A Strategy for Improving Safety Related Software Engineering Standards

Norman E. Fenton, Member, IEEE Computer Society
and Martin Neil, Member, IEEE Computer Society

Abstract—There are many standards which are relevant for building safety or mission critical software systems. An effective standard is one that should help developers, assessors, and users of such systems. For developers the standard should help them build the system cost-effectively, and it should be clear what is required in order to conform to the standard. For assessors it should be possible to determine, objectively, compliance to the standard. Users and society at large should have some assurance that a system developed to the standard has quantified risks and benefits. Unfortunately, the existing standards do not adequately fulfill any of these varied requirements. We explain why standards are the way they are and then provide a strategy for improving them. Our approach is to evaluate standards on a number of key criteria that enable us to interpret the standard, identify its scope, and check the ease with which it can be applied and checked. We also need to demonstrate that the use of a standard is likely either to deliver reliable and safe systems at an acceptable cost or help predict reliability and safety accurately. Throughout the paper we examine, by example, a specific standard for safety critical systems (namely IEC 1508) and show how it can be improved by applying our strategy.

Index Terms—Standards, IEC1508, safety-critical, measurement, prediction, assessment.

1 Introduction and Background

Software standards exist to protect customers and the public from poor quality products and shoddy design practices. They should encourage developers to produce systems of acceptable reliability and safety. As well as helping users and developers of software systems, standards aim to help assessors by encoding those rules by which "quality" can be defined and evaluated. However, given the important role that standards play in systems engineering we must strive to continually improve them for the benefit of the user, developer, and assessor.

The goal of this paper is to describe a strategy to evaluate current standards and make recommendations for improving standards in the short and long term. The improvement strategy is based on the simple principle that a software standard is effective if, when used properly, it improves the quality of the resulting software products cost-effectively. The strategy is applicable to *any* software standards, but is especially pertinent to the safety critical ones. The latter can be viewed as simply the most demanding of the software standards; if you remove the safety integrity requirements material from such standards then they can be applied to any software system with high quality requirements. We considered evidence from the literature and also conducted a small number of empirical studies of specific company standards [17]. We found no evidence that any of the existing standards are effective according to the criteria that we will describe in this paper. This will come as no surprise to anybody who has sought quantitative evidence about the effectiveness of *any* software engineering method or tool. However, what concerned us more was that, in general, software standards are written in such a way that we could never determine whether they were effective or not.

There was certainly no shortage of standards to review. We came across over 250 standards (from various international and national bodies) that we considered to fall within the remit of software engineering. The common feature of all of them was that they defined some aspect of perceived "best practice" relevant for developing or assuring high quality software systems or systems with software components. Unfortunately, there is no consensus about what constitutes best practice, and it follows that there is no consensus about how to distinguish those best practice techniques that should always be applied. Thus, for standards of similar names and objectives we came across very different models of software quality and the software development process.

We discovered the following general problems in the standards we reviewed:

1) *Heavy overemphasis on process rather than product*. Traditional engineering standards assure product quality by specifying properties that the product itself must satisfy. This includes the specification of extensive product testing. Software engineering standards almost entirely neglect the product and concentrate on the development process. Unfortunately, there is no guarantee that a "good" process will lead to a good product, and there is no consensus that the processes mandated in many of the standards are even "good."
2) *Imprecise requirements.* The traditional notion of a standard is of a set of mandatory requirements. Such requirements must be sufficiently precise in definition so that conformance can be determined objectively by appropriate tests. Where no such precision is possible,

• *N.E. Fenton and M. Neil are with the Centre for Software Reliability, City University, Northhampton Square, London EC1V 0HB UK. E-mail: {nf, martin}@csr.city.ac.uk.*

Manuscript received 3 Apr. 1997; revised 21 Apr. 1998.
Recommended for acceptance by J. Rushby.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number 104809.

Authorized licensed use limited to: Vrije Universiteit Amsterdam. Downloaded on May 25,2025 at 10:40:28 UTC from IEEE Xplore. Restrictions apply.
0098-5589/98/$10.00 © 1998 IEEE

and hence where mandatory enforcement is impossible, standards bodies traditionally defined documents as "codes of practice" or "guidelines." In this respect software engineering is subjected to a proliferation of "standards" which are at best guidelines that could never be mandated realistically.

3) *Non-consensus recommendations*. Many of the standards proscribe, recommend, or mandate the use of various technologies which have not themselves been validated (in the sense of having been shown to be practically useful and effective). The standards may, therefore, be mandating methods which are not effective for achieving the aim of high quality systems.
4) *Unclear about risks and benefits*. It is not clear from standards how to predict the risks and benefits presented by the system. The language in standards studiously avoids and discourages any assertions of this kind and thus leaves the user unsure about whether the standard indicates a system's fitness for purpose or otherwise.
5) *Standards too big and poorly organized*. Most standards attempt to address the complete system development life-cycle. This results in extremely large documents containing sets of unrelated requirements, of which many will be irrelevant in a given application. Such standards are almost impossible to apply, and generally stay "left-on-the-shelf."

In Section 2 we define the scope of the paper, including a background of the key standards, and the scope of what we mean by *assessment*. In Section 3 we explain why standards are the way they are by describing the different philosophies underlying their construction. Section 4 summarizes our framework for evaluating standards, the steps of which are detailed in the five subsequent section. Thus, in Section 5 we discuss the notion of clarity and objectivity in respect to conformance assessment; our objective is to provide recommendations on how to rationalize and refine standards in such a way that we move toward the scenario where at least the obligations for the assessor are clear and objective. In Section 6 we explain how to classify requirements according to whether they focus primarily on one of three categories: process, product, or resource. Using this classification, we show how the safety critical standards concentrate on process and resource requirements at the expense of clear product requirements. We explain how to shift the focus towards the product requirements. In Section 7 we clarify the differing roles played by prediction and measurement in standards and touch on how subjective judgment informs risk assessment. Section 8 addresses the need for balance between differing types of requirements, while Section 9 explains how conformance assessment to a standard could involve greater objectivity, especially for the assessor.

Our emphasis is on how we can interpret and use standards *despite* their current weaknesses. We do not question the importance of standards to safety critical systems development, and a lot of very good work has gone into the existing standards. Thus, in Section 10 we make concrete recommendations on how standards could be improved in the short-term, and in Section 11 we address more difficult problems that relate to creating domain specific standards in the longer-term.

2 Scope: Standards and Assessment

In this section we define first the scope of the standards covered by the paper and then provide a brief background to the key standards. We finally define the scope of what we mean by assessment.

2.1 Scope of Standards Covered by This Paper

The scope of our work is the large set of standards which come under the umbrella of software engineering. This includes the set of IEEE/ANSI standards that are contained in the collection [10], such as IEEE/ANSI 983-1986 "Guide for Software Quality Assurance Planning," IEEE/ANSI 1008-1987 "Standard Software Unit Testing," and IEEE/ANSI 1012-1986 "Standard Software Verification and Validation Plans." Excluded from our scope are the standards for programming languages (such as the ANSI C standard) or standards for interfacing to programming languages or operating systems (such as the POSIX family of standards [11]). Of special relevance, however, are the standards which have been proposed for developing safety-critical software applications. The need for, and hence prevalence of, such standards has been more widely recognized in Europe than in the United States. Hence, the specific standards that we will refer to in depth in this paper originate in Europe.

2.2 Background to IEC 1508 and Other Safety Related Standards

The most important relevant standard is the generic IEC1508 "Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems" [8]. This standard is produced by the International Electrotechnical Commission, (IEC) a world wide organization promoting international standardization and cooperation in the electrical and electronic fields. In the 1980s the IEC recognized that computer-related systems were being used widely to perform safety functions. They decided that, if computer system technology was to be effectively and safely exploited, it was essential that those responsible for making decisions had sufficient guidance for the safety aspects. The standard IEC1508 was developed to set out a generic approach for ensuring the safety of programmable systems. It proposes an overall safety lifecycle for both hardware and software—addressing all stages from initial concept, through design, implementation, operation and maintenance to decommissioning, of a programmable electronic system used to achieve safety. A risk-based approach is adopted to determine the required "safety integrity," and numerical targets are set for safety performance. The standard adopts a broad range of approaches intended to achieve safety. The standard IEC1508 was the result of an extensive period of consultation lasting many years, and which included publication of an interim version IEC 65A [9] that was widely reviewed.

The standard IEC 1508 has created intense interest in Europe and is likely to be adopted widely in diverse industries. It is, therefore, timely to both publicze and scrutinise the standard. Hence, throughout this paper we use IEC 1508 as an example of applying our method. Needless to

Authorized licensed use limited to: Vrije Universiteit Amsterdam. Downloaded on May 25,2025 at 10:40:28 UTC from IEEE Xplore. Restrictions apply.

say there are potential improvements to be found in all software standards but since IEC1508 is not yet widely in use our suggestions may prove helpful to those involved in its future development and application.

By way of comparison, the most closely related standard to IEC 1508 of United States origin is DO-178B [18] which was produced especially for software in avionics systems. The major similarities in the standards are general assumptions about best practice software development and testing which evolve from the traditional software engineering life-cycle. In this respect both standards differ from the highly contentious United Kingdom draft defence standard for safety critical software systems Def-Stan 00-55 [14]. The latter assumes that the traditional software engineering life-cycle is inadequate for critical applications, and mandates the use of formal methods instead. Such methods are mentioned in both IEC 1508 and DO-178B only as possible approaches for the most critical components. The key difference between IEC 1508 and DO-178B is that most of the requirements are mandatory in IEC 1508 rather than guidelines as in DO-178B.

2.3 The Notion of Assessment

The word *assessment* is given many different interpretations in the standards community. Typically, in relation to safety critical software, there are three broad categories to consider:

1) Assessment in the sense of testing, reviewing, or proving a product;
2) Assessment in the sense of determining the integrity requirements to place on a software component based on system level hazards;
3) Assessment in the sense of comparing actual quality delivered against quality requirements.

We believe that standards already address the first two types of assessment adequately. However, it is the third, assessing against requirements (whether conformance or predictions), where most improvements can be made. Consequently, in this paper we concentrate on the third type of assessment.

3 Why Standards Are the Way They Are

It is important to understand why software engineering standards are the way they are before we attempt to evaluate them. In this section, we examine the standards-making process and describe the conflicts that lead to standards which are a confusing mix of different philosophies.

Creating a standard is a lengthy collaborative effort between industry, users, consultants, and academia, all with different and potentially conflicting ideas. This process can take many years and the eventual procedures and requirements have to be widely agreed upon before a standard is released. Each of the different parties to this process brings particular commercial and engineering perspectives, interests and experience. From an engineering perspective we can identify different, and sometimes opposing, forces at work on standards:

1) *Prescriptive vs. nonprescriptive regulation*. The majority of traditional engineering standards are highly prescriptive in the sense that they dictate the what's and how's of the product or the process used to develop and test that product. This can breed inflexibility and may present an unwelcome constraint on innovation. On the other hand, nonprescriptive approaches place appropriately wide bounds on what products should do and what test and development processes should be used and so allow the product developer a freer hand. Of course the ultimate form of nonprescription is where no standards exist to govern system development. Essentially prescriptive regulation may be more effective in areas which are technologically mature or where products or processes may be fairly uniform and already well understood.
2) *Conformance vs. risk-based assessment*. Conformance assessment involves asking to what extent developers have adhered to the requirements placed upon them by the standard. We can think of this act as primarily involving measurement of either the developed product against what the standard says the product should look like, or the development process against a set of process requirements given by the standard. Risk-based assessment involves predicting the possible future risks incurred by users of the system. Put simply, under a risk-based regime[1] the developer or assessor will assess the likelihood and severity of the hazards, or failures, presented by the system and compare them against what society considers acceptable. Conformance assessment approaches work well when we know what product forms or process approaches work well and provide adequate safety or quality. Risk-based approaches identify unknown risks and demonstrate how far the system can be trusted and are especially relevant to novel situations and unusual designs.
3) *Product vs. process assessment*. Product assessment involves measuring and evaluating the properties a product possesses against what is required. Sometimes such assessment is called "type" assessment because we are assessing a product against a generic type of product. For example, in shipping we might assess an oil tanker against a standard specifically targeted at oil tankers rather than one relevant to all ships. There are two levels of product assessment; functional and nonfunctional. Functional assessment involves examining what the system actually does. For example, evaluating a word processor might involve determining whether it allows text editing, file storage, and font selection. Nonfunctional assessment addresses properties such as safety, reliability, and maintainability. Obviously the nonfunctional properties are predicated on the functional properties—it only makes sense to evaluate reliability once you have some idea of what the system's functional properties are. Process assessment rests on the assertion that "a good process produces a good product." Attention is therefore given to the relevant facets of the software process *and* the resources used during the process. Process oriented standards will describe essential features of these processes such as configuration

1. The terms goal-setting or safety-case are equivalent to a risk-based approach.

Authorized licensed use limited to: Vrije Universiteit Amsterdam. Downloaded on May 25,2025 at 10:40:28 UTC from IEEE Xplore. Restrictions apply.

management, quality policies and testing and the resources used during these processes such as staff effort, development tools and budgets. Process oriented standards have shown themselves to be effective in industries like car manufacturing, where the products are standardized and the production repeatable. If the process produces identical products of required quality it is claimed there is little need to assess product quality directly. Total Quality Management (TQM) and Statistical Process Control (SPC) partly rest upon this assumption but can only succeed where the products are well defined and standardized.

Of course, the above three classifications are mutually dependent in many ways. Standards will vary in their mix of emphases. For example, as a result of the Piper Alpha accident Lord Cullen recommended a nonprescriptive, risk-based product approach be applied to health and safety matters in North sea oil and gas platforms [13]. This has been encapsulated in the idea of a *safety case*, defined by Shaw as:

> *"a clear, defensible, comprehensive, and convincing argument, developed by an operator, aimed at identifying the risks inherent in operating a system, demonstrating that the operating risks are fully understood, that they have been reduced to an acceptable level and are properly managed"* [19]

Central to all of the above issues are the roles played by novelty and technological change. Where we have mature well understood designs, such as in traditional hardware artifacts, we encode this understanding in prescriptive, product-oriented standards. The risks of using such products are known, so the assessment process simply involves evaluating conformance against the standard. If we know a product conforms we may trust that the product presents an acceptable risk. However, for novel systems, such as the majority of software systems, our experience is much smaller and assessment much more difficult. Even where we have mature, well-understood product designs of known quality, such as in word processing systems, the IT industry forces rapid technological change. This rate of change produces an explosion of variety of system types needing to be assessed. For regulators, assessors, and users as well as developers, this presents a challenging situation and has had a profound influence on the software standards we have.

Software engineering and safety critical system standards have failed to meet the challenge presented by system novelty and change. The prevailing attitude is that software engineering should at least present itself as being as mature as manufacturing and traditional engineering. Therefore, software standards typically pretend to be traditional and conformance oriented because this is what people expect. However, software standards can only be as prescriptive as traditional product engineering standards if we understand the properties of software products sufficiently well. This understanding is clearly not currently available even for the most common application domain products such as databases, real-time control systems, and word processors. Furthermore, the problem presented by system novelty dictate that a very large number of product specific software standards would need to be developed.

As a reaction to the problem of novelty some standards have embraced the risk-based approach where a system is assessed against its specific mode of use rather than against a generic product type. Under a risk-based approach the developer and user must analyse the system in its intended environment, identify the practical potential hazards presented by the system, predict their consequences and take action to reduce risk to an acceptable level with respect to the costs involved [7]. In this way the risk presented by novelty and complexity can be analyzed and accounted for, but at an expense felt appropriate for safety-critical systems only. The act of assessment in a risk-based approach moves from compliance against the standard (although this remains to some extent) to assessment of the system itself. In this way an assessment becomes a meaningful measure of trust for the eventual user of the system.

This is clearly a positive development but has led to the danger that some standards may present a confusing mixture of different philosophies. By attempting to combine conformance and risk-based clauses, prescriptive requirements, and nonprescriptive guidelines software standards may be in danger of losing their authority and the support of the developers it is trying to enlighten.

4 Framework for Evaluating Standards

In order to interpret and evaluate standards we consider a number of steps that must reflect the various requirements placed upon a standard by system developers, assessors, and users. These steps are:

1) *Determine the clarity of requirements in standards.* Developers are obliged to comply to the requirements set forth in a standard if they wish to claim conformance to it. Clear requirements make the system construction task easier and the evaluation of conformance more straightforward.
2) *Classify the requirements in standards.* We can classify requirements in a standard according to whether they are process, resource or product oriented. Requirements can also be separated into those that focus on compliance and those that are risk-based.
3) *Identify the role of measurement and prediction.* Assessing processes and products is more meaningful and useful where concrete measurements and sensible prediction models are required by the standard when assessing compliance and predicting risk
4) *Determine the balance of requirements.* Having classified requirements as product, process, or resource, it is important to consider the overall balance between these classes. For example, a standard whose aim is to improve product safety should not be made up primarily of just process and / or resource requirements.
5) *The need for objectivity in conformance assessment.* The extent to which an assessor can determine whether the standard has genuinely been followed will depend on whether the requirements, definitions and descriptions listed in the standard are observable and verifiable.

Authorized licensed use limited to: Vrije Universiteit Amsterdam. Downloaded on May 25,2025 at 10:40:28 UTC from IEEE Xplore. Restrictions apply.

Each of these criteria will be separately addressed in Sections 5, 6, 7, 8, and 9.

5 Clarity of Requirements in Standards

A standard is a collection of individual requirements. Our main concern is to consider the clarity of each mandatory requirement in the following two key respects:

1) *The developer's obligations for compliance*. Is it clear what is required in order to conform to the requirement? If not then the standard cannot be used in a consistent and repeatable way.
2) *An assessor's obligation for determining conformance*. Is it possible to determine conformance to a requirement reasonably objectively? If not then we may not be able to trust the assessor's results.

Generally, obligation 2) will follow from 1). For example, in IEC 1508,[2] Part 1, there are a number of requirements concerning the Safety Plan. Of these 6.2.2e asserts that the Safety Plan shall include a "description of the safety lifecycle phases to be applied and the dependence between them." The developer knows that certain specific information must appear in the document. To evaluate conformance the assessor only has to check that this information is there.

Conversely, however, it is not necessarily true that 1) will follow from 2). For example, for the software safety lifecycle we have:

> *Requirement 7.1.6.* "Each phase shall be terminated by a verification report."

Obligation 2) is clear. The assessor has, strictly speaking, only to check the existence of a specific report for each specified phase. However, the developer's obligations for the requirement is unclear; a subsequent requirement (in the software verification section) sheds little light on what constitutes an acceptable verification report:

> *Requirement 7.9.2.4.* "A Software Verification report shall contain the evidence to show that the phase being verified has, in all respects, been satisfactorily completed."

Unfortunately, in a key standard like IEC 1508 most requirements are unclear in both respects. For example, requirement 7.4.6.1a asserts that:

> "The source code shall be readable, understandable, and testable."

It is unclear what is expected of developers, while an assessor could only give a purely subjective view about conformance.

In traditional engineering standards it is widely accepted that the necessary clarity for both obligations 1) and 2) have to be achieved for all requirements [5]. Partly because of the immaturity of the discipline, software engineering standards do not have this clarity. Our objective here is to provide recommendations on how to rationalize and refine standards in such a way that we move toward the scenario where we can make the obligations for the assessor clear and objective where it is possible to do so.

6 Classifying Requirements in Standards

6.1 Processes, Products, and Resources

Our approach to interpreting standards begins by classifying individual requirements according to whether they focus primarily on *processes*, *products*, or *resources*:

A **Process** is any specific activity, set of activities, or time period within the manufacturing or development project. Examples of process requirements are:

> *7.1.4.* "Quality and safety assurance procedures shall run in parallel with lifecycle activities" (process here is *Quality and Safety Assurance*).

> *7.9.2.12.* "The source code shall be verified by static methods to ensure conformance to the Software Module Design Specification, the Coding Manual, and the requirements of the Safety Plan (process here is *Static Analysis*).

A **Product** is any new artifact, deliverable or document arising out of a process. Examples of product requirements are:

> *7.2.2.5a.* "The Software Requirements Specification shall be expressed and structured in such a way that it is as clear, unequivocal, verifiable, testable, maintainable and feasible as far as possible commensurate with the safety integrity level" (product here is *Requirements Specification Document*) *7.4.6.1b.* "The source code shall satisfy the Software Module Design Specification" (product here is *Source Code*).

A **Resource** is any item forming, or providing input to, a process. Examples include a person, a compiler, and a software test tool. Examples of resource requirements are:

> *Part 1, 5.2.1.* "All persons involved in any life-cycle activity, including management activities, shall have the appropriate training, technical knowledge, experience and qualifications relevant to the specific duties they have to perform." (resource here is *People*)

> *7.4.4.3a.* "The programming language selected shall have a translator/compiler which has either a "Certificate of Validation" to a recognized National/International standard or an assessment report which details its fitness for purpose." (resource here is the *Programming Language Compiler*)

> *7.4.4.3b.* "The programming language selected shall be completely and unambiguously defined or restricted to unambiguously designed features." (resource here is the *Programming Language*)

Ideally, it should be absolutely clear for each requirement which process, product, or resource is being referred to and which property or attribute of that process, product, or resource is being specified. The example requirements above are reasonably satisfactory in this respect (even though they do not all have the desired clarity discussed in Section 5). However, in many requirements, it is necessary to "tease out" this information. Consider the following examples,

> *7.4.5.3.* "The software should be produced to achieve modularity, testability, and maintainability."

2. Unless otherwise stated all examples are taken from IEC 1508 Part 3.

Authorized licensed use limited to: Vrije Universiteit Amsterdam. Downloaded on May 25,2025 at 10:40:28 UTC from IEEE Xplore. Restrictions apply.

Although this refers explicitly to the software production *process*, this requirement really only has meaning for the resulting *product*, namely the *source code*. Moreover, the three specified product attributes are quite different and should be stated as separate requirements (preferably in measurable form as discussed below in Section 5.3).

7.4.2.5. "The design method chosen shall possess features that facilitate software modification. Such features include modularity, information hiding and encapsulation."

Although this requirement refers to two *processes* (*design* and *modification*) its primary focus is a *resource*, namely the *design method*. Three very different attributes of the method are specified.

7.4.7.1. "Each module shall be tested against its Test Specification."

This is, strictly speaking, a combination of two separate requirements (and should be treated as such). One is a product requirement: the existence of a document (Software Module Test Specification) to accompany each module. The other is a process requirement that specifies that a certain type of testing activity has to be carried out.

The above classification of standards' requirements represents only the first stage in our proposed means of interpreting standards. It is important because it forces us to identify the specific object of the requirement, and to seek clarification where this is unclear. As a final example, consider the following requirement:

7.4.2.8. "Where the software is to implement both safety and nonsafety functions then all of the software shall be treated as safety-related unless adequate independence between the functions can be demonstrated in the design."

By thinking about our classification we can interpret this rather vague and confusing requirement. First of all we tease out the fact that this is a product requirement, but that there are two levels of product being considered: the software as a whole; and the set of individual functions which are being implemented. We need to break up the requirement into the following subrequirements:

1) The individual functions in the software shall be identified and listed in the Software Architecture Specification; safety-related functions shall be marked as such. (*This is a product requirement; the product is the Software Architecture Specification.*)
2) An independence check will be performed on each <safety, nonsafety> pair of functions identified in 1). (*This is a process requirement: how the check is to be performed needs to be further expanded.*)
3) A <safety, nonsafety> pair of functions are defined to be independent provided that... (needs to be further expanded). Two functions that are not independent are defined to be dependent. (*Product requirement*)
4) The whole system shall be partitioned into two groups of functions: Group A will contain all safety-related functions together with all nonsafety related functions which are dependent on at least one safety related function. Group B will contain all remaining nonsafety-related functions. The whole software system shall be classified as safety-related if Group B is empty. (*Product requirement*)

7 The Role of Measurement and Prediction

For product requirements, we make a distinction between attributes which are internal and those which are external. An *internal attribute* of product X is one that is dependent only on product X itself (and hence not on any other entity, be it another product, process or resource). For example, where X is source code, *size* is an internal attribute. An *external attribute* of a product X is one that is dependent on some entities other than just product X itself. For example, if product X is source code then the *reliability* of X is an external attribute. Reliability of X cannot be determined by looking only at X; it is dependent on the machine running X, the person using X, and the mode of use. If any of these are changed then the reliability of X can change. We have already seen numerous examples of external attributes in the above requirements (testability, maintainability, readability). Attributes like *modularity* (in 7.4.5.3) can, with specific definitions, be regarded as internal [4].

We have already mentioned that standards are influenced by two forces; the requirement to assess conformance objectively and the need to anticipate risks presented by novel systems. The act of determining conformance objectively is a form of measurement, where the assessor is attempting to find the "distance" between actual practice, as performed by the developer or evident in the product, and expected practice as embodied in the standard. For internal attributes, whether they are of process, resources or products, improving conformance assessment means that we must make their definitions and descriptions more rigorous and precise. However, anticipation of future risks and failures involves prediction, and is markedly different from conformance assessment. Prediction of the future dependability of software-based systems is an uncertain business requiring considerable human judgment and skill [15]. Assessors have little concrete evidence of the historical dependability of similar systems to base predictions on and even were this to exist the novelty of many systems may make this evidence necessarily weak [12]. Thus, the assessor is forced to make predictions conditional on the particularities of the system's design, the competence of the developers, and the methods and processes deployed. The weight that can be assigned to each of these differing sources of evidence is unknown and highly uncertain. Likewise, how we combine this evidence is an open research issue.

The distinction between internal and external attributes is now a widely accepted basis for software evaluation. Clearly, external attributes are the ones of primary concern, especially as, from a user's perspective, the ultimate goal is to determine acceptance criteria for safety critical systems. This means that we have to determine whether the system's external attributes like safety, reliability, and maintainability are acceptable for the system's purpose. In practice, these attributes cannot be measured directly until the system is being used so we are forced to make a decision about the acceptability of these attributes before the system is even extensively tested. This means that we are forced to look for evidence in terms of internal product attributes, or process and resource attributes.

Authorized licensed use limited to: Vrije Universiteit Amsterdam. Downloaded on May 25,2025 at 10:40:28 UTC from IEEE Xplore. Restrictions apply.

In IEC1508 the role of the assessor is explicitly to address conformance to the standard. As a measurement task this has its problems as described in Sections 5 and 6. However, where the assessor must address safety, the need to perform prediction emerges. Central to this is the notion of a SIL (Safety Integrity Level), which is a probability of failure interval. IEC1508 recommends the use of particular methods, techniques and measures[3] to developers if they want to increase the likelihood of achieving a particular SIL. For higher SIL levels supposedly more rigorous methods are recommended.

Table 1 shows the recommendations made for independent validation in Section 7.7.2.8 of IEC1508:

TABLE 1:
RECOMMENDATIONS FOR INDEPENDENT VALIDATION IN IEC1508

SIL	1	2	3	4
Independent Person	HR	HR(1)	NR	NR
Independent Department	–	HR(1)	HR(2)	NR
Independent Organization	–	–	HR(2)	HR

HR *highly recommended*
R *recommended*
– *neither recommended or not recommended*
NR *not recommended HR(1), HR(2): whether*
HR(1) *or* HR(2) *is selected will depend on higher complexity, similar design, novelty of design etc.*

If a developer wants to achieve a higher level of safety, say SIL 4, then it is highly recommended that an independent organization performs validation. For SIL 1 only an independent person is recommended and they are not required to be from another organization or department. IEC1508 is careful to state that these recommendations are guidelines only but expects deviation from them to be recorded and justified to the assessor.

We can make two important points about IEC1508 based on this example:

1) It uses deterministic *recommendations* as surrogates for what are really *predictions*. Rationally we would only recommend one act over another if we believed that it increased the likelihood of achieving a particular SIL. Furthermore, we would want these likelihoods ideally to be based on real experience and observation.
2) These recommendations are conditional on the situation to hand. The standard recognizes that factors such as the design novelty, complexity and poor experience of similar products should be accounted for when determining whether a recommendation should be followed. This is equivalent to determining whether these factors will affect detrimentally the likelihood of achieving the desired SIL.

From these points we can see that the important role played by prediction is left unclear and implicit because the language does not *appear* to admit uncertainty. IEC1508 attempts to force prediction into the straightjacket of conformance assessment where evidence is black and white rather than in shades of grey.

In IEC1508 it is left up to the assessor to determine *subjectively* whether a SIL target will be achieved and with what likelihood. To do this assessors must combine mentally the various recommendations, uncover their complex interrelations and use this evidence to inform their prediction. How, for example, is an assessor to determine the effects of novelty on whether the developer should employ an independent validation team from within or outside the organization? Should an organization be able to claim SIL 3 when they have no experience of similar products and the system requirements are complex? This will always be an uncertain task requiring considerable judgment but it is clearly difficult and one where the standards offers no guidance on how this should or could be done. Furthermore the assessor is given significant responsibility, beyond mere conformance evaluation, in the ultimately important process of predicting risk without the need to document or justify it.

We believe that those standards which set out to embrace a risk-based approach to assessment of complex systems should make explicit either the prediction systems to be used or the prediction calculus to be applied by the assessor. Doing this should make standards more relevant to the users of systems since assessment would be more solidly related to prediction of the risks and failures that they might face. The ESPRIT funded SERENE project [3] is using Bayesian Belief Networks to model the safety assessment procedures in various industrial settings as a way of making predictions about safety using diverse sources of evidence [16].

8 BALANCE BETWEEN TYPES OF REQUIREMENTS

The Oxford Encyclopedia English Dictionary defines a standard as:

> *"an object or quality of measure derived as a basis or example or principle to which others conform or should conform or by which the accuracy or quality of others is judged."*

This definition conforms to the widely held intuitive view that standards should focus on specifying measurable quality requirements of products. Indeed, this is the emphasis in traditional engineering standards. This point was discussed in depth in [5] which looked at specific safety standards for products (such as pushchairs). These explicitly specify tests for assessing the safety of the products. That is, they provide requirements for an external attribute of the final product. The measurable criteria for the testing process are also specified. There is, therefore, a direct link between conformance to the standard and the notions of quality and safety in the final product. Standards such as BS4792, the safety of pushchairs [2][4] also specify a number of requirements for internal attributes of the final product, but only where there is a clearly understood relationship between these and the external attribute of safety.

We contrast this approach with software safety standards. Very few requirements in these standards are well-defined product requirements. For example, [5] provided a detailed comparison of the requirements in BS 4792 with those of the software safety standard DEF-STAN 00-55. The latter consists primarily of process requirements (88 out of a

3. The word "measures" is used in IEC1508 to mean tasks or actions rather than the act of measuring something.

4. Commonly called a "baby stroller" in the United States.

Authorized licensed use limited to: Vrije Universiteit Amsterdam. Downloaded on May 25,2025 at 10:40:28 UTC from IEEE Xplore. Restrictions apply.

total 115 with 14 internal product and 13 resource requirements). There is not a single external product requirement. In contrast, BS 4792 consists entirely of product requirements (28 in total) of which 11 are external.

The distribution of requirements in DEF-STAN 00-55 seems fairly typical of software standards studied. The standard IEC 1508 is slightly different in that there is a very large number of resource requirements, but again we find far more process than product requirements. The problem with most of these requirements is that they are not demonstrated to be fit for purpose because, unlike BS 4792, there is no evidence that they enhance safety. For example, the following are typical internal product requirements from IEC 1508:

7.4.4.6. "The coding standards shall specify good programming practice, proscribe unsafe language features, and describe procedures for source code documentation."

7.4.2.11. "The software design shall include, commensurate with the required safety integrity level, self-monitoring of control flow and data movements."

7.4.5.3. "The software should be produced to achieve modularity, testability, and maintainability."

Each of these requirements (which would need further clarification to be usable anyway) represent particular viewpoints about internal structural properties that may impact on system safety. Unfortunately, there is no clear evidence that any of them really does [6]. The many process and resource requirements in standards such as IEC 1508 have an even more tenuous link with final system safety.

9 The Need for Objectivity in Conformance Assessment

The classification of standards" requirements into process, product, or resource represents a small step in clarifying standards. We need to classify further the requirements according to the ease with which we can assess conformance. Our objective is to identify the "rogue" requirements by making definitions and descriptions more precise. Rogue requirements are those for which the assessor's obligation (as discussed in Section 2) is unclear; that is, where an assessment of conformance has to be purely subjective. Note that we are only considering the objectivity of the act of conformance assessment, not the objectivity of predictions.

Assuming that a requirement refers to some specific, well-defined process, product or resource, we distinguish four degrees of clarity for each requirement (as shown in Table 2):

TABLE 2
CODES FOR DEGREE OF DETAIL GVEN IN ANY REQUIREMENT

Code	Interpretation
R	A reference only with no indication of any particular attribute(s) which that entity should possess
*	A reference for which only a subjective measure of conformance is possible
**	A reference for which a partially subjective and partially objective measure of conformance is possible
***	A reference for which a totally objective measure of conformance is possible

Ideally, the vast majority of requirements should be in categories '**' and '***' (with a small number of necessary "R's for definition). In the IEE pushchair safety standard BS4792 every one of the 28 requirements is in category '***' Although IEC 1508 is more objective than the vast majority of software standards reviewed (and is indeed a significant improvement on its earlier draft IEC SC65A), many requirements (including most of the examples presented so far) still fall into the "R" and '*' category. This means that conformance to such requirements can only be assessed purely subjectively. For example, it would be near impossible to assess conformance to the following requirement:

7.4.6.1a. "The source code shall be readable, understandable, and testable."

Here the requirement is effectively redundant unless we can achieve consensus amongst a large number of experts (which would be uneconomic). Alternatively, we could attempt to re-write it in a form which enables us to check conformance objectively. If there is mutual agreement (between developer and assessor) in the overall value of a requirement (however vague) then this is the option we propose. However, we stress that there is a considerable difference between

1) making a requirement objective, and
2) being able to assess conformance to a requirement objectively.

Option 1) is generally very difficult and often impossible; in an immature discipline there is even some justification for allowing a level of subjectivity in the requirements. It is only option 2) that is being specifically recommended. The following example explains the key difference between 1) and 2) and shows the different ways we might interpret requirements to achieve 2). There are generally many ways in which this can be done:

EXAMPLE 1. We consider how we might interpret requirement 7.4.6.1a above in order that we can assess conformance objectively. First of all we note that there are actually three separate product requirements, namely:

1) Each software module shall be readable;
2) Each software module shall be understandable;
3) Each software module shall be testable.

We concentrate on just 1) here. Consider the following alternative versions:

1) To accompany each software module a report justifying readability will be produced.
2) To accompany each software module a report justifying readability will be produced. This report will include a section that explains how each variable in the module is named after the real-world entity that it represents.
3) The ratio of commented to noncommented code in each software module shall be at least 1 to 4, and the total size shall not exceed 1000 LOC.
4) An independent reviewer, with a degree in Computer Science and 5 years experience of technical editing, shall devote a minimum of 3 hr to reviewing the code in each software module. The reviewer shall then rate the module for readability on the following 5-point ordinal scale:

Authorized licensed use limited to: Vrije Universiteit Amsterdam. Downloaded on May 25,2025 at 10:40:28 UTC from IEEE Xplore. Restrictions apply.

- 0 (totally unreadable)
- 1 (some serious readability problems detected, requiring significant rewrite);
- 2 (only minor readability problems detected, requiring rewrite);
- 3 (only trivial readability problems detected);
- 4 (acceptable).

The module must achieve a rating of 3 or higher.

Each of the above versions can be checked for conformance in a purely objective manner even though a large amount of subjectivity is still implicit in each of the requirements. In the case of 1) we have only to check the existence of a specific document. This is a trivial change to the original requirement since we have still said nothing about how to assess whether the document adequately justifies if the module is readable. Nevertheless, we have pushed this responsibility firmly onto the developers and not the assessors. Alternative 2) is a refinement of 1) in which we identify some specific criteria that must be present in the document (and which might increase our confidence in the readability argument). For alternative 3) we have only to check that the module has the right "measures." A simple static analysis tool can do this. In alternative 4) we have only to check that the rating given by the independent reviewer is a 3 or 4 and check that this person does indeed have the specified qualifications and experience.

In each of the alternative versions measurement plays a key, but very simple role. In the case of version 4) the requirement is based on a very subjective rating measure. Nevertheless, we can determine conformance to this requirement purely objectively. None of the alternative requirements except 3) is a requirement for which the module itself (a product) is the focus. Alternatives 1) and 2) are both requirements of a different product, while Alternative 4) concentrates on the results of a reviewing process.

Example 1 confirms that being able to assess conformance to a requirement objectively does not mean that the requirement itself is objective. Nor, unfortunately, does it always mean that assessment will be easy. The approach that we are proposing is to move toward identifying measurable criteria to replace ill-defined or subjective criteria. This is consistent with the traditional measurement-based approach of classical engineering disciplines. Texts such as [4] explain how to move toward quantification of many of the subjective criteria appearing in a standard such as IEC 1508. The following example further illustrates the method:

EXAMPLE 2. Requirement 7.2.2.5a asserts "To the extent required by the integrity level the Software Safety Requirements Specification (SSRS) shall be expressed and structured in such a way that it is as clear, precise, unequivocal, verifiable, testable, maintainable and feasible as possible commensurate with the safety integrity level." Each of the required attributes here (which need to be treated as separate requirements) are ill-defined or subjective. In the case of "maintainable" there are a number of ways we could interpret this so that we could assess conformance objectively. The most direct way is to specify a mean or maximum time in which a change to the SSRS can be made. Since such measures are hard to obtain it may be preferable to specify certain internal attributes of the SSRS, such as: the electronic medium in which it must be represented; the language in which it has to be written; that it has to be broken up into separately identifiable functions specified using less than 1000 words each; etc. Specification measures such as Albrecht's Function Points [1] might even be used. A radically different approach is that of alternative 4) in Example 1 where we simply specify what expert's rating of maintainability has to be achieved.

10 A Strategy for Improving Standards

So far we have concentrated on how we can interpret and use standards despite their many weaknesses. We do not question the general importance and value of standards to safety critical systems development. Nevertheless, there are very wide differences of emphasis in specific safety-critical standards. For example, DEF-STAN 00-55 and IEC1508 are totally different in their underlying assumptions about what constitutes a good software process; DEF-STAN 00-55 mandates the use of formal specification (and is structured around the assumption that formal methods are used), while IEC1508 mentions it only as a technique which is "highly recommended" at the highest safety integrity level (level 4). Clearly the standards cannot all be equally effective. They are certainly not equally easy to apply or assess.

It follows that there is a need for evaluating the effectiveness of standards, especially when we consider the massive technological investments which may be necessary to implement them. What we have described so far may be viewed as a "front-end" procedure for standards evaluation. This is like an intuitive quality audit, necessary to establish whether a given standard satisfies some basic criteria. It also enables us to interpret the standard, identify its scope, and check the ease with which it can really be applied and checked. However, for proper evaluation we need to demonstrate that, when strictly adhered to, the use of a standard is likely either to deliver reliable and safe systems at an acceptable cost, or help predict reliability and safety accurately.

We looked at how to assess standards in some of these respects [17]. The basic impediment to proper evaluation is the sheer flabbiness of the relevant standards. Many of the standards address the entire development and testing lifecycle, containing massive (and extremely diverse) sets of requirements. It makes no scientific sense, and is in any case impractical, to assess the effectiveness of such large objects. Thus we use the notion of a *ministandard*. Any set of requirements, all of which relate to the same specific entity or have the same specific objective, can be thought of as a standard in its own right, or a *ministandard*. Rather than assess an entire set of possibly disparate requirements, we instead concentrate on ministandards.

From our discussion on the forces at work during the creation of standards there emerged three clear ministandard types:

Authorized licensed use limited to: Vrije Universiteit Amsterdam. Downloaded on May 25,2025 at 10:40:28 UTC from IEEE Xplore. Restrictions apply.

- *Product ministandards*. A product ministandard would cover a specific *product* entity, such as a test plan or a software module, describe and define its desirable attributes and specify a measurement procedure whereby conformance could be assessed.
- *Process ministandards*. A process ministandard would cover similar ground to the product ministandard except it would address process and resource entities rather than product entities. For example we could imagine *mini*standards for the requirements specification phase, the maintenance process or the design method.
- *Risk-based ministandards*. A risk-based ministandard would act as the glue to combine the product and process ministandards together. The risk-based ministandard would combine the differing evidence from both process and product ministandards to make a prediction about *another* product or process attribute, which might have its own ministandard. For example, we might take evidence about conformance to a requirements specification phase ministandard to predict whether conformance to a requirements document ministandard is likely to be achieved as a result of following the specification process. In this way we can invoke complex cause-effect chains of inference between standards and evaluate one standard's usefulness in terms of its effects on others.

The need to decompose standards into manageable ministandards is a key stage in the evaluation strategy described in [17]. Many software-related standards are written in a way which makes this decomposition extremely difficult. Any system developer or user would select those ministandards relevant to the systems development and use.

However, the parts of IEC 1508 relevant to software are structured in a naturally decomposable way. We can identify seven key ministandards in the relevant parts of IEC 1508:

Process Ministandards

- Process of Specifying Safety Integrity Levels (Part 1, Section 8)
- The Safety Plan (Part 1, Sections 6 and 7, and Part 3, Sections 7.1 and 7.2)
- Resources (Part 1, Section 5 and Part 3, Section 5 which concentrate on people; and those requirements in Part 3 which describe the requirements of the design method, programming language and other tools)
- The Design Process (Part 3, Section 7.4)
- The validation and verification and testing process (Part 3, Sections 7.3, 7.7, and 7.9)
- The maintenance process (Part 3, Sections 7.6 and 7.8)

Product Ministandards

- The Software Requirements Specification (Part 3, Section 7.2)

Risk-Based Ministandards

- Methods for Determining Safety Integrity Levels (Annexes C to E)

Developers and assessors would require a road-map and guidelines on how to select ministandards on the basis of relevance. We can think of this road-map as forming an argument for the dependability of the system and a justification of why particular ministandards were selected [15].

The formal obligations for evaluating the efficacy of process and product ministandards reduce to evaluating the following criteria in a given application of the standard:

- *Benefits*. What observable benefits are supposed to result from the application of the ministandard? Specifically, which external attributes of which products, processes or resources are to be improved? For example, is it reliability of code, maintainability of code or designs, productivity of personnel? We would also ask whether those benefits were actually achieved. Was the system safe to an acceptable degree? What was its reliability?
- *Degree of conformance*. To what extent have the requirements been conformed to (note that to measure this properly we need to be able to assess conformance objectively)?
- *Cost*. What is the extra cost of following the ministandard for the expected benefits to be achieved?

For a risk-based ministandard the evaluation will focus on the extent to which it helps risk-based prediction of some attribute based on evidence from others. The relevant criteria here would be:

- *Predictive Accuracy*. Are the event likelihoods and predictions realistic and trustworthy?
- *Informativeness*: Is the chain of argument meaningful and useful to assessor, developer and user?
- *Representativeness*. Does it employ a realistic and meaningful way of representing and manipulating uncertain evidence and the predictions made using this evidence?

Essentially, a successful product or process ministandard is one where, for a given environment, it has been demonstrated that the greater the degree of conformance to the standard, the greater are the benefits, providing that such improvements merit the costs of applying the standard. The costs and benefits of a risk-based standard on the other hand can only be evaluated by its second-order effects on other standards.

11 Improving Standards in the Long-Term

The recommendations we have made in this paper have covered the need to evaluate standards using a formal strategy. In particular we have focused on:

- making requirements more objective for conformance assessment
- classifying requirements according to the entities they are addressing (process, product, resource)
- balancing the requirements in standards according to their intended purpose
- recognizing the differing roles played by measurement and prediction in conformance and risk-based assessment
- reorganizing standards into more meaningful ministandards with clearer goals

Authorized licensed use limited to: Vrije Universiteit Amsterdam. Downloaded on May 25,2025 at 10:40:28 UTC from IEEE Xplore. Restrictions apply.

All of the above goals are achievable in the short-term given appropriate resources and commitment from either the standards making bodies or organizations actually using standards. However, improvement in the long-term will require a greater understanding of software systems and the potential risks they present.

The recommendations we have made here have concentrated largely on tightening up and clarifying definitions of systems entities and their attributes. For example, by following our recommendations standards would have to list objective definitions of reliability, testability, structure, quality etc. In traditional engineering such abstract attributes are typically only standardized for products where concrete descriptions of purpose, functionality, and structure already exist. Inclusion of measures such as reliability and safety in a software standard will only make sense when purpose, functionality and structure have been clearly defined for a specific software product. Database systems, expert systems and real-time control systems are all solving different problems yet are covered by the same standards. A user would not choose a database system over an expert system simply because the expert system has a higher reliability. The users first wants functionality and *then* reliability.

If we were to take the model of current software engineering and systems safety standards and apply it to traditional engineering products we would end up with bizarre results. For example, we would not want to develop a single *product* standard for "transportation systems" in the same way we do for software systems. To do so would mean effectively amalgamating standards for trains, cars, aeroplanes, bicycles, boats and ships and generalizing reliability and safety targets to apply to abstract "generic mode of transport." It would not make sense to describe known designs and safety features of ships alongside those of cars. The reliability targets for cars would not be the same as it is for aeroplanes. A generic "transportation system" standard is clearly absurd because we lose the whole notion of a standard as a benchmark to compare similar artifacts. Yet software standards treat *all* software as belonging to the same generic domain regardless of the purpose behind its use.

The community needs standards that mandate minimum desirable functionality for specific types of products. Only in novel or complex situations should the user have to rely on developers applying "best practice" and expensive risk assessments. Where we do have such product standards they are largely informal, as in the case of Commercial Off-The-Shelf (COTS) products like word-processors, databases and spreadsheets. The functionality of each of these is fairly well understood and accepted, but they often fall down on their unknown or low reliability. We should be identifying such product types and standardizing them to ensure minimum levels of functionality and reliability. Of course standardizing the wide variety of different software systems will not be easy but should be more relevant to the majority of users' needs.

The appeal of software is often its capacity for novelty; that ability to do something new. Indeed it could be claimed that it is questionable to apply product standards to systems that are hybrids of various product types. For instance, a pharmaceutical system may contain databases, neural nets, and real-time control modules all of which work together to form the system. It is for situations such as these that process oriented and risk-based standards would still be needed to help predict the consequences of novelty and unknown complexity. However, where knowledge about particular product types has been accumulated it should be encoded in product ministandards which would in turn act as valuable inputs to a risk-based assessment exercise for any hybrid system formed from their union.

12 Summary and Conclusions

For software standards to be usable we expect the individual requirements to be clear to:

- *Users* so that they know what benefits the standards deliver;
- *Developers* so that they know what they are required to do and
- *Assessors* so that they know how to determine conformance and predict risk.

Unfortunately, many requirements in the relevant standards are not clear in any of these three respects. This was partly explained by the different philosophies acting upon standards development.

We have shown how to interpret unclear requirements in both respects, but with special emphasis on the assessors' needs. There is a significant difference between:

1) making a requirement objective,
2) being able to assess conformance to a requirement objectively, and
3) whether conformance implies that the product will be fit for purpose.

While 1) is generally very difficult, we have shown how to achieve 2) in a rigorous manner. We have also stated how conformance assessment is no substitute for risk assessment when addressing 3). Evaluating fitness for purpose of novel and complex software systems makes extensive use of human expertise. However, existing standards cannot easily accommodate human judgment and have ignored the dominant role it plays.

The vast majority of all requirements in existing systems standards are unnecessarily unclear. Our approach to interpreting such requirements begins by teasing out the relevant process, product or resource that is the primary focus. In many cases this means breaking down the requirement into a number of parts. This technique alone can often achieve greatly improved clarity. This strategy also helps isolate the differing roles played by risk prediction and conformance assessment. We provided some examples drawn from IEC 1508 on how to do this.

When the requirements in standards are classified according to products, processes and resources, we found a dearth of external product requirements (in stark comparison with safety-related standards in traditional engineering disciplines). The emphasis was on process and resource requirements with a smaller number of internal product requirements. This balance seems inappropriate for standards whose primary objectives are to deliver products

Authorized licensed use limited to: Vrije Universiteit Amsterdam. Downloaded on May 25,2025 at 10:40:28 UTC from IEEE Xplore. Restrictions apply.

with specific external attributes, namely safety and reliability. Moreover we suggest that the long-term development of standards should attempt to overcome this process bias by constructing domain specific standards for products.

Finally, we discussed the need for assessing the effectiveness of standards. The sheer size of existing standards makes them too large to assess as coherent objects. Thus, we used the notion of ministandards, whereby coherent subsets of requirements all relating to the same specific process, product, or resource. The identification of ministandards helps us not only in assessment but also in rationalising and interpreting standards. We proposed a decomposition of IEC 1508 into ministandards.

We have presented some simple practical advice on how to improve software standards. Unfortunately, the standards-making process is long and tortuous. In many cases this process itself contributes to some of the problems highlighted earlier. Perhaps, in addition to the consensus approach, it is time that the software industry validated its own standards against a universal set of objective criteria before releasing them for general use.

ACKNOWLEDGMENTS

The contents of this report have been influenced by material from the SMARTIE project (funded by EPSRC and DTI) in which the authors was involved, and also by an earlier assessment of IEC SC65A that the authors performed as part of the ESPRIT project CASCADE project (funded by the CEC). The new work carried out here was partly funded by the ESPRIT projects SERENE, DeVa, and the EPSRC project IMPRESS. The authors are indebted to Colum Devine, Miloudi El Koursi, Simon Hughes, Heinrich Krebs, Bev Littlewood, Stella Page, Armstrong Takang, Shari Lawrence Pfleeger, Linda Shackleton, Roger Shaw, Jenny Thornton and the *IEEE Transactions on Software Engineering* reviewers and editor for comments that have influenced this work.

REFERENCES

[1] A.J. Albrecht, "Measuring Application Development," *Proc. IBM Applications Development Joint SHARE/GUIDE Symp.*, Monterey Calif., pp. 83–92, 1979.

[2] British Standards Institute, *Specification for Safety Requirements for Pushchairs*, BS 4792, British Standards Inst. 1984.

[3] ESPRIT Project 22187 SafEty and Risk Evaluation Using Bayesian Nets, 1996. http://www.hugin.dk/serene/

[4] N.E. Fenton and S. Lawrence Pfleeger, "Software Metrics: A Rigorous and Practical Approach, second edition. Int'l Thomson Computer Press, 1996.

[5] N.E. Fenton, B. Littlewood, and S. Page, "Evaluating Software Engineering Standards and Methods, in Software Engineering: A European Perspective, T.R. McGettrick AD, ed., IEEE CS Press, pp. 463–470, 1993.

[6] N.E. Fenton, S. Lawrence Pfleeger, and R. Glass, "Science and Substance: A Challenge to Software Engineers," *IEEE Software*, vol. 11, 4, pp. 86–95, July, 1994.

[7] Health and Safety Executive, UK, "The Tolerability of Risk from Nuclear Power Stations," HMSO, 1992.

[8] International Electrotechnical Commission (IEC), "Functional Safety of Electrical/Electronic/Programmable Systems: Generic Aspects, IEC 1508, 1996.

[9] International Electrotechnical Commission (IEC), "Software for Computers in the Application of Industrial Safety Related Systems," IEC 65A, 1992.

[10] IEEE ed., "Software Engineering Standards (third edition)," New York: Institute of Electrical and Electronics Engineers, 1991.

[11] IEEE, "Standard 1003.1: Portable Operating System Interface (POSIX)-Part 1: System Application: Program Interface (API) [C Language]," (1-55937-061-0), IEEE Computer Society, 1990.

[12] B. Littlewood, M. Neil, and G. Ostrolenk, "Uncertainty in Software-Intensive Systems," *High Integrity Systems J.*, vol. 1 no. 5, pp. 407–413, 1996.

[13] L. Cullen, "The Public Inquiry into the Piper Alpha Disaster, HMSO, 1990.

[14] Ministry of Defence Directorate of Standardisation, "Interim Defence Standard 00-55: The Procurement of Safety Critical Software in Defence Equipment; Parts 1-2," Kentigern House, Glasgow, 1991.

[15] M. Neil and N.E. Fenton, "Predicting Software Quality Using Bayesian Belief Networks," *Proc 21st Ann. Software Eng. Workshop*, NASA Goddard Space Flight Centre, pp. 217–230, Dec 1996.

[16] M. Neil, B. Littlewood, and N.E. Fenton, "Applying Bayesian Belief Networks to Systems Dependability Assessment," *Proc. Safety Critical Systems Club Symp.*, Leeds, pp. 71–93, Springer-Verlag, Feb. 1996.

[17] S. Lawrence Pfleeger, N.E. Fenton, and P. Page, "Evaluating Software Engineering Standards," *Computer*, vol. 27, no. 9, pp. 71–79, Sept. 1994.

[18] Requirements and Technical Concepts for Aviation, Inc. (RTCA), "Software Considerations in Airborne Systems and Equipment Certification," DO-178B, Washington DC, 1992.

[19] R. Shaw, "Safety Cases: How Did We Get Here?" *Safety and Reliability of Software Based Systems, 12th Ann. CSR Workshop*, R. Shaw, ed., London: Springer-Verlag, pp. 43–95, 1995.

Norman E. Fenton has worked for 16 years in the software engineering profession. He has been involved in research, development, and teaching. His text books and many research papers on software metrics and formal methods are well known internationally. He has held academic posts at the University College Dublin, Oxford University and South Bank University where he was director of the Centre for Systems and Software Engineering. He is currently professor of computing science at the Centre for Software Reliability, City University. He is a chartered engineer (member of the IEE) and an associate fellow of the IMA. He is one of the founder members of the EPSRC Computing College. He has managed many major collaborative projects. His current research projects are concerned with applications of Bayesian nets, software measurement, and safety critical systems. Professor Fenton is secretary to the (national) Centre for Software Reliability and managing director of Agena Ltd. a consulting company specializing in decision support and risk assessment of safety and business critical systems. URL: http://www.agena.co.uk

Martin Neil holds a first degree in 'mathematics for business analysis' from Glasgow Caledonian University and has achieved a PhD degree in 'Statistical Analysis of Software Metrics' jointly from South Bank University and Strathclyde University. Dr. Neil is a lecturer in computing at the Centre for Software Reliability (CSR) City University, London. Before joining CSR, Neil spent three years with Lloyd's Register as a consultant and researcher and a year at South Bank University. He has also worked with J.P. Morgan as a software quality consultant. His research interests cover software metrics, Bayesian probability and the software process. Dr. Neil is a member of the CSR Council, the IEEE Computer Society, and the ACM. Dr. Neil is also a director of Agena, a consulting company specializing in decision support and risk assessment of safety and business critical systems. http:/www.agena.co.uk

Authorized licensed use limited to: Vrije Universiteit Amsterdam. Downloaded on May 25,2025 at 10:40:28 UTC from IEEE Xplore. Restrictions apply.