: *claims* nodes link to supporting *evidence* nodes via layers of *argument* (the notation is also known as Claims, Arguments, Evidence (CAE)). GSN [73] is similar, with layers of *goals* and sub-goals linking to *solutions* via *strategy* that explains the argument, as shown in Figure 2.1. GSN also has node types to represent *context*, *assumptions* and *justification*; extensions for modular safety arguments; and argument patterns that can be instantiated with optional or repeated elements.

While some choose ASCAD for its simplicity [68], GSN appears to have greater expressive power: with more node types available, it is better able to represent the reasoning behind arguments. In choosing methods for producing software artefacts, Ould [35] recommends selecting those that have a notation with well-defined semantics, heuristics for development, and good V&V potential. GSN appears to better fit this recommendation and was used in the majority of work discussed here [55, 69, 74]. Its notation is well-defined in a public standard that also contains recommended processes for constructing and evaluating goal structures [73].

Constructing an argument model from a standard is not without its challenges. Some standards like the Common Criteria are written in a way that makes the reasoning behind them relatively obvious [55, 68]. In others like DO-178B, the argument is implicit and difficult to infer from the requirements [68, 69, 74]. The *Explicate '78* project attempted to build an assurance argument from DO-178C [75]. It followed principles of minimal speculation and faithfulness to the text, and used expert review to ensure the fidelity of the argument. Holloway and Graydon [74] report that while experts agreed on the need

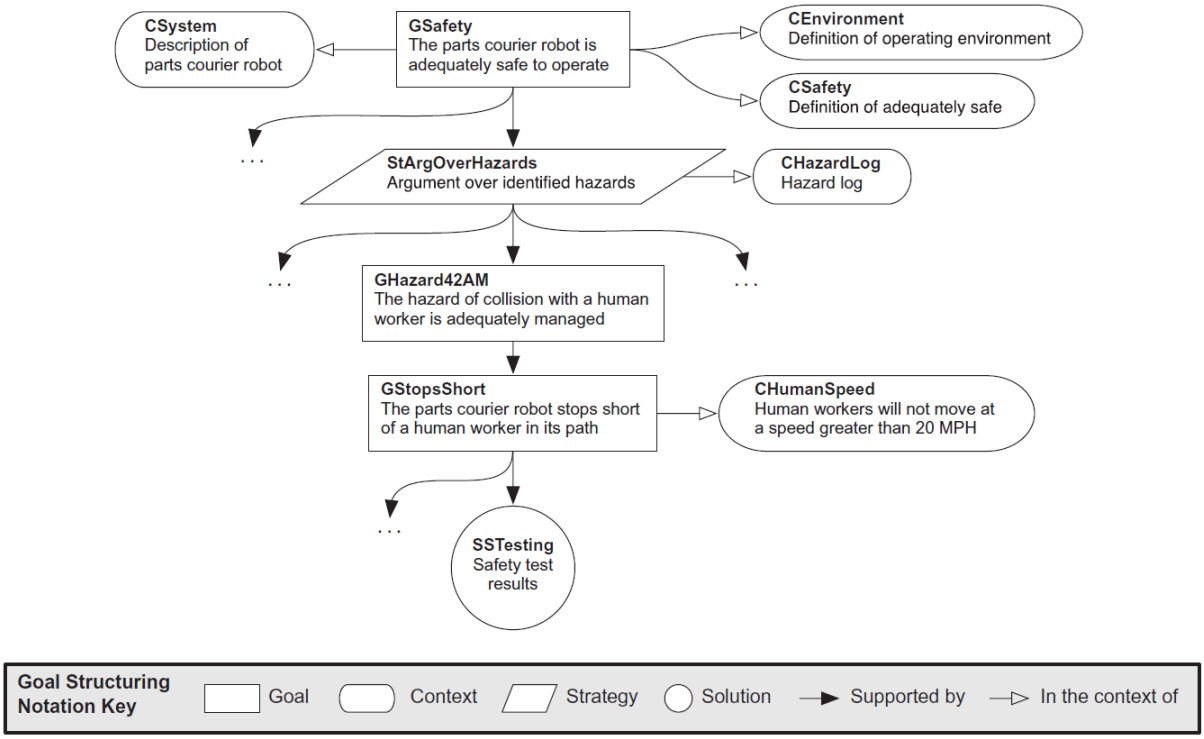

Figure 2.1: Example GSN argument fragment, from [55].

for various DO-178C requirements, there was less consensus on the reasoning behind them, weakening their contribution to the overall argument. Despite this, the project made useful observations, including that the adequacy of the standard depended on the specifics of the project involved.

The GSN Community Standard [73] describes how to construct a GSN argument either top-down or bottom-up. The top-down method works by elaborating and decomposing an initial top-level goal into the low-level goals that will satisfy it, and the evidence that will show it has been achieved. It might be useful for helping in the design of a standard, but is inappropriate for review. We need an argument that represents the document under review, rather than the reviewer's opinion on how to meet the top-level goal. The bottom-up method involves identifying evidence to use as the argument's solution nodes, inferring the claims they support, then deriving higher-level claims from the low-level arguments. While some kind of bottom-up strategy seems more appropriate, we cannot use this method either. There are potentially many claims that a given piece of evidence could support, leading to many potential argument structures. The GSN standard does not explain how to consistently derive the higher-level argument. We need a method with less potential for variation, which will generate an argument closer to the text of the standard under review.

It appears that the only source that gives practical, tailored advice on building arguments for review of software assurance standards is provided by Graydon and Kelly [55]. They propose a review method for evaluating standards that could be applied to Def Stan 00-055, which includes guidance on capturing the underlying argument of a standard. An analyst captures the argument by identifying a top-level goal that the standard supports, then its low-level requirements and the evidence that is supposed to show that they have been satisfied. The analyst then attempts to identify intermediate goals from the objectives of groups of requirements, before subjecting the argument to criticism.

Care must be taken with this approach, to ensure that the evaluation tests the standard itself, rather than the faithfulness of the model. Graydon and Kelly warn the analyst to follow the text of the standard as accurately as possible when creating the argument. Confusing text and logic gaps should be left in if necessary, so that criticism of the argument will be valid criticism of the standard. However, they also suggest that the top-level and intermediate goals may not be explicit in the standard and should be determined by the analyst. Similarly, the evaluation process asks the analyst to review the reasonableness of implicit assumptions that they have themselves inferred from the text [55]. If the analyst is evaluating the standard to assess its fitness for the purpose of a particular application, it seems reasonable for them to set a top-level goal that matches that application. But when reviewing the standard on its own merits, they should use the intended purpose of the standard—which hopefully should be apparent from the text. Either way, they must avoid the temptation to fabricate too much of the intermediate argument where it is not supported by explanation in the text of the standard or supporting guidance.

Once the argument has been built, Graydon and Kelly suggest evaluating it a fragment at a time using the following heuristics, which address both the standard's adequacy and structure [55]:

1. Consider how the argument might be vague or subject to misinterpretation.
2. Draw out implicit assumptions.
3. Judge the necessity and reasonableness of each assumption (implicit or explicit).
4. Search the argument for well-known fallacies.
5. Identify where 'independent' lines of reasoning depend upon common sub-arguments.
6. Consider expected but omitted practice to see if the argument could practicably be strengthened.
7. Determine whether negative experience with similar systems might provide counter-evidence.
8. Judge (subjectively) the strength of the argument.

Hawkins and Kelly [65] took a different approach, applying the Hazard and Operability Study (HAZOPS) guidewords to the stages of Kelly's "six-step method" [55] for argument construction. From this they derived sixteen potential areas where confidence in a safety argument could be strengthened. HAZOPS is a structured method used in safety and reliability to improving the robustness of a process. It uses guidewords (*no* or *none*, *more*, *less*, *as well as*, *part of*, *other than* and *reverse*) to suggest potential deviations that can be addressed [76]. When applied to the argument construction process, these guidewords suggest potential weaknesses in the resulting argument that Hawkins and Kelly term "assurance deficits". As an example, applying the keyword *none* to the step of defining the supporting basis for a goal suggests a lack of scope or context for that goal. This could increase the uncertainty around whether it could be claimed to be satisfied. Although proposed for the context of software safety arguments, Graydon and Kelly's findings appear generic and could be distilled into guidance on potential pitfalls for use when evaluating the argument behind a standard.

Further guidance on evaluating arguments is available in the GSN Community Standard [73]. This contains a four-step process for reviewing an argument:

1. Argument comprehension; where the reviewer checks that they understand the argument, and that it represents the written text.

2. Well-formedness checks; where the reviewer looks for gaps in the argument, or parts that do not contribute to the overall goal.
3. Expressive sufficiency checks; to reveal where the standard is ambiguous or does not make its strategy clear.
4. Argument criticism and defeat; to check whether it is plausible that the argument can be satisfied. The lower-level goals and strategies should credibly support the top-level goal; and it should be reasonable to generate the necessary supporting evidence.

Helpfully, the first stage recognizes that the argument may need to be extracted from a plain text document, rather than being initially presented as a GSN structure. It also makes no assumption about the type of argument under review. These factors make it more relevant to the application of reviewing a text-based standard than some other sources. It gives detailed high-level guidance, providing tips for the conduct of the review at document level. In contrast, the papers by Graydon, Hawkins and Kelly [32, 55] focused on specific nodes of the argument structure.

2.3.2 Relationship modelling

While argument structures are useful, as Ankrum and Kromholz point out [68], they do not well represent the dynamic interactions involved or the processes that are required for compliance. They present a picture that represents the outcome of successfully applying a standard, but are not well suited to assessing how practical this might be to achieve.

Applying a standard such as Def Stan 00-055 requires various interactions between two contracting parties: agreements, decisions, and exchange of information between both parties. These must take place at logical points in the lifecycle of the software product. As Groom [77] points out, the contract for provision of the software (which should cite the Defence Standard), also has a lifecycle of its own that impacts on the software development process. An assessment of the practical effectiveness of the standard must take these factors into account, and requires a modelling technique that is more expressive than an argument structure.

There is a growing trend in industry to management of complexity through Model-Based Systems Engineering and model-based techniques for software development. DO-178 is now supported by guidance (DO-331 [78]) on how to treat software that has been developed through such techniques, although it does not cover use of models for verification purposes, or how models can support automatic verification [79]. There is no industry consensus yet on how these should link to safety engineering [80], but several lines of research have investigated applying similar techniques to standards.

In 2009, the MOD Future Logistics Information System (FLIS) team built a model of Def Stan 00-56 Issue 4 using the MOD Architecture Framework (MODAF) which the author was invited to review. MODAF is an enterprise architecture framework. It defines 46 standard views for presenting information about an enterprise, grouped into seven 'viewpoints' for different stakeholders[5] [81]. The FLIS team was already using MODAF to plan their business processes and enterprise architecture. They intended to use the 00-56 model to help communicate how their delivery partner would have to interface with the MOD in order to meet their obligations for safety management.

[5]'All Views' (AV), Strategic (StV), Operational (OV), System (SV), Technical (TV), Acquisition (AcV) and Service Oriented (SOV) Views

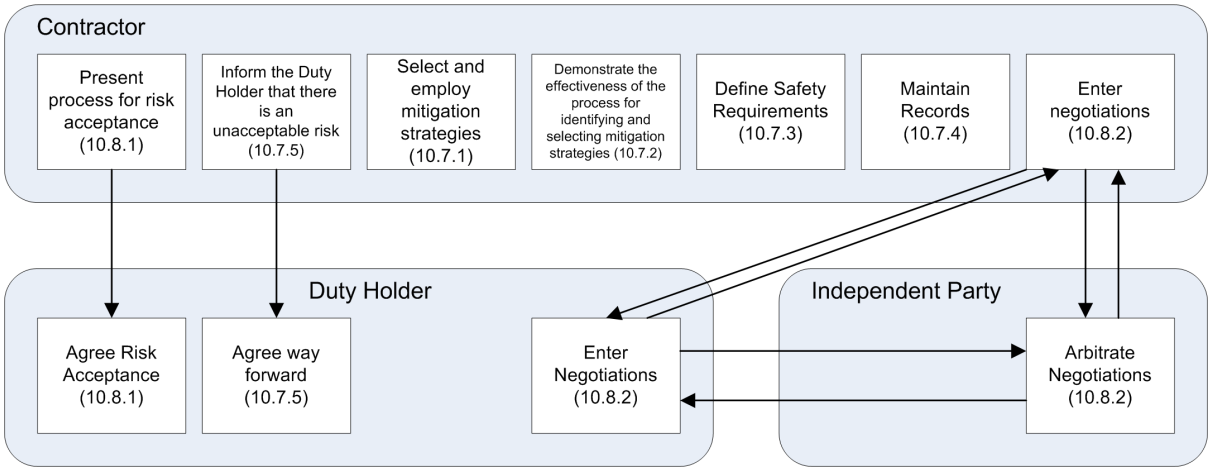

Figure 2.2: MODAF OV-5 view of the Def Stan 00-56 Issue 4 Risk Reduction process, from [82].

Inspired by the FLIS work, in 2011 the author tasked the MOD Systems Engineering and Integration Group (SEIG) to review Def Stan 00-56 using MODAF views to highlight inconsistencies or ambiguities in the standard. This resulted in a strategic view of the standard's vision and key requirements (StV-1); a set of operational (OV-5) views that showed which requirements of the standard were fulfilled by different actors (such as the MOD, the Contractor, etc.); and a stakeholder map (OV-2) [82]. An example is shown at Figure 2.2.

The level of detail presented in the SEIG work was relatively limited, showing some interactions (e.g. requirements for agreements between actors) and other linkages, but not the detail of sequencing or flows of process or information. Despite this, they were still able to reveal a number of issues with the standard including duplications, ambiguities and omissions. The report also recommended structuring the standard better to group relevant requirements, helping contractors differentiate the normative parts (requiring evidence of compliance) from the purely informative [82]. MODAF is a primarily MOD tool, but others such as Watkinson [83] have also found frameworks like The Open Group Architecture Framework (TOGAF) [84] useful to structure work on safety critical systems. TOGAF does not define standard views [84, Ch 31], and while MODAF does do so, it does not specify how to present information within these views. As one is free to choose suitable notations, the MODAF approach could be used to present information from different viewpoints with other, more expressive notations than were used by SEIG.

De la Vara et al. [85] describe a Reference Assurance Framework (RAF) metamodel developed as part of the European Union-funded OPENCOSS safety assurance research project. We could use this to model the relationships in Def Stan 00-055 more expressively. Their metamodel provides a framework for modelling different RAFs (such as assurance standards or similar), and mapping these to project-specific models that show how their requirements are satisfied. It was motivated by the need for suppliers to understand multiple safety assurance standards and cost-effectively demonstrate compliance with each. The RAF metamodel includes elements for modelling common features of safety assurance standards such as requirements, processes, techniques and information artefacts; and the relationships between them. Having validated their work through industrial case studies, De la Vara et al. claim that "models can facilitate the understanding of safety standards, the identification of inconsistencies in their text, the determination

of the evidence to collect, the specification of traceability requirements, and compliance assessment". This sounds promising for our purposes, as a tool that can make a standard easier to understand should help reviewers identify problems with it. Unfortunately, the majority of the validation effort seems to have focused on the latter three benefits. The authors admit that little work has been done so far to model safety standards for the purpose of analysing them to identify issues. In fact, the three papers they cite as safety regulation modelling examples were all aimed at enabling a regulated community to identify relevant requirements in a large regulatory corpus. One paper they cited as seeking to identify conflicts and inconsistencies in the text of standards did use a similar domain metamodel, but only to improve the consistency of search results in a regulatory documentation browsing application [86]. Rather than modelling, another paper used natural language processing to structure regulatory documents by the similarity of their requirements using lexical clustering [87]. The third described annotating regulatory documents with metadata to help users identify relevant requirements [88].

One area not covered by de la Vara et al. [85] is the modelling of the process implied by a standard. Their paper presents models of IEC 61508 and the RAF metamodel itself using Unified Modelling Language (UML) [89] class and object diagrams, which show static properties of the models. They also discuss modelling the mappings between models of different assurance frameworks and models of projects' compliance efforts. Process aspects may have been overlooked since IEC 61508 and its derivative standards already contain high-level process diagrams, so the need may be less obvious. However, they do not consider that different requirements might apply at different times, and that there can be issues of sequencing and refinement in how artefacts are developed. Joliffe [17] reports that a process diagram was generated as part of the early drafting of Def Stan 00-055 Issue 3. The authoring team found this useful to explain how Def Stan 00-055 and Def Stan 00-056 were supposed to work together in safety requirement management process. Unfortunately this work does not appear to have been taken forward into the published versions of either standard. Process diagrams built from elements modelled using the RAF meta-model could be more consistent with the overall model of a standard, and would fit into frameworks such as MODAF as OV-5 Operational Activity Models. UML includes notations such as sequence diagrams and activity diagrams that can be used to capture interactions and processes. Other notations such as the Business Process Model and Notation (BPMN) [90] are available. These could present the dynamic aspects of standards in a more expressive way than the simple OV-5 example presented in Figure 2.2, allowing better opportunity for review and constructive criticism.

2.3.3 Filter model

Steele and Knight [56] propose a different modelling approach to enable a "relatively rigorous standards development and assessment process". They claim that the approval process for a safety-critical system can itself be characterized as a safety-critical system, and modelled as a set of filter planes. The approval process, which they term 'certification', is captured in a standard and envisaged to involve creation of a 'submission package' of information artefacts associated with a system. This is then scrutinized by some kind of authority (e.g. a customer or regulator). The submission package passes through the filter planes, each of which is supposed to catch different potential flaws in the submission. Each filter has an intended way through (pass criteria), but may be flawed and not catch all the problems it is intended to. The results of the filtering process inform the final Certified / Not Certified decision. In treating this model as a

safety-critical system, they identify the hazard as a submission with unacceptable flaws passing undetected through all the filters and obtaining certification. Steele and Knight suggest that standard system safety techniques such as Fault Tree Analysis (FTA) [91] or Failure Modes Effects and Criticality Analysis (FMECA) [92] can be used to investigate how that might happen. The results of this analysis could indicate the likely effectiveness of the standard that defines the process, and suggest areas that could be strengthened.

Although Steele and Knight use software examples throughout their paper and aim their model at standards with similar purposes to Def Stan 00-055, the filter model does not look directly applicable to the Def Stan. The example standard used in their paper is the avionic certification standard, DO-178B [26], which includes large tables of recommended practices which can easily be represented as filters. While 00-055 does define a type of approval process, it does not recommend specific practices in this way, or a specific acceptance process. It is hard to link high-level objectives in 00-055 (such as having a management system in place or ensuring that safety requirements are met) to the prevention of specific flaws in software. Def Stan 00-055 bridges the gap by recommending use of a lower-level standard (such as DO-178 or IEC 61508). This hand-off of requirements makes it much harder to build the filter mod el.

Despite this, the filter model may be useful as a representation of the standards review process, rather than the standard itself. It suggests that in standards reviews, one is attempting to filter out flaws. Rather than trying to catch every flaw with a single filter, we can try to increase the effectiveness of the process by using multiple filters optimized to catch different types of flaw. We must just decide which filters to use and in what order.

2.4 Conclusions from literature review

Having discussed the guidance currently available for standards reviews and explored the literature on methods that might improve their results, we must decide which methods to test out on the standard. Published advice from standardization organizations such as DStan [60, 62], ISO and IEC [61] gives some guidance on criteria for standards to meet—in particular for formatting—but relatively little advice on how to conduct a review of a standard. It focusses more on process issues such as when reviews should occur, rather than how they should be carried out. The software engineering literature is more helpful, with various sources describing good practices for review or inspection of software artefacts [35, 50–52, 59]; these also appear applicable to assurance standards. We cannot investigate the social aspects of the reviews described in these sources, as the author does not have the opportunity to organize or observe group reviews in the scope of this project. However, we can use some general concepts from these methods. In particular, we can use the principles of prompting reviewers to consider a document from different viewpoints, and using checklists or heuristics to focus the review on important issues. The requirements engineering field suggests criteria for good software requirements [35, 50, 63], which seem directly applicable to the requirements set in standards. We can use this material to develop guidance for checklist-based reviews that improve on the traditional, naïve 'read-and-comment' approach to standards review.

Beyond traditional review styles, we have also identified model-based review techniques using argument structures [55] or metamodels [85] that promise a more thorough, structured approach to standards review. Both of these appear worth experimenting with. A third modelling method, using a filter model [56], looks like it might have appli-

cation to some standards, but is not especially appropriate for Def Stan 00-055. It seems to assume a style of standard that recommends many specific techniques, whereas 00-055 is more goal-based and recommends higher-level processes. Given the time constraints on the project, this method will not be investigated.

To get valid results, we need a firm baseline to work from. Graydon and Kelly [55] claimed that ad hoc reviews were not appropriate for assurance standards and that reviews using argument structures were more effective and could identify more issues. While we would like to be able to make a similar claim, their paper does not provide a good supporting argument. They distinguished between specification-setting standards (that directly describe the properties of a compliant artefact) and assurance standards (that attempt to influence a desirable property like safety by setting requirements on development processes or other software attributes). However, they did not present any argument about why this distinction makes ad hoc reviews inappropriate. Moreover, the authors did not demonstrate that their method revealed more issues, as they do not compare their findings against any baseline data. Their claim is based on having found issues in DO-178B that had escaped previous reviews. They do not consider that some issues may have been known but deliberately left, or attempt to compare the number of issues revealed by different methods. Patton [50] notes that software bugs are not always fixed, either due to lack of time, risk of causing further problems, expense, or a willingness to live with the bug as a 'feature'. In Storey's Darlington nuclear power plant case study [58], only errors affecting safety were fixed—others were allowed to remain. Similarly, many issues identified by Graydon and Kelly could have been identified by traditional ad hoc reviews. They might have been known to DO-178B's authors, but consciously not addressed. Known issues in standards might not be fixed if there was disagreement that they were problematic, they were not found important enough to spend resource to fix within publication deadlines, or no consensus could be reached on a resolution.

To ensure that this project makes a fair comparison of methods, we will need to take the above issues into account. Def Stan 00-055 Issue 4 has already been subject to a reasonably thorough ad hoc review process. This included internal review by an authoring team, scrutiny by the MOD SSRC, public consultation, and the opportunity of a year's experience of Interim Issue 3 to generate feedback on practical problems. While there are doubtless problems with the published version, we do not know how many of these might already have been identified, or could be found through a more rigorous application of traditional techniques. To use a review of the published Issue 4 to demonstrate the power of advanced structured review techniques like argument or relationship modelling, we need to generate a baseline to compare our results against. We can do this by first carrying out a naïve review (reading the standard and noting any apparent problems), before applying the methods selected above. We will then be able to compare the number and type of issues identified by the different techniques to determine how much they improve over traditional methods.

3 Application of standards review methods

Much of the material discussed in Chapter 2 appears to have potential as a source of guidance for standards review. However, we wish to determine whether this is actually the case, and which of the methods discussed will be most useful. To do this, we need to experiment by practically applying the methods. In this chapter, we describe how the material from the literature review has been adapted and applied in practice, and summarize the results.

3.1 Traditional naïve review

An initial 'naïve' review was carried out to provide a baseline for comparison of other review methods. The author printed a copy of Def Stan 00-055 and read through it, using a coloured pen to highlight concerns and note issues in the margins. The review was primarily carried out on hard copy, the author finding it easier to spot errors in printed documents than on-screen. It is also easier to flick between pages to compare text. An electronic version was also used to allow easier searching for cross-references.

A second pass was then carried out, transcribing the paper annotations into a tabular comment form in the style of those used by DStan or the British Standards Institute (BSI) [15, 54]. A reference, comment and proposal were listed for each issue. To allow production of summary statistics, additional columns were used to record the type and rough severity of each issue, the results of which are summarized in Table 3.2. The only classification guidance identified in the literature review had been to classify issues as general, technical or editorial [54]. This did not appear to give sufficient resolution to allow a good comparison of the types of issue found by different methods, so classifications were suggested by the author based on the issues found. No guidance was found on classifying the impact of an issue, so the scale shown in Table 3.1 is proposed.

The first pass review took approximately 4.5 hours, mainly consisting of reading time. The second pass took a further 6 hours, with the majority of the time spent describing the problems identified and potential fixes. Both passes were spread over a number

Impact	**Descriptor**
High	Issues that appear to compromise the intent of the document.
Medium	Issues that affect the meaning of the document, but do not appear to compromise its intent.
Low	Incorrect, or makes the text harder to understand, but does not significantly affect the meaning of the document.
Readability	Issues of punctuation, grammar, style and similar that detract from the readability of the document, but are not otherwise incorrect.

Table 3.1: Issue impact descriptors.

Issue type	**Impact**					**Total**
	High	Medium	Low	Readability	Stet	
Grammar			1	37	1	39
Ambiguous	3	18	8		1	30
Punctuation			1	23		24
Outdated	1		11		1	13
Factual error		3	9		1	13
Acronym issue			4	3	2	9
Formatting			1	7		8
Omitted words			7			7
Inconsistency		3	1		3	7
Style				5		5
Speculation		3	1			4
Potential for enhancement			3			3
Duplication			2			2
Omission	2					2
Non-atomic requirement			2			2
Spelling				1		1
Incomplete list	1					1
Grand Total	**7**	**27**	**51**	**76**	**9**	**170**

Table 3.2: Summary of issues from naïve review, by impact and type classification

Review pass	**Impact**					**Total**
	High	Medium	Low	Readability	Stet	
Pass 1	6	22	44	73	9	154
Pass 2	1	5	7	3		16
Grand Total	**7**	**27**	**51**	**76**	**9**	**170**

Table 3.3: Summary of issues from naïve review, by impact and identification point.

of sessions to avoid the effect of fatigue. (After more than approximately 45 minutes reviewing, the author found himself losing concentration or falling asleep). Although the second pass was only intended to record issues that had already been identified, further issues spotted at the same time were also recorded. At this stage the author also decided that on reflection, some issues identified during the first pass were not valid (classified 'stet' in Table 3.2). Table 3.3 summarizes the results from the two passes.

The majority of the issues identified were of low significance. An ironic example is the first paragraph of the Foreword. This contains a punctuation error, a spelling mistake and bad grammatical construction in the revision note that is supposed to explain that errors in the text have been fixed. It actually reads that the standard has been updated to "include . . . minor grammatical and textural [sic] errors" which, while true, is presumably not what was intended! This example is interesting from two viewpoints: it illustrates that grammatical errors can be significant as they can change or invert the meaning of the text; and equally that they are not always worth fixing. The paragraph does not affect the requirements of the standard, and will in any case be replaced by a new revision note when the next issue is published.

The review did identify some more significant issues, the most numerous being ambiguity in the text. An example is the requirement for a Programmable Element Safety Management Plan. §6.1.4 makes it clear that the plan is not a mandatory deliverable requirement of the standard, but also mentions that its contents may be included in other plans. Later clauses in the standard refer to it in both mandatory and optional terms, leaving the reader unclear on what is required in order to claim compliance.

3.2 Checklist-based reviews

3.2.1 ISO/IEC checklist review

An initial checklist-based review was carried out using the *Checklist for Writers and Editors* from the ISO/IEC Directives [61, Annex A]. The literature review found that DStan's guidance on standard development says that "Defence Standard requirements and layout align as closely as practical to the ISO/IEC Directives" [62]. It is not clear whether this is a mandate that the Directives are to be followed or is merely intended to give provenance for the style conventions used in DStan templates. Nonetheless, it was decided to apply the checklist from the Directives to investigate whether it would help find issues missed by the initial baseline review. Its effectiveness could also then be compared with checklists based on software engineering practice.

The checklist gives a set of review tasks, listing several assessments under each task. Most of the assessments require the reviewer to read a particular part of the text with a specific question in mind. Many require a qualitative assessment (e.g. "is it concise?"). Others can be systematically verified (e.g. "is each figure referred to in the text?")

Issue type	**Impact**				**Total**
	High	Medium	Low	Readability	
Normative/informative confusion		29			29
Structure			9	4	13
Acronym issue				7 (3)	7 (3)
Speculation			4 (1)		4 (1)
Omission				1	1
Ambiguous		(1)			(1)
Total issues found		30	14	15	59
Previously identified issues		(1)	(1)	(3)	(5)
New issues		29	13	12	54

Table 3.4: Summary of new and previously identified issues found during ISO/IEC checklist review, by impact and type classification (previously identified issues in brackets).

The detailed findings of this review are reported in Appendix A. In all, 54 new issues were found, shown in Table 3.4. The most significant category of issues, by number and by impact, related to the identification of which parts of the text were normative (mandatory in order to claim compliance with the standard) or purely informative. These issues included labelling of informative references as normative, and use of mandatory contractual language ('shall' clauses) in parts of the standard that should have been informative. These had at least a medium impact as they made it hard to understand which parts of the standard were actual requirements. There were also numerous minor problems with

the structure of the document that made it harder to follow or reference. Examples included poor subdivision of sections, hanging paragraphs without unique clause numbers, or missing captions on tables. These had a generally low impact. Neither category of issue had previously been identified in Table 3.2.

The checklist review also revealed several general issues around the use of language in the document. The standard defines 'shall' and 'should' to have specific contractual meaning (denoting mandatory or recommended clauses [18, § 4.2]). However, it was clear that the standard's authors had not followed the ISO/IEC convention of using 'may', 'can' or 'must' in a specific way. This was counted as a single general issue, rather than flagging up each use of these terms. Had they been used more consistently, and in line with the ISO/IEC definitions, the document might have been clearer. This issue compounds the problem mentioned above about identification of the standard's normative requirements.

The other general issue was around use of plain language. Rather than re-read the entire text, the author chose to review this using the readability statistics in Microsoft Word. The document scored a Flesch-Kincaid grade level of over 14, indicating a difficult document to read. The grade level is calculated using an empirical formula based on the average number of syllables per word and words per sentence [93]. This metric is analogous to cyclomatic complexity in code reviews. Code which is more complex is not necessarily incorrect, but may well be harder to understand and more prone to errors. A spreadsheet was also used to analyse the proportion of text made up by acronyms, finding 1201 instances of 48 acronyms making up over 6% of the word count of the standard. It could be argued that these issues are not important: the target audience of the standard is likely to have a relatively high reading age. While all the acronyms may be a little daunting, writing them longhand would have made the document somewhat longer. That said, a standard that is easier to understand is better, in that it is more likely to be followed correctly. Contractors are also likely to charge less in their tender if they see less risk in ensuring that they can understand and comply with the standard.

Using the ISO/IEC checklist did not identify as many issues as the naïve review, but it was a more focused review. It took less time to carry out (approximately 2.5 hours), and found fewer types of issue. However, the issues it did find were new: most were in two new issue categories. Of the remainder, there was relatively little overlap between the issues it found and those previously identified. On average, these issues were also of higher impact than those from the first review. This appears to support Fagan's argument [59] that using checklists makes traditional reviews more effective.

3.2.2 Enhanced checklist

Having reviewed Def Stan 00-055 using a checklist from a standardization body, we now wish to see whether a software engineering-inspired checklist will be more effective. Fagan [59] recommends using lists of likely issues (with clues to help find them), as an 'inspection specification'. His examples are too focussed on low-level coding issues to be relevant to standards reviews, but he suggests using statistics from previously identified errors to produce new lists for the problem in hand: We can analyse the results of a preliminary review, to generate an inspection specification that "condition[s the inspectors] to seek the high-occurrence, high-cost error types". We should be able to use the results of our previous two reviews (Tables 3.2 and 3.4) to bootstrap this process.

Unfortunately, turning the previous results into a new checklist is not straightforward. We want to either find common types of error more efficiently or reveal more important types of issue that were not previously obvious. The most numerous issues in Table 3.2,

relate to grammar, punctuation, omitted words or spelling (42%). One might hope to detect them more efficiently using the proofing tools in a word processor. This was not the case here: the standard's authors had presumably already used Microsoft Word's spelling and grammar check. It did not reveal any of the issues identified in the manual review and it is not clear that a checklist would have helped either. To find these issues more efficiently than manual proofreading, we need a more powerful proofing tool instead. Improving on the issues from Table 3.4 does not look promising either. These have already been generated using a checklist. Re-applying similar checks might reveal some errors that were missed before, but is unlikely to lead to much improvement. There are some opportunities though. Another large group of issues from the first review concerns ambiguity, currency of information, or factual errors (33%). Detecting these issues seems to need qualitative human judgement. However, there may be clues we can put in a checklist to find them more efficiently. We may also be able to provide prompts for classes of issue that were not previously found.

To build this new checklist, we can consider the attributes of good requirements discussed in Section 2.2.2. We can synthesize together the guidance and criteria in DO-178B [26, § 6.3.1] and suggested by Hull et al., Ould and Patton [35, 50, 63] to produce a new checklist tailored for review of standards. Many of these sources suggest similar criteria for drafting or reviewing requirements. Appendix B shows how these have been mapped to create the following list of 17 positive attributes of good requirements for standards:

- Abstraction
- Accuracy
- Atomicity
- Clarity
- Completeness
- Consistency
- Currency
- Feasibility
- Legality
- Modularity
- Non-redundancy
- Precision
- Relevance
- Satisfaction
- Structure
- Uniqueness
- Verifiability

This list of attributes was then applied as a checklist for a further review of Def Stan 00-055. Where possible, keyword searches were used to quickly identify potential problems, e.g. searching for 'and' to identify non-atomic requirements. The search results were then reviewed manually, to look for issues relevant to the attribute in question. This approach was not possible for all the attributes. Some, such as 'legality' did not have obvious keywords to act as clues. Others, such as 'consistency' or 'structure', required comparisons to be made between separate clauses, and could not be evaluated individually. To address these attributes, the document was read through, using the checklist as a prompt when reviewing each clause.

Overall this exercise took around six hours. The results are shown in Table 3.5. It can be seen that using this checklist resulted in more issues being identified than the ISO/IEC checklist. The average impact of the issues identified was also higher. Interestingly though, only 10% of the issues it found had been identified by a previous review. That said, although this review focussed on issues around the attributes included in the list above, a few new issues (8%) were identified from other categories. Some of the more significant issues are summarized below against the relevant checklist attributes:

atomicity §6.1.6 says "The Contractor should define a Programmable Element Safety Management Plan (PESMP) and carry out a preliminary PE Failure Assessment as part of the tendering process and formalize and agree the plan and assessment findings with the MOD at Contract award." This combines at least four requirements: defining

Issue type	**Impact**				**Total**
	High	Medium	Low	Readability	
Atomicity		1	19 (2)		20 (2)
Completeness		10	2		12
Precision	4	4 (2)	2		10 (2)
Clarity	3	2 (3)	2 (1)	2	9 (4)
Feasibility	3	6			9
Verifiability		7			7
Consistency	1	3	1		5
Relevance		2			2
Satisfaction		2			2
Currency			(1)		(1)
Abstraction					
Accuracy					
Legality					
Modularity					
Non-redundancy					
Structure					
Uniqueness					
Normative/informative confusion	5				5
Grammar				1	1
Speculation	1				1
Total issues found	17	42	30	3	92
Previously identified issues		(5)	(4)		(9)
New issues	**17**	**37**	**26**	**3**	**83**

Table 3.5: Summary of new and previously identified issues found during enhanced checklist review, by impact and type classification (previously identified issues in brackets).

the plan, doing an assessment, formalizing the documents, and agreeing the findings. Some of these have to be done before contract award (to include in a tender). The others have to be done after the tender has been assessed. This means that a potential contractor can only show partial compliance with the requirement at the time their tender is assessed, and will have to revisit it later. (Medium impact)

clarity §8.3.1 says that "where there is a shortfall in evidence ... the Contractor shall make use of replacement or enhanced PE Open Standards". This is probably trying to say that if the main open standard does not address the 'military delta', additional measures need to be used. However, it can be read to mean that when the contractor fails to provide evidence to satisfy one standard, they can use a different one.

completeness The PESMP Data Item Definition (DID) says that the PESMP "should clearly identify important aspects" of the management arrangements. It lists some things to be included, but the list ends with "etc." It is not clear what other things might be necessary, or if the plan would be adequate without them. (Medium impact)

consistency The PESMP DID asks the contractor to set the terms of reference and membership of a software project's Safety Committee. However, the body of the standard implies the Committee should be MOD-led, as it adjudicates on the adequacy of the contractor's work. This ought to be a customer responsibility. (High impact)

feasibility §7.2 requires the contractor "to identify and assess the consequences of PE behaviour within the intended military operational environment". It is not clear that a contractor will necessarily understand enough about the operational environment or the wider system containing the PE to be able to do this. (High impact)

§6.1.6 requires a preliminary PE Failure Assessment to be delivered at the tendering stage. As above, this is not feasible without knowledge of the higher-level system and the military environment in which it will be deployed. This information is even less likely to be available at tender stage, before the software developer has a contractual relationship with the MOD. (Medium impact)

precision The PESMP "should identify both primary and ancillary PE . . . where they are safety-related". However, there is no definition of what safety-related means, or how closely related 'ancillary PE' needs to be to fall into the scope of the PESMP.

verifiability The standard calls for use of 'recognized good practice'. The contractor has to justify the practices they propose to use, but the standard does not reveal how the MOD will verify if they are sufficiently good or well-recognized. (Medium impact)

Despite use of the checklist, no issues were observed against some attributes listed in Table 3.5. For attributes like 'legality', it may be that there are no problems to be found. For others, it seems more likely that issues exist, but were not discovered. Many of these attributes seem to apply to comparisons of multiple requirements ('non-redundancy', 'uniqueness') or the requirement set as a whole ('modularity', 'structure').

To check the truth of the assumption that more of these issues existed, a more focused experiment was carried out to look for 'modularity' issues. One deliverable required by Def Stan 00-055 is a Programmable Element Safety Summary (PESS) report. The requirements, scope and content of this report are described in a DID in the standard's annexes. If 00-055 were highly modular, all the requirements for the PESS would be in the DID, with only loosely coupled references elsewhere. To check this, a keyword search was carried out for 'PESS' in the main body of the standard. The clauses mentioning it were then compared against the DID. This check found six mentions of the PESS in the body of 00-055 that did not correspond to anything in the DID (low impact 'modularity' issues). There was also a requirement where the body referred to 'interim versions' and the DID to 'preliminary versions' ('consistency' issue with readability impact). More importantly, while the body required the PESS report to "record the results of PE Failure Assessment", the DID says it "supports the conduct of PE Failure Assessment" or "supplements the PE Failure Assessment report". This contradiction means the intent of the report is not clear (a high impact 'consistency' issue). Only in one case did a reference to the PESS report from the body of Def Stan 00-055 map cleanly to text in the DID. These results suggest that, while some attributes like 'modularity' may be valid for use in the checklist, more guidance is needed to help a reviewer to understand how to apply them effectively.

3.3 Goal structure review

To check if Def Stan 00-055 meets the MOD's requirements, a goal structure was constructed for review. Figure 3.1 shows the general method, which builds on Graydon and Kelly's process [55] discussed in Section 2.3.1. The Adelard ASCE tool [94] was used to draw the GSN diagram. It was created by reading through the standard, then précising fragments of information into the nodes of a GSN goal structure. The clause numbers were used as node references and the full text of each clause copied into the node descriptions for later reference. (This is only shown in the tool, not the diagrams). The

top-level goal was taken from notes in the standard's foreword. Much of the higher-level argument and strategy was also taken from the foreword, scope and introduction. Outside the introductory sections, requirements in notes were not modelled, as they are stated to be non-mandatory. The requirement clauses generally mapped to lower-level goals. Solution nodes could have been drawn from either bottom-level requirements (satisfying the argument by requiring contractor actions) or deliverable work products (documentary evidence that the requirements were met). Graydon and Kelly's paper [55] does not state a preference, but includes diagrams showing both approaches. In this review, work products were used. This choice lets us check the structure for unverifiable requirements (goals with no solution). Finally, clues in the notes, document structure and requirement clauses were used to connect the nodes into a logical structure.

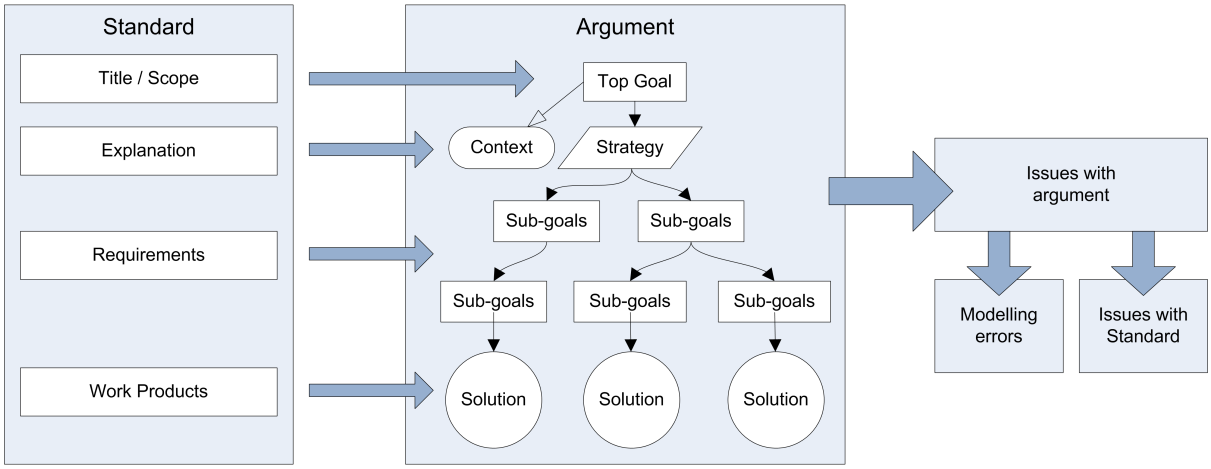

Figure 3.1: Overview of argument modelling method.

The full argument structure for the body of Def Stan 00-055 includes 111 nodes and is illustrated in Figure 3.2. One disadvantage of GSN is that non-trivial argument structures tend to be unreadable unless printed on very large pieces of paper! Without reading the text, we can still see that the network of nodes is complex. It is hard to identify the detailed structure, but the criss-cross of lines implies a tight coupling between different parts of the standard. ASCE provides a method of constructing custom views of parts of the diagram, using the same base data. In Figure 3.3, this has been used to show the top-level structure (from the top-right of Figure 3.2) at a readable scale. Figure 3.4 shows the strategy of achieving safety by satisfying the standard's five PE Safety Objectives.

About half of the issues were found while actually constructing the argument structure, including many low- and medium-impact atomicity and satisfaction issues. Most of the other issues were found using the structure checking tool in ASCE (Figure 3.5). This also found five modelling errors, where links or nodes had been missed from the structure. The checker was not foolproof though: it identified several cases of missing evidence where evidence might not be appropriate, e.g. a requirement for the contractor to ask for information if they found it necessary. Only a few issues were found by manually checking the goal structure afterwards. In this respect, the 4-step process in the GSN Community Standard [73] was more practical to use than Graydon and Kelly's 7-step review process [55], although the latter might be more systematic. Hawkins and Kelly's HAZOPS-inspired assurance guidelines [65] turned out to be too specific to evidence of a particular instantiation of a standard. Its prompts were not useful at all. Overall the argument structure review took around ten hours, similar to the naïve review. Its results

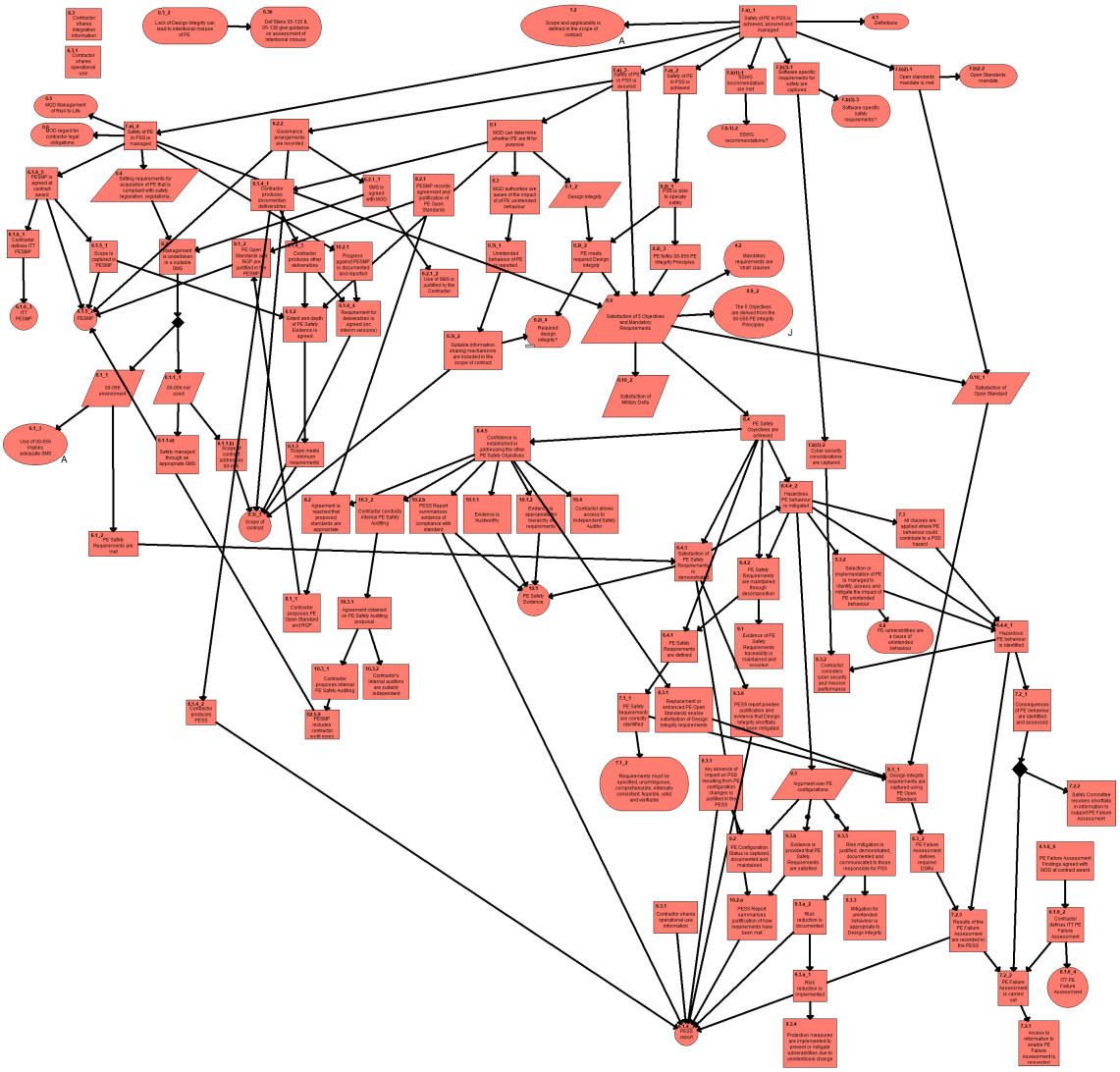

Figure 3.2: Overview of complete Def Stan 00-055 goal structure (not readable).

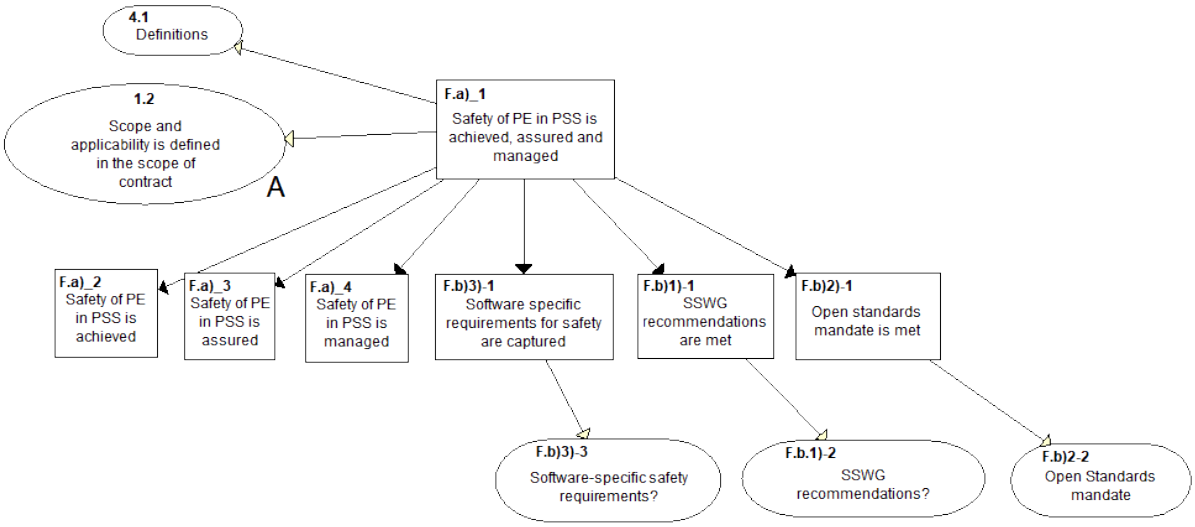

Figure 3.3: Top-level argument fragment for Def Stan 00-055.

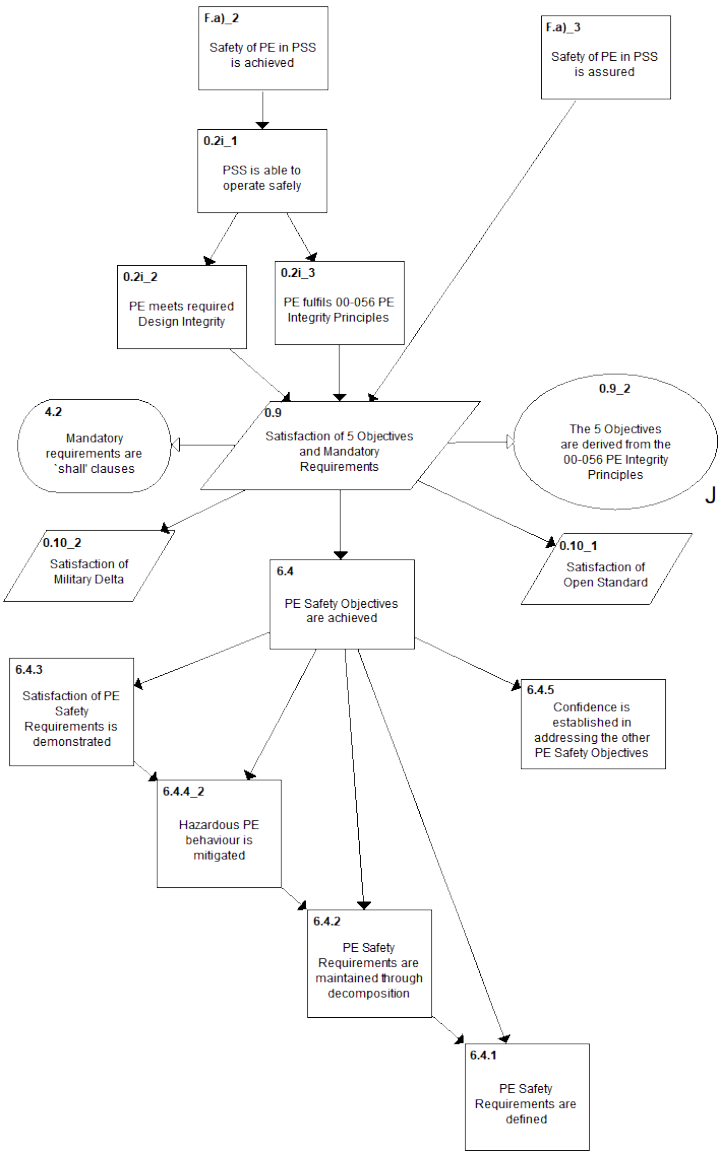

Figure 3.4: Argument fragment for Def Stan 00-055 showing the strategy for achieving safety.

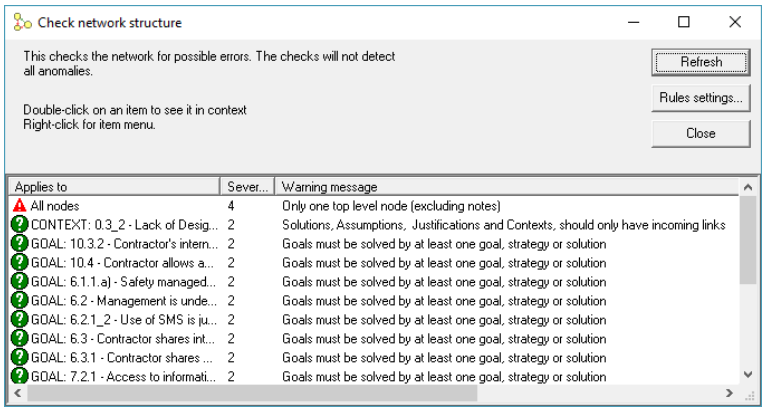

Figure 3.5: Example of errors found by ASCE argument structure checking tool.

Issue type	Impact				Total
	High	Medium	Low	Readability	
Atomicity			2 (7)		2 (7)
Completeness	1	2			3
Precision	1		1		2
Clarity	5				5
Consistency		1 (1)			1 (1)
Relevance	2		3		5
Satisfaction	1	3 (2)			4 (2)
Non-redundancy			1		1
Total issues found	10	9	14		33
Previously identified issues		(3)	(7)		(10)
New issues	**10**	**6**	**7**		**23**

Table 3.6: Summary of new and previously identified issues found during goal structure review, by impact and type classification (previously identified issues in brackets).

are summarized in Table 3.6. A key finding was that various requirements that appear important to the standard are not actually well-defined (high impact):

- The standard is not consistent about whether its aim is to achieve 'safety' (as per its title and foreword), or 'design integrity' (as per §1.1 of the introduction)
- The necessity of a deliverable PESMP is not clear.
- It is unclear if the contractor or the MOD should set PE Safety Requirements.
- No mandatory clause invokes the strategy of satisfying the military delta.
- Concepts such as data safety and intentional misuse are only referred to in non-normative parts of the standard.
- Several high-impact clarity issues related to the lack of explanation of the 'design integrity' concept. Although defined as "the extent to which the design is free from flaws which could give rise to or contribute to hazards or failure modes that contribute to a hazard" [18], it is sometimes used as a synonym for safety, but at other times as shorthand for a set of quality criteria.

Many of the results stem from the text of the standard being unclear or badly structured, in a way that makes it hard to model. The implication is that the text will also be hard to use in practice. The goal structure review focused on whether, if achieved, the requirements would satisfy the aim of the standard. However, it was less helpful in highlighting where those requirements might be easy to model but impractical to achieve.

3.4 Relationship modelling review

In this review, we try to use the RAF metamodel [85] to model the relationships between concepts in Def Stan 00-055, to test that they make sense. The standard was read through and key concepts noted to form elements of the model. Not all elements of the metamodel were used: the standard did not identify any criticality kinds (such as safety integrity levels), or any techniques. However, 63 elements (listed at Appendix C) were identified in the following classes of the metamodel:

- Requirements
- Activities
- Roles
- Artefacts
- Artefact attributes
- Artefact relationships
- Applicability kinds

These model elements were entered into an Eclipse-based integrated development environment called the Architecture-driven, Multi-concern and Seamless Assurance and Certification of Cyber-Physical Systems (AMASS) toolset [95]. This prototype system offers standards modelling tools implementing the RAF metamodel described by de la Vara et al. [85]. The different concepts were entered in AMASS and relationships assigned. e.g. The 'configuration status' artefact was set as a required input of the 'configuration management' process and assigned as a 'Contractor' responsibility. Unfortunately, the system proved slow and difficult to use, with poor documentation. While a model of the standard could be produced, it could only be viewed in a text-based hierarchy, which was not easy to review. Although it advertised functionality to draw relationship diagrams based on the modelling rules, this did not appear to work in practice.

Next, an attempt was made at constructing a process diagram. This was abandoned due to an absence of detail in the standard. Def Stan 00-055 seems to have three main phases. At the tendering or ITT phase, the PESMP is defined and a preliminary PE Failure Assessment carried out. At the contract award phase, the Failure Assessment findings are agreed and the PESMP formalized. In the third phase, everything else happens. Some things appear to need to happen once, such as justifying the choice of open standard used to satisfy Def Stan 00-055's requirements. Other things, such as iterating the Failure Assessment or reflecting operational use in the PESS report, seem to be ongoing tasks for the duration the software is in service. The standard does not generally say when things should happen, or in what order. While various activities can be identified, it is difficult to see clear processes that take inputs and produced defined outputs. With this lack of clarity, it was decided that a process diagram would not be helpful.

Instead, UML diagrams were drawn to show relationships between concepts, as illustrated in Figure 3.6. These were drawn by picking one of the model elements, e.g. the 'PE Failure Analysis' activity, and searching for references to it in the text. These were used to identify relationships with other concepts. This approach was more successful and yielded useful results, some of which are summarized below:

- The standard mentions many types of requirement: mandatory requirements, safety requirements, design integrity requirements, assurance requirements and derived safety requirements. It is not always clear which are which, or whether they are supposed to be set by the MOD or the contractor. For instance, §6.4.1 states "PE Safety Requirements shall be defined ...". The notes then talk about how they are derived, but it is not clear who should derive them or whether they are the same as 'Derived Safety Requirements'.
- The 'scope of contract' is a key artefact in the standard. It appears to contain many important parameters for the software project, such as the safety requirements, management systems and means of governance, etc. However, the format and extent of the content of the scope of contract is not clearly defined.
- The 'PE Failure Analysis' activity (Figure 3.6) is said to impact the 'scope of analysis' and confirm the 'scope of contract'. One would expect the opposite: the scope of analysis and contract should bound what we expect the contractor to analyse.

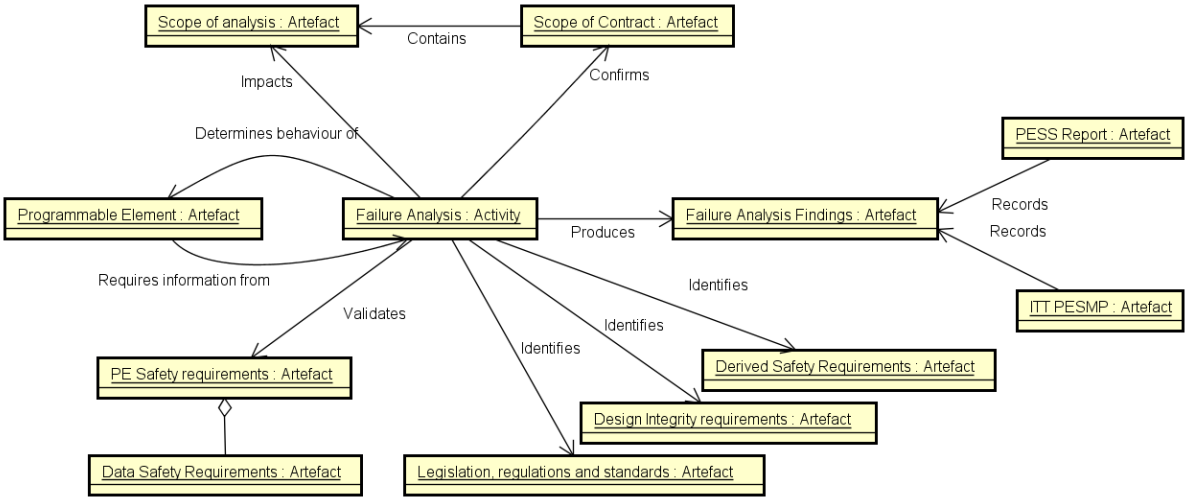

Figure 3.6: UML object diagram showing relationships to the PE Failure Analysis activity.

- The 'Safety Committee' is given various endorsement and approval roles, and so would be expected to be a MOD-led group. However, it is not defined in the main body of the standard. Its composition and terms of reference are left for the Contractor to determine through the PESMP (itself an optional artefact), and need not necessarily contain MOD stakeholders.

The results from this review were difficult to classify. Most of the issues identified by the other methods could be linked to specific portions of the text, with just a few general comments. They were also relatively easy to assign categories to. This review identified fewer, more far-reaching issues. They typically related to problems that affected several different parts of the text, often blending issues of consistency, completeness and clarity. Table 3.7 summarizes the results, classifying them by the dominant issue type.

The three experiments with software engineering-inspired review methods show that applying them to standards can yield useful results. They found fewer issues than traditional methods, but different types of problem. We must now evaluate how to use this information to improve future reviews.

Issue type	**Impact**				**Total**
	High	Medium	Low	Readability	
Completeness	4 (1)				4 (1)
Precision	2				2
Clarity	(1)				(1)
Consistency	3	1			4
Total issues found	11	1			12
Previously identified issues	(2)				(2)
New issues	**9**	**1**			**10**

Table 3.7: Summary of new and previously identified issues found during relationship review, by impact and type classification (previously identified issues in brackets).

4 Evaluation of standards review methods

Having experimented with various review methods, we need to evaluate the results. We need to decide if ideas taken from software engineering are beneficial for standards review and if so, how they can inform a guidance framework to help people carry out better reviews. First though, we discuss the requirements for this guidance. With these in mind, we then evaluate the findings from the experiments described in Chapter 3 to decide which are best. On this basis we decide which methods to include in the framework and how the practical experience of applying them can add value to the guidance.

4.1 Requirements for a standards review guidance framework

Before putting together the framework, we should consider its purpose and scope, and the requirements for its contents. The aim of creating a standards review guidance framework is to give people methods that will let them review standards more effectively and efficiently. 'More effectively' means telling them how to find more issues compared to traditional methods. 'More efficiently' means telling them how to use these methods in a way that minimizes the time and effort required. Ideally, the scope of the guidance would be to apply to all types of standard. However, this project has focussed on a software safety assurance standard. The methods we discuss may not necessarily be helpful in other cases, potentially limiting the scope of the guidance we produce. While this may be true, the material discussed in Chapter 3 does not seem to be safety-specific. Many of our sources in Chapter 2 were either generic [35, 50–52, 59], or applicable to other topics beyond safety (e.g. security) [55, 68]. We will therefore scope the guidance to apply to assurance standards in general.

There are three main potential audiences for the framework document. We can write standard format user stories [96] for each one:

1. *As a* review organizer, *I want to* know which methods to use, when and why *so that* I can direct people to review standards in the most effective and efficient way.
2. *As a* review participant, *I want to* understand how to carry out reviews using the different methods *so that* I can take part without wasting time.
3. *As a* standard author, *I want to* structure standards to facilitate review *so that* my work can be checked, maintained and understood more easily.

Organizers are the key audience, as they task individual reviewers and sequence the reviews in the standards development process. These factors may significantly impact the effectiveness and resource consumed by reviews. Helping reviewers understand what to do is obviously important too. Authors' decisions can influence how easy it is to review standards; they may wish to understand how they can structure the standard to ease later review. Easier to review tends to mean easier to understand, and hence probably also easier to maintain. Authors may also want to use review techniques to check their work as they go, potentially adopting a strategy similar to test-driven development.

For usability, the guidance framework needs enough detail for readers to understand how to use its methods without consulting other sources. However, it is intended to be a brief, practical guide rather than a textbook. Detailed theory should be signposted as further reading, rather than included in the text. To make it more widely applicable, the document is not intended to be specific to the MOD. It should use general terms rather than refer to specific products or organizations.

4.2 Evaluation of experimental findings

The baseline naïve review of Def Stan 00-055 Issue 4 revealed numerous issues. These were of a generally low significance, such as formatting, punctuation and style issues. Fixing these would have made the standard a little more readable. It would not have made it much easier to understand though, or changed its meaning. However, these issues were not found by the other methods, and addressing them would have made the standard look more professional. This could be important from the point of view of acceptance of the document by its stakeholders. The naïve review also identified various more significant types of issue not found by the other methods, such as use of incorrect terms or outdated references.

Given the results of the two proofreading rounds shown in Table 3.3, one might speculate that further rounds of similar review might expose even more issues. Potentially, more of the identified issues might also be recategorized as non-issues. Effectiveness might be improved slightly, but the law of diminishing returns would likely apply. Clearly a review carried out in this manner is fallible. It depends on the expertise and knowledge of the reviewer. Without applying any particular guidance, the results also depend on the reviewer's mental models of the purpose of the document and its desired quality criteria. One might reasonably assume that other reviewers would find different issues, or might take a different view on their validity or relative importance. This implies that to increase the robustness of this review method, one should use multiple reviewers with different backgrounds and knowledge. One could also task them to approach the review from the viewpoint of different roles, as suggested by Fagan and Patton for code inspections [50, 59]. However, this does not sound a particularly efficient use of resource. While it might be more effective at finding the same types of issues as before, it would not necessarily be any more effective at finding other, more significant issues.

The two checklist-based reviews increased both efficiency and effectiveness—for those issues they addressed. They were quicker to conduct than thorough proofreading, but found more medium- and high-impact issues. While effective at finding issues in their scope, their focus on these issues meant other problems got overlooked. With the ISO/IEC checklist, we mainly found issues with the presentation and structure of the standard. But with the checklist based on software requirements engineering principles, we started to find more substantive issues with the standard's actual requirements.

The argument review produced interesting findings. Trying to identify the top-level goal revealed inconsistency in whether the standard was more about ensuring safety or 'design integrity' (freedom from flaws that might contribute to hazards). This recalls Graydon and Holloway's discussion [49] about needing a clear purpose for software safety standards, and opens a debate about the difference between software correctness and safety. There was generally sufficient descriptive text to allow the lower-level argument structure to be constructed easily. However, the notes also contained what appeared to be implicit requirements (as Ankrum and Kromholz had found [68] with DO-178B).

It was not clear whether these should be included in the model. They were excluded, as the standard states "These notes are not mandatory", but this highlights an area of potential confusion. Another modelling challenge was modelling IF / THEN / ELSE constructs. In several places, Def Stan 00-055 assumes that its requirements might not be met and makes provision for fallbacks or alternatives. For instance, an open standard is supposed to be chosen to meet Def Stan 00-055's objectives, but if there is a shortfall, some other standard can be used, or another mitigation proposed. Conditional or partial fulfilment of requirements is hard to model in an argument structure and thus likely to be difficult for a contractor to understand how to achieve, and to demonstrate conclusively that they have done so. The combination of the ambiguous status of the requirements in the notes and the many options for compliance does not give a high degree of confidence that complying with Def Stan 00-055's mandatory requirements will in itself ensure a sufficient level of safety. This seems to support Graydon and Holloway's suggestion [49] that the achieved level of safety is as much due to the contractor's 'recipe' for compliance as the standard itself.

Building models of the relationships between the concepts in Def Stan 00-055 was more difficult than anticipated. The problem was not that the method of modelling seemed unsuited. Indeed, the RAF metamodel provided a helpful way to think of the constituent elements of the standard's requirements, and the widespread use of UML in software engineering meant that easy-to-use modelling tools were widely available. Rather, the standard did not contain the anticipated information to support coherent models. While this review produced fewer specific issues than expected, the issues it did find were more fundamental, revealing several areas where what appeared to be important concepts in the standard were not addressed properly. It also highlighted that the standard ought to be written in a way that makes the relationships between its concepts more obvious. A potential way of solving this problem would be to design models of the processes, obligations, interactions and other relationships described in the standard before starting to draft the next iteration.

The purpose of the experiments was to find out whether software engineering-inspired methods can improve on the baseline naïve review, and to determine the most effective and efficient methods of reviewing standards. Figure 4.1 compares the results of the different review methods described in Chapter 3. It shows that on average, the checklist-based methods found higher-impact issues than a naïve review, but fewer issues in total. The model-based methods found fewer issues again—contrasting with Graydon and Kelly's claim [55] that a structured argument review would reveal more issues. However, the issues identified during the argument and relationship modelling reviews appear to be more important. Although the impact scale proposed in Table 3.1 does not offer any distinction, the high-impact issues found during modelling appear qualitatively to be more fundamental to the purpose and intent of the standard.

This is not the whole story. Figure 4.2 shows that while there is some overlap in coverage, the different review methods found different types of issue. Naïve proof reading found the greatest range of types, including some not found by any other method. However, it did not find some types of issue identified in the checklist-based or argument reviews. The greatest overlap was between the enhanced checklist and the two model-based reviews, which all focused on requirements issues. The results tables in Chapter 3 also show that where the methods do overlap, there are few duplicate issues found.

These results imply that there is not one single most effective method: different methods are good for different things. This recalls Steele and Knight's filter model [56]

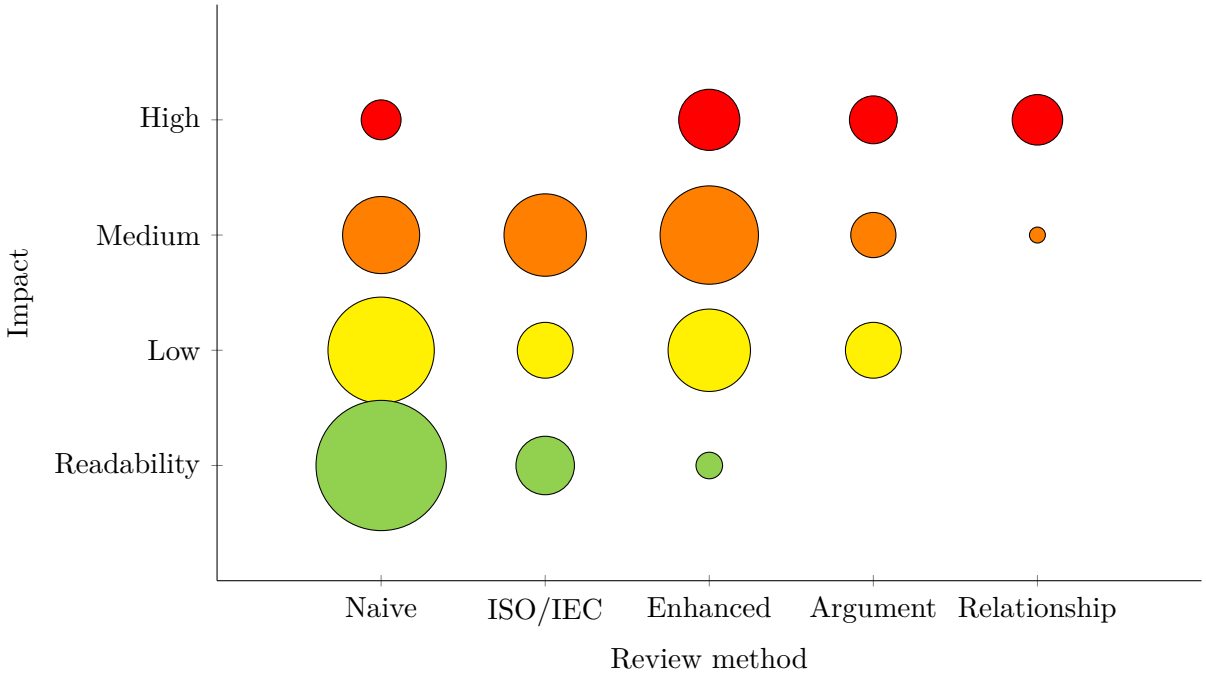

Figure 4.1: Relative impact of issues found by different review methods. Bubble area is proportional to number of issues found. Impact classified as per Table 3.1.

(see Section 2.3.3). Rather than representing an assurance standard as a filter system, we can use the model to represent the standards review process. Each of our review methods becomes a filter to remove different types of issue from the standard. Instead of selecting a single 'best method', we can improve effectiveness by using the methods in combination. While this ought to make a review process more thorough, using multiple methods will take more resource. We must also make the process more efficient.

If we extend the filter analogy beyond Steele and Knight's basic model, we can see a way to reduce overall resource. Some of our filters are fine, catching lots of very small issues like typographic or formatting errors. Others are coarse, catching a lesser number of chunkier issues like inconsistent requirements. To improve efficiency, we use the coarse filters first, to stop the fine filters getting clogged. Reviewing time is only part of the resource cost of the overall standards review process. The issues found in a review have to be examined and, if found valid, fixed in the standard. If we fix the fine-filter issues first, the work may be nugatory if we later discover coarse-filter issues that require whole sections to be re-written. We therefore need to arrange our filter model so that the methods we use first are the ones most likely to catch high-impact issues. By 'high-impact' we mean something slightly different from the definitions in Table 3.1. We are now interested in the impact on the drafting of the standard, rather than on its meaning. The two concepts are not directly linked. In this respect, a typographic error would always be low-impact. Even if it totally changed the meaning of a requirement, a small edit would fix it. In contrast, a relatively minor inconsistency could have a higher impact, requiring more extensive changes to be made in several places in the document.

In both the naïve and checklist-based reviews, it was difficult to find consistency issues because the inconsistent text was in different parts of the standard. The model-based reviews were better at finding these types of problem, because the modelling process drew together related requirements. This implies that a standard would be easier to review if

it were more modular. If all the requirements on a topic appeared together, reviewers would not have to rely so much on their memory of other parts of the text. This would make it easier to split the review into chunks, either to review in short sessions (to avoid fatigue), or to allocate to separate reviewers. Achieving this might be difficult: one would have to choose appropriate groupings to form the modules. Grouping all the requirements about particular artefacts might seem a sensible arrangement—this may have been the intent of the DIDs in Def Stan 00-055. However, this might scatter the requirements for processes that generate the artefacts. Model-based drafting could help: artefact-based and process-based requirements could be written to describe the same model from different viewpoints. Care would have to be taken to maintain consistency and avoid introducing redundant text that could cause configuration issues later. It could help to cross-reference rather than re-write text, analogously to the software concept of capturing repeated code in reusable methods or functions.

In the model-based reviews, more issues were found in constructing models than in reviewing them. Partly this seems to be because the modelling process requires the modeller to scrutinize the standard in detail to build the mode. They are forced to think about what it means. With most of the methods, it was difficult to deduce the absence of necessary requirements, rather than criticize what was there. But with the model-based methods, the process of transforming the text into a model revealed the gaps. The relative lack of issues found in reviewing the resulting models may be because they can be complicated and difficult to interpret. 'Stepping back' from the detail helped. In this respect, the simpler review process in the GSN Community Standard [73] was more useful than the apparently more rigorous proposal from Graydon and Kelly [55]. It helped one look at the generalities of an argument, rather than the specifics of individual logic steps. These factors point to a needed for guidance on organizing review of models and explaining them to reviewers, as it is reasonable to assume they would appear even more opaque to a reviewer who had not been involved in constructing them.

4.3 Potential sources of experimental error

A potential source of error in our comparison of review methods is that the results are based on experiments carried out by a single person with particular skills and experiences. At one level, this will have biased which issues were found using different review methods, to the extent that those methods rely on the competence of the reviewer. At another level, it means that the comparison of the different methods is not entirely fair. As the reviewer experiments with each review method, they build up expectations and biases that may influence the results of later experiments. In a larger study, these factors could be compensated for by using multiple reviewers. Using people with a range of different backgrounds would help control for the effects of differing prior knowledge and skills. Tasking them to carry out reviews in a different order would help compensate for effects of a build-up of familiarity. Tasking each to only use a single method would remove that source of bias, but require more reviewers.

In the model-based reviews, the modelling process can introduce error. Storey [58] reported that as well as code errors, the Darlington formal review found errors in their formal requirements model. In our reviews, detail will be lost when précising the standard down to the titles of GSN nodes or labels on a RAF model. Using the ASCE consistency checker in the argument review also revealed several modelling errors where argument nodes or relationships between nodes had been omitted. If the model is not faithful, flaws

found in it may not correspond to issues in the standard. These modelling errors can be tackled in two ways. At a simple level, potential issues can be triaged by checking them against the text of the standard to see if they still appear valid. At a more sophisticated level, the review could be split into two parts. In the first, multiple reviewers would check that the model was a valid representation of the standard (this corresponds to the 'argument comprehension' stage in the GSN Community Standard review process). In the second part, they would examine the model to look for problems with the standard.

Another potential source of error is the use of only one standard for review during the project. This decision was made in order to allow the results of the different experiments to be compared. It would not have been practical to test each method on multiple standards given the time constraints on the author. However, this could potentially have resulted in a guidance framework that is over-specific to Def Stan 00-055. This could be improved by repeating the review experiments using different standards, and updating the guidance framework to reflect the new findings. However, the problem can also be partially accommodated in the guidance framework itself. We can recommend users tailor which methods and checklists they use according to their experience applying them in their own problem domain (as suggested by Fagan, Humphrey and others [51, 59]). While this may be an effective mitigation for software reviews, it is less likely to be effective for standards, given that standards are reviewed far less frequently than code. If an organization is only responsible for a few standards, with long review period, they may not build up enough experience to tailor their reviews quickly enough to be useful.

4.4 Guidance material content

We must now decide what to include in our guidance framework (Appendix D). Based on the experimental results, we can make standards reviews more effective by using different methods together, rather than just one. We can make them more efficient by ordering the methods to help us find issues that cause bigger changes earlier in the process. From our experience of this project, we can also give tips to make individual methods more effective and efficient. However, we need to try to achieve a balance of breadth and depth. Measures that focus review methods tend to make them more efficient at finding some types of problem, but less effective at identifying others. To address these points in our framework, we will give both general advice on reviewing and specific guidance on individual methods. The methods will be organized using the filter model in Figure 4.3.

The first half of the filter will contain software engineering-inspired methods that deal with a standard's content. Argument modelling comes first. It appears to have the highest potential for disruptive changes, as it concerns getting the standard's intent right. Even though the issues it found in Def Stan 00-055 were more specific than those found by relationship modelling, if the standard's intent is not captured correctly, fixing relationship issues could still be nugatory. Steele and Knight's filter method was not trialled on Def Stan 00-055 but still appears useful enough to include next. It could be better than argument modelling for standards with simple aims met by many detailed requirements. We then list relationship modelling, to check drafts include the right processes and responsibilities to make them workable. The first three methods check that the right concepts are in the standard. The remaining methods check that these concepts are properly captured and communicated well. The last software engineering-inspired method, the 'requirement-level quality checks', uses the checklist from Section 3.2.2. The more traditional review methods are in the second half of our filter model. The tasks

from the ISO/IEC checklist are split into three separate methods with their own guidance. Finally, we include traditional proofreading. Although previously characterized as 'naïve', as our last filter stage it acts as a catch-all for remaining issues that the heuristics in the other methods do not trap. It is also the chance to confirm that fixing the issues trapped by the other filters resulted in a well-written, professionally presented document.

It will not always be proportionate to use every method. To help readers select which one to use when, each method description starts with a brief summary and bullet point pros and cons. To indicate how effective and efficient they might be, we give a rough qualitative assessment of their rigour and automation potential. We then describe how to carry out the method. From the experience of our experiments, we add practical advice to the basic principles, e.g. for the checklists, we offer interpretations of the guide words, and potential search terms to help find issues. We also talk about the differences between model-based and text-based reviews, and the need for both.

As groups will carry out practical standards reviews rather than individuals, we need to advise the best way to coordinate their efforts. Although this project has not experimented with them, we can explain how principles from social review methods in software engineering can be applied to standards. In particular, social review techniques such as peer reviews, walkthroughs or inspection scan be used to review the models and drafts output by the methods in our filter model. A guided walkthrough or inspection is likely to be more productive than asking stakeholders to independently review a model they were not involved in building. This could overcome unfamiliarity with other software engineering techniques, such as UML models used in relationship modelling. Finally, as the structure and drafting of a standard affects how easy it is to review, it seems worthwhile to add some notes on how standards can be designed to make their review easier. Doing this is also likely to make the standard easier to understand, and hence easier to use.

Appendix D gathers together all this material. With general guidance on how to carry out reviews and specific advice on particular methods, it draws on the theory from the literature review and the practical experience from our experiments. The intent is to preserve good practice from traditional standards review methods, but build on it with new ideas from software engineering to make reviews more effective and efficient. The next challenge is to check whether it is actually helpful.

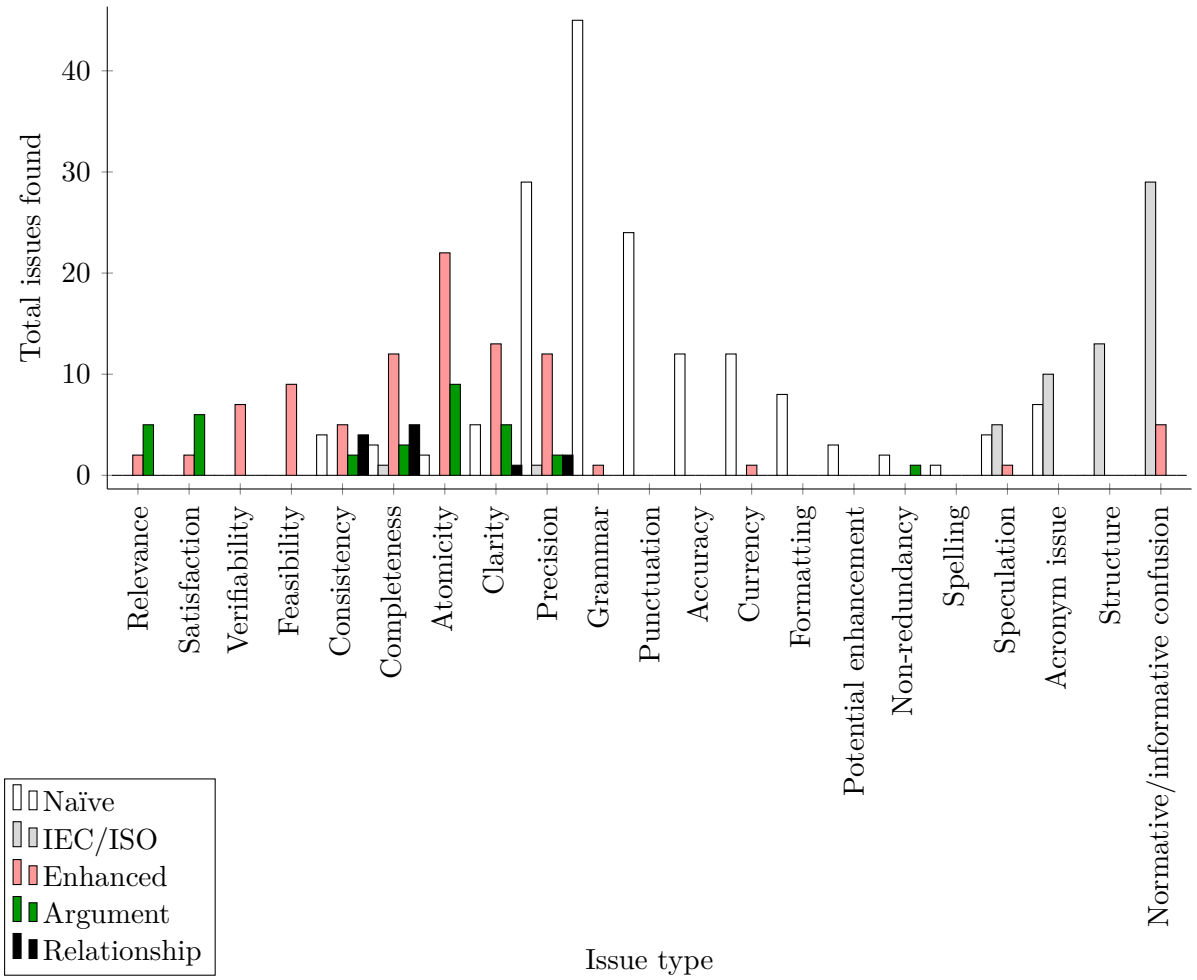

Figure 4.2: Total number of issues of each type found by each review method.

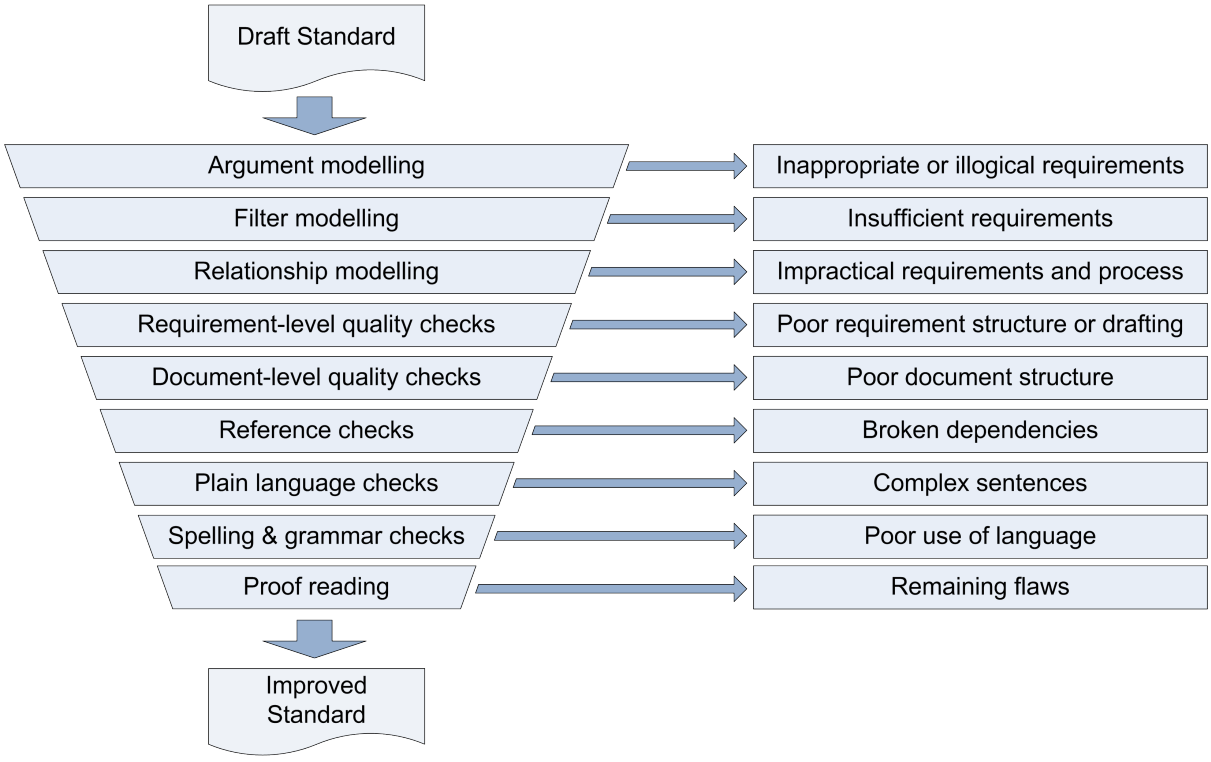

Figure 4.3: Review methods as a set of filters for removing potential flaws in standards.

5 Reflection

5.1 Reflection on production of the standards review guidance framework

5.1.1 Feedback from subject matter experts

To validate whether the guidance framework developed in Chapter 3 is likely to be useful, we need user feedback. Appendix D was sent to a variety of stakeholders. These people were known to have been involved in previous reviews of MOD assurance standards, with different roles and viewpoints: sponsoring standards, coordinating their publication, implementing or providing guidance on implementing standards, or providing subject matter expertise for their content. They were asked to provide brief feedback on whether the guidance framework looked useful for helping review and develop assurance standards; whether it could be applied more broadly to other types of standard; and whether there might be any practical problems in using it. Table 5.1 shows the respondents, who represent a reasonable selection of viewpoints. Their detailed feedback is in Appendix E.

Number	Respondent
1 & 2	A combined response from two members of DStan, who are involved in coordinating the publication and review process for Def Stan 00-055 and similar standards.
3	An independent safety assurance consultant with considerable experience scrutinizing MOD projects, including software-intensive projects using Defence Standards 00-055 and 00-056.
4	A member of the MOD team responsible for providing advice on software support. Prior to this role, she had extensive experience leading software assurance projects for a major prime contractor in the defence industry.
5	A member of the MOD team that sponsors Def Stan 00-055. He is not a software specialist, but is regularly involved in drafting and reviewing internal standards and policy documents, including reviews of Def Stan 00-055 and 00-056.

Table 5.1: Respondents to feedback request.

Overall, the feedback was positive. Respondents 1 & 2 said they recognized the methods as good practice. Respondent 4 said "I believe the combination of the methods discussed can be a powerful tool to help reviewers and authors to produce and deliver good products, not only standards but other engineering documents". However, several responses expressed trepidation about the effort needed to carry out structured reviews. Respondent 5 said he would be concerned about "frightening the horses" with something that looks complex and specialized, while Respondent 3 commented he would have liked more detailed definitions of the terms and concepts used, and an explicit standard development cycle.

Showing an explicit lifecycle for standards could help link the filter model to the development of a standard and explain the aims of different review stages. Appendix D was written from the viewpoint of trying to make a review more effective and efficient. It assumed the decision to undertake the review had already been taken. Other than the general purpose of improving the standard, it does not explain why a review should be carried out at a particular time. Feedback from Respondents 3 & 5 also points towards recommending the use of modelling and review in the development process. These two ideas can be combined. The guidance already talks about using modelling early in the design process. We can go further and recommend model-based design, but will also have account for legacy standards that have not yet been modelled.

To address the concerns about resourcing complex, model-based reviews, the framework includes the notion that models should be built by individual reviewers, then scrutinized by a larger group. The section on social reviews (D.1.1) says "It is likely to be more effective to socially review the outputs of these methods ..., rather than to carry out the analytical part of a method in a group". This may mean that some issues will not be spotted so quickly (many are identified during the construction of the model, rather than its review), but it means that the review does not rely on everyone being adept at constructing good models. If model-based development is used, it makes sense if the people who construct the models are part of the authoring team. This will reduce the overall resource required and make the reviews more efficient. Further automation would also help address resource concerns. This is a topic for further research, discussed in Section 5.3.

Respondents 1 & 2 also point out the difference in rationale between civil and military standards: civil standards address a market need, while military standards address the 'military delta'. This implies a difference in purpose when reviewing the two types. For civil standards we are mainly interested in fitness for purpose. For military standards, we want to leverage civil standards where possible, but add to them to meet military needs. This means we need to check military standards for compatibility with the civil standards they draw on. We also need to avoid duplicating material that is already covered elsewhere. As far as review methods are concerned, the military delta does not make much difference. However, this discussion emphasizes the need to be clear about the aim of the standard and the purpose of review.

Respondent 5 noted that none of the review methods in the framework really addresses the correctness of the standard: whether it recommends measures that will actually ensure its aim is met. He made the case that ad hoc review by subject matter experts is still needed. To an extent this is captured through the argument review method (checking that each level of argument plausibly satisfies the level above), and the inclusion of 'accuracy' as a guideword in the requirements-level checks. However, the need for reviewers to bring their experience to bear still remains. The last method in the framework is proofreading. This may need to be retitled 'expert review', to emphasize that this is about using reviewer expertise to find the errors that got through all the other filters. It is not just about looking for spelling and grammar issues (which are covered in a separate method).

5.1.2 Way ahead

The guidance framework seems useful enough to be worth developing further. First we would add a standards development cycle. Figure 5.1 gives an outline of how this might look. The cycle would need to cater both for new standards that start with a fresh model at the Design phase, and existing standards that have to be modelled retrospectively.

It would need to be expanded with information about which review methods are most appropriate during the different phases. With a standards development cycle included in the guidance framework, it would make sense to expand the design for review material (Section D.1.5). This would now need to cover model-based design and explain the interactions between development and reviews through the cycle.

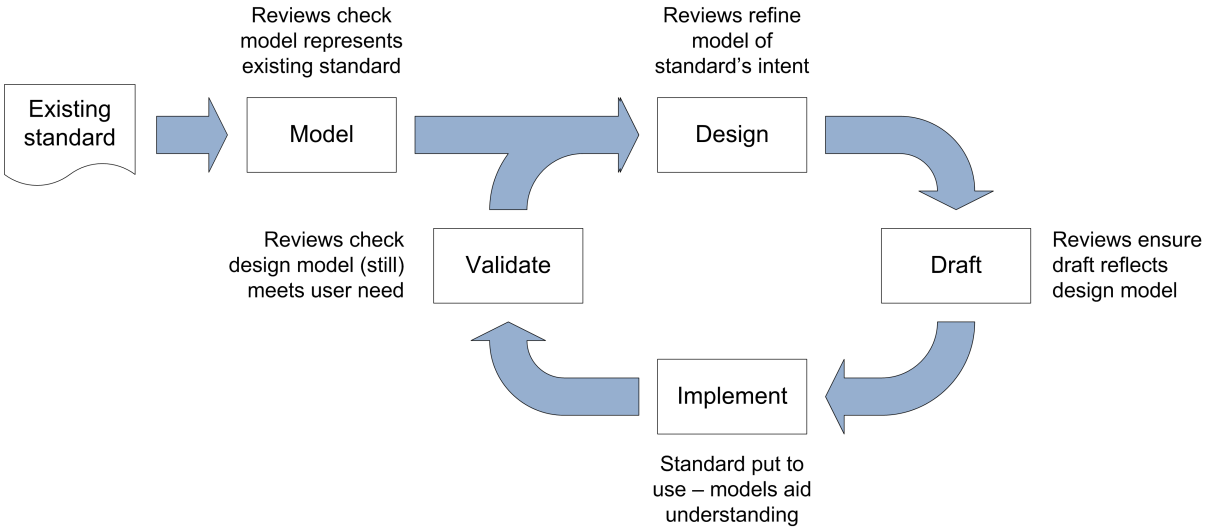

Figure 5.1: Outline model-based standards development lifecycle, showing intent of reviews at different lifecycle phases.

This model-based approach to designing, writing and reviewing a standard appears to be novel. Other than their filter approach, Steele and Knight [56] state that they were not aware of any methods of designing standards, beyond drafting by "ad hoc gatherings of experts". With the guidance updated, we would then look to put the theory into practice by applying it to a real review project.

5.2 Reflection on Def Stan 00-055

5.2.1 Key findings

Tony Hoare said "There are two ways of constructing a software design: One way is to make it so simple that there are *obviously* no deficiencies, and the other way is to make it so complicated that there are no *obvious* deficiencies" [97]. Def Stan 00-055 seems to have gone down the second path. It is hard to spot the more major deficiencies because the standard talks about all the right topics. On closer inspection though, it often does not actually say what one might expect it should. Some key concepts, such as design integrity and PE failure assessment, are used inconsistently and not well explained. Requirements for some of its main artefacts, like the scope of contract or the PE safety summary report, are scattered through the document; it is hard to know whether they are complete. And some important-looking concepts such as the treatment of intentional misuse, data safety, or satisfaction of the military delta, do not appear to be part of the normative requirements. Cyber security is only considered with respect to unauthorized change to programmable elements: the potential for harm through unauthorized access to systems is not addressed.

In principle, Def Stan 00-055 is quite simple. It asks the Contractor to satisfy the safety requirement principles shown in Table 2.1 by following an open standard of their

choice. It also gives a minimum set of topics that the open standard should cover. In practice, the standard is much more complicated, as can be seen from the argument structure in Figure 3.2. Part of the explanation is that the standard tries to cover all eventualities. It is supposed to be applied to projects ranging from selection of simple off-the-shelf software to development of complex, bespoke cyber-physical systems; and from quick one-off buys to long-term support and maintenance arrangements. It may either be applied to satisfy Def Stan 00-056 for the programmable elements of a wider system, or on its own. Unfortunately, the text does not make it clear which assumptions are being made at what point. It does not provide any kind of baseline process that would make it easy to see where tailoring was needed to suit the circumstances. And although the standard is intended to be called up in a contract between the MOD and a supplier, in most instances it only describes the Contractor's role, not the interface with MOD processes. The temptation for MOD staff is to merely cite the standard in a contract, without understanding the implicit need for MOD to set safety requirements, provide information and tailor aspects of the standard.

5.2.2 Way ahead

As Def Stan 00-055 Issue 4 was published in April 2016, work on a formal review should start in the next six months to meet DStan's expectations for a five-year review cycle. This project has not discovered any fundamental problems with the standard that do not appear to have solutions, but it is clear that there is a lot of scope for clarification and simplification. We propose that the key to doing this is to follow the guidance in Section 5.1.2: to set out simple models of what the standard is supposed to achieve, then re-draft the text to follow the models. Re-writing the standard to make it clearer and simpler will mean that many of the specific issues we have identified in this dissertation will become irrelevant for the new draft. However, they will still be useful as pointers for what to avoid, and as evidence of the need for change. The other driver for change will need to be user experience from implementing Issue 4. This project has identified theoretical problems, but we will need to engage with stakeholders to capture practical issues, and ensure we build resolutions into the new models.

Review will then follow the filter model, checking that the essence of the standard's intent has been correctly captured before refining the detailed drafting. A new argument structure will help ensure that the things the standard is trying to achieve are captured as normative requirements. Process and interaction models will help make it clear what a contractor is supposed to do to implement the standard, and when. As well as providing the baseline structure to draft the new standard against, it is likely that the models can also be used as the basis for training and guidance documents. This would help contractors understand how to implement the standard, and help MOD people understand how to use it effectively.

5.3 Further opportunities for study

A key challenge in standards review is management of input from a wide stakeholder community. Face-to-face meetings are good for shared understanding. This applies both to presenting the reasoning behind a standard, and explaining the issues found with it. Sansone and Rocca-Serra [53] maintain they are "crucial to galvanize a new group, enable participants to evaluate their commitment, design the initial workplan, harmonize different perspectives and explore available options". However, dozens of people may

provide input to a standards review, compared to Fagan's recommendation [59] of four people for an inspection team. Large meetings are generally inefficient. Offline reviews allow everyone to give input, but tend to generate many hundreds of comments. In both cases, many opinions will be duplicated or conflicting. It has not been possible to investigate this area within the scope of this project, but it appears worthy of future research.

In software engineering, despite the emphasis on automation, reviews are still a group activity. Improved communications with customers and within development teams is a key theme of recent agile methodologies. The growth of distributed development projects has spawned a range of collaboration tools, including issue management tools like Bugzilla and Jira, and source control services like GitHub and SourceForge. These allow multiple reviewers to comment on, tag and classify issues, which can be linked directly to the source text. Some of these tools and practices could help manage standards review and development. Research could also look at the balance between committee and individual work, and measures for making review meetings more effective.

Another broad area for future research is automation of reviews. Spelling, grammar and readability checkers are the main static analysis tools currently available for reviewing standards. When we use models to represent a standard, we can also use consistency checking tools. These can check our model against the rules for that modelling technique, such as the metamodels for goal structures or reference assurance frameworks. The difficulty with further automation is the use of natural language for writing standards. There are two broad avenues for improvement: making standards more formal, or dealing better with natural language.

In the first approach, the idea of modelling standards could be taken further, to model the standard's intent as well as its structure. Sansone and Rocca-Serra [53] advocate making standards digital objects that can be found and used automatically. This would also aid automated checking. Ideally, formal representations of standards could be produced. Habli and Rae [98] have proposed an experiment to test whether the act of representing standards requirements formally finds errors. Beyond this, a formal representation might also be amenable to logical proof of correctness. In the second approach, natural language processing or machine learning techniques could be developed for reviewing standards. Sentiment analysis techniques have already been used to identify potential new requirements from online user reviews [99]. Similar techniques might be used to find problematic text in standards. It is not clear how feasible any of this would be to achieve. The research would have to demonstrate that new analysis methods were practical.

A further potential research area is automatic checking and enforcement of style guides. However, this is likely to have less beneficial impact on standards. It can partly be achieved already through the use of drafting templates. DStan uses an automated template system to format boilerplate sections of standards (such as the front matter, reference lists, etc.), but does not enforce this on the main content of their standards. Solutions in this area are likely to be specific to particular drafting tools.

Beyond these topics, there is still much research to be done to demonstrate whether software assurance standards like Def Stan 00-055 actually improve safety. The work covered in this project will help ensure that these standards impose requirements and practices that are believed to improve safety. However, it has not addressed whether they actually do so, or examined how the methods contractors use for compliance impact the safety outcome. Graydon and Holloway discuss the challenges in this area in detail in their paper, *Planning the unplanned experiment* [49].

6 Conclusions

6.1 Software safety assurance standards as software engineering artefacts

This project set out to test the notion that Def Stan 00-055 is a software engineering artefact, and thus should be amenable to software engineering methodologies. Our findings strongly suggest that this is the case. Whether it actually is a software engineering artefact is a moot point, although as Def Stan 00-055 sets requirements on the software engineering process, it can be seen as an extension of that process. At a more philosophical level, policies and standards can be seen as software that is executed by an organization rather than a computer. More practically though, this project has shown that there are enough similarities between the standard and artefacts such as software specifications and requirements documents to make it useful to apply software engineering principles to standards reviews.

We have seen that ideas from software inspections can be applied to make standards reviews more effective than traditional proof-reading. The requirement-setting parts of the standard are similar enough to software requirements that we can use good practice from software requirements engineering to find issues with them. We have also found that the standard contains enough internal structure to make software safety engineering modelling methods useful. It appears that some software design principles are also relevant to standards and can make them easier to understand and review, and thus also easier to use. Reducing complexity in how a standard is written will make it easier to understand, and thus easier to find problems in and to follow correctly. The 'code once' principle (using functions or methods rather than duplicate code) can be used to avoid introducing inconsistencies when similar text is edited in one place but not another. Writing standards in a modular way that reduces coupling between sections would make it easier to identify inconsistent requirements, by bringing requirements on similar topics closer together. This will also make it easier to understand how to implement the standard. Although it has not been investigated in this project, there is scope to apply collaborative software development tools to the development process for standards. Tools such as issue tracking, source code management and task management systems could help improve communication between reviewers. This might make the review process more efficient and effective.

While there are useful similarities between software and standards, there are some key differences. The review cycle of standards is very long, often measured in years. In contrast, software reviews are much more frequent: potentially every fortnight in a sprint-based agile project. This difference in timing can make some software methods less practical for standards. Reviewers get less return on investment for learning complex techniques. The long cycle time can also make continuous improvement harder, as there is less opportunity to tailor checklists and methods using feedback from common issues.

Another major difference is the use of natural language in standards. This makes it harder to produce automated review tools. Some static analysis tools do exist though, such as spelling, grammar and readability checkers. Representing standards using structured models can improve this situation, as automatic model-checking can then be used. Future research could investigate formal, machine-readable representation of standards, to allow verification that correct implementations would logically imply meeting the standard's goals. However, it would be challenging to make this practical given the point above about return on investment. Current formal software verification methods are likely to require too much effort to learn and deploy for a review task that is only repeated every few years. Nonetheless, research on methods such as RAF metamodelling is starting to allow standards to be described more formally.

6.2 Learning points

In terms of effective review of standards, several points come out of this work:

1. It is easier to find problems when you know what you are looking for.
2. Having a structure makes systematic reviews easier, and helps avoid omissions.
3. Different methods typically find different types of issue.

The first point is the basis behind checklist-based methods. While an imaginative human reviewer can potentially find any manner of problems by reading the text of a standard, they are likely to find specific types of issue more effectively when prompted. This appears to be just as true for standards now as it was 40 years ago for software code, when Fagan [59] recommended using checklists in his inspections. The counterpoint is that when checklists focus a review on one set of issues, other potential problems can get overlooked. This leads to the need to optimize checklists to find important categories of problem, and to update them when new types of problem become apparent. 'Important' needs to be defined according to the context. For software, correctness is likely to be a key issue. For standards, how easily they can be understood may be just as important. In both cases, the likely expense of fixing the issue will also be relevant.

It is less obvious that some special structure is needed for systematic review. Why is reading a standard linearly from end to end not sufficiently systematic? Part of the answer is the need to maintain focus on the task in hand. Human attention wanes quickly, so methods that can help direct focus to the important points can be more effective. This is why the moderator role is part of a Fagan inspection. It is also why methods that itemize and step through particular features of a document are helpful. The other part of the answer is that standards contain internal structure, and a broad category of potential issues relates to problems with these structures. Just as software can have branches and control flows, or hierarchies of classes and objects; standards can describe processes and hierarchies of requirements. When these structures are represented as linear, natural language text, the structures are hidden and it is difficult to spot issues such as inconsistencies between requirements. Methods that make these structures more explicit make it possible to systematically review for these types of problem.

The third point flows from the last two. Checklists, heuristics or structures that improve the effectiveness of a method for finding one class of problem tend to reduce its effectiveness for finding others. At the extreme, if you know clearly enough what you are looking for, it is likely to be possible to automate a check for a particular type of issue. However, the automatic review will only find the class of issues it has been programmed

for. An example is an automatic grammar check, which can test some rules of grammar, but not others.

6.3 Standards review guidance

We have produced a guidance framework for review of standards. This takes good practice from traditional review methods and enhances it with methods and guidance built on software engineering principles. It explains how combinations of different review methods can be used to identify different types of problems in standards. It also explains how to order the use of these methods to make the review process more efficient, by identifying the more disruptive types of issue earlier in the overall review process. Feedback from stakeholders in the field implies that it will be useful in future reviews. However, it could usefully be developed to include model-based drafting of standards and an explicit lifecycle that explains how the drafting and review processes are linked.

The principles set out in the guidance will be taken forward into the next update of Def Stan 00-055, to guide the drafting and review of the standard. The issues identified during the review experiments carried out in this project will also form a valuable input to the process. If it proves useful, it will also be applied more widely to other MOD assurance standards.

6.4 Personal Reflection

This project has been a great chance to take ideas that I have learnt from the Software Engineering MSc and apply them to make improvements in other areas of my work. In particular, I have been able to build on material discussed in the *Managing Risk and Quality*, *Software Process Quality and Improvement*, and *Requirements Engineering* modules. It seems fitting that we use software engineering ideas to improve reviews of software assurance standards, as those standards will then be used to improve software. The results have not gone as I had expected though: from the literature, I had expected that using more structured review methods would reveal more problems, but this was not supported by my findings. Instead, these methods led to more specialized results. However, it was fortuitous to find the filter model, which proved a useful tool for thinking about my results and organizing my guidance framework.

With hindsight, I should perhaps have looked more at the reasons for carrying out reviews earlier in the project. I took it for granted that reviews occurred, and hence focused on how to improve their results. If I were to repeat the project, I would look more at how reviews form a part of the development cycle of a standard. This might lead more directly to the idea of using model-based design methods for standards. I think there is a lot of scope to learn from software engineering in this area, using similar design tools to describe the systems and processes that assurance standards are supposed to put in place.

The project has taken a lot of effort to complete, and was a challenge to fit in around the demands of family and working life. If I did it again, I would set aside more dedicated blocks of time for the work, rather than attempting to fit so much in my spare time. In the end though, I am pleased to have produced what I think will be a valuable tool for improving the quality of software safety assurance standards, and assurance standards more widely. I am looking forward to being able to put it to use.

Acronyms

ALARP	As Low As Reasonably Practicable.
AMASS	Architecture-driven, Multi-concern and Seamless Assurance and Certification of Cyber-Physical Systems.
ASCAD	Adelard Safety Case Development methodology.
ASCE	Assurance and Safety Case Environment.
BPMN	Business Process Model and Notation.
BSI	British Standards Institute.
CAE	Claims, Arguments, Evidence.
Def Stan	Defence Standard.
DID	Data Item Definition.
DOD	Department of Defense.
DStan	Defence Standardization.
FAA	Federal Aviation Administration.
FLIS	Future Logistics Information System.
FMECA	Failure Modes Effects and Criticality Analysis.
FTA	Fault Tree Analysis.
GSN	Goal Structuring Notation.
HAZOPS	Hazard and Operability Study.
IEC	International Electrotechnical Commission.
ISO	International Organisation for Standardization.
ITT	Invitation To Tender.
JSP	Joint Service Publication.
KiD	Knowledge in Defence.
MOD	Ministry of Defence.
MODAF	MOD Architecture Framework.
PE	Programmable Elements.
PESMP	Programmable Element Safety Management Plan.
PESS	Programmable Element Safety Summary.
RAF	Reference Assurance Framework.
SEIG	Systems Engineering and Integration Group.
SME	Subject Matter Expert.
SSRC	Safety Standards Review Committee.
TOGAF	The Open Group Architecture Framework.
UK	United Kingdom.
UML	Unified Modelling Language.
US	United States.
V&V	Verification and Validation.

References

[1] J. McDermid, "Report of a working party on software management in acquisition and support," Defence Scientific Advisory Council, London, Internal Report D/DST/01/14/16/22, Apr. 2014.

[2] R. V. Carlone, "Patriot missile defense—software problem led to system failure at Dhahran, Saudi Arabia," US General Accounting Office, Washington, D.C., Report to the Chairman, Subcommittee on Investigations and Oversight, Committee on Science, Space, and Technology, House of Representatives GAO/IMTEC-92-26, Feb. 1992, last accessed 25 Dec 18. [Online]. Available: https://www.gao.gov/products/IMTEC-92-26.

[3] J.-L. Lions, "Ariane 5—Flight 501 failure," Paris, Report by the Inquiry Board, Jul. 1996, last accessed 25 Dec 18. [Online]. Available: https://www.esa.int/For_Media/Press_Releases/Ariane_501_-_Presentation_of_Inquiry_Board_report.

[4] Defence Accident Investigation Branch, "Service inquiry into the Watchkeeper (WK006) Unmanned Air Vehicle (UAV) accident at Boscombe Down Aerodrome on 2 November 2015," Defence Safety Authority, Service Inquiry, Aug. 2016, Redacted, last accessed 25 Dec 18. [Online]. Available: https://www.gov.uk/government/publications/service-inquiry-into-the-watchkeeper-wk006-unmanned-air-vehicle-uav-accident-at-boscombe-down-aerodrome-on-2-november-2015.

[5] B. Zhang, "A software problem caused a brand-new Airbus military plane to crash," *Business Insider*, Jun. 2015, last accessed 25 Dec 18. [Online]. Available: https://www.businessinsider.com/a-software-problem-caused-an-airbus-a400m-to-crash-2015-6?r=US&IR=T.

[6] A. Morse, "Major projects report 2012," National Audit Office, House of Commons report HC 684-I 2012–13, Jan. 2013, last accessed 29 May 18. [Online]. Available: https://www.nao.org.uk/wp-content/uploads/2013/03/Major-Projects-full-report-Vol-1.pdf.

[7] T. Burr, "Chinook Mk3 helicopters," National Audit Office, House of Commons report HC 512 2007–2008, Jun. 2008, last accessed 29 May 18. [Online]. Available: https://www.nao.org.uk/wp-content/uploads/2008/06/0708512.pdf.

[8] S. J. Bourn, "Battlefield helicopters," National Audit Office, House of Commons report HC 486 2003–2004, Apr. 2004, last accessed 29 May 18. [Online]. Available: https://www.nao.org.uk/wp-content/uploads/2004/04/0304486.pdf.

[9] Ministry of Defence, "The procurement of safety critical software in defence equipment—Part 1: Requirements," Directorate of Standardization, Glasgow, Interim Defence Standard 00-55 Part 1 Issue 1, Apr. 1991.

[10] Ministry of Defence, "Requirements for safety related software in defence equipment—Part 1: Requirements," Directorate of Standardization, Glasgow, Defence Standard 00-55 Part 1 Issue 2, Aug. 1997.

[11] M. Tierney, "Software engineering standards: The 'formal methods debate' in the UK," *Technology Analysis & Strategic Management*, vol. 4, no. 3, pp. 245–278, Jan. 1992. DOI: 10.1080/09537329208524097.

[12] M. A. Ould, "Software development under Def Stan 00-55: A guide," *Information and Software Technology*, vol. 32, no. 3, pp. 170–175, Apr. 1990. DOI: 10.1016/0950-5849(90)90174-p.

[13] A. Adam and P. Spedding, "Trusting computers through trusting humans: Software verification in a safety-critical information society," in *Social and Human Elements of Information Security: Emerging trends and countermeasures*, M. Gupta and R. Sharman, Eds., Information Science Reference, 2009, ch. V, pp. 61–75, ISBN: 978-1-60566-037-0.

[14] C. Howard, "The MOD's new system safety standard: Interim Defence Standard 00-56 issue 3," *Safety Systems*, vol. 15, no. 1, Sep. 2005, last accessed 3 Jun 18. [Online]. Available: https://scsc.uk/scsc-75.

[15] Ministry of Defence, "Requirements for safety of Programmable Elements (PE) in defence systems—Part 1: Requirements and guidance," UK Defence Standardization, Glasgow, Interim Defence Standard 00-55 Part 1 Issue 3, Dec. 2014.

[16] P. Williams and J. McDermid, "Reincarnation of Def Stan 00-55," *Safety Systems*, vol. 25, no. 1, Sep. 2015, last accessed 3 Jun 18. [Online]. Available: https://scsc.uk/scsc-145.

[17] G. Jolliffe, "Re-issuing Def Stan 00-55," in *Addressing Systems Safety Challenges: Proceedings of the Twenty-second Safety-critical Systems Symposium, Brighton, UK, 4–6th February 2014*, T. Anderson, Ed., ser. Safety-critical Systems Symposium, last accessed 3 Jun 18, vol. 22, Feb. 2014, pp. 25–34. [Online]. Available: https://scsc.uk/scsc-126.

[18] Ministry of Defence, "Requirements for safety of Programmable Elements (PE) in defence systems—Part 1: Requirements and guidance," UK Defence Standardization, Glasgow, Defence Standard 00-055 Part 1 Issue 4, Apr. 2016.

[19] Ministry of Defence, *Knowledge in Defence website*, last accessed 2 Dec 18, 2018. [Online]. Available: https://www.aof.mod.uk/.

[20] Ministry of Defence, *Software quality management guidance*, last accessed 9 Apr 2018, last accessed 9 Apr 18, Jan. 2009. [Online]. Available: https://www.aof.mod.uk/aofcontent/tactical/quality/downloads/sqmg.pdf.

[21] S. F. Mattern, "Increasing the likelihood of success of a software assurance program," *Journal of System Safety*, vol. 44, no. 4, pp. 19–25, Jul./Aug. 2008, ISSN: 0743-8826.

[22] Ministry of Defence, "Hazard analysis and safety classification of the computer and programmable electronic system elements of defence equipment," Directorate of Standardization, Glasgow, Interim Defence Standard 00-56 Issue 1, Apr. 1991.

[23] Ministry of Defence, "MOD standardization management policy—Part 1: Directive," Defence Authority for Technical & Quality Assurance, Joint Service Publication JSP920 Part 1 V3.0, Aug. 2017, last accessed 15 Dec 18. [Online]. Available: https://www.dstan.mod.uk/policy/JSP920_Part1.pdf.

[24] B. Hendrix, "New system safety standard ANSI/GEIA-STD-0010 is available," *Journal of System Safety*, vol. 46, no. 2, pp. 36–37, Mar./Apr. 2010, ISSN: 0743-8826.

[25] IEC Subcommittee 65A: System Aspects, "Functional safety of electrical/ electronic/ programmable electronic safety-related systems—Part 0: Functional safety and IEC 61508," International Electrotechnical Commission, Tech. Rep. IEC/TR 61508-0:2005, 2005.

[26] RTCA Special Committee 167, "Software considerations in airborne systems and equipment certification," RTCA Inc., Washington, Recommendation DO-178B, Jan. 1992.

[27] R. Bates, "Software safety in the MOD—a comprehensive review of the acquisition of safety critical software," Master's thesis, University of Oxford Software Engineering Programme, Oct. 2017.

[28] Department of Defense, "Standard practice for system safety," DOD, Military Standard MIL-STD-882D, Feb. 2000.

[29] G. Williamson, *Health, safety and environmental protection in defence: Policy statement by the secretary of state for defence*, last accessed 5 Dec 18, Jun. 2018. [Online]. Available: https://www.gov.uk/government/publications/secretary-of-states-policy-statement-on-safety-health-environmental-protection-and-sustainable-development.

[30] *Health and safety at work etc. act*, 1974 c. 37, last accessed 18 Dec 18, Jul. 1974. [Online]. Available: http://www.legislation.gov.uk/ukpga/1974/37/contents.

[31] Ministry of Defence, "Safety management requirements for defence systems—Part 1: Requirements," UK Defence Standardization, Glasgow, Defence Standard 00-56 Part 1 Issue 7, Feb. 2017.

[32] R. D. Hawkins, I. Habli, and T. P. Kelly, "The principles of software safety assurance," in *31st International System Safety Conference*, last accessed 8 Dec 18, International System Safety Society, Boston, MA, Aug. 2013. [Online]. Available: https://www-users.cs.york.ac.uk/rhawkins/papers/HawkinsISSC13.pdf.

[33] T. Kelly, "Software certification: Where is confidence won and lost?" In *Addressing Systems Safety Challenges: Proceedings of the Twenty-second Safety-critical Systems Symposium, Brighton, UK, 4-6th February 2014*, T. Anderson, Ed., ser. Safety-critical Systems Symposium, last accessed 8 Dec 18, vol. 22, Feb. 2014, pp. 255–267. [Online]. Available: https://scsc.uk/scsc-126.

[34] Ministry of Defence, "Safety management requirements for defence systems—Part 2: Requirements," UK Defence Standardization, Glasgow, Defence Standard 00-056 Part 2 Issue 5, Feb. 2017.

[35] M. A. Ould, *Managing software quality and business risk*. Chichester: John Wiley & Sons Ltd, 1999, ISBN: 047199782X.

[36] D. R. Wallace, D. R. Kuhn, and L. M. Ippolito, "An analysis of selected software safety standards," in *COMPASS '92 Proceedings of the Seventh Annual Conference on Computer Assurance*, Jun. 1992, pp. 123–136. DOI: 10.1109/CMPASS.1992.235757.

[37] J. Bowen and M. Hinchey, "Seven more myths of formal methods," *IEEE Software*, vol. 12, no. 4, pp. 34–41, Jul. 1995. DOI: 10.1109/52.391826.

[38] M. A. Ould, "Testing—a challenge to method and tool developers," *Software Engineering Journal*, vol. 6, no. 2, p. 59, Mar. 1991. DOI: 10.1049/sej.1991.0008.

[39] P. R. Caseley and T. A. D. White, "The MOD procurement guidance on software safety assurance—assessing and understanding software evidence," in *4th IET International Conference on Systems Safety*, Oct. 2009, pp. 1–12. DOI: 10.1049/cp.2009.1547.

[40] M. Tamos, "Applicability of safety critical systems techniques to business domain," Master's thesis, University of Oxford Software Engineering Programme, 2011.

[41] Ministry of Defence, "Safety management requirements for defence systems—Part 1: Requirements," UK Defence Standardization, Glasgow, Interim Defence Standard 00-56 Part 1 Issue 3, Dec. 2004.

[42] V. Hamilton, "A new concept in defence safety standards: The revised UK Defence Standard 00-56," in *Proceedings of the 10th Australian Workshop on Safety Related Programmable Systems*, T. Cant, Ed., ser. Conferences in Research and Practice in Information Technology, last accessed 13 Feb 17, vol. 55, Sydney, Australia: Australian Computer Society, Inc., Apr. 2006, pp. 77–83, ISBN: 1-920-68237-6. [Online]. Available: http://dl.acm.org/citation.cfm?id=1151816.1151824.

[43] J. A. Mcdermid and P. Williams, "Defence standard 00-56 issue 5: Concepts, principles and pragmatics," in *9th IET International Conference on System Safety and Cyber Security (2014)*, Oct. 2014, pp. 1–6. DOI: 10.1049/cp.2014.0984.

[44] Ministry of Defence, "Defence standards 00-56, 00-55 and 00-27," Defence Equipment & Support, Bristol, Safety & Environmental Bulletin SEB/008, Apr. 2015, last accessed 13 Feb 17.

[45] W. E. Wong, T. Gidvani, A. Lopez, R. Gao, and M. Horn, "Evaluating software safety standards: A systematic review and comparison," in *2014 IEEE Eighth International Conference on Software Security and Reliability-Companion*, Jun. 2014, pp. 78–87. DOI: 10.1109/SERE-C.2014.25.

[46] Ministry of Defence, "Safety management requirements for defence systems—Part 1: Requirements," UK Defence Standardization, Glasgow, Defence Standard 00-56 Part 1 Issue 4, Jun. 2007.

[47] Federal Aviation Administration, "System safety handbook," FAA, Handbook, Dec. 2000, last accessed 8 Nov 18. [Online]. Available: https://www.faa.gov/regulations_policies/handbooks_manuals/aviation/risk_management/ss_handbook/.

[48] National Aeronautics and Space Administration, "Software safety standard," NASA, Washington, DC, Technical Standard NASA-STD-8719.13B w/Change 1, Jul. 2004, last accessed 13 Feb 17. [Online]. Available: https://system-safety.org/Documents/NASA-STD-8719.13B.pdf.

[49] P. J. Graydon and C. M. Holloway, "Planning the unplanned experiment: Assessing the efficacy of standards for safety-critical software," NASA Langley Research Center, Technical Memorandum NASA/TM-2015-218804, Sep. 2015, last accessed 8 Nov 18. [Online]. Available: https://ntrs.nasa.gov/search.jsp?R=20150018918.

[50] R. Patton, *Software Testing*, 2nd. Indianapolis IN: Sams Publishing, Aug. 2005, ISBN: 067232798-8.

[51] W. S. Humphrey, *PSP(SM): A Self-Improvement Process for Software Engineers*. Addison Wesley, Mar. 11, 2005, 368 pp., ISBN: 0321305493.

[52] J. C. Knight and E. A. Myers, "An improved inspection technique," *Commun. ACM*, vol. 36, no. 11, pp. 51–61, Nov. 1993, ISSN: 0001-0782. DOI: 10.1145/163359.163366.

[53] S.-A. Sansone and P. Rocca-Serra, *Interoperability standards—digital objects in their own right*, Oct. 2016. DOI: 10.6084/m9.figshare.4055496.

[54] British Standards Institute, *National comment template*, On BSI eCommittees System, last accessed 11 Dec 18.

[55] P. J. Graydon and T. P. Kelly, "Using argumentation to evaluate software assurance standards," *Information and Software Technology*, vol. 55, no. 9, pp. 1551–1562, Sep. 2013. DOI: 10.1016/j.infsof.2013.02.008.

[56] P. Steele and J. Knight, "Analysis of critical systems certification," in *2014 IEEE 15th International Symposium on High-Assurance Systems Engineering*, Jan. 2014, pp. 129–136. DOI: 10.1109/HASE.2014.26.

[57] N. G. Leveson, *Safeware: System Safety and Computers*. Addison-Wesley Professional, 1995, ch. 12 & 18, ISBN: 0201119722.

[58] N. Storey, *Safety Critical Computer Systems*. Pearson Education (US), Jul. 18, 1996, 472 pp., ISBN: 0201427877.

[59] M. E. Fagan, "Design and code inspections to reduce errors in program development," *IBM Systems Journal*, vol. 15, no. 3, pp. 182–211, 1976. DOI: 10.1147/sj.153.0182.

[60] Ministry of Defence, "Standards for defence—Part 2: Management and production of defence standards," UK Defence Standardization, Glasgow, Defence Standard 00-00 Part 2 Issue 5, Dec. 2010, Withdrawn 12 Dec 2012.

[61] "Principles and rules for the structure and drafting of ISO and IEC documents," International Organisation for Standardisation, Geneva, ISO/IEC Directives Part 2 Edition 7, May 2016, last accessed 1 Dec 18. [Online]. Available: https://www.iec.ch/members_experts/refdocs/iec/isoiecdir-2%7Bed7.0%7Den.pdf.

[62] Ministry of Defence, *Defence standard development*, last accessed 1 Dec 18. [Online]. Available: https://www.dstan.mod.uk/policy/defstan_development.html.

[63] E. Hull, K. Jackson, and J. Dick, *Requirements Engineering*, 2nd. Springer-Verlag, 2005, ISBN: 978-1-85233-879-4. DOI: 10.1007/b138335.

[64] P. K. Wilkinson, "Dealing with conflicting contract system safety requirements," *Journal of System Safety*, vol. 49, no. 2, pp. 14–17, Mar./Apr. 2013, ISSN: 0743-8826.

[65] R. D. Hawkins and T. P. Kelly, "A systematic approach for developing software safety arguments," *Journal of System Safety*, vol. 46, no. 4, pp. 25–33, Jul./Aug. 2010, ISSN: 0743-8826.

[66] Ministry of Defence, "Management of ship safety and environmental protection—Part 1: Policy," Ship Safety Management Office, Bristol, Joint Service Publication JSP430 Part 1 Issue 4, Jun. 2011.

[67] *Common criteria for information technology security evaluation*, v3.1 revision 3, last accessed 8 Nov 18, Jul. 2009. [Online]. Available: https://www.commoncriteriaportal.org/cc/.

[68] T. S. Ankrum and A. H. Kromholz, "Structured assurance cases: Three common standards," in *Ninth IEEE International Symposium on High-Assurance Systems Engineering (HASE'05)*, Institute of Electrical and Electronics Engineers (IEEE), 2005. DOI: 10.1109/hase.2005.20.

[69] A. Galloway, R. Paige, N. Tudor, R. Weaver, I. Toyn, and J. McDermid, "Proof vs testing in the context of safety standards," in *24th Digital Avionics Systems Conference*, IEEE, Oct. 2005. DOI: 10.1109/dasc.2005.1563405.

[70] C. Reed, D. Walton, and F. Macagno, "Argument diagramming in logic, law and artificial intelligence," *The Knowledge Engineering Review*, vol. 22, no. 01, p. 87, Mar. 2007. DOI: 10.1017/s0269888907001051.

[71] O. Scheuer, F. Loll, N. Pinkwart, and B. M. McLaren, "Computer-supported argumentation: A review of the state of the art," *International Journal of Computer-Supported Collaborative Learning*, vol. 5, no. 1, pp. 43–102, Jan. 2010, ISSN: 1556-1615. DOI: 10.1007/s11412-009-9080-x.

[72] Adelard, *Claims, arguments and evidence (CAE)*, last accessed 12 Feb 2017. [Online]. Available: http://www.adelard.com/asce/choosing-asce/cae.html.

[73] The Assurance Case Working Group (ACWG), *GSN Community Standard version 2*, last accessed 2 Dec 18, Jan. 2018. [Online]. Available: http://scsc.uk/SCSC-141B.

[74] C. M. Holloway and P. J. Graydon, "Explicate '78: Assurance case applicability to digital systems," NASA Langley Research Center, Final report to the FAA DOT/FAA/TC-17/67, Jan. 2018, last accessed 4 Nov 18. [Online]. Available: https://www.faa.gov/aircraft/air_cert/design_approvals/air_software/media/TC-17-67.pdf.

[75] RTCA Special Committee 205, "Software considerations in airborne systems and equipment certification," RTCA Inc., Washington, Recommendation DO-178C, Dec. 2011.

[76] IEC Subcommittee 56: Dependability, "Hazard and operability studies (HAZOP studies). Application guide," International Electrotechnical Commission, International Standard IEC61882, 2016. DOI: 10.3403/30309555u.

[77] S. Groom, "The life cycle and legal cycles in software engineering," last accessed 12 Jan 17, Master's thesis, University of Oxford Software Engineering Programme, Mar. 2002. [Online]. Available: https://www.cs.ox.ac.uk/signin/dissertation/lifecycle.pdf.

[78] RTCA Special Committee 205, "Model-based development and verification supplement to DO-178C and DO-278A," RTCA Inc., Washington, Recommendation DO-331, Dec. 2011.

[79] S. Jacklin, "Certification of safety-critical software under DO-178C and DO-278A," in *Infotech@Aerospace 2012*, NASA Ames Research Center, American Institute of Aeronautics and Astronautics, Jun. 2012. DOI: 10.2514/6.2012-2473.

[80] B. Hendrix, S. Dwyer, and D. West, "Model-based systems engineering and software system workshop," *Journal of System Safety*, vol. 53, no. 3, pp. 24–29, Winter 2017, ISSN: 0743-8826.

[81] Ministry of Defence, *MOD architecture framework*, last accessed 3 Dec 18, Dec. 2012. [Online]. Available: https://www.gov.uk/guidance/mod-architecture-framework.

[82] Systems Engineering and Integration Group (SEIG), "SEIG review of Defence Standard 00-56: Safety management requirements for defence systems—Part 1," Ministry of Defence, Tech. Rep., Nov. 2011, Unpublished.

[83] P. Watkinson, "Software engineering—methodology for critical systems," Master's thesis, University of Oxford Software Engineering Programme, 2012.

[84] The Open Group Architecture Forum, *TOGAF version 9.2, enterprise edition*, http://pubs.opengroup.org/architecture/togaf9-doc/arch/index.html, last accessed 10 Jan 19, Apr. 2018. [Online]. Available: http://pubs.opengroup.org/architecture/togaf9-doc/arch/index.html.

[85] J. L. de la Vara, A. Ruiz, K. Attwood, H. Espinoza, R. K. Panesar-Walawege, Á. López, I. del Río, and T. Kelly, "Model-based specification of safety compliance needs for critical systems: A holistic generic metamodel," *Information and Software Technology*, vol. 72, pp. 16–30, Apr. 2016, ISSN: 0950-5849. DOI: 10.1016/j.infsof.2015.11.008.

[86] N. Sannier and B. Baudry, "INCREMENT: A mixed MDE-IR approach for regulatory requirements modeling and analysis," in *Requirements Engineering: Foundation for Software Quality*, ser. Lecture Notes in Computer Science, C. Salinesi and I. van de Weerd, Eds., vol. 8396, Springer International Publishing, Apr. 2014, pp. 135–151, ISBN: 978-3-319-05843-6. DOI: 10.1007/978-3-319-05843-6_11.

[87] A. Ferrari, S. Gnesi, and G. Tolomei, "Using clustering to improve the structure of natural language requirements documents," in *Requirements Engineering: Foundation for Software Quality*, ser. Lecture Notes in Computer Science, J. Doerr and A. L. Opdahl, Eds., vol. 7830, Springer Berlin Heidelberg, Apr. 2013, pp. 34–49, ISBN: 978-3-642-37422-7. DOI: 10.1007/978-3-642-37422-7_3.

[88] E. Uusitalo, M. Raatikainen, M. Ylikangas, and T. Männistö, "Experiences from an industry-wide initiative for setting metadata for regulatory requirements in the nuclear domain," in *2014 IEEE 7th International Workshop on Requirements Engineering and Law (RELAW)*, IEEE, Aug. 2014, pp. 2–9. DOI: 10.1109/relaw.2014.6893474.

[89] Object Management Group, *About the unified modeling language specification version 2.5.1*, last accessed 10 Jan 19, Dec. 2017. [Online]. Available: https://www.omg.org/spec/UML/.

[90] Object Management Group, *Business process model and notation*, last accessed 10 Jan 19, Jan. 2011. [Online]. Available: http://www.bpmn.org/.

[91] IEC Subcommittee 56: Dependability, "Fault Tree Analysis (FTA)," International Electrotechnical Commission, International Standard IEC61025, 2006.

[92] IEC Subcommittee 56: Dependability, "Failure Modes and Effects Analysis (FMEA and FMECA)," International Electrotechnical Commission, International Standard IEC60812, 2018.

[93] J. P. Kincaid, R. P. Fishburne Jr, R. L. Rogers, and B. S. Chissom, "Derivation of new readability formulas (automated readability index, fog count and Flesch reading ease formula) for Navy enlisted personnel," Naval Technical Training Command, Millington, TN, Research Branch Report 8-75, Feb. 1975, last accessed 12 Jan 19. [Online]. Available: https://stars.library.ucf.edu/istlibrary/56/.

[94] Adelard, *Asce software*, last accessed 18 Jun 18, 2018. [Online]. Available: https://www.adelard.com/asce/choosing-asce/index/.

[95] PolarSys, *Opencert website*, last accessed 10 Jan 19, Dec. 2018. [Online]. Available: https://www.polarsys.org/opencert/.

[96] M. Cohn, *User stories*, http://www.mountaingoatsoftware.com/agile/user-stories, last accessed 10 Jan 19. [Online]. Available: http://www.mountaingoatsoftware.com/agile/user-stories.

[97] C. A. R. Hoare, "The emperor's old clothes," *Communications of the ACM*, vol. 24, no. 2, pp. 75–83, Feb. 1981. DOI: 10.1145/358549.358561.

[98] I. Habli and A. Rae, "Formalism of requirements for safety-critical software: Where does the benefit come from?" In *Proceedings of Planning the Unplanned Experiment: Assessing the Efficacy of Standards for Safety Critical Software (AESSCS)*, last accessed 22 Feb 19, May 2014. arXiv: http://arxiv.org/abs/1404.6802v1.

[99] W. Maalej, Z. Kurtanović, H. Nabil, and C. Stanik, "On the automatic classification of app reviews," *Requirements Engineering*, vol. 21, no. 3, p. 311, Sep. 2016, ISSN: 1432-010X. DOI: 10.1007/s00766-016-0251-9.

A ISO/IEC Checklist results

In the table below, the 'task' entries are taken directly from the table in the ISO/IEC Directives [61, Annex A]. The 'method' describes what was done to carry out the 'assessment' described in the Directives, and the 'comments' summarize the findings.

Task	**Method**	**Comments**
Structure	Read-through of contents list, skim heading structure.	OK
	Skim read headings for subdivision consistency.	Confusing subdivision in Foreword
	Skim read text for hanging paragraphs	9 identified (makes precise clauses referencing harder).
Plain language	Reviewed using Microsoft Word readability statistics.	Average sentence length: 23 words. Flesch Reading Ease: 29.7 Flesch-Kincaid Grade level 14.6.
	Reviewed acronyms using spreadsheet	6% of word count: 1201 instances of 48 acronyms.
Title	Read-through for scope and conciseness.	OK.
Foreword	Check content drafting.	OK.
Introduction	Check it describes document and why it is needed.	OK.
	Word search for requirement-like vocabulary ('shall', 'should') to check if purely informative.	6 'shall' statements identified in introductory sections using word search.
Scope	Check it describes what the document is for.	OK.
	Check it states where it applies.	OK - but lots of references to a 'scope of contract' which isn't well explained.
	Check it only contains factual statements.	5 of the notes and clauses appear more speculative than factual (one of which was previously identified).

Normative references	Word search for use of references in text.	Most of the references were actually referred to in the standard, but of the 19 documents listed, only 6 (ARP 4754A, ARP 4761, Def Stan 00-056, IEC 61508, DO-178 and DO-254) are actually normative. Def Stan 00-056 should be a dated reference (previously identified in naïve review), as specific details are cited. ISO 9001 is listed but not referred to at all.
Definitions & acronyms	Check for validity, word search for use of references in text.	All the 'definitions' are used somewhere in the document, but three of the acronyms were unused (ASEMS, IP and POSMS), and four more (DSRP, FHA, IPR, and PLD) were unnecessary as they only appeared in one place, alongside their full definitions. Several mistakes in acronym definitions had already been picked up by the naive review.
Figures and symbols	Not reviewed.	Not present in document.
Tables	Check for title, numbering and cross-references.	The only 'tables' in the standard are the definitions, acronyms and reference lists. None of these have titles or reference numbers other than for the clause in which they appear.
Annexes	Word search for references and status.	All are referred to in the text, but it is not obvious which ones are normative.
Bibliography	Not reviewed.	There isn't a bibliography, but there probably needs to be one to hold the non-normative references.
Requirement keywords	Word search for 'shall', 'should', 'may', 'can', 'must'.	'Shall' or 'should' appear 3 times in the Foreword, once in the Introduction and 11 times in notes. 'May', 'can' and 'must' appear numerous times, but are not defined to have special significance.
Legal issues	Read-through to check there are no requirements to comply with legislation.	OK.
	Read-through to check for patent, trademark or copyright issues.	OK.

Table A.1: Results of review using the ISO/IEC checklist for writers and authors [61]

B Requirements-level checklist mapping

The requirements-level checklists in Appendix D were drawn from a number of sources, supplemented by the author's own experience. Table B.1 shows how the sources were combined.

		Checklist attribute															
Source	**Attribute**	**Abstraction**	**Accuracy**	**Atomicity**	**Clarity**	**Completeness**	**Consistency**	**Currency**	**Feasibility**	**Legality**	**Modularity**	**Non-redundancy**	**Precision**	**Relevance**	**Satisfaction**	**Structure**	**Uniqueness**	**Verifiability**
Hull [63]	Abstract	X																
	Atomic			X														
	Clear				X													
	Complete					X												
	Consistent						X											
	Feasible								X									
	Legal									X								
	Modular										X							
	Non-redundant											X						
	Precise												X					
	Qualified														X			X
	Satisfied														X			
	Structured															X		
	Unique																X	
	Verifiable																	X
Ould [35]	Behavioural equivalence														X			
	Completeness					X												
	Internal consistency						X											
	Economy								X									
	Feasibility								X									
	Robustness																	
	Flexibility																	
Patton [50]	Accuracy		X															
	Code-free																	
	Complete					X												
	Consistent						X											
	Feasible								X									
	Precise, clear unambiguous				X								X					
	Relevant													X				
	Testable																	X
DO-178B [26]	Compliance with requirements		X												X			
	Algorithm aspects		X						X									
	Compatibility								X									
	Accurate & consistent		X				X											
	Traceable														X			
	Verifiable																	X
	Conformant to standards		X				X											

Table B.1: Mapping from sources to requirements-level checklist attributes.

C Relationship Model Concepts

Requirements

- Legal obligations
- Integrity principles
- PE Safety Objectives
  - PE Safety Requirements defined
  - PE Safety Requirements maintained through decomposition
  - PE Safety Requirements satisfaction demonstrated
  - Hazardous behaviour identified and mitigated
  - Confidence commensurate to risk
- Mandatory requirements
- Safety requirements
- Design integrity requirements
- Assurance requirements
- Derived safety requirements

Activities

- Management of Risk to Life
- Management of Design Integrity
- Whole life support
- PE Failure Assessment
- Tendering
- Contract award
- Information sharing
- Requirements Definition
- Standards selection
- PE management
- PE risk reduction and mitigation
- PE configuration
- PE implementation or selection
- PE safety audits
- Independent audits

Applicability levels

- Shall (mandatory)
- Should (tailorable)

Roles

- Contractor
- MOD
- Safety Committee
- Internal auditor
- Independent Safety Auditor

Artefacts

- Programmable Element
- Product, Service or System
- Non-PE components
- Open standard
- Scope of contract
- Scope of analysis
- Scope of supply
- Product, service or system info
- Safety Management System
- safety evidence
- PE Safety Summary report
- PE Safety Management Plan
- Configuration status
- Progress reports
- PE information set

Artefact attributes

- Risk to Life
- Design integrity
- Cyber security
- Mission performance
- Traceability

Artefact relationships

- Agrees
- Defines
- Describes
- Is independent from
- Justifies
- Meets shortfall in
- Records
- Satisfies
- Supports

D Standards Review Framework

D.1 Introduction

This framework provides guidance for reviewing assurance standards and similar documents that place standardized requirements on parties to achieve a particular aim. The first part of the document provides an introduction and general guidance. The second part consists of the methods that make up the review framework.

The framework describes eight sets of review methods, which we recommend are used in combination. Each method has different strengths and weaknesses and is likely to filter out different types of problem, as shown in Figure D.1. They are presented in order of their likely impact on a draft standard, so that methods that are likely to reveal fundamental problems appear before those that only find localized problems in small sections of text. If the issues that are found are fixed as soon as possible, using the methods in this order should avoid nugatory work. Ideally, the more powerful techniques should also be used earlier in the standard's development cycle, to help guide its design while causing minimal re-work.

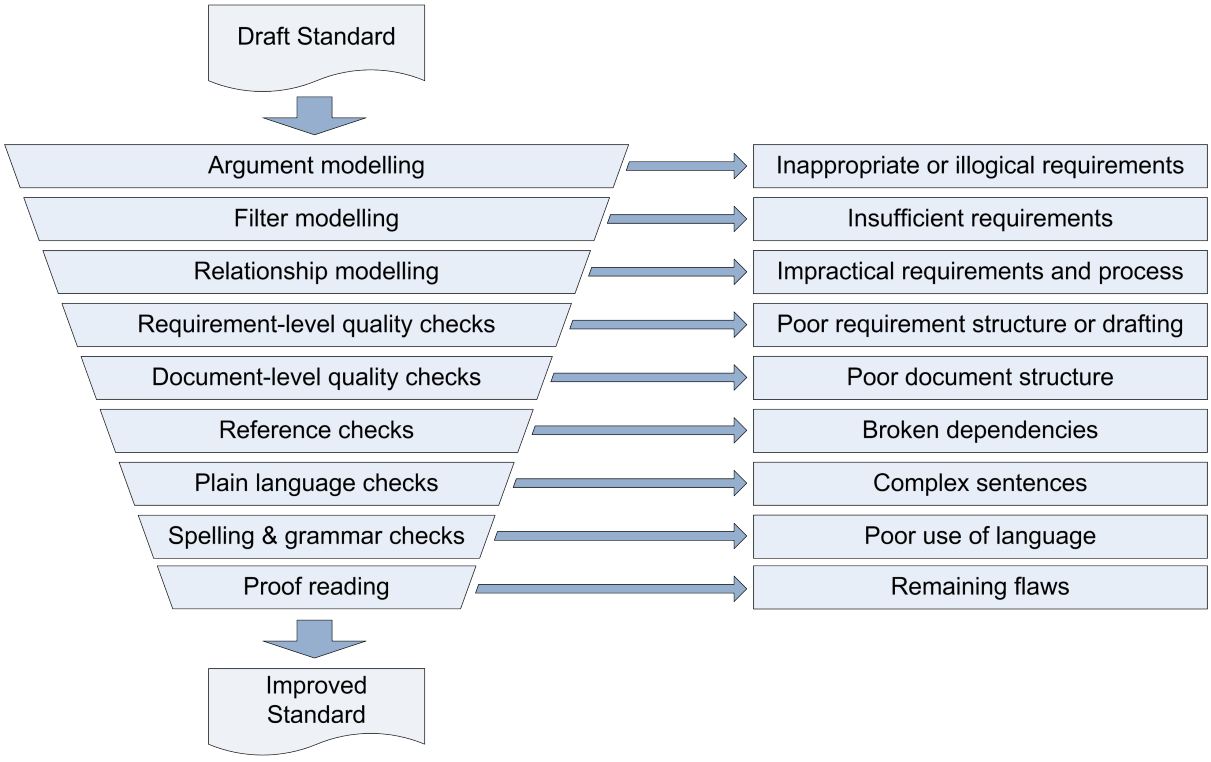

Figure D.1: Review methods as a set of filters for removing potential flaws in standards.

There is a trade-off between the time and resource taken to carry out a review and the rigour of the result. It isn't always proportionate to use every method described here,

but sometimes additional measures may be necessary for further rigour. To help with method selection, each method description includes notes on its intent and pros and cons.

D.1.1 Social reviews

All the review methods described here can be carried out by a single reviewer, however humans are fallible and make mistakes and omissions. Tool support and automation can help to an extent, but many of the methods depend on applying human knowledge, experience and judgement. To increase their rigour, multiple reviewers can conduct a method independently, then compare their results. There is often even more benefit to carrying out reviews as a social exercise, allowing reviewers to build off each other's ideas. Group reviews can be powerful, but they are resource intensive and need to be managed carefully to keep them focussed and efficient. It is likely to be more effective to socially review the outputs of these methods (e.g. a model or draft text, and the accompanying findings), rather than to carry out the analytical part of a method in a group.

Social reviews can be carried out at different levels of rigour. In a peer review, the group simply gathers to provide comment and feedback on the work product under review. In a walkthrough, the reviewer who did the work presents their findings to the group, explaining their reasoning while the other reviewers ask questions. Fagan introduced inspections as the gold standard of software reviews. These are more structured, with the participants assigned to represent specific roles or viewpoints. For standards, these inspectors would include the author, a standardization expert, user representatives from the communities that will impose and implement the standard, as well as subject matter experts from the problem domain. While it is important to get the necessary breadth of stakeholder input, Fagan recommended keeping the inspection team small to maintain their effectiveness. One of the inspectors is appointed as *moderator* to facilitate the review, and another (not the author) as *reader* to present the work under review. A *recorder* keeps track of the outcomes.

In any type of social review, prior preparation is vital to an effective process. The participants should be given an overview of the purpose of the standard and the review first, and have read through the material before the group part of the process. To reduce fatigue, the review should be scoped in relatively small chunks that can be tackled within a couple of hours. To maintain focus, the reviewers should limit themselves to identifying flaws in the standard, rather than getting distracted by trying to develop solutions; and the review needs to generate a tangible outcome, in terms of either an agreement that the material under review is acceptable, or a list of issues that must be fixed. Just asking the reviewers to find problems is unlikely to be especially effective. Fagan and others recommend prompting reviewers with a checklist of common types of issue, and gathering statistics on the results to help improve these lists. The methods in this guidance framework include a number of prompts which can be used in this way. Without such a mechanism, peer reviews and walkthroughs can easily focus only on what is there and overlook omissions.

Where it is not practical for reviewers to meet as a physical group, modern collaborative working tools can assist in coordinating distributed reviews. These range from tools that allow multiple reviewers to mark up a shared document with their comments, to cloud-based issue management systems that allow reviewers to raise issues, share their comments and track progress through to resolution.

D.1.2 Model-based review methods

Models provide abstractions of standards that can make it easier to understand their main concepts and allow them to be considered from different viewpoints. They can also provide alternative visualizations of the standard, which some reviewers may find easier to use than text-based documents. These factors mean that they can help identify more fundamental problems with their requirements than traditional text-based reviews.

Modelling methods do have some drawbacks though. As detail is abstracted away and the modelling process can introduce errors of its own, issues found in the model may not actually represent problems with the source standard, so must be carefully sentenced. Carrying out model-based reviews late in the development cycle can also be disruptive, as relatively small changes in a standard's underlying model can cause far-reaching changes to its text. To address this, model-based reviews should be carried out before text-based reviews, and ideally done at the design stage, before detailed drafting has started.

D.1.3 Text-based review methods

While model-based reviews are potentially more powerful, text-based reviews are still essential for standards written in natural language, to ensure that they correctly represent their requirements, are easy to use and understand, and look professional. Without checking the text, a conceptually good standard can be undermined by poor drafting.

D.1.4 Collating results

For a simple review process, it can be sufficient for each reviewer to mark up their comments on a copy of the document under review, either electronically or on paper. This works for a very small number of reviewers, but becomes rapidly more cumbersome to manage as more reviewers take part, as there may be multiple conflicting comments on the same parts of the document. To aid collation and sentencing of the results, it is helpful to use a form to collect their comments in a standard format. At the minimum, this should require a description of the problem and a reference to where it occurs in the document under review. The following fields are also helpful:

Proposed resolution Requesting a proposal for each issue makes resolution easier and makes it more likely that the standard's editors will understand the point being raised by the reviewer.
Reviewer identity To allow feedback, and help identify potential bias in the results (e.g. due to commercial interests).
Identification of document under review For configuration control. When multiple versions of a document are in circulation, it is important to understand which version was reviewed.
Impact category To help triage and prioritize issues for resolution.
Issue category To allow statistics to be gathered on the type of problems found, which can help inform the techniques to be used in future reviews.

Rather than use a stand-alone form, similar information can be captured in an online issue management system, using tags to capture metadata and comments to capture discussion on issues. This can be a more efficient method of providing feedback and allowing reviewers to see each other's comments.

D.1.5 Design for review

The design and layout of a standard can affect how easy it is to review and update. While the primary aim should be to create a document that is usable by its intended audience, the same factors that can make it easy to review are also likely to make it easier to understand, and hence to use. They are also likely to reduce the time and effort required to draft and maintain the document.

- Include explanatory text to justify requirements. This will help a reviewer understand whether a requirement is necessary or sufficient, and help a user of the standard understand the intent and importance of compliance.
- Make it clear which parts of the standard are normative (must be complied with) and which are purely informative.
- Aim for loose coupling between sections, so that changes made to one part of the text are less likely to require changes to other parts.
- Ensure that each part of the text can be uniquely referenced. Ideally, use a referencing scheme that will enable comments to be automatically sorted into the same order they occur in the standard.

D.2 Review Methods

Argument modelling	
Automation potential: Low	**Rigour:** Medium–high

Modelling an assurance standard as a logical argument can help review how the standard's requirements contribute to its overall goal, and whether they are likely to be sufficient to meet its intent. Visualizing this argument using a graphic notation can help stakeholders understand and constructively criticize the requirements and structure of the document.

Pros:

- Examines whether the standard's requirements logically meet its goals.
- Prompts reflection on what the standard is supposed to achieve.

Cons:

- Does not address process issues or readability.
- Requires familiarity with methodology.
- May focus on what is there, rather than potential omissions.

Method:

To use this review method, the standard is modelled as an argument structure, which is then subjected to criticism, as shown in Figure D.2. Various notations are available for the model, but Goal Structuring Notation (GSN) is recommended, as guidance and tool support is available. This guide assumes a basic familiarity with GSN; readers who need a background on the notation should consult the GSN Community Standard for an introduction.

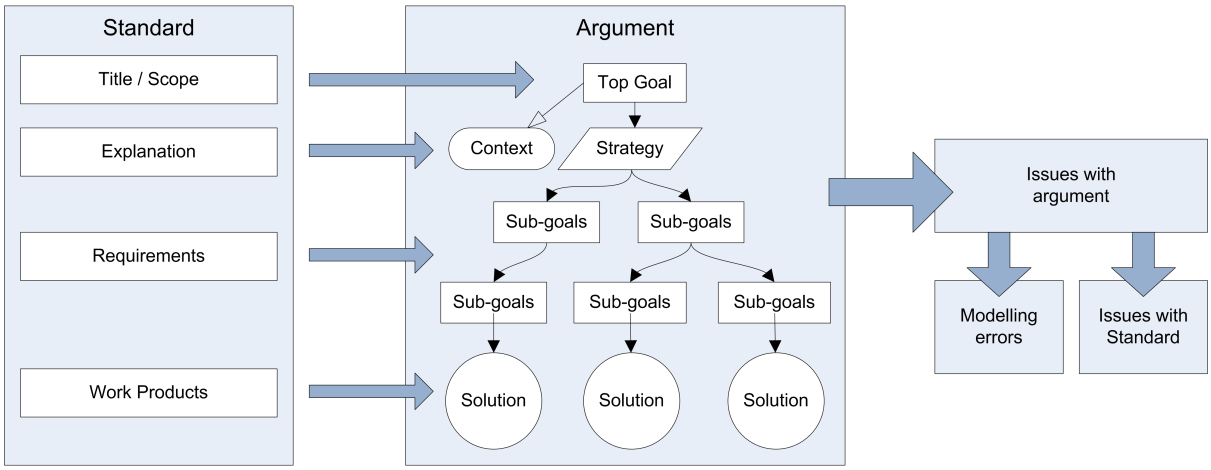

Figure D.2: Overview of argument modelling method.

First read the standard to identify which parts might form part of its argument. The top-level goal or argument claim should reflect the overall intent of the standard, and is likely to be found in its title, scope or introduction. The requirements of the standard are modelled as sub-goals: they are things that must be achieved in order to achieve the top-level goal. The sub-goals are arranged to show how they logically support one another. For example, three separate requirements in a standard might require a plan to be proposed, agreed and enacted. The goal of enacting the plan depends on it having been agreed, which in turn depends on it having been proposed. Intermediate levels of the argument might also be implied by the structure of the document, with sections or groups of requirements contributing to higher-level goals around topics like safety management or risk assessment. Work products (such as plans, designs, test results, etc.) are modelled as solutions, since when the standard is implemented, they will provide the tangible evidence that its requirements have been met. Explanatory notes and other informative text may provide information to explain why the requirements are necessary, or how they contribute to the overall effect. These can be included in the argument as strategies or context. Context nodes may also be used to model links out to other documents that are used as references. In some cases, dependencies on requirements in other documents might be modelled as assumptions or justifications.

e.g. an assumption that something will have occurred due to a process outside the scope of the standard, or a justification that something is appropriate because the standard requires an external reference to be followed.

To help capture the standard's argument, it is helpful first to highlight relevant parts of its text in different colours for each GSN node type (*goals, strategies, solutions, context, assumptions* and *justifications*), then annotate the links between them before redrawing as a diagram. To make the diagram readable, précis and re-write the text from the standard into ⟨noun-phrase⟩⟨verb-phrase⟩ argument form. e.g. "The contractor shall implement appropriate protection measures" might become a goal title that "Appropriate protection measures are implemented". During this process, take care to follow as closely as possible to the actual text of the standard, not what you might hope to have been written. Similarly, while there may appear to be missing portions of the resulting argument, you should not attempt to infer the missing segments at this stage. By capturing the standard faithfully, flaws in the text should be exposed by review of the argument structure.

Simple GSN diagrams can be drawn in a basic graphics package, but more powerful tools allow metadata to be captured alongside each argument node. If tool support allows, it is useful to annotate each node with a reference to the associated clause in the standard, and to copy the verbatim text into the node's description. This makes it easier to check that the standard has been properly captured.

The intent of the argument review is to check that—if the necessary evidence is provided to support the low-level claims—it is reasonable to assume the top-level goal would be satisfied. A simple review process is described in the GSN Community Standard:

1. **Argument comprehension.** Check that the argument structure faithfully represents what is actually written in the standard, and that errors have not been introduced in the modelling process.
2. **Well-formedness.** Ensure that the argument is structured properly, with solutions to support each bottom-level goal, and each lower-level goal somehow supporting the top-level goal. In some tools, these checks can be performed automatically.
3. **Expressive sufficiency checks.** Check that the argument is written in a way that can be understood, and there is enough context and strategy to explain the reasoning behind it.
4. **Argument criticism and defeat.** Check that the argument makes logical sense. Are the sub-goals relevant to the goals they support, and sufficient to satisfy them? Are there gaps in the reasoning, or goals that have no supporting argument or solution? Does the argument look convincing?

To make the criticism step of the process more robust, Graydon and Kelly suggest a carrying out the following steps for each small fragment of the argument:

1. Check the argument is not too vague or able to be misinterpreted.
2. Identify any unstated assumptions that the argument makes.
3. Review whether each assumption is reasonable.
4. Check that there are no obvious fallacies in the argument, such as omitting key evidence, or using reasoning that does not support the goal.
5. Look for common dependencies in arguments that are supposed to be independent.
6. Question whether the argument could be strengthened by using additional good practice.
7. Check that nothing in the argument is known to be ineffective or bad practice.
8. Qualitatively evaluate the strength of the argument—how convincing is it?

Further reading:

The Assurance Case Working Group (ACWG), *GSN Community Standard version 2*, last accessed 2 Dec 18, Jan. 2018. [Online]. Available: `http://scsc.uk/SCSC-141B`

P. J. Graydon and T. P. Kelly, "Using argumentation to evaluate software assurance standards," *Information and Software Technology*, vol. 55, no. 9, pp. 1551–1562, Sep. 2013. DOI: `10.1016/j.infsof.2013.02.008`

Filter model

Automation potential: Low **Rigour:** Low–Medium

This method models an assurance standard as a set of filters designed to remove flaws from a work product such as a software certification submission. Each technique or practice required by the standard is treated as a filter for different classes of flaw. The reviewer applies system safety techniques to the model to investigate how the filters can be optimized.

Pros:

- Good for prescriptive standards that recommend many techniques.
- Forces consideration of why particular techniques are required.

Cons:

- Requires a good understanding of the range of types of flaw that may be present in the work product that the standard is applied to, and the effectiveness of particular techniques in addressing them.
- Does not address the process aspects of the standard.
- Difficult to apply rigorously without detailed information in the standard.

Method:

First, the reviewer builds a filter model of the standard under review by identifying which techniques and practices it requires or recommends. They then determine which types of problem they are expected to prevent; and how they contribute to the standard's decision process to influence whether the product is declared to be acceptable or not.

Next, the reviewer investigates how this filter model could fail. In the context of the filter model, 'failing' means allowing flaws in a candidate product to progress into a certified or approved product. In general, this could happen due to a combination of both the presence of flaws in the candidate, and the failure of any filter to catch them. Informally, this can be investigated by comparing the list of likely flaws in candidate submissions against the types of flaw that the filters are expected to trap. Where detailed information is available in the standard under review, it may be possible to apply structured techniques such as Failure Modes Effects and Criticality Analysis (FMECA) or Fault Tree Analysis (FTA) to review the filter model in more detail. FTA can help reveal which combinations of problems could lead to certification failures. FMECA—treating shortfalls in the standard as failure modes—can help investigate the impact of these problems on the robustness of the decision process.

If particular types of problem are addressed ineffectively or not at all by the filter set, the requirements of the standard can be updated to compensate. This might be done by adding additional techniques, to ensure each type of potential flaw is covered by at least one technique; or adjusting the decision criteria to require more rigorous evidence, e.g. greater test coverage levels. The changes can either introduce measures to make it less likely that the flaws will be present in the candidate material, or make it more likely that existing flaws will be discovered.

Further reading:

P. Steele and J. Knight, "Analysis of critical systems certification," in *2014 IEEE 15th International Symposium on High-Assurance Systems Engineering*, Jan. 2014, pp. 129–136. DOI: 10.1109/HASE.2014.26

IEC Subcommittee 56: Dependability, "Fault Tree Analysis (FTA)," International Electrotechnical Commission, International Standard IEC61025, 2006

IEC Subcommittee 56: Dependability, "Failure Modes and Effects Analysis (FMEA and FMECA)," International Electrotechnical Commission, International Standard IEC60812, 2018

Relationship modelling	
Automation potential: Low	**Rigour:** Low–Medium

Metamodels can provide frameworks for modelling standards. The models produced using such frameworks can represent the concepts and entities involved in a standard, and the relationships between them. This gives new viewpoints to help visualize and constructively criticize a standard.

Pros:

- Can reveal process and logic issues not revealed by other methods.
- Flexible method.

Cons:

- Little guidance on method currently available.
- Relies on reviewer's experience to decide what models will be helpful and how to represent them.

Method:

Recent research has developed metamodels to describe assurance standards consistently, like the OPENCOSS Reference Assurance Framework (RAF) metamodel. RAF models of standards, compliance evidence, and the mappings between them have been used to help projects show compliance with multiple standards simultaneously. These models can also be used to help identify internal inconsistencies in a single standard.

The metamodels provide a framework of building blocks for constructing a model of an assurance standard or similar documentation, referred to generically as 'reference assurance frameworks'. In the RAF metamodel, the key elements are:

- **Reference requirements:** the requirements or objectives that have to be met in order to implement the standard.
- **Reference activities:** the processes or behaviours that need to be carried out under the standard.
- **Reference techniques:** the specific techniques required to be applied to carry out an activity or produce an artefact.
- **Reference role:** the actors or agents who are involved in implementing the standard.
- **Reference artefact:** the deliverables generated or provided in order to meet the standard. These may have particular **reference artefact attributes**, or there may be **reference artefact relationships** between artefacts.
- **Reference criticality kinds**: categorizations such as safety integrity levels that determine under what circumstances requirements apply.
- **Reference applicability kinds**: categorizations such as 'recommended' or 'mandatory' that determine the extent to which requirements apply.

To put the method into action, the reviewer first reads the standard to identify these elements, then assembles them into a model. A basic overview can be generated using a Unified Modelling Language (UML) object diagram, as in Figure D.3. Other helpful models could include UML sequence diagrams to explain the flow of information between actors, or Business Process Model and Notation (BPMN) diagrams showing process flow. These models are then examined to identify omissions, inconsistencies, or other problems.

Depending on the type of model, the following prompts could be used:

- Are there any entities which do not interact with the rest of the model?
- Is it clear which role(s) are supposed to participate in each activity?
- Is it clear when activities are supposed to occur and artefacts be produced?
- Do the activities occur in a logical order?
- Is there a logical flow of information artefacts?
- Are artefacts likely to be available at the time they are required as input for activities or techniques?
- Is it clear which role is responsible for producing or supplying each artefact?

Further reading:

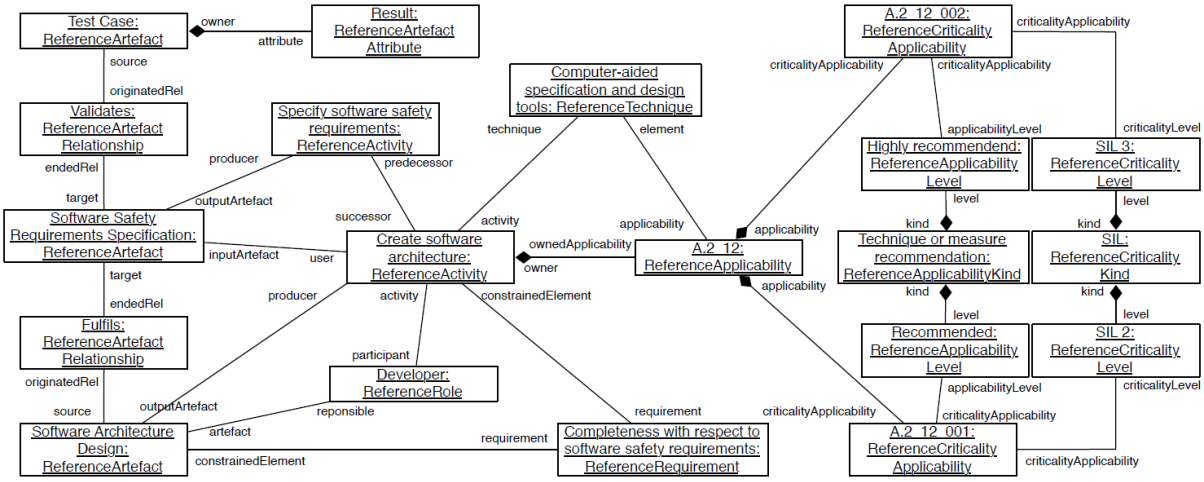

Figure D.3: RAF model of IEC 61508, from de la Vara et al. 2016.

J. L. de la Vara, A. Ruiz, K. Attwood, H. Espinoza, R. K. Panesar-Walawege, Á. López, I. del Río, and T. Kelly, "Model-based specification of safety compliance needs for critical systems: A holistic generic metamodel," *Information and Software Technology*, vol. 72, pp. 16–30, Apr. 2016, ISSN: 0950-5849. DOI: 10.1016/j.infsof.2015.11.008

Requirement-level checks

Automation potential: Low–medium **Rigour:** Low–Medium

Standards need to be written in a way that makes clear what to do to meet their requirements and how to tell if it has been achieved, without asking for either any more or any less than is necessary.

Pros:

- Easy to carry out without familiarity with the method.

Cons:

- May require domain knowledge to be effctive.

Method:

The testing and requirements engineering fields suggest various attributes for good requirements, which are captured in the lists below. The first set of checks applies to individual requirements and the second to the set as a whole. Checking some of these attributes requires domain knowledge (such as those for *currency*, *feasibility* or *legality*), but many issues can be identified by looking at the way the requirements are drafted. Some of these can be found quickly by using the suggested keyword searches. The checks can be improved over time if statistics are kept about the problems found, and used to improve the keyword lists.

There is a close link between requirements and the goals in an argument structure, so it can be helpful to carry out these checks in conjunction with the Argument Modelling review method. Missing requirements can correspond to gaps in the argument; non-atomic requirements expand to several separate goals; and goals with no solution may indicate non-verifiable requirements.

Checks for individual requirements:

Attribute	**Intent**	**Look for**
abstraction	Although the purpose of standards is to bring commonality, they should avoid unnecessarily constraining their users where the standard's goal does not require it. They should describe required outcomes abstractly rather pre-determining particular solutions.	Use of specific techniques or practices without specifying their aims.
accuracy	Requirements should be correct in what they ask for?	Do requirements specify the correct processes or algorithms to meet their goal? Are there factual errors in contextual information?
atomicity	Avoid compound requirements, which make it more difficult to demonstrate whether a requirement has been satisfied.	Could one part of the requirement be satisfied, but not another? Conjunctions: 'and'.
clarity	The language of the requirement must be understandable for the target audience.	Is the wording concise and jargon-free?
currency	Requirements should match the configurations of references and dependencies.	Are there dependencies on outdated policies, standards or organizations?
feasibility	It should be practical to achieve each requirement.	Inappropriate absolutes: 'always', 'every', 'all', 'none', 'never'. Requirements that are incompatible with common business practices.
legality	Complying with the requirement must not entail illegal behaviour.	
precision	Avoid vagueness; the requirement should not be open to multiple interpretations.	Let-out clauses: 'if necessary', 'as required' Vague wording: 'usually', 'generally', 'often', 'normally', 'typically', 'some', 'sometimes', 'ordinarily', 'customarily', 'most', 'mostly'.

relevance	Irrelevant requirements can add unnecessary cost; those that just appear irrelevant may not be complied with.	Check each requirement contributes to the overall goal, and it is clear why it is necessary.
uniqueness	It should be possible to refer to each requirement individually.	Check each requirement has a unique identifier.
verifiability	It should be practical to determine whether the requirement has been achieved.	Untestable requirements: 'cheap', 'efficient', 'fast', 'flexible', 'good', 'reasonable', 'small', 'stable', 'versatile'. Is it clear how satisfaction could be demonstrated?

Checks for sets of requirements:

The checks described below can be applied directly to plain text lists of requirements, but will be easier to carry out with tool support. Databases or requirements management tools can help improve the quality of requirement sets by maintaining traceability between requirements, structuring them into hierarchies or by priority, or providing features to organize similar requirements.

Attribute	**Intent**	**Look for**
completeness	The set of requirements should cover the entire problem space.	Logical omissions and shortfalls. Missing *else* clauses in *if . . . then . . . else* constructs. Incomplete lists: 'etc.', 'and so forth', 'and so on', 'such as', 'for example', 'e.g.'
consistency	Requirements should not contradict each other.	
modularity	Requirements on the same topic should be close to each other.	
non-redundancy	Separate requirements should not ask for the same thing.	Duplicate text.
satisfaction	It should be possible to see how each requirement ought to be satisfied and helps satisfy the overall goal.	Traceability to some kind of outcome or lower-level requirement.
structure	There should be a logical structure to help navigate the requirement set.	

Further reading:

E. Hull, K. Jackson, and J. Dick, *Requirements Engineering*, 2nd. Springer-Verlag, 2005, ISBN: 978-1-85233-879-4. DOI: 10.1007/b138335

R. Patton, *Software Testing*, 2nd. Indianapolis IN: Sams Publishing, Aug. 2005, ISBN: 067232798-8

Document-level checks

Automation potential: Medium **Rigour:** Low

Standards should be well-structured documents that present information in a way that is easy to use, so that the reader can easily understand what parts they need to comply with.

Pros:

- Relatively quick to use.
- Improves the usability of the standard.
- Makes it easier to demonstrate compliance.

Cons:

- Does not address problems with the subject matter of the standard.

Method:

These checks are mainly based on the drafting guidance used by the International Organisation for Standardization (ISO) and International Electrotechnical Commission (IEC). The reviewer uses them as a series of prompts to skim through the standard under review, looking for problems. Many of the checks can be aided by software tools that allow the document structure to be viewed and navigated, and by using keyword searches. Some checks can be made redundant by using strongly enforced templates with features such as automatic tables of contents and paragraph numbering.

Structural checks:

- Does the heading structure of the document make sense? Is the subdivision logical and consistent between sections?
- Is the heading structure accurately reflected in the table of contents?
- Does each clause have a reference number so that it can be uniquely identified? Are there hanging paragraphs, which are difficult to refer to separately from the rest of the section they appear in?
- Does each Figure and Table have a caption, a reference number and (if necessary) a key or legend?
- Is each Figure, Table, Appendix or Annex referred to somewhere in the main text?
- Is it easy to tell which parts of the standard are supposed to be normative, and which informative or provided for guidance only?

Content checks:

- Do the title of the standard and section titles make sense?
- Does the title match the scope of the standard?
- Does the Foreword make sense, and explain the background to this edition of the document?
- Does the Scope explain clearly what the standard is supposed to do and where it should apply? This section should be factual, rather than speculative or requirement-setting.
- Does the Introduction explain the intent and general approach of the standard? Is it purely informative?
- Are acronym and glossary definitions correct? Is each term used somewhere in the standard? Are particular concepts referred to consistently using the same terms throughout the document?
- Are contractually significant terms defined clearly, such as 'shall', 'should', 'may', 'must' or 'can'? Are they used consistently? Do they appear in parts of the standard that are not supposed to be normative?
- Do expressions of quantities use standard symbols and units?
- Are there any copyright, patent or trademark issues with any material incorporated in the text, such as use of proprietary processes?

Further reading:

"Principles and rules for the structure and drafting of ISO and IEC documents," International Organisation for Standardisation, Geneva, ISO/IEC Directives Part 2 Edition 7, May 2016, last accessed 1 Dec 18. [Online]. Available: `https://www.iec.ch/members_experts/refdocs/iec/isoiecdir-2%7Bed7.0%7Den.pdf`

Reference checks

Automation potential: Low–medium **Rigour:** Medium

To reduce duplication and redundant text in standards, if relevant requirements are already captured elsewhere, it is desirable to reference out to them, rather than repeat them. However, this introduces dependencies in standards, which need to be checked.

Pros:

- Relatively simple process.

Cons:

- Difficult to automate for external references.
- May require extensive review of referenced documents to check their relevance.

Method:

These checks focus on references to external documents, but the principles apply also to internal cross-references within a standard. Reviewing references is a largely manual process, but may be aided where good records have been kept of what has been referred to and why. Ideally, this should also be apparent from the text.

- Identify where references occur. Referenced external documents should be included in a reference list or bibliography, but it is also necessary to check individual instances of references in the text. Searching for keywords such as 'chapter', 'section' and 'clause' can help identify internal cross-references. Keywords such as 'Def Stan', 'ISO', 'IEC', 'Std', etc. can help locate external references—it may be helpful to build a domain-specific list to aid future checks.
- Are all the documents included in the reference list actually referred to by the standard?
- Do all the documents referred to in the text appear in the reference list or bibliography?
- Is enough bibliographic information provided to allow the reader to correctly identify and locate the referenced document?
- Is the material being referred to still available and relevant? Is a better alternative now required?
- Do all hyperlinks work, including internal hyperlinks within the document?
- Is it clear whether each reference is normative (must be followed in order to claim compliance) or informative?
- Does the reference refer to specific text in an external document and hence require the date or version of the document to be specified? Later versions may have a different structure or omit the relevant text.
- If the reference is not to a specific version of a document, is it clear whether it will be acceptable to use any subsequent version?
- Is the reference specific enough: i.e. does it require compliance with a whole standard, when only a particular section is actually necessary?
- Are the references too vague, e.g. referring to entire sets of regulations, or 'all legislation'?

These checks can be simplified to an extent if good practices have been followed in constructing the standard. These include using bibliographic databases to maintain external references, and using the cross-referencing features of word processing packages, rather than hard-coding paragraph numbers and other references.

Plain language checks

Automation potential: High **Rigour:** Low

Various automated checks can give an indication of the complexity of passages of text, allowing identification of parts that may be difficult to read.

Pros:

- Can assist in identifying parts of the standard that may require improvement.
- Quick to carry out.

Cons:

- Low accuracy.
- May drive wasted effort on over-simplification.

Method:

True readability is probably best judged by a human proofreader, but tools exist to help highlight complex portions of text that may be more difficult to read. Simple metrics like the Flesch reading ease score or Flesch-Kincaid grade level indicate complexity based on average sentence and word length. Other tests identify complex grammatical constructs. Such tests are built into many word processing packages. They are also available through online services or stand-alone software.

For the purposes of identifying areas of a standard to improve, tools that can find problem sentences or paragraphs are more useful than those that just give a summary metric for the whole document. However, one can use document-level readability metrics to set targets or acceptance criteria based on the reading ability of the target audience. Assurance standards are likely to have a well-educated, technical audience. Even so, it helps to keep them simple and easy to grasp for people not already familiar with their concepts. This Method is "fairly difficult to read". It has a Flesch reading ease score of 50 and a Flesch-Kincaid grade level of 10.

Simple language makes standards easier to use and understand. Hence, it should lead to more consistent results. However, as per Einstein's quote, "everything should be made as simple as possible, but not simpler". The text of standards should not be simplified to the point where their technical meaning is compromised.

Further reading:

Advice on Plain English is available from the Plain English Campaign:
`https://plainenglish.co.uk/`

Spelling and grammar check

Automation potential: Medium–high **Rigour:** Low

Spelling and grammar mistakes are often trivial, but can be significant when they change the meaning of the text. Aside from this, they cause distractions that make documents harder to read; and look unprofessional, reducing the credibility of the document. Spelling mistakes can also cause problems with searching or indexing documents automatically.

Pros:

- Can be automated to a reasonable extent.
- Can find gross errors of spelling or grammar.

Cons:

- Time-consuming and error-prone.
- Can fail to find usage that is inappropriate but technically correct.

Method:

Many word processing systems have automatic spelling and grammar checkers, which can quickly highlight potential errors. These checkers are not 100% accurate, due to limited dictionaries and rule sets. They may also fail to find misspellings that result in valid but incorrect words, or grammar that, while correctly constructed, does not convey the intended meaning. Some checkers also skip certain types of text, e.g. text that is embedded in graphics, capitalized, or inadvertently marked as in a foreign language or not to be checked.

To avoid these pitfalls, it is recommended to use a human proofreader in addition to using an automated checker—ideally not the original author.

It is also helpful to build up lists of commonly misspelt or confused words from your problem domain (e.g. 'discrete'/'discreet'), which can be searched for to check proper usage.

Proof reading

Automation potential: Low **Rigour:** Low

Simply reading a document and making comments is not always reliable, but can find problems with standards that other methods fail to find.

Pros:

- Can reveal many different types of problem.
- No special preparation or tool support required.

Cons:

- Time-consuming.
- Relies on knowledge and expertise of reviewer.
- High variability in results.
- Tends to focus on relatively low-impact issues.
- Poor at identifying omissions.

Method:

A proofreader reads the document, and records their observations. Proof reading can potentially reveal any type of issue with a standard, depending on the experience of the reviewer and the amount of attention paid to the task. If the reviewer has the appropriate subject-matter expertise, it can reveal problems such as outdated references or impractical requirements, which require human knowledge to identify and may not be suggested by more structured techniques. Typically though, this type of review misses many errors and focuses on issues that relates to small portions of the text, such as spelling or grammar errors; or structural problems like non-atomicity in individual clauses. The following points can help improve effectiveness:

- To maintain concentration, review small sections at a time but beware that this can make it harder to spot inconsistencies between sections.
- Consider working on paper rather than electronic documents to make it easier to spot errors and compare different parts of the text.
- Prompt reviewers to consider the standard from particular viewpoints (e.g. as a commercial officer or a software developer), or to look for certain classes of problem. This can increase detection of specific issues of interest, but may mean that other types of issue are overlooked.

E Feedback on Standards Review Framework

This appendix contains stakeholder feedback on the Standards Review Framework (Appendix D). Stakeholders were asked to provide brief feedback on whether the framework looked useful to help review and develop assurance standards, whether they thought it could be applied more broadly and whether they saw any practical problems applying it.

Respondents 1 & 2

This is a combined response from two members of the Defence Standardization (DStan) group. They are involved in coordinating the publication and review process for similar standards to Defence Standard (Def Stan) 00-055.

In our discussions we certainly recognised the models you refer to and the pros and cons associated with each, albeit we had never thought about these steps as discrete review models rather as good practice that could be employed dependant on the situation.

Our thoughts mainly focussed on why you carry out the reviews, is it really just to produce an improved standard?

I ask as this would be dependent on the context of the standard being drafted; this would be the same for any standard not just one dealing with assurance.

The rationale behind a civilian standard and military standard are different; the first wants an improved standard that meets the demand of the market (dare I say it potential sales defines the need of the market) and the second delivers an improved standard to specifically meet a military need (removing gold plating and over specifying to deliver a standard that delivers the mil delta, civil as poss and only mil as necessary).

These points are important and set the expectation behind what the improved standard will look like, and as such temper the way the review models are applied while offering the context of the verification and validation that the resulting improved standard must meet.

Our only other point relates to those involved, your paper I think relates to the drafter and the Subject Matter Expert (SME), and for the civil market this would be quite correct. However for a military standard we would look to involve the standard user, to ensure the improved standard meets the specific and defined defence need.

Respondent 3

Respondent 3 is an independent safety assurance consultant. He has considerable experience scrutinizing Ministry of Defence (MOD) projects, including software-intensive projects using Defence Standards 00-055 and 00-056.

A couple of points that I think are significant:

- It would be very useful to have a defined lifecycle model for a Standard (1.1 does mention "the standard's development cycle" but doesn't flesh out what that looks like or consider whether there are variants) and even better if that included Review processes at key stages in the lifecycle. I imagine that the type and purpose of the reviews would be different at different lifecycle changes. For instance at very early stages, you might want to decide whether the concept and purpose of a proposed Standard are good or fatally flawed and the whole development should be cancelled. Later on, the reviews are more concerned with improving the document and deciding whether it is fit to go ahead to the next stage (like decision gates).
- I was surprised that Figure 1.1 didn't include:
  - Omissions
  - Inconsistencies
  - Conflicts
- There is a good deal of technical and technical-sounding language used in the document without having been specifically defined, but maybe definitions are elsewhere. Some examples I found are:
  - Rigour, Levels of rigour
  - Flaws
  - Social review
  - Assurance standard
  - Proportionate
  - Problem domain
  - Work product
  - Model-based abstraction
  - Issue
  - Problem (type)
  - Filter
  - Triage
  - Sentence

Respondent 4

Respondent 4 is a member of the MOD team responsible for providing advice on software support. Prior to this role, she had extensive experience leading software assurance projects for a major prime contractor in the defence industry.

Thanks for the opportunity to comment on your MSc work. It is very interesting to see all the methods that can be used to review standards. A particular challenge I have always found is that everyone has their own way of reviewing documents and standards,and, sometimes it is difficult for the author to address the comments if the reviewer doesn't have a clear understanding of the aim of the document. This takes unnecessary time and effort, and I think this is well covered in your approach as a "set of filters".

The guidance framework looks good and useful, I do however have some reservations in terms of the practicality of using the ones I see as more complex to use such as the Filter Model and the relationship model. Obviously this is my personal opinion and I am happy to discuss further if you wish. My main concern is that some of the requirements

subject to review, e.g. RAF model of IEC 61508 figure are being segregated in a way that the reviewer will have to spend a lot of time trying to assemble a parallel picture before starting the actual review.

I believe the combination of the methods discussed can be a powerful tool to help reviewers and authors to produce and deliver good products, not only standards but other engineering documents. I also believe that depending on the size and complexity of the standards these tools could be well received by the various engineering groups as a way to harmonise the review process.

Respondent 5

Respondent 5 is a member of the MOD team that sponsors Def Stan 00-055. He is not a specialist in software, but is regularly involved in drafting and reviewing internal policy documents. He has previously been involved in reviews of Def Stan 00-055's sister standard, 00-056.

Overall I found it an interesting approach to something that I don't believe we currently do particularly well. My main concern is one of "frightening the horses" and introducing something which, on the face of it, looks hugely complex and deeply specialised. I think it's crying out for a degree of automation, although your comments seems to suggest the potential is low for the most complex of the review methods. More specific issues:

- It's unclear to me whether the process you define will result in a standard that is factually correct. The use of techniques such as GSN and relationship modelling will show to what extent the top-level goals of the standard are underpinned by a comprehensive set of supporting requirements, but I can't see anything that would verify that the basic principles are correct. Is there not a risk that you will end up with a carefully crafted standard that is fundamentally flawed and factually incorrect?
- Expecting reviewers to generate GSNs and FTAs/FMECAs introduces an additional risk of error because you're relying on them to get their GSN/FTA/FMECA correct. Would it not be better for the originator to use such tools and techniques as part of the development process and to present their argument that the standard is fit for purpose?
- A common issue in the review process is dealing with comments that conflict, where reviewers find themselves continually reviewing and revising each other's changes and the document goes round in circles rather than progressing. I was expecting to see something about dealing with feedback, capturing why some proposals may not be taken forward, and managing conflict.
- Where does an experiential review fit in to this process? What you describe is a structured approach to standards review which provides the reviewer with tools and techniques that generate a clear audit trail for all comments. Does there not remain a need for a more ad hoc, unstructured review by acknowledged SMEs? I couldn't necessarily see how that fitted into your model.