# Can software engineering methods give us better software safety standards?

**James Inge**

Defence Equipment & Support

Bristol, UK

**Abstract** *Are safety assurance standards actually software engineering artefacts, part of the decomposition of organisational goals into software requirements and designs? Loosely speaking, aren't they just software that is executed by an organisation rather than a computer? And if so, can we use software engineering methods to improve them? Software safety standards have a vital role in delivering safe products, services and systems. In critical systems, software failures can lead to significant loss of life, so it is 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. This implies that the review process for such standards is not entirely effective. Building on the author's MSc research at the University of Oxford, this paper examines how techniques from the domain of software engineering and allied fields can be used to improve the review of standards, potentially leading to better safety standards and safer systems. It presents a selection of potential techniques, evaluates the results of applying them to Def Stan 00-055, (the Ministry of Defence's Requirements for Safety of Programmable Elements in Defence Systems), shows how they can be helpful, and discusses the practicalities of applying them to review of new and existing standards.*

## 1 Introduction

### *1.1 Why standards for software safety assurance are important*

With software playing an ever-increasing role in delivering the functionality of critical systems, evidently it is important to ensure that software will operate safely. For critical systems, it is also important to gain confidence that this will

be the case before deploying the system. Failing to adequately plan to achieve this can be a notable cause of cost rises and delays to major projects.

A well-documented example of such delays was the UK Ministry of Defence (MOD) procurement of the Chinook Mk 3 helicopter (Fig. 1). 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 (Burr 2008). 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 (Bourn 2004). 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 (Morse 2013).

**Fig. 1**. Chinook Mk 3 (© Crown copyright 2016)

This Chinook example demonstrates that it is important for software not just to work, or even just to work safely: it needs to be demonstrably safe. Achieving this does not happen by accident. When an organisation needs to acquire new safety-critical software, it needs to communicate its safety requirements and its assurance requirements to the supplier as part of the contract. Using assurance standards for safety assurance such as Def Stan 00-055 (MOD 2021) or IEC 61508 (IEC 2010) is an efficient, repeatable way of doing this that captures accepted good practice, avoids the need for each project or organisation to work up its own requirements from scratch.

## *1.2 Why quality and review of standards matters*

To lead to good outcomes, standards need to have good functional content: what they prescribe needs to be technically effective. As the state of the art develops, standards need maintenance to keep them relevant. It is important to review and update them to incorporate new good practice and remove material which has become outdated. However, other non-functional aspects of standards are also important.

Standards need to be easy for their audience to understand and put into practice, or else risk their technical merit getting lost and the cost of their use rising. If a standard is ambiguous or hard to use, organisations implementing it are likely to budget more to account for the extra time required to understand its requirements and the risk of getting them wrong. If a standard is hard to understand, it is likely to be hard to review and technical problems may go un-noticed. This means that it is desirable for reviews of standards to look at not just their technical merit, but other quality factors that contribute to their practical effectiveness.

## *1.3 Are software safety standards software (and would it help if they were)?*

A case can be made to argue that assurance standards for software safety are in fact software artefacts themselves. Philosophically, they can be seen as sets of instructions, processes and supporting data that are executed by an organisation, rather than a machine (Fig. 2). More practically, by setting assurance requirements for software, they are a part of its high-level specification – part of the decomposition from high-level organisational goals to low-level software requirements.


**Fig. 2**. Standards as an input to the process of an organisation.

According to IEC 61508 (IEC 2010), the definition of 'software' includes 'any associated documentation pertaining to the operation of a data processing system'; and authors such as Ould (1999) and Patton (2005) include specifications among the set of artefacts to consider as part of the software quality assurance and testing processes.

Regardless of whether you accept the argument that standards actually are a type of software, the analogy is helpful. Both software and standards are abstract information products that have important functional and non-functional attributes. It is important for both software and standards to be technically correct to achieve their intent. As documents, it is also important for software code and standards to be easy to understand, so that problems can easily be identified and fixed, and so that they can be maintained efficiently in the future. With these similarities in mind, it is reasonable to ponder whether the discipline of software engineering can teach us lessons for improving standards.

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. In contrast, 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.[1] 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.

This paper reports on the results of MSc research carried out by the author at the University of Oxford (Inge 2019), investigating whether software engineering techniques could indeed inspire a more effective approach to review of safety assurance standards.

## 2 In search of a better review method

To attempt to identify a better way of reviewing software safety assurance standards and test the hypothesis that software engineering-inspired methods could provide improvements, the author carried out a literature review to identify potential methods, then applied each of these methods to Def Stan 00-055 Issue 4 (MOD 2016). The results were then compared to evaluate the quantity and type

[1] While the ISO/IEC Directives (ISO/IEC 2021) do contain some guidance that can help reviews (see section 2.2.2), in practice, the author has not been aware of this being applied proactively in the committees in which he has taken part.

of issue found by each technique, and the practicalities involved in carrying them out.

## *2.1 Approaches to evaluating standards*

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. An editor or editorial committee determines their resolution and the document is amended accordingly. A meeting or workshop may be held to resolve the comments, or this may be left to the editors. A 'comments form' often guides reviewers to respond in a certain format. The review template used by the British Standards Institute for comments on international standards asks for comments, proposed changes, a reference to the location in the standard's text, and a classification of the comment as general, technical or editorial. 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 interrelations between different parts of the standard.

While relatively little appears to have been published concerning review of standards, the author's experience does not seem uncommon, with other authors also bemoaning the quality of ad hoc standard review processes and seeking more rigorous methods (Graydon and Kelly 2013, Steele and Knight 2014).

Wong et al. (2014) did attempt to carry out a systematic review of five standards used in software safety. They scored them against twelve evaluation criteria that questioned how thoroughly each standard covers topics they deemed important like quality assurance and complexity management; if techniques like cost-benefit analysis or integrity levels were included; and other factors such as ease of use and active maintenance of the standard. They found some standards scored higher against some criteria and some against others, and suggested that projects should select their standards carefully to suit their needs. This analysis seems a little unsatisfactory: there was a justification for each criterion, but no explanation of how they chose the set as a whole. Their results seem less indicative of the quality of the standards, and more a consequence of the fact that the standards they evaluated had been written for different purposes, to fit into different regimes. Of the five standards evaluated, three were general system safety

standards, rather than being software-specific (Def Stan 00-56[2], the Federal Aviation Administration System Safety Handbook and Mil-Std-882D[3]). One was not safety-specific (DO-178B, which addresses development of safety-critical software, but assumes the safety analysis will be performed and safety requirements set according to other standards). Of the five, only NASA-STD-8719.13B[4] was a dedicated software safety standard.

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 clear guidance on how to evaluate a standard on its own.

Graydon and Holloway (2015) also investigated the evaluation of software safety standards, motivated by the lack of evidence for their efficacy. They argued 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, Graydon and Holloway claimed 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 (Ould 1999, Patton 2005). However, it can also be argued that the usefulness of V&V techniques extends beyond code to other artefacts used in the software development process. Taking this idea, we will explore how use of methods from software engineering, systems engineering and software safety can assist in evaluating and improving the quality of standards.

[2] The UK Ministry of Defence Safety management requirements for defence systems

[3] The US Department of Defense Standard practice for system safety.

[4] The NASA Software Safety Standard.

## *2.2 A review of reviews*

### 2.2.1 Naïve Review

To provide a baseline for comparison, a 'naïve' review of Def Stan 00-055 Issue 4 was carried out by the author. This involved reading a hard copy version of the standard and marking up apparent issues in red pen (approx. 4.5 hours work), then re-reading and recording a description of each issue, with a proposed resolution, into a comments table (a further 6 hours). The impact of the identified issues were scored according to Table 1; issues were also classified into 17 types of problem.

**Table 1**. Issue impact descriptors.

| 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. |

170 issues were recorded, the majority being of Low impact or only affecting readability. Grammar, ambiguity and punctuation were the most common categories. 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. Instead of explaining 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 would inevitably have been replaced by a new revision note when the next issue is published, even had the mistakes not been spotted.

The issues identified with the greatest impact tended to be ambiguities (especially with regard to which requirements were mandatory) or omissions in the text.

### 2.2.2 ISO/IEC checklist review

Use of checklists is recommended in code inspections to prompt checks for omissions and avoid reviewers focusing just on what is there, rather than what is missing. Checklists also provide the ability to capture learning for experience (Fagan 1976, Ould 1999, Patton 2005). The International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) include a *Checklist for Writers and Editors* as an annex to their Directives (ISO/IEC 2021), which appears appropriate for review of an assurance standard.

Applying the ISO/IEC checklist to Def Stan 00-055 took less time than the naïve review (2.5 hours), but resulted in fewer issues being identified (59). However, over 90% of these were new issues, and the majority of the issues were of Medium impact. The most significant category of issues, both in terms of quantity and impact, related to confusion around which parts of the standard were normative and which informative.

### 2.2.3 Enhanced Checklist

Having used a checklist from the standards community, the author then looked to software engineering for inspiration to produce an improved checklist. The checklists used in a Fagan inspection are supposed to condition inspectors to seek high-occurrence, high-cost error types (Fagan 1976). The previously obtained results identify these types of error, theoretically allowing a more efficient checklist to be generated – we want to identify the common problems more easily and also reveal more significant problems that are harder to spot.

In practice, this is easier said than done. The most common problems (42% of the total) related to grammar, punctuation, omitted words and spelling; and it was not clear how a checklist would help. One might have hoped that these issues could be detected automatically, but the Microsoft Word spelling and grammar check did not reveal any of the issues found by the manual reviews (it had may well have already been used in the drafting process). Many of the other issues had already been found using the ISO/IEC checklist – how could this be improved further?

Taking inspiration from software requirements engineering, a list of positive quality attributes of assurance standards was compiled, drawing on suggestions from multiple sources (Ould 1999, Hull 2005, Patton 2005), as shown Table 2.

**Table 2**. Attributes of good requirements for standards

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

These attributes were then applied as a checklist, both as general prompts for a read-through of the standard, and where possible, to inspire keyword searches for specific issues (e.g. finding 'and' to identify where requirements were non-atomic). Overall this process took approximately six hours and the issues it revealed were both more numerous and higher impact than using the ISO/IEC checklist. There was some overlap (10%) with previously identified issues, but the vast majority of the issues fell into the new categories listed in Table . For 7 of the 17 categories, no issues were found, hinting that there is perhaps room for refinement of the list.

### 2.2.4 Argument review

Ould 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 (Ould 1999). A manual review of 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 (Ould 1999). 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. One approach is to use diagrams and abstraction to help identify high-level faults in strategy (Ould 1999). This points to reviewing some kind of abstract version of the standard, suggesting that a modelling approach might be useful.

Models of argument structures are often used to construct safety cases for software. Making the argument explicit is intended to help the author explain their safety case and let reviewers identify problems in its logic. Argument structures have also been used to model assurance standards (Ankrum and Kromholz 2005, Galloway et al. 2005), and as a basis for their review (Graydon and Kelly 2013). 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 has used this method successfully to draft policy documents. 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 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 the question of 'what it means for a standard to "work"' (Graydon and Holloway 2015).

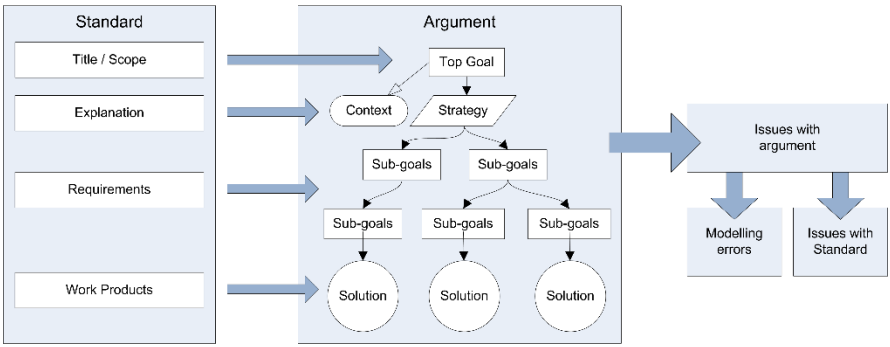


**Fig. 3**. Overview of argument modelling method.

To test out argument modelling as a review method, an argument was constructed using the Goal Structuring Notation (GSN) (ACWG 2018) in the ASCE tool (Adelard 2018), mapping parts of Def Stan 00-055 to GSN elements as shown in Fig. 3. This produced a complex network of 111 nodes, with a criss-cross of lines implying a tight coupling between different parts of the standard. Approximately half the issues found in the review were identified during the process of constructing the model, typically relating to requirements that were non-atomic requirements or had no obvious method of satisfaction. Further issues were found using the structure-checking tool built into ASCE. Manual review of the argument structure was carried out using guidance from (Hawkins and Kelly 2010), (Graydon and Kelly 2013), and the GSN Community Standard (ACWG 2018) – the latter document proving to provide the most practical advice. However, visual inspection of the structure did not reveal many further issues. Overall, the process took around ten hours, similar to the initial naïve review. A key high-impact findings was that various requirements that appear important to the standard are not actually well-defined.

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 practical to model but impractical to achieve.

### 2.2.5 Relationship modelling review

Goal structures appear to give useful insight into software safety assurance standards, but only present one view of the standard. Software design reviews are often supported by modelling that presents multiple coherent views of the software, using notations such as the Unified Modelling Language (UML). Different views can reveal potential problems with different types of relationship, such as temporal relationships between activities required by a standard. One framework developed specifically for use with assurance standards is the Reference Assurance Framework (RAF) metamodel (de la Vara et al. 2016).

Reviewing Def Stan 00-055 against the RAF metamodel identified 63 elements referred to in the standard that could be mapped to classes within the metamodel, including requirements, activities, roles and artefacts. The author attempted to construct a model using these classes in the Opencert toolset (Polarsys 2018), but found the functionality it provided impractical to use for a review. Attempts at constructing a process diagram to represent the activities required by the standard also failed, due to a lack of detail in Def Stan 00-055 about the sequencing of its activities. Instead, UML class diagrams were constructed, showing a static view of the types of relationship between entities described in the standard.

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. Modelling relationships identified fewer, but 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.

## *2.3 Filtering out problems*

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 but would not have made it much easier to understand 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, which depend on the expertise and knowledge of the reviewer. 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 for code inspections (Fagan 1976, Patton 2005).

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 types of problem got overlooked. The ISO/IEC checklist mainly found issues with the presentation and structure of the standard, but the checklist based on software requirements engineering principles found 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 the discussion about needing a clear purpose for software safety standards (Graydon and Holloway 2013), and opens a debate about the difference between software correctness and safety.

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 possible way to do this would be to design models of the processes, obligations, interactions and other relationships described in the standard before starting to draft the next iteration.

Fig. 4 shows the relative quantity of issues revealed by the different review methods. It shows that the traditional method of naively reading a standard and commenting did indeed highlight more potential problems, but these were generally less important than those found by the more sophisticated methods. However, this hides the fact that the different methods tended to find different types of problem: each method was valuable in a different way.

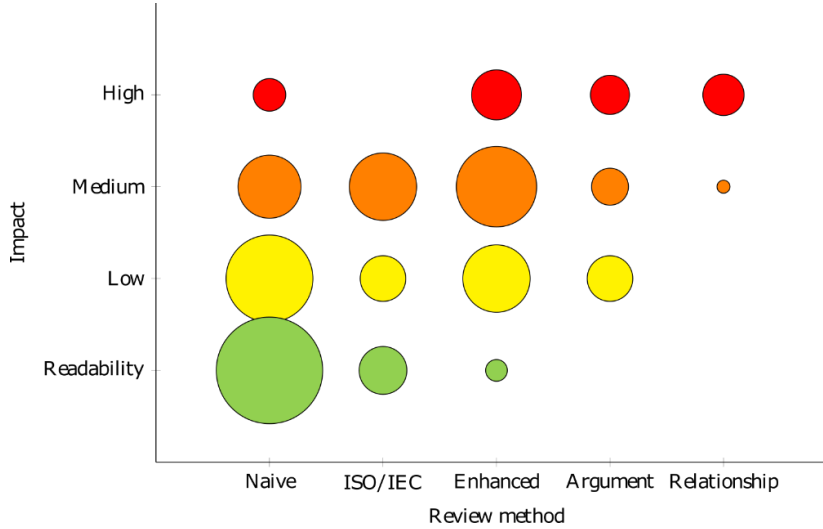


**Fig. 4.** Relative quantity of issues found broken down by impact and review method.

Graydon and Holloway (2015) suggested that one approach to understanding the intent of a software assurance standard is to consider it as part of a filter model, similar to that previously been proposed for regulation (Steele and Knight 2014). Graydon and Holloway saw standards as successful if they 'filter out' certain problems from software, either by making safety issues easier to spot, or by encouraging practices that reduce their likelihood or avoid them altogether.

We can adapt this filter model to the review of standards, by selecting review techniques to filter out different issues from the standard. As with a physical filter, it is more efficient to use coarse filters first: techniques that are likely to find major issues fast, rather than clog up the process with fine detail. Fig. 5 illustrates the principle, showing how different types of review checks can be used to filter out different types of issues, starting with those likely to require the most fundamental changes to put right if present in a standard.

In practice, it is likely to be impractical to apply all the different types of technique shown in Fig. 5 to a given review project. While the issues revealed through this work (Inge 2019) were included in the update of Def Stan 00-055 from Issue 4 to Issue 5, constraints on resource and timescales meant that it was not considered feasible to deploy the techniques described here more widely in the formal review.

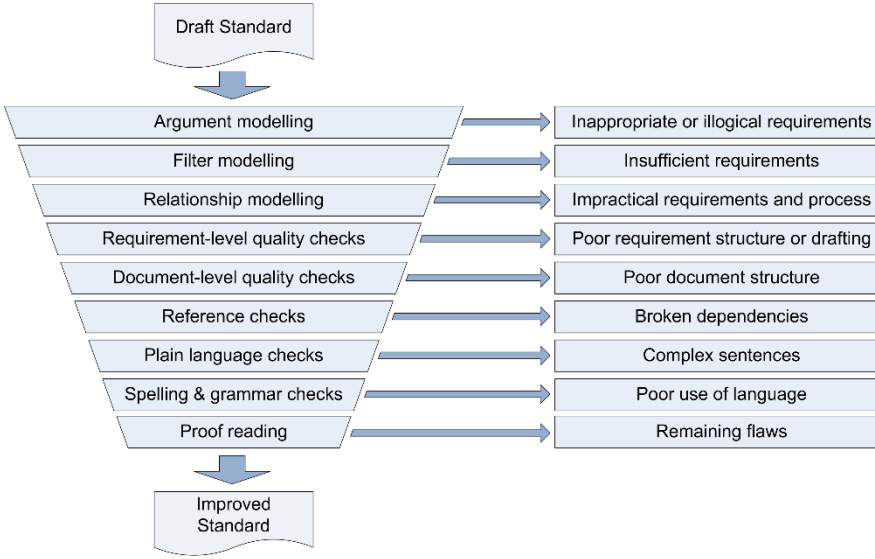


**Fig. 5**. A filter model for reviewing standards

Partly, this would have been a duplication of effort, and partly it was thought that construction of models and training reviewers in the use of the more sophisticated review methods would take disproportionate effort. However, use of these techniques might have more merit when designing a new assurance standard from scratch.

## 3 Conclusions

While it is a moot point whether software safety assurance standards are actually software, this research has shown that concepts from software engineering can be applied to make reviews of standards more effective. The requirements-setting parts of standards are similar enough to software requirements to make good practice from software requirements engineering applicable; and standards contain enough internal structure to make modelling methods useful. For both software and standards, there is a benefit to reducing complexity and making artefacts easier to understand, both to avoid errors and to aid maintenance.

However, there are some important differences. The use of natural language in standards makes it much harder to automate the review process, and the long iteration period of standards documents limits the return on investment provided by developing tools or learning involved techniques for their review. Typically assurance standards are only updated every few years, while agile software development may iterate versions every few weeks.

A potential area for both improving the drafting of standards and making them more amenable to review is the greater use of modelling in their construction (Model-Based Standards Engineering?) Representing standards using structured models could make automated checking more feasible, as well as potentially presenting different views of their requirements, that might reveal problems more easily to human reviewers.[5] However, to be cost effective, this kind of model would be likely to need to be built from the inception of the standard, rather than reverse engineered later.

A more practical way of improving reviews would be greater use of checklists as a prompt to reviewers to look for particular types of problem, and guide them as to how these might be found. Checklists based on desirable aspects of software or software requirements appear to have merit here, and could be applied more widely without undue cost.

## *3.1 Areas for further research*

A key area for research would be automated reviews for standards. While some tools such as spelling and grammar checkers are available, standards writers have nothing to compare to the range of automated tools for testing and verification available to software developers. International standards organisations are researching machine-readable standards (Bielfeld and Rodier 2021); combining semantically marked-up standards with formal models using methods such as the Reference Assurance Framework (de la Vara et al. 2016) would help bring useful tools for standards review a step closer.

Due to its scope as an MSc project, the research discussed in this paper has been limited to review methods that can be accomplished by a solo reviewer, but there is scope to research the benefits of group methods. Fagan recommended four people as the optimum size for a software code review (Fagan 1976), and many of the more recent practices grouped under the "agile methods" banner are intended for use by small development teams. However, standards are intended for re-use on multiple projects and have a much broader range of stakeholders (and hence potential reviewers) than typical software code. The uplift of DO-178B to DO-178C involved 374 active participants, creating a tension between the need to build wide consensus and the need to draft a coherent document (Daniels 2011). The software industry has developed distributed collaboration tools such as Bugzilla and Jira for issue management, and GitHub and SourceForge for source control; there is scope to investigate whether similar tools and methodology could help coordinate input from participants in large standards reviews.

[5] A model-based approach to standards might also prompt more re-use of common patterns or templates. This could perhaps help spread good practice and increase consistency between standards, aiding both reviewers and users of standards .

Finally, this work has focused mainly on the 'non-functional' aspects of standards: qualities such as self-consistency or readability. More research is needed to verify that safety assurance standards actually have the desired impact in terms of improving safety.

**Acknowledgments** The author is grateful to his supervisor Peter Bloodsworth and the Software Engineering Programme at the University of Oxford for their support in the MSc research on which this paper is based, and to Defence Equipment & Support for funding the MSc through their upskilling fund.



**Disclaimers** Views expressed in this paper are those of the author, and not necessarily those of his employer.

**References**

ACWG (2018) GSN Community Standard version 2, Assurance Case Working Group. https://scsc.uk/scsc-141B. Accessed 17 September 2022.

Adelard (2018), ASCE software, https://www.adelard.com/asce/choosing-asce/index/. Accessed 18 June 2018.

Ankrum TS and Kromholz AH (2005) 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). DOI: 10.1109/hase.2005.20.

Bieldfeld A and Rodier K (2021) What's next in Standards and Standards Publishing at ISO and IEC. In Typefi Standards Symposium 2021. https://www.typefi.com/standards-symposium-2021/whats-next-in-standards-publishing-iso-iec/. Accessed 15 November 2022

Bourn SJ (2004) Battlefield helicopters. House of Commons report HC 486 2003–2004. National Audit Office. https://www.nao.org.uk/wp-content/uploads/2004/04/0304486.pdf. Accessed 29 May 18

Burr T (2008) Chinook Mk3 helicopters. House of Commons report HC 512 2007–2008. National Audit Office. https://www.nao.org.uk/wp-content/uploads/2008/06/0708512.pdf. Accessed 29 May 18

Daniels D (2011) Thoughts from the DO-178C committee. In Proceedings of the 6th IET International Conference on System Safety 2011. Institution of Engineering and Technology. DOI: 10.1049/cp.2011.0266

De la Vara JL, Ruiz A, Attwood K, Espinoza H, Panesar-Walawege RK, López Á, Del Río I and Kelly T (2016) Model-based specification of safety compliance needs for critical systems: A holistic generic metamodel. Information and Software Technology, vol. 72, pp. 16–30. DOI: 10.1016/j.infsof.2015.11.008.

Fagan ME (1976) Design and code inspections to reduce errors in program development. IBM Systems Journal, vol. 15, no. 3, pp. 182–211. DOI: 10.1147/sj.153.0182

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

Graydon PJ and Holloway CM (2015) Planning the unplanned experiment: Assessing the efficacy of standards for safety-critical software. Technical Memorandum NASA/TM-2015-218804. NASA Langley Research Center.

Graydon PJ and Kelly TP (2013) Using argumentation to evaluate software assurance standards. Information and Software Technology, vol. 55 no 9 pp. 1551–1562. DOI: 10.1016/j.infsof.2013.02.008.

Hawkins RD and Kelly TP (2010) A systematic approach for developing software safety arguments. Journal of System Safety, vol. 46, no. 4, pp. 25–33. ISSN: 0743-8826.

Hull E, Jackson L and Dick J (2005) Requirements Engineering. Springer-Verlag. ISBN: 978-1-85233-879-4. DOI: 10.1007/b138335

IEC (2010) IEC 61508 series – Functional safety of electrical/electronic/programmable electronic safety-related systems. International Electrotechnical Commission

Inge JR (2019) Improved Methods for Review of Software Assurance Standards using Def Stan 00-055 as a Case Study. University of Oxford

ISO/IEC (2021) ISO/IEC Directives, Part 2:2021 – Principles and rules for the structure and drafting of ISO and IEC documents. International Organization for Standardization, International Electrotechnical Commission. https://www.iec.ch/news-resources/reference-material. Accessed 17 September 2022

MOD (2016) Def Stan 00-055 Issue 4 – Requirements for safety of Programmable Elements (PE) in defence systems. Ministry of Defence

MOD (2021) Def Stan 00-055 Issue 5 – Requirements for Safety of Programmable Elements (PE) in Defence Systems. Ministry of Defence

Morse A (2013) Major projects report 2012. House of Commons report HC 684-I 2012–13. National Audit Office. https://www.nao.org.uk/wp-content/uploads/2013/03/Major-Projectsfull-report-Vol-1.pdf. Accessed 29 May 18

Ould MA (1999) Managing software quality and business risk. John Wiley & Sons Ltd. ISBN 047199782X

Patton R (2005) Software testing. Sams Publishing. ISBN 067232798-8

PolarSys (2018) Opencert website. https://www.polarsys.org/opencert/. Accessed 10 January 2019

Steele P and Knight K (2014) Analysis of critical systems certification. In 2014 IEEE 15th International Symposium on High-Assurance Systems Engineering, pp. 129–136. DOI: 10.1109/HASE.2014.26

Wong WE, Gidvani T, Lopez A, Gao R and Horn M (2014) Evaluating software safety standards: A systematic review and comparison. In 2014 IEEE Eighth International Conference on Software Security and Reliability-Companion. pp. 78–87. DOI: 10.1109/SERE-C.2014.25