Safety-Critical Systems, Formal Methods and Standards

Jonathan Bowen
Oxford University Computing Laboratory
Programming Research Group
11 Keble Road, Oxford OX1 3QD, UK
Tel: +44-865-273838 (272574 direct)
Fax: +44-865-273839
Email: <Jonathan.Bowen@comlab.ox.ac.uk>

&

Victoria Stavridou
Department of Computer Science
Royal Holloway and Bedford New College
University of London
Egham Hill, Egham, Surrey TW20 0EX, UK
Tel: +44-784-434455 (443429 direct)
Fax: +44-784-443420
Email: <victoria@cs.rhbnc.ac.uk>

Revised December 1992.

To appear in the *Software Engineering Journal*.

Safety-Critical Systems, Formal Methods and Standards

Jonathan Bowen   Victoria Stavridou

Abstract

Standards concerned with the development of safety-critical systems, and the software in such systems in particular, abound today as the software crisis increasingly affects the world of embedded computer-based systems. The use of formal methods is often advocated as a way of increasing confidence in such systems. This paper examines the industrial use of these techniques, the recommendations concerning formal methods in a number of current and draft standards, and comments on the applicability and problems of using formal methods for the development of safety-critical systems of an industrial scale. Some possible future directions are suggested.

1 A Brief Historical Perspective

Lives have depended on mathematical calculations for centuries. In the 19th century, the scientific community was facing the 'tables crisis' [144] due to the problem of errors in numerical tables such as logarithms and navigation tables, calculated by human 'computers'. It was rumoured that ships had been wrecked as a result of such errors. Charles Babbage was so concerned that he decided to try to alleviate the situation by attempting to mechanize the process of generating such tables using 'difference engines' and later more versatile and programmable 'analytical engines', the forerunners of modern computers.

The first true 'real-time' computer to be developed was on the Whirlwind project at MIT [5]. Started in 1944, the project produced an embryonic (military) air traffic control system in 1951. The short lifetime of the large number of vacuum tubes used in the computer was a considerable problem. Initially, the mean time between failures was about 20 minutes. Fault-tolerance was achieved by detecting weak tubes before they failed and redirecting signals to other components, thus enabling continued operation even in the event of partial hardware failure [149]. At this time, such failures were a dominant feature of the system.

Computer-based industrial process control followed by the late 1950s. The problems of software development and revision became recognized, but the solutions remained ad hoc and unreliable [134]. Even in the mid 1950s, the cost of producing software had already outstripped that of the computers themselves.

The physical hardware became increasingly reliable. The problem of frequent breakdowns, bulkiness and high power consumption of vacuum tubes was alleviated by the invention of the transistor. Despite considerable improvement, the connections between components (e.g., 'dry joints' between soldered wires) remained a serious source of failure. The advent of integrated circuits in 1959, while helping with this problem, was initially not cost-effective. However the US space programme demanded low-weight and high-reliability components – almost irrespective of cost – for the (safety-critical) computer required on board the space craft. This enabled the US to gain the lead in the microelectronics world at the time; subsequently the price of integrated circuits dropped and the number of transistors per chip increased dramatically year by year.

Similar advances were not forthcoming for software that also became more complex but less reliable.

As computers became physically smaller, it was more and more feasible to embed them within the systems that they controlled. In 1971, Intel announced the first complete microprocessor on a single chip, little realising what an enormous impact such an idea would have on the world of computing. At the beginning of the 1980s, embedded software had still not really been considered seriously by theoretical computer science researchers (but see [156]), despite the fact that it is capable of inflicting physical damage [14]. However in the last decade, such systems have come under more and more scrutiny as computers pervade all areas of our lives, especially in embedded applications.

The software currently used in computers has itself become so complex that it is not trustworthy and has caused human injury and death as a result. [108] provides a list of incidents that is updated annually. Up until relatively recently it has not been considered feasible to use formal techniques to verify such software in an industrial setting [61]. Now that some case studies and examples of real use are available, formal methods are becoming more acceptable in industrial circles. Even populist accounts of the computing industry are mentioning the problems of software errors in relation to critical systems and the possibility of applying mathematical techniques to reduce such errors in a wide variety of forums [90, 106, 112, 141].

This paper briefly discusses safety-critical systems, examines the use of formal methods as a possible technique for increasing safety and reliability, and surveys some standards in this area. The objective of the paper is to provide information on the current safety issues, particularly with regard to software, as reflected by a number of current and emerging standards and to examine ways in which formal methods technology can and has been used to improve system safety. It should be noted that formal methods are only one of a gamut of important techniques, including many classical safety analysis methods, that can be applied in safety-related software development; the scope of this paper excludes many of these other techniques.

This is a fast moving area in which rapid change is the norm; therefore, this paper should be seen as representative snapshot rather than as a comprehensive and definitive guide.[1] It is also our contention that the subject of software safety and the contribution of formal methods is in its infancy; we will, therefore, conclude this paper with a summary of open research questions. A substantial bibliography is included at the end of the paper, with a list of relevant standards, to enable interested readers to investigate the issues further.

2 Safety-Critical Computer Systems

Safety is closely coupled to the notion of risk. Charette [28] defines risk as an event or action:

- *Having a loss associated with it.*
- *Where uncertainty or chance is involved.*
- *Some choice is also involved.*

Safety can then be defined as the freedom from exposure to danger, or the exemption from hurt, injury or loss. But in most situations, one is concerned with the *degree* of safety and therefore safety is a subjective measure which makes safety provision and measurement extremely difficult and contentious tasks.

[1] Probably the best survey paper of the 1980s in the area of software safety is provided by [84]. However this is now somewhat out of date because of recent developments. For an update, [85] and [113] are recommended.

Safety concerns in computer systems are even more confusing. Such systems consist of many subcomponents which are tightly coupled and have highly complex interactions. The binding of application to operating system to architecture is a prime example of a tightly coupled system. When such a system is further embedded within larger contexts, such as command and control systems, the probability of failure quickly approaches unity. Myers [103] estimates that there are approximately 3.3 software errors per thousand lines of code in large software systems. This figure is not surprising given that there are as many as 10^{20} unique end-to-end paths in a moderate-sized program [137]. What is worse is that not all errors in software are created equal, as small errors do not necessarily have small effects. The picture becomes much bleaker when the software/hardware interactions in computer systems are taken into account. In two studies [48, 70], nearly 10% of all software errors and 35% of all software failures identified were later found to be hardware-related, such as transient faults corrupting data. In fact, it appears that hardware can fail three times as often as software under some circumstances [6].

The most effective means to avoid accidents during a system's operation is to eliminate or reduce dangers *during* the design and development, not afterwards when the complexity becomes overwhelming. Safety cannot be considered as an add-on after the system has been developed much the same way that it does not make sense to design an aircraft and then think about its weight. We strongly believe, that *safety must be designed in a system and dangers must be designed out.* We also feel that software and hardware safety are inextricably intertwined and must be considered as a whole with special attention paid to the interfaces.

2.1 Dependable computer systems

Despite these considerable problems, the advantages of the added versatility and flexibility provided by digital systems as opposed to other means are overwhelming and, therefore, insufficiently understood software and hardware are often used. However implemented, we require that safety-critical systems are *dependable.* There are many terms associated with dependability, and considerable international effort has been expended to standardize these [83]. The accepted definition of the overall concept is [82]:

Dependability is that property of a computing system which allows reliance to be justifiably placed on the service it delivers.

The life of a system is perceived by its users as an alternation between proper and improper service. Delivery of proper service (service which adheres to specified requirements) is normally termed *correctness.* Therefore a "correct" system is not necessarily a dependable system. Dependability is an overall property which has other measures such as safety, reliability and availability. Laprie [82] defines these terms as follows:

Safety is a measure of the continuous delivery of service free from occurrences of catastrophic failures.

Reliability is a measure of the continuous delivery of proper service (where service is delivered according to specified conditions) or equivalently of the time to failure.

Availability is a measure of the delivery of proper service with respect to the alternation of proper and improper service.

Formal methods address correctness issues and these will be considered in the remainder of this paper. We do not address other dependability measures here although safety, reliability and availability should be modelled and measured independently of verification, using probabilistic techniques and testing. However, it is worth pointing out that confusion often arises between the above concepts which are, in fact, distinct. For instance, a safe system is not necessarily reliable (an airplane in flight may be safe for only some of the time), a reliable system is not necessarily correct (the autopilot may reliably compute an incorrect course) and a safe system is not necessarily available (the safest airplane is one that never leaves the ground).

The key notion in dependability is that reliance must be justifiable [82]. This means that we need explicit and testable requirements and specifications which are refined to a system using rigorous or formal development techniques, as well as credible analytical and experimental evidence demonstrating the satisfaction of the requirements by the system. We believe that if no such evidence can be obtained, the wise developer will use human or purely hardware resources instead of software or software/hardware systems.

There are four approaches to achieving system dependability [82]:

Fault avoidance: How to prevent, by construction, fault occurrence or introduction.

Fault tolerance: How to provide, by redundancy, a service complying with the specification in spite of faults.

Fault removal: How to minimize, by verification, the presence of faults.

Fault forecasting: How to estimate, by evaluation, the presence, the creation and the consequences of faults.

It is commonly agreed that a combination of these approaches must be used in order to achieve ultra-high dependability. Software testing (fault removal) alone may be able to reduce the failure rate of a program to about 10^{-4} per hour (approximately 1 failure per year) and faster more complex computers can only make matters worse. It has been suggested that only an improvement factor of about 10 may be achievable using fault tolerance approaches such as N-version programming [101]. In fact, the benefits of these techniques are still a matter of some contention [78]. Combining these gives a figure of around 10^{-5} but most safety-critical situations demand a figure of nearer 10^{-9} or even 10^{-10} (e.g., see [FAA82]). This leaves us with an enormous gap between what is desired and what is attainable with current practice. A viable means of narrowing this gap in the future is the use of fault avoidance in the form of formal methods in conjunction with the other techniques, although the quantification of dependability improvements obtained by such combinations is an open research issue. It is particularly fortunate that existing formal methods technology is sufficiently developed and well suited for use during the crucial requirements and specification stages of development when the system is still relatively abstract and therefore less complex than the final implementation. Later in this paper, a number of examples of realistically sized uses of formal methods for the verification of (parts of) safety-critical systems are briefly outlined and discussed.

2.2 Formal methods

Formal methods have been a topic of research for many years; however they are rarely used in commercial contexts [34]. Whilst industrial research laboratories are investigating formal methods, there are relatively few examples of real use in the computing industry. Even in companies where formal methods are used, it is normally only to a limited extent and is often resisted (at least initially) by engineers, programmers and managers. This situation is hardly surprising since formal methods technology is largely perceived to consist of a collection of prototype notations and tools which are difficult to use and do not scale up easily; there are many widely held misconceptions about the use of formal techniques [56]. It may be fair to say that formal methods research has to some extent been dominated by the fundamental scientific aspects rather than by problems in application.

Even if we knew how to fit formal methods with current development techniques there is still the problem that a lot of software is currently produced by "chaotic" processes (as defined by the Software Engineering Institute's process maturity metrics [68]). The use of formal methods is no substitute for good software production management. Furthermore, the adoption of these techniques requires up-front committal of resources to projects which runs contrary to industrial practice where most projects consume the bulk of their resources toward the end of system development (testing and debugging). Additionally, present management practice may be entirely inadequate for management of the early stages of full formal development. For example, the simplification and reduction in size of a formal specification is certainly progress, but it is measured by a reduction in the amount of paper produced.

Finally, the use of formal methods requires mathematical expertise which is simply not currently available in industry where most practitioners have little or no formal computer science training. In this paper, we will argue that it is possible, and in the case of safety-critical systems highly desirable, to obtain benefits from formal methods even in constrained contexts.

It is often said that the use of formal techniques in the production of systems should be viewed as a means of delivering enhanced quality rather than establishing correctness. This difference of perception is crucial and is particularly highlighted in the case of safety-critical systems. Formal methods can deliver correctness – that is adherence to some requirements – and therefore enhanced quality; but correctness is not the end of the story.

As pointed out by Cohn [32], correctness involves two or more models of a system (designer intentions and software/hardware system), where the models bear a tentative but uncheckable and possibly imperfect relation to both the user requirements and the final implementation. Even under the best possible circumstances, when we have an accurate interpretation of the requirements, at best we can assert that the model of the implementation satisfies these requirements. Whether the system will work satisfactorily *in situ* also depends on factors ranging from communication, training, and behaviour to the performance of mechanical, electrical and chemical components both within the system and its operational environment.

Formal methods may be characterized at a number of levels of use and these provide different levels of assurance for the software developed by such methods [151]. At a basic level, they may simply be used for specification of the system to be designed. The development process itself may be informal, but benefits are still gained since many bugs can be removed by formalizing and discussing the system at an early stage. Proofs may be undertaken to confirm properties of the system that are required or assumed to be true. Such proofs help to increase the design team's understanding of the system and this is an important component of the increased confidence that such validation provides. The Z notation [139] is often used in this manner, and has proved to be beneficial.

The next level of use is to apply formal methods to the development process, using a set of rules or a design calculus that allows stepwise refinement of the specification to an executable program. For example, VDM [76] was specifically designed for the development of programs, as opposed to just their specification or proof in general.

At the most rigorous level, the whole process of proof may be mechanized. Hand proofs or design inevitably lead to human errors occurring for all but the simplest systems. Checking the process by computer reduces the possibility of error, although it never eliminates it since the program that does the checking may itself be incorrect. In addition, it is always possible that the basic underlying axioms may themselves be inconsistent.

Mechanical theorem provers such as HOL [52] and the Boyer-Moore theorem prover [19] have been used to verify significant implementations [32, 100], but need to be operated by people with skills that very few engineers possess today. Such tools are difficult to use, even for experts, and significant improvements will need to be made in their usability before they can be widely accepted in the computing industry. However, proof tools are now becoming commercially available (e.g., the B tool [1, 2] and Lambda [98]). Thus commercial pressures will hopefully improve such tools which up until now have mainly been used in research environments. In particular, the user interface and the control of proofs using strategies or 'tactics' are areas that require considerable further research and development effort. Despite the present inadequacies, safety-critical software is the one application domain where the added confidence of mechanical proofs may be justifiable if feasible, even though the development cost of such an approach (of which more later) is high. [87] provides a good snapshot of what is currently available.

Perhaps an indication of the seriousness with which the UK formal methods community now takes safety-critical computer systems is that the first article of the first volume of the relatively recently introduced journal *Formal Aspects of Computing* is devoted to this subject [146]. This gives an authoritative, reasoned and balanced account of the state of the safety-critical industry. Many workshops specifically concerned with safety-critical systems and formal methods are now being held (e.g., [36, 38]). Safety concerns are considered an important part of state-of-the-art software engineering (e.g., [10, 142]) and formal methods promise to help with some aspects of the problems encountered. Safety-related journals also consider how formal methods can help (e.g., [12]). There are now entire books on the use of formal methods in safety-critical systems (e.g., [132]), and most more general books at least address the subject (e.g., [89, 123]). [99] is especially recommended for further reading; it includes several case studies using different formal notations. Further recent and relevant papers are [7, 18, 79, 110].

2.3 The cost of software safety

Thoreau, in *Walden* [28], wrote that:

> *"the cost of a thing is the amount of what I will call life which is required to be exchanged for it, immediately or in the long run"*

In other words, the cost of safety is determined, ultimately, by what people are willing to pay. Table 1 [77] illustrates the wide range of amounts of money spent on trying to save a human life in various situations. Although various methods for calculating the objective cost of safety have been proposed [15, 58], the problem is largely unresolved. One cannot ever be certain, when an accident *does not* occur, whether it is because of the safety devices or because the system was designed "well enough" so that the safety devices are superfluous. On the other hand, when accidents *do* occur, the penalties for ignoring software safety can be very severe.

A recent report [125] values "a statistical life" at £2–3M (around $4M) in that this is the sort of amount of money that should be spent on saving one life. However the value placed on a human life varies widely from country to country, and even within a single country depending on the situation and the people involved.

The overall system safety cost includes a multitude of factors and components. From this point on, we will concentrate on the cost of software defects for the *producer* of the software; some of these will affect safety and others will not. We will attempt to estimate how much it costs to eliminate software defects at large since the proportion of safety-critical defects to benign ones cannot be quantified for all systems. Note that we do not take into account the liability costs incurred by producers of safety-critical software that has failed causing accidents.

Software defect costs can be investigated using a variety of approaches. Ward [153] uses data focused on the cost per prerelease software defect that is found and fixed during the integration through to the release phases of project development. The calculation of the cost is based on the formula

$$\text{Software defect cost} = \text{Software defect rework cost} + \text{Profit loss}$$

Based on data from an extensive software project database maintained at the Hewlett-Packard Waltham Division for product releases over the past five years, Ward has produced software defect costs for typical software projects which give an average cost of around $10,000 per *corrected* defect.

The figures supplied by Ward give us an approximation of the cost of software defects in projects where current practice is used. We are interested in a similar cost approximation when fault avoidance techniques, and in particular formal methods, are used. Calculations based on a substantial railway system of around 70 man years effort [54] which will be deliberated upon further in the next section have been made, using similar assumptions to Ward [17]. These result in an estimated order of magnitude greater cost per *avoided* defect. On the other hand, only about half the total effort of the development was devoted to proofs.

One reason for high costs may be the necessity of training, which could be amortized over several projects if formal methods are adopted on a permanent basis. Other substantial industrial examples [63] have demonstrated that formal methods *can* be the most cost-effective technology if used in an appropriate manner. Some of these are mentioned later in the paper. In particular, full verification may not be worthwhile in many cases, but may be helpful for the most critical parts of an application; formal *specification* alone may be more appropriate for the rest of the system, with *rigorous* rather than formal development of implementation.

Such conflicting evidence on the cost-effectiveness of formal methods makes the need for proper deployment and quantification of the technology even more pressing. It is hoped that an international ongoing study of the industrial applications of formal methods [120] will help shed light on this issue.

3 Industrial-scale Examples of Use

Whilst safety-critical systems may have the most to gain from the use of formal methods, such techniques are in fact being used in a wide variety of application areas in industry, although still in very few projects overall [8]. Formal methods have been used to improve the quality of software in a number of non safety-critical areas. They have been shown to have the potential to both reduce the development cost (e.g., the IBM CICS project [67] where a saving of 9% in costs – which runs into millions – has been claimed) and to reduce the time to market (e.g., the inmos T800 transputer floating-point unit [97], where a saving of 12 months development time is claimed).

A notable example of the application of formal methods to a number of related levels of abstraction is provided by the work at CLInc. in Austin, Texas. The Boyer-Moore theorem prover [19] has been used to verify a compiler, assembler, kernel and microprocessor (that has since been fabricated) [100]. Most of these components form a 'stack' of verified parts that link together to form several related levels in a verified system [51]. This is the first such effort in the world and although the work is very much based around the Boyer-Moore theorem prover, and thus perhaps difficult to generalize directly, further work in Europe is now building on these foundations along similar lines [11].

It is worth noting that Europe (and in particular the UK) has a leading position in formal methods research and use [145]. These techniques have not gained an equal foothold in North America, even in the safety-critical sector. The notable exception is the security field where most US formal methods work has concentrated. This work is primarily championed by CLInc, ORA Corporation and SRI International.

Safety-critical systems make up a minority of industrial applications using formal methods [8]. Despite the fact that such systems have the most to gain potentially, industry is wisely cautious in adopting new untried techniques in this area [106]. The following sections give a brief overview of some companies and significant projects involved in the development of safety-critical systems that have used formal methods over the past few years. In general the results have been successful, but comments concerning individual cases are included below. Table 2 summarizes these experiments.[2]

For a recent comprehensive market study of safety-related computer controlled systems in general, resulting from a survey of many businesses and other relevant organizations concerned with safety-critical systems in the UK, see [44]. The formal specification technique most often quoted as being used was Z [139], but HOL [52] and OBJ [49] were also mentioned. Software houses often referred to the potential of formal methods. In many cases the main pressure to use formal techniques is coming from the defence sector, following the publication of 00-55 [MoD91a]. They are seen as one way of providing greater assurance that errors have been minimized in both software and hardware design. In practice few suppliers, especially outside the software houses and consultancies, make use of formal methods. This is mainly because formal methods are relatively new to industry and there is little experience in their use. The cost of training is seen as prohibitive.

3.1 Aviation

An early example of the application of formal methods to real life systems, was the SIFT project, which probably represents the most substantial US experiment in the safety-critical sector. SIFT [155] is an aircraft control computer which was commissioned by NASA in the mid-seventies. The safety requirements proposed by NASA and the FAA (Federal Aviation Administration) are extremely stringent; they permit a probability of life-threatening failure of less than 10^{-10} per hour during a ten hour flight [FAA82]. Formal methods were used in the SIFT project in order to try and bridge the gap between this failure rate and the 10^{-5} which can be achieved with other techniques such as testing and fault tolerance [101].

SIFT is designed to operate safely in the presence of hardware faults by replication of processors and adaptive majority voting. In contrast to other majority voted systems, the voting mechanism that detects and masks hardware faults is implemented entirely in software. It is this software, or rather its design, that was subjected to formal verification. The verification was conducted in two stages. The first involved verifying the I/O specification against the pre/post specification and was done initially using the Boyer-Moore theorem prover [19] and subsequently the STP (Shostack Theorem Prover) system [133]. The second, more substantial proof additionally dealt with the transition specification as well as the fault model specification. This proof

[2] A • under a column heading indicates whether a particular activity was undertaken as part of the project. The machine support heading indicates whether machine support was used in this particular case for either specification or verification or both, *not* whether the whole method is supported by tools.

was done using the specially built EHDM system [128] and involved approximately one man year of effort.

The SIFT system was delivered to the NASA Langley Research Center AIRLAB facility where it has formed the basis for other evaluation projects. It has been found to be a very reliable system but as [101] points out this was the result of the simplification of the design rather than the design verification exercise (which was done post code development). This same simplification of the system has led to a number of criticisms of the SIFT project; it was widely felt that the verification exercise involved oversimplification which eventually rendered the system "unfit for purpose" [104]. So, although the SIFT episode was a successful research exercise it was a failure as far as subsequent actual deployment of the processor was concerned.

More recently, NASA commissioned work involving the application of formal methods to support fault tolerance in digital flight control systems (DFCS). [127] contains the formal specification and verification of a model for fault masking and transient recovery in DFCS applications. [129] describes the formal verification of the interactive convergence algorithm for Byzantine fault tolerant clock synchronization. [140] discusses the formal verification of the FtCayuga fault tolerant microprocessor system. It appears that NASA has found this line of investigation fruitful and preferable to experimental quantification of software reliability [23]. Another recent project for the FAA, undertaken by Nancy Leveson *et al.* has been working to produce a formal specification of the TCAS collision avoidance system [RTCA90].

3.2 Railway systems

In 1988 GEC Alsthom, MATRA Transport and RATP started working on a computerized signaling system for controlling RER commuter trains in Paris. Their objective was to increase traffic movement by 25% while maintaining the safety levels of the conventional system. The resulting SACEM system (partly embedded hardware and software) was delivered in 1989 and has since been controlling the speed of all trains on the RER Line A in Paris. The dependability of SACEM has obvious safety implications for 800,000 passengers per day.

The SACEM software consists of 21,000 lines of Modula-2 code. 63% of the code is deemed as safety-critical and has been subjected to formal specification and verification [54]. The specification was done using Abrial's B language [2] and the proofs were done manually using automatically generated verification conditions for the code. The validation effort for the entire system (including non safety-critical procedures) was of the order of 100 man years and therefore, this experiment represents a substantial investment in formal methods technology.

The authors of [54] believe that the system is safer as a result of the formal specification and verification exercise. It is certainly instructive to observe that even within the current constraints, mature formal methods technology can contribute to system safety. The SACEM work has primarily benefited from formal specification which enabled precise and unambiguous statements about the system to be made. It is interesting to note that a difficult problem which the project team had to resolve was communication between the verification personnel and the signaling experts who were not familiar with the formal notations employed. They overcame this problem by providing the engineers with a natural language description derived manually from the formal specification. Similar techniques are now being applied to other train control systems [27] using the B-tool [1].

3.3 Nuclear power plants

Rolls-Royce and Associates have been applying formal methods (mainly VDM) to the development of software for safety-critical systems, and nuclear power plants in particular, for a number of years [62, 63]. This approach has proved very successful and has produced the following conclusions based on practical experience [63]:

- The main problem is the system as a whole, rather than the software alone.
- A combination of modern techniques, including formal methods, can help in the development of safety-critical software, even though their scope may be limited at present.
- There are many myths and few facts concerning software engineering methods (see also [56]). [63] lists some myths and facts concerning the use of formal methods in the development of software.
- Improvements in time-scales, costs and quality are possible in practice using such techniques.

In a (subjective) table giving the contribution of 11 (not necessarily incompatible) software development methods, formal methods is considered the most helpful option. Formal methods are also considered important for the demonstration of software (using animation techniques) and for subsequent changes, although structured methods and documentation configuration control are considered more important for the latter. Some approaches, such as the use of independent project teams or the use of diverse software are considered of little or no help.

Tests are still undertaken, but it is noted that these have normally found mistakes in the test software rather than the software under test, since the former has not been developed rigorously, whereas the latter has.

A comparison of cost-effectiveness of different methods has been made (see Table 3, reproduced from [63]). Using formal methods has doubled the specification and analysis stage, but eliminated the redevelopment stage. Since the latter can be of the order of half the costs whereas the former is a much smaller percentage, the overall saving is up to half the cost. One problem which project managers face is that a project using formal methods may be two thirds complete without any sign of the final software in sight. This can be unnerving the first time round, and may be one reason for the lack of uptake of formal methods in general; unfortunately people have a proclivity towards wanting to see some actual running results, even if they are the wrong results! An answer to this problem is to use rapid prototyping tools (perhaps through the use of executable specifications) for parts of the system that must be demonstrated to the customer before development of the actual software begins.

[114] provides an approach to the design, documentation and evaluation of computer-based safety-critical systems that have been used by the Atomic Energy Control Board of Canada (AECB) to assess the safety of software in a nuclear power station. Ontario Hydro and AECB are advocating the use of systematic mathematical verification techniques and rigorous arguments of correctness [75]. Indeed, they have already applied formal methods in the analysis of the shutdown system for the Darlington Nuclear Reactor plant [4].

3.4 Medical systems

A number of medical instruments have life critical functionality and require a high degree of dependability. Two Hewlett-Packard divisions have used formal specification in order to enhance the quality of a range of cardiac care products. In both cases the formal notation used was HP-SL [9].

The first instance [81] involves a bedside instrument which is used to monitor vital signs of patients in intensive care units and operating rooms. Formal specification was used in a product enhancement involving monitoring a segment of the patient's electrocardiogram (ECG) which can be clinically significant since a change in that segment can indicate reduced blood flow which can be asymptomatic and hence difficult to detect. The team found that using formal specification resulted in higher precision in the development process which helped uncover a number of problems and ambiguities in the informal product specification. They believe that the use of formal notation played a significant role in realising a high quality product. Although they found it difficult to quantify the benefits of formal specification, they do report improved defect densities as shown in Table 4 (reproduced from [81]).

The second example [41] relates to an instrument involving approximately 16 Kbytes of safety-critical code in ROM. In this case, the formal specification provided a process model of the system, specifying the relationship of inputs and outputs over time. The specification was subsequently used to produce SA/SD diagrams (structured analysis/structured design) from which the code was derived. The team again found that using formal specification helped them to uncover and resolve a number of ambiguities in the informal product definition. They also found that although they had an increased design time, the effort paid back in terms of shorter coding times.

It is interesting to note that both teams were particularly concerned with the impact of formal specification on their project schedules. Notwithstanding the significant benefits that were realised during development, they felt that adoption of formal techniques would have been very difficult if they had not introduced the approach very early on in the project and had they not had managerial commitment and support from the expert HP-SL team in Hewlett-Packard Research Laboratories, Bristol, UK. They found that retrospective specification of existing products was problematic and not very successful.

Tektronix in Oregon, USA, are also producing software for safety-critical applications involving medical equipment (as well as developing oscilloscopes) and have started applying formal methods in both areas. Real-time kernels can often be particularly tricky and error prone. Attempts have been made to formalize existing kernels and this has drawn attention to possible problem areas [138]. In particular, it is possible to calculate the preconditions (prior assumptions) of operations and if these are not vacuously true (i.e., the operation can be called at any time) it is possible that an error may be caused by activating that part of the software at an inappropriate time. These potential error conditions may not be a problem in the initial release of the software since they never occur in practice; however, if they have been inadequately documented (which is often the case), then subsequent maintenance of the software (quite often by a different team) could result in catastrophic failure.

Deaths due to software in medical equipment have been documented elsewhere (e.g., the Therac 25 radiotherapy machine [72, 86, 112, 142], where the dosage editor was poorly designed) and as a result others are also resorting to formal techniques in this area. [71, 73] discuss formal specification and verification issues regarding a clinical cyclotron control system which is currently under development at the University of Washington Hospital Cancer Treatment Center.

3.5 Ammunition control

The control of operations concerning the storage and use of explosive articles is an area with obvious safety-critical implications. In fact, the Ordnance Board of the UK Ministry of Defence (MoD) contributed substantially to the impetus towards the standardization of safety-critical software by articulating the problems inherent in using software in fusing applications such as torpedoes and missiles. Explosives safety concerns are not however limited to weapons delivery systems. Over the years, the variety of explosive substances and articles has increased significantly and the MoD has a continuing problem with the storage of explosives which is driven by the increasing complexity and variety of weapon systems. Although the MoD does not publish its internal directives, these are known to be consistent with publicly available regulations such as the United Nations Regulations for the Transport of Dangerous Goods ("Orange Book") [UN64]. Similar arrangements operate in other NATO countries.

The ammunition holdings of a number of MoD ranges in the UK are managed by an ammunition control software system (ACS) which has been subjected to extensive validation as well as formal specification [102]. Although ACS is not a real-time system, it is nonetheless safety-critical since incorrect storage combinations can lead to massive explosions. The ACS software became more safety critical as experienced technical staff were replaced by operators who needed to rely implicitly on the computer output. It is therefore vital that the system correctly implements the appropriate MoD regulations since human intervention is not a feasible backup option.

As an example of what can be done, Mukherjee and Stavridou [102] have produced a formal specification of the safety requirements in the Orange book and related it to the operations performed and controlled under ACS in VDM. In particular, they address additions of explosives to magazines and extensions to facilities by means of additional magazines. They have found a number of contradictions in the UN rules.[3] The formal specification is significant because the MoD can now generate a formal specification of their own regulations which when verified by the regulation authority can be incorporated into the Operational Requirement for a planned ACS replacement in accordance with Def Stan 00–55. Furthermore, the work has demonstrated that the use of formal methods can be successfully extended to areas such as planning and prediction as well as a "change facilitator".

3.6 Embedded microprocessors

The Viper (Verifiable Integrated Processor for Enhanced Reliability) is a microprocessor that has been specifically developed for safety-critical applications [39]. The HOL (Higher Order Logic) mechanized theorem prover [52] has been used to verify parts of the processor. However the methods used and the (sometimes misinterpreted) claims about the correctness of this processor have caused some controversy in industrial and even the formal methods communities [20, 33].

Viper was the first "real" microprocessor subjected to formal verification and intended for serious use. This fact, coupled with its Ministry of Defence parentage, makes Viper a high profile hardware verification experiment in the UK. In [37], Cullyer outlines the reasons behind the chip's development, namely MoD control of chip design, manufacture and marketing. The constraints on the architecture and manufacture on Viper can be summarized under the heading of "simplicity". This is understandable as simplicity is a very sensible requirement for systems used in safety-critical applications; it is also fortunate because in this instance it made formal verification experiments possible. Simplicity coupled with the need for quality are potentially fertile ground for the use of formal methods.

In [31, 32], Cohn presents proofs in HOL [52] relating the top specification with the host machine (state machine) and the block level description of Viper (register transfer level description). The block level itself, is still very abstract as one has to go through the gate and electronic levels before arriving at the manufactured chip. The proof relating the top level functional specification

[3]The UN rules are currently being redrafted, independently of the work described in [102]. It is expected however that the findings of [102] will be incorporated in the revised regulations.

with the host machine revealed a type error, a confused definition and an incomplete check [31] while the (only partially completed) block level proof did not reveal any problems. Neither proof was of direct concern to the fabricators of Viper chips and indeed, the fact that the first batch of chips from one of the manufacturers contained errors cannot be attributed to the flaws discovered by Cohn. The chips and the manufacturer literature appeared a long time before the conclusion of the formal verification work. It is clearly therefore the case that formal verification in the case of Viper was a *design quality assurance* exercise. The associated costs were prohibitive and therefore the next generation of the chip, Viper2, was not subjected to similar hardware verification (although other verification techniques have been used[4]).

The lessons from the Viper experiment can be summarized as follows:

- The dependability of the chip was enhanced at a price, although no statistics are available.
- Formal methods can certainly help deliver *slower* chips because efficient but complex structures are hard to verify.
- Although no comparative figures are available, it is difficult to imagine that the formal verification produced a cheaper chip.
- The formal verification work was not directly utilized in the development of Viper2 (which is very similar to the original with the addition of a multiply instruction) so there is no evidence of the work aiding design backtracking.
- The experiment has certainly shown that HOL is scalable even if painfully so [32].

Some of the ensuing controversy as to what exactly constitutes a proof is discussed from a sociological viewpoint in [91, 92].

4 Areas of Application of Formal Methods

As has just been illustrated, formal methods are applicable in a wide variety of contexts to both software and hardware, even within the confines of safety-critical systems. They are useful at a number of levels of abstraction in the development process ranging from requirements capture, through to specification, design, coding, compilation and the underlying digital hardware itself. Some examples of current research work in these areas are given in this section. An example of a suggested overall approach to project organization using formal methods is provided by [122].

The *Cleanroom* approach is another technique that incorporates the use of rigorous methods to produce highly reliable software by means of non execution-based program development that is establishing itself as an effective means of drastically reducing the number of errors in software [131, 45]. The rigorous development stage is clearly split from the certification stage, that replaces the normal testing phase, and is used to check for the absence of errors rather than for correcting them. The combination of this approach with the use of formal notations such as Z would be a useful area of study that IBM is starting to investigate [109].

In 1991 the UK Department of Trade and Industry (DTI) instituted the 'SafeIT' initiative in order to establish a unified approach to the assurance of software-based safety-critical systems by encouraging and financing collaborative projects in the area [SafeIT90a]. This sponsors industrial (and to a lesser extent academic) organizations to undertake collaborative projects in this area.

[4]Logica Cambridge was contracted to develop and use Meta-Morph, a tool for reasoning about functional language specifications.

A second phase of the initiative has been launched in 1992 [94]. An associated Safety Critical Systems Club has been formed and judging by the attendance of 255 delegates at the inaugural meeting, interest in this area is very strong in the UK. A club newsletter includes articles on the application of mathematical methods to safety-critical systems (e.g., see [3]) as well as issues concerning standards. The research effort in the area is gathering momentum and a number of interesting projects are currently under way. The SafeIT MORSE and SafeFM projects aim to build models for analyzing safety requirements and find practical ways of using formal methods in the development of safety-critical systems. The SafeIT initiative is particularly interested in safety standards and has produced a framework for such standards [SafeIT90b].

The European ESPRIT programme is sponsoring two research projects that have a particular interest in the safety of computer-based systems. The ProCoS (Provably Correct Systems) [11] and PDCS (Predictably Dependable Computing Systems) [80] Basic Research Actions are examining different approaches to safety, the former concentrating on qualitative and the latter on quantitative aspects.

The upsurge of interest in the area is also evident from the emphasis placed on criticality by major international conferences. The ACM SIGSOFT '91 Conference was devoted to safety-critical software and the 1992 International Conference on Software Engineering is concentrating on trusted systems.

4.1 Requirements capture

Accurate requirements capture is very important in the design of any system. A mistake at this stage will be carried through the entire development process and will be very expensive to correct later. Studies have shown that a modification in service can cost up to 1,000 times more than a modification at the requirements stage [26]. Even worse, two thirds of all errors are made at the requirements stage [26]. So it is hardly surprising that the US Government Accounting Office has calculated requirements defects cost \$6·5 million on 9 projects alone. It clearly makes sense to ensure that the requirements are correct before proceeding with development. When formalizing requirements, there is nothing to validate them against except the real world. Thus it is important that the requirements language is simple enough to be easily understandable, but expressive enough to describe the desired requirements fully. This is a difficult balance to achieve, and the language used will vary from project to project depending on which aspects of the system need to be captured.

There is now a considerable interest in this aspect of design in the formal methods community (see, for example, [50]); it forms, for example, a major goal of both the SafeFM and MORSE projects. For safety-critical systems, timing is often of great importance. This has proved to be a difficult area to formalize in a manner that is usable in practice. However research in this area is gathering momentum (e.g., using the Duration Calculus [161]) [59, 121].

4.2 Design

The design process refines a specification down to a program using (possibly) provably correct transformations or some other kind of rigorous refinement method. In general this must involve input from the engineer since there are many programs that meet a particular specification. Most formal methods until now have not considered the problem of timing issues in these transformations, partly because of its intractability. However research is active in this area (e.g., [93]). It is crucial to keep things as simple as possible while still addressing the problems that actually matter. In a *hard real-time* system (which includes most safety-critical systems), a missed response is as bad as functional failure, whereas in a *soft real-time* system the occasional delay in response is tolerable. In the former type of system, it is very important to prove that the desired response time will be met under all circumstances.

Research into real-time formalisms such as Timed CSP (Communicating Sequential Processes) [42] is currently very active and is being applied in the area of robotics, for example, to help ensure correctness of the design [118]. Student textbooks for real-time formalisms are also now appearing [96]. However, there are a number competing formalisms to reason about real-time aspects of systems [130]; there are many problems yet to be solved and it remains to be seen which of the existing formalisms will be most useful in practice. Recently interest in *hybrid systems* [95] has increased, in which continuous variables (as well as time) are considered. The best interface between the differential equations of the control engineer that define the overall system and the controlling computer program of the software engineer has yet to be determined.

4.3 Compilation

Compilers produce code that it is notoriously difficult to analyze, particularly as far as timing aspects are concerned. They themselves may be unreliable and introduce an extra unknown into the development process. The development of the compiler needs to be as strictly controlled and reliable as the development of the high-level safety-critical code itself. Thus in the past, software safety standards and directives have normally insisted that all software is written in assembler that can be transliterated almost directly into machine code. Since this is the actual code that is executed, this is the code that needs to be verified [29]. However, this simply shifts the burden of responsibility, since the programmer must ensure that the assembler program meets its specification, and this is more difficult than the equivalent process for a high-level program.

Nowadays, safety standards are recognizing that programmers can produce high-level programs much more reliably than low-level programs and thus some are even insisting that high-level languages are used, a complete reversal of the previous guidance issued to engineers. Recent research has demonstrated that it is possible to verify compiling specifications elegantly and even produce a rapid prototype compiler that is very close to the original specification in the form of a logic program [66]. Other related research is investigating methods to verify an actual compiler, including the bootstrap process, but significant barriers remain before such an approach can become viable in practice [24, 40].

Since the machine code itself is the final program that actually matters, decompilation is sometimes used to ensure that it is correct. Decompilation can be tricky, but very similar (or even identical) programs for compilation and decompilation can be used if a declarative approach is adopted [16].

4.4 Programmable hardware

Programmable Logic Controllers (PLCs) are often used in process control and work has been undertaken to formalize the design process for these devices [55]. Another relatively new digital hardware technology, which may be of interest to safety-critical engineers who currently use embedded computer systems, is the *Field Programmable Gate Array* (FPGA, e.g., [159]). This allows the possibility of directly programming hardware almost as easily as general purpose computers are programmed today. A memory within the FPGA contains a pattern of bits (similar to the object code of a program) that determines the interconnections of a number of digital components such as boolean gates and latches within the chip.

Compilers from high-level programming languages down to a '*netlist*' of components are now being produced [111], and it seems a realistic goal that such compilers could be formally proved correct. A particularly attractive feature of this direct implementation in hardware for safety-critical systems is that the timing aspects of the program can be considerably simplified if synchronous circuits are used. For example, in [111] all assignment statements take one clock cycle and (perhaps surprisingly) all control statements take no clock cycles since the control operates between the clock edges. Additionally, the natural parallelism of hardware can be used to great advantage. Parallel programs can run truly concurrently and parallel assignments of several variables (up to the entire state of the program) still only take one clock cycle.

This looks like a very promising research area for the 1990s and it is foreseen that programmable hardware will be used increasingly during the coming years. Formal verification of the overall system will be simplified since the high-level program is related directly to gate-level hardware without the complexity of an intermediate instruction set.

4.5 Documentation

An important part of a designed system is its documentation, particularly if subsequent changes are made. Formalizing the documentation leads to less ambiguity and thus less likelihood of errors. In the case of safety-critical systems, timing issues become significant and methods for documenting these are especially important [115].

Formal methods provide a precise and unambiguous way of recording expected/delivered system functionality and can therefore be used as a powerful documentation aid. The normal expectation would be that the system documentation contains both the requirements and the system specification in a suitable formal notation, accompanied where appropriate with English narrative. The latter is particularly important for conveying information on system aspects which are not formally specified for various reasons.

4.6 Human-Computer Interface

The human-computer interface (HCI) is an important part of most software systems. In safety-critical systems, it becomes ever more important that the interface is both dependable [22] and ergonomically sound.[5] Formalizing HCI in a realistic and useful manner is a difficult task since the problem has widely divergent facets such as task allocation and cognition, but progress is being made in categorizing features of interfaces that may help to ensure their reliability in the future. However, as it is recognized by the technical workplan of the second phase of the UK SafeIT research programme [94], there seems to be considerable scope for work in this area. Investigation of human errors [124] and how computers can help to avoid them is now being undertaken in a formal context [60].

4.7 Object-oriented Methods

Object-oriented approaches to software development have been advocated as a way to improve the design and reusability of software components, and hence increase their reliability. Recently there has been much discussion on combining object-oriented and formal methods, especially in critical applications [43]. Much research work has been undertaken in extending formal notations such as Z and VDM to include object-oriented concepts (e.g., see [143]). Currently there are many variants, and it remains to be seen which if any will emerge to be used in practice.

[5] Witness, for example, the HCI issues surrounding the fly-by-wire A320 Airbus. Some pilots have consistently criticized the ergonomics of the cockpit instrument layout which they have identified as a possible contributing factor to the pilot errors which have caused at least 2 crashes so far.

4.8 Artificial Intelligence

Despite the complexities and difficulty of understanding the exact nature of AI, there is interest in including artificial intelligence in safety-critical systems. In particular, *blackboard systems* are being used as a method of communication within AI systems, for example in the area of robotics [118]. Blackboard systems have previously been rather vaguely described, but this problem is now being recognized and an attempt to formalize them has been undertaken [35]. The formal verification of AI software is further discussed in [126, 150].

4.9 Static Analysis

Static analysis techniques and tools such as MALPAS and SPADE are used in industry for the rigorous checking of program code. Such techniques are sometimes used for *post hoc* validation of (safety-critical) code. It is a matter of engineering judgement as to how much effort should be expended to design the system correctly in the first place and how much checking should be undertaken after the design has been implemented. The identification and the discharging of proof obligations are two phases of the design process [30].

5 Safety Standards

There are a wide variety of standards bodies – perhaps too many[6] – throughout the world concerned with software development. The IED/SERC funded SMARTIE project is investigating a standards assessment framework [46] and [57] gives an overview of existing standards. Many have emerged or are currently emerging in the area of software safety, because this is now of such widespread importance and public concern. Formal methods are being increasingly mentioned in such standards as a possible method of improving dependability. This section gives some examples of such standards.

In addition, formal specification languages and their semantics are themselves being standardized (e.g., LOTOS [ISO89], VDM [BSI91] and Z [ZIP91]). Formal notations are also becoming increasingly accepted in standards as it is realized that many existing standards using informal natural language descriptions alone (e.g., for programming language semantics) are ambiguous and can easily be (and often are) misinterpreted.

An important trigger for the exploitation of research into formal methods could be the interest of regulatory bodies or standardization committees (e.g., the *International Electrotechnical Commission* [IEC91, IEC92], the *European Space Agency* [ESA91], and the UK *Health and Safety Executive* [HSE87a, HSE87b]). Many emerging standards are at the discussion stage (e.g., [RIA91, IEEE91]). A major impetus has already been provided in the UK by promulgation of the Ministry of Defence interim standard 00-55 [MoD91a], which mandates the use of formal methods and languages with sound formal semantics. Previous guidelines [EWICS88/89] have been influential in the contents of safety standards and a standards framework [SafeIT90b] may help to provide a basis for future standards.

5.1 Formal methods in standards

Up until relatively recently there have been few standards concerned specifically with software in safety-critical systems. Often software quality standards such as the ISO9000 series have been

[6] During the December 1991 ACM SIGSOFT conference on safety critical software, Mike De Walt of the US Federal Aviation Administration mentioned that a recent count revealed 146 different standards relevant to software safety!

used instead since these were the nearest relevant guidelines. Now a spate of standards in this area have been or are about to be issued. [135] gives a good overview (in 1989) and also covers a number of formalisms such as VDM, Z and OBJ. Many standards do not mention a formal approach specifically (e.g., MIL-STD-882B [DoD84]) although most are periodically updated to incorporate recent ideas (e.g., a draft version of MIL-STD-882C is currently under discussion).

The software engineering community became acutely aware of the introduction of formal methods in standards in the late 80s and particularly since the introduction of the UK MoD DefStan 00–55 which will be commented upon later in this section. Although the debate on the exact formal methods content of standards like 00–55 is bound to continue, we feel that there are certain aspects such as formal specification which cannot sensibly be ignored by standardizing bodies.

This section introduces the recommendations concerning the use of formal methods in a number of software safety standards. The selection, which is summarized in Table 5, is somewhat eclectic, but demonstrates the range of areas and organizational bodies that are involved. Overviews of current standards concerned with software safety from an American point of view are provided by [152, 158].

The US and Europe are the major sources of software safety standards and research in this area. In Canada, [OH90, AECB91] have been produced in relation to the nuclear industry. Standards Australia is recommending adoption of the IEC Draft Document 65A [IEC91]. [105] gives a rare overview of dependability and safety issues in Japan, including details of an abortive attempt to produce their own JIS standard sponsored by MITI in this area, although guideline reports exist.

RTCA DO-178

The US *Radio Technical Commission for Aeronautics* (RTCA) produced a guideline on software considerations in airborne systems and equipment certification (DO-178A) in 1985 [RTCA85]. This does not explicitly recognize formal methods as part of accepted practice. However a new guideline (DO-178B) is currently under consideration by a committee and is likely to include a brief section on *Guidelines for the Use of Formal Methods*.

UK HSE

The UK *Health and Safety Executive* issued an introductory guide [HSE87a] and some general technical guidelines [HSE87b] concerning Programmable Electronic Systems (PES) in safety related applications in 1987. Two pages are devoted to software development (pp. 31–32) and a further two to software change procedures (pp. 32–33). No mention is made of formal methods; it simply states that software should be of high quality, well documented, match its specification and be maintainable. It does list the necessary phases of software development and includes in these requirements specification, software specification, design, coding and testing, and system testing. It goes on to state that modifications to the software should be strictly controlled.

IEC

The *International Electrotechnical Commission* has issued two standards in the area of safety-critical system development [IEC91, IEC92]. These documents were originally issued in 1989, but have recently been updated and reissued. The former deals specifically with software for computers in the application of industrial safety related systems, while the latter is concerned with the functional safety of programmable electronic systems in general. These are generic international standards designed to be applied in many different industrial sectors. An example of a particular instantiation of the IEC65-WG9 standard is included below.

The "formal methods" CCS, CSP, HOL, LOTOS, OBJ, Temporal Logic, VDM and Z are specifically mentioned in [IEC91] (with a brief description and bibliography for each) as possible techniques to be applied in the development of safety-critical systems in an extensive section (B.30, pp. B-14–18) under a *Bibliography of Techniques*. A shorter section on "formal proof" (B.30, p. B-18) is also included.

ESA

The *European Space Agency* has issued guidelines for software engineering standards [ESA91]. This suggests that formal notations such as Z or VDM should be used for specifying software requirements in safety-critical systems (p. 1-27). A natural language description should accompany the formal text. A short section on formal proof (p. 2-25) suggests that proof of the correctness of the software should be attempted if practicable. Because of the possibility of human error, proofs should be checked independently. Methods such as formal proof should always be tried before testing is undertaken.

UK RIA

The *Railway Industry Association* consisting of a number of interested organizations and industrial companies in the UK have produced a consultative document on safety-related software for railway signaling [RIA91]. It is a draft proposal for an industry specific standard that has yet to be ratified. It makes extensive reference to the IEC65-WG9 standard [IEC91]. Formal methods are mentioned briefly in several places in the document. Rigorous correctness argument is advocated as a less detailed and formal proof method to demonstrate the correctness of a program by simply outlining the main steps of the proof. In general, formal techniques are only recommended or mandated when the very highest levels of safety are required.

MoD 00-55 and 00-56

The UK Ministry of Defence have recently published two interim standards concerning safety. 00-55, on the procurement of safety-critical software in defence equipment [MoD91a] is split into two parts, on requirements and guidance. The 00-56 standard is concerned with hazard analysis and safety classification of the computer and programmable electronic system elements of defence equipment [MoD91b]. These standards, and particularly 00-55, mention and mandate formal methods extensively and have, therefore, created many ripples in the defence software industry as well as the software engineering community in the UK.[7] The standards are currently in interim form. The MoD, which had previously set 1995 as the goal date for the introduction of fully mandatory standards [21], has now withdrawn a specific introduction date.

00-55 mandates the expression of safety-critical module specifications in a formal language notation. Such specifications must be analyzed to establish their consistency and completeness in respect of all potentially hazardous data and control flow domains. A further fundamental requirement is that all safety-critical software must be subject to validation and verification to establish that it complies with its formal specification over its operating domain. This involves static and dynamic analysis as well as formal proofs and informal but rigorous arguments of correctness.

[7] See [147] for a very interesting account of the evolution of 00-55 and the associated debate in the UK.

AECB, Canada

The Atomic Energy Control Board (AECB) in Canada have commissioned a proposed standard for software for computers in the safety systems of nuclear power stations [AECB91]. This has been prepared by David Parnas who is well known in the fields of both software safety and formal methods. The standard formalizes the notions of the environment ('nature'), the behavioural system requirements, and their feasibility with respect to the environment. It is based on the IEC Standard 880 [IEC86]. AECB have not, at the time of writing, decided to adopt Parnas's proposal and discussions are continuing.

IEEE P1228

The P1228 Software Safety Plans Working Group, under the Software Engineering Standards Subcommittee of the IEEE Computer Society, is preparing a standard for software safety plans. This is an unapproved draft that is subject to change. The appendix of [IEEE91] includes headings of "*Formal/Informal Proofs*" and "*Mathematical Specification Verification*" under techniques being discussed for inclusion. The latest version of the draft (Draft G of January 1992) omits all mention of formal methods so it is unclear what the final position will be.

5.2 Education, certification and legislation

Standards are a motivating force that *pull* industry to meet a minimum level of safety in the development of critical systems. Another complementary force that could be seen to *push* industry is the education of engineers in the proper techniques that should be applied to such systems. A safety-critical software engineer should have an an appreciation of a great deal more areas than the average programmer. Such engineers must typically interface with control and hardware engineers for example.

Despite the above, safety-critical software is still either not mentioned at all, or mentioned in passing as being too specialized for inclusion, in many standard text books on software engineering, although formal methods are being included more now (e.g., see [136]). [53] bemoans the lack of mathematical content in many software engineering courses. [157] includes a recent report on education and training with respect to safety-related software. Professional bodies can provide assistance in the form of up-to-date information aimed at practicing engineers (e.g., [69]).

It is a paradox of current avionics practice that the engineer who fixes bolts on airframes must be accredited whereas the programmer who writes the autopilot software needs no such qualification. [119] discusses the accreditation of software engineers by professional institutions. It is suggested that training is as important as experience in that *both* are necessary. In addition, (software) engineers should be responsible for their mistakes if they occur through negligence rather than genuine error. Safety-critical software is identified as an area of utmost importance where such ideas should be applied first because of the possible gravity of errors if they do occur.

Currently a major barrier to the acceptance of formal methods is the fact that many engineers and programmers do not have the appropriate training to make use of them and many managers do not know when and how they may be applied. This is gradually being alleviated as the necessary mathematics (typically set theory and predicate calculus) is being taught increasingly in computing science curricula. Educational concerns in the UK are reflected in the SafeIT strategy document [SafeIT90a]. The UK Department of Trade and Industry has commissioned a special study to stimulate the development of education and training in the area. In addition, the British Computer Society and the IEE have established working groups which are aiming to produce proposals on the content of courses aimed at safety-critical systems engineers.

Some standards and draft standards are now recognizing the problems and recommending that appropriate personnel should be used on safety-critical projects. There are suggestions that some sort of certification of developers should be introduced. This is still an active topic of discussion [107], but there are possible drawbacks as well as benefits by introducing such a 'closed shop' since suitable able engineers may be inappropriately excluded (and vice versa).

The education/accreditation debate has been particularly active in the UK, in the wake of Def Stan 00-55. The MoD, having commissioned a report on training and education to support 00-55 [160], has chosen to withdraw from the consequent controversy stating that it is beyond the remit of the standard to set a national training agenda. Perhaps the central issue here is not formal methods education *per se*, but the identity of the whole software engineering profession; in other words, what precisely *is* a software engineer is a question that will no doubt be debated for some time to come.[8]

Finally, legislation is likely to provide increasing motivation to apply appropriate techniques in the development of safety-critical systems. For example, a new piece of European Commission legislation, the Machine Safety Directive, is effective in the UK from 1st January 1993 [106]. This encompasses software and if there is an error in the machine's logic that results in injury then a claim can be made under civil law against the supplier. If negligence can be proved during the product's design or manufacture then criminal proceedings may be taken against the director or manager in charge. There will be a maximum penalty of three months in jail or a large fine. Suppliers will have to demonstrate that they are using best working practice. This could include, for example, the use of formal methods.

6 Discussion

This section offers a discussion of the current situation and some possible ways forward in research. It represents the opinions of the authors as opposed to an impartial and objective survey.

The subject of software safety has profound technical, business, professional and personal aspects for the individuals who research, develop, sell, use and rely upon computer controlled systems. So it is hardly surprising that the introduction and use of a technology such as formal methods in this context is accompanied by vigorous if not heated debate. What is at stake ranges from substantial industrial investment, to 'closed shop' interests and professional pride in the job, and ultimately to our very lives. The arguments surrounding the value and use of formal methods for safety-critical systems are a prime example of the potential for controversy.[9]

The complexity of critical systems is rising as more and more functionality is provided by software solutions. The gap between the dependability requirements and what we can achieve in terms of delivering and measuring such dependability is huge. We believe that, on the evidence of past experiments, formal methods technology deployed in conjunction with other techniques can help narrow this gap. The factors that diminish the effectiveness of formal methods in this context are:

- Some aspects of the technology, such as formal specification, have been widely used and are relatively well understood. Other practices, however, such as machine-supported theorem proving, have not benefited from real-world use and are correspondingly less well developed.

- Formal methods can be expensive when compared with traditional defect removal techniques. It is naive to assume that "money is no object" given that the cost of safety is

[8] See [148] for a discussion of the drift of many kinds of professionals into software engineering.

[9] See [117] for a discussion of the 'radical' and 'reformist' factions in software engineering.

highly subjective, varies from system to system even within the same sector and depends on the perception and the politics of risk [116].[10] Clearly the cost-effectiveness of formal methods will need to be established on a case by case basis. The UK SafeIT DATUM project [88] is currently investigating this issue.

- Although it is accepted that the use of formal methods increases dependability margins, we cannot measure by how much. In fact, even if we could, we would not be able to measure global dependability since we do not know how to combine formal methods assurance with metrics collected from other techniques such as fault tolerance.

In spite of these problems, we feel that *mature* formal methods can and should be used to produce safer software because benefits can be obtained without wholesale adoption. The mere act of writing a formal specification, for instance, can help to clarify system design and requirements; it can be used to improve or simplify a design; it can even be used to produce a rapid prototype in order to evaluate the projected system behaviour. However, in the context of safety-critical systems, it is profoundly important to recognize the limitations of any technology. Formal methods cannot do much, for example, in a chaotic software production environment.

If the issues surrounding the applicability of formal methods to critical systems are so complicated, it is hardly surprising that educational provision and standardization are equally complex matters. Currently, there is no universally agreed curriculum for safety critical software professionals. On the contrary, there is a plethora of standards and this domain is beginning to look surprisingly similar to the state of the art in formal methods; too many standards that are not industrially used and assessed.

In this paper, we have tried to present an objective account of the state of the art of formal methods as reflected by recent industrial practice and standardization activities. In our opinion, the areas that need to be addressed in the future are research, technology, education/accreditation and standardization for the use of formal methods in the development of safety-critical software.

Formal methods research

Research in formal methods to date has largely addressed the functional and/or temporal correctness of system. We believe that as well as continuing to strive for better formal models [64] there is a need to interface formal models with safety engineering techniques such as hazard analysis and risk engineering. We also believe that research needs to focus more on safety-critical system issues which we collectively call *provable dependability*. This viewpoint affords many research dimensions, including amongst others:

- Dependability requirements analysis/capture. Integrity, reliability, security, safety, functional behaviour, temporal behaviour.

- Dependability specification. Can dependability requirements be formally stated? Is it possible to develop problem specific calculi for the different aspects of dependability (such as fault tolerance and security)?

- Development of dependable systems. Can we develop the necessary theories for refinement/transformation? If not, how should high integrity systems be built?

[10] For instance, the MoD in the UK places different criticality on the importance of saving a life threatened by military aircraft in flight depending on whether the individual is a civilian, a pilot in flight at peace time and a pilot in flight at war time.

- Machine aided formal verification of dependability properties. To what extent can we use theorem proving tools for verifying the dependability properties of systems? Which existing technologies are relevant?
- Qualitative and quantitative analysis of the dependability that can be achieved using the combination of formal verification and fault tolerance. Can we increase confidence in systems by combining assurance methods?
- Case studies drawn from a wide spectrum of high integrity systems. Real-time embedded systems, distributed systems, high integrity transaction processing systems, theorem proving systems.

Formal methods technology

Formal methods research is abundant, although we believe that focusing on safety-critical systems is important since provable dependability has not been sufficiently addressed in the field. We must distinguish between the issues relating to technology and those relating to research. By *technology* we mean the transition from research results to methods and tools which are "fit for purpose" with regard to the needs of industry. Understanding the difference between technology and research results is crucial and can go some way in explaining the reluctance of industry to adopt formal methods. The fact that a highly trained expert proves the correctness of a simple computer-based system in a research environment does not imply that a real safety-critical developer will use a theorem prover. Suitable technology must be produced before the process is enabled and as with any other endeavour the users as opposed to the research community must be the driving force.

In summary, in order to strengthen the technology and contribute to its maturity, the following are desirable:

- An engineering approach to formal specification and verification.
- Better tools.
- Investment in technology transfer.
- Unification and harmonization of the engineering practices involved in building safety-critical systems.
- More practical experience of the industrial use of the methods.
- Assessment and measurement of the effectiveness of formal methods.

Education and accreditation

The educational debate is also set to continue. It is likely that there will be a skills shortage in this area for the foreseeable future, although most computer science degree programs now contain at least elements of discrete mathematics and formal methods. The contentious issue is the education of the safety-critical software professional; work in this area is currently undertaken by the professional institutions and learned societies. Although, even for those of us teaching in higher education, there is no established consensus on this issue, it seems to us that software engineering education must be widened with safety engineering and dependability issues at the very least. The most fundamental question that has to be answered is whether the professionals writing the safety-critical software of the future should have a software or hardware or systems education. It is precisely the multidisciplinary nature of most safety-critical systems that makes educational provision such a thorny issue.

A closely related issue is the accreditation of such professionals. In our view, future accreditation is inevitable because of the massive stakes in resources and human lives involved in safety critical systems. Happily, the professional institutions are actively examining this issue in conjunction with the educational requirements. Although the outcome of these deliberations is uncertain, any reasonable accreditation procedure can hardly fail to take into account a combination of educational qualifications coupled with training and responsible experience requirements.

It appears that Europe is leading the US and the rest of the world in the field of formal methods education, so this may be a good sign for the long term development and reliability of safety-critical software in Europe.

Standards

The role of standards for safety related software has critical implications for all the aspects that we have discussed above. Witness the impact of the MoD Def Stan 00-55 both in terms of research and development, and education in the United Kingdom [147, 148]. The current level of standardization activity is encouraging. We note, however, that the proliferation of standards is not in itself sufficient to ensure the production of safer software. These standards need to be used and their impact on software safety assessed and quantified. Moreover, research is needed in order to establish precisely what standards should contain and how various sector specific standards interact when they are used simultaneously on a system. Work in this direction is reported in [46].

It is important that standards should not be over-prescriptive, or that prescriptive sections are clearly separated and identified as such (perhaps as an appendix or even as a separate document). These parts of a standard are likely to date much more quickly that its goals, and thus should be monitored and updated more often. Dependability goals should be set and the onus should be on the software supplier to ensure that the methods used achieve the required level of confidence. If particular methods are recommended or mandated, it is possible for the supplier to assume that the method will produce the desired results and blame the standards body if it does not. This reduces the responsibility and accountability of the supplier and may also result in a *decrease* of safety.

Standards have the dual effect of reflecting current best practice and normalizing procedures to the highest commonly acceptable denominator. As such, a significant number of software safety standards (at least half in this study) reflect the importance and relative maturity of formal methods. We believe that this trend is set to continue and standards will increasingly provide more motivation for the research, teaching and use of formal methods. We hope that this will eventually lead to some improvement in the safety of people and resources that depend upon computer software.

Acknowledgements

The European ESPRIT Basic Research Action **ProCoS** project (BRA 3104) and the UK Information Engineering Directorate safemos project (IED3/1/1036) provided financial support. Members of these projects provided intellectual support, encouragement and advice. We are particularly indebted to Prof. Tony Hoare (Oxford, UK) and Anders Ravn (DTH, Denmark). The following have also helped by supplying advice, information, papers, standards and feedback on earlier drafts which have been used as input to this survey: G.J.K. Asmis, Steve Bear, Steve

Cha, Simon Chadwick, Bernie Cohen, Derek Coleman, Dan Craigen, Ben Di Vito, Susan Gerhart, Pat Hall, Guenter Heiner, Jill Hill, Jim Horning, Jonathan Jacky, Paul Joannou, Nancy Leveson, John McDermid, Peter Neumann, David Parnas, Ted Ralston, John Rushby, Debra Sparkman, Richard Stein, Martyn Thomas, Brian Wichmann, Geoff Wilson, Cynthia Wright, Janusz Zalewski, Tony Zawilski. Finally, the anonymous reviewers provided helpful suggestions and references that have been incorporated here.

Bibliography

Standards, draft standards and guidelines

[AECB91] 'Proposed Standard for Software for Computers in the Safety Systems of Nuclear Power Stations'. Final Report for contract 2.117.1 for the Atomic Energy Control Board, Canada, March 1991 (By David L. Parnas, TRIO, Computing and Information Science, Queen's University, Kingston, Ontario K7L 3N6, Canada. Based on IEC Standard 880 [IEC86].)

[BSI91] 'VDM Specification Proto-Standard'. Draft, BSI IST/5/50 document, British Standards Institute, March 1991

[DoD84] 'Military Standard: System Safety Program Requirements'. MIL-STD-882B, Department of Defense, Washington DC 20301, USA, 30 March 1984

[ESA91] 'ESA Software Engineering Standards'. European Space Agency, 8–10 rue Mario-Nikis, 75738 Paris Cedex, France, ESA PSS-05-0 Issue 2, February 1991

[EWICS88/89] REDMILL, F. (Ed.): 'Dependability of Critical Computer Systems 1 & 2'. European Workshop on Industrial Computer Systems Technical Committee 7 (EWICS TC7), Elsevier Applied Science, London, 1988/1989

[FAA82] 'System Design Analysis'. US Department of Transportation, Federal Aviation Administration, Washington DC, USA, Advisory Circular 25.1309-2, September 1982

[HSE87a] 'Programmable Electronic Systems in Safety Related Applications: 1. An Introductory Guide'. Health and Safety Executive, HMSO, Publications Centre, PO Box 276, London SW8 5DT, UK, 1987

[HSE87b] 'Programmable Electronic Systems in Safety Related Applications: 2. General Technical Guidelines'. Health and Safety Executive, HMSO, Publications Centre, PO Box 276, London SW8 5DT, UK, 1987

[IEC86] 'Software for Computers in the Safety Systems of Nuclear Power Stations'. International Electrotechnical Commission, IEC 880, 1986

[IEC91] 'Software for Computers in the Application of Industrial Safety Related Systems'. International Electrotechnical Commission, Technical Committee no. 65, Working Group 9 (WG9), IEC 65A (Secretariat) 122, Version 1.0, 1 August 1991

[IEC92] 'Functional Safety of Programmable Electronic Systems: Generic Aspects'. International Electrotechnical Commission, Technical Committee no. 65, Working Group 10 (WG10), IEC 65A (Secretariat) 123, February 1992

[IEEE91] 'Standard for Software Safety Plans'. Preliminary – subject to revision, P1228, Software Safety Plans Working Group, Software Engineering Standards Subcommittee, IEEE Computer Society, USA, July 1991

[ISO87] 'JTC1 Statement of Policy on Formal Description Techniques'. ISO/IEC JTC1 N145 and ISO/IEC JTC1/SC18 N13333, International Standards Organization, Geneva, Switzerland, 1987

[ISO89] 'ISO 8807: Information Processing Systems – Open Systems Interconnection – LOTOS – A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour'. First edition, International Organization for Standardization, Geneva, Switzerland, 15 February 1989

[MoD91a] 'The Procurement of Safety Critical Software in Defence Equipment' (Part 1: Requirements, Part 2: Guidance). Interim Defence Standard 00-55, Issue 1, Ministry of Defence, Directorate of Standardization, Kentigern House, 65 Brown Street, Glasgow G2 8EX, UK, 5 April 1991

[MoD91b] 'Hazard Analysis and Safety Classification of the Computer and Programmable Electronic System Elements of Defence Equipment'. Interim Defence Standard 00-56, Issue 1, Ministry of Defence, Directorate of Standardization, Kentigern House, 65 Brown Street, Glasgow G2 8EX, UK, 5 April 1991

[OH90] 'Standard for Software Engineering of Safety Critical Software'. 982 C-H 69002-0001, Ontario Hydro, 700 University Avenue, Toronto, Ontario M5G 1X6, Canada, 21 December 1990

[RIA91] 'Safety Related Software for Railway Signalling'. BRB/LU Ltd/RIA technical specification no. 23, Consultative Document, Railway Industry Association, 6 Buckingham Gate, London SW1E 6JP, UK, 1991

[RTCA85] 'Software Considerations in Airborne Systems and Equipment Certification'. DO-178A, Radio Technical Commission for Aeronautics, One McPherson Square, 1425 K Street N.W., Suite 500, Washington DC 20005, USA, March 1985

[RTCA90] 'Minimum Operational Performance Standards for Traffic Alert and Collision Avoidance System (TCAS) Airborne Equipment – Consolidated Edition'. DO-185, Radio Technical Commission for Aeronautics, One McPherson Square, 1425 K Street N.W., Suite 500, Washington DC 20005, USA, 6 September 1990

[SafeIT90a] BLOOMFIELD, R.E. (Ed.): 'SafeIT1 – The Safety of Programmable Electronic Systems'. Safety-Related Working Group (SRS-WG), Interdepartmental Committee on Software Engineering (ICSE), Department of Trade and Industry, ITD7a – Room 840, Kingsgate House, 66–74 Victoria Street, London SW1E 6SW, UK, June 1990

[SafeIT90b] BLOOMFIELD, R.E., and BRAZENDALE, J. (Eds.): 'SafeIT2 – A Framework for Safety Standards'. Safety-Related Working Group (SRS-WG), Interdepartmental Committee on Software Engineering (ICSE), Department of Trade and Industry, ITD7a – Room 840, Kingsgate House, 66–74 Victoria Street, London SW1E 6SW, UK, June 1990

[UN64] UN Committee for the Transport of Dangerous Goods, Technical Report, 1964

[ZIP91] NICHOLLS, J.E., and BRIEN, S.M. (Eds.): 'Z Base Standard'. ZIP Project Technical Report ZIP/PRG/92/121, SRC Document: 132, Version 1.0, 30 November 1992 (Available from the Secretary, ZIP Project, Oxford University Computing Laboratory, PRG, 11 Keble Road, Oxford OX1 3QD, UK.)

Other references

[1] ABRIAL, J.R.: 'The B reference manual', Edinburgh Portable Compilers, 17 Alva Street, Edinburgh EH2 4PH, UK, 1991

[2] ABRIAL, J.R., LEE, M.K.O., NEILSON, D.S., SCHARBACH, P.N., and SØRENSEN, I.H.: 'The B-method' *in* PREHN, S., and TOETENEL, W.J. (Eds.): 'VDM '91, Formal Software Development Methods', Volume 2: Tutorials (Springer-Verlag, *Lecture Notes in Computer Science*, 1991) **552**, pp. 398–405

[3] ANDERSON, S., and CLELAND, G.: 'Adopting mathematically-based methods for safety-critical systems production' *in* REDMILL, F. (Ed.): 'Safety Systems: The Safety-Critical Systems Club Newsletter', Centre for Software Reliability, University of Newcastle upon Tyne, UK, January 1992, **1**, (2), p. 6

[4] ARCHINOFF, G.H., HOHENDORF, R.J., WASSYNG, A., QUIGLEY, B. and BORSCH, M.R.: 'Verification of the shutdown system software at the Darlington nuclear generating station'. International Conference on Control and Instrumentation in Nuclear Installations, The Institution of Nuclear Engineers, Glasgow, UK, May 1990

[5] AUGARTEN, S.: 'The Whirlwind project' *in* 'Bit by Bit: An Illustrated History of Computers', chapter 7 (Ticknor & Fields, New York, 1984) pp. 195–223

[6] BABEL, P.S.: 'Software integrity program'. Aeronautical Systems Division, Airforce, U.S., April 1987

[7] BARROCA, L., and MCDERMID, J.: 'Formal methods: use and relevance for the development of safety critical systems', *The Computer Journal*, **35**, (6), December 1992

[8] BARDEN, R., STEPNEY, S., and COOPER, D.: 'The use of Z' *in* NICHOLLS, J.E. (Ed.): 'Z User Workshop, York 1991' (Springer-Verlag, Workshops in Computing, 1992) pp. 99–124

[9] BEAR, S.: 'An overview of HP-SL' *in* PREHN, S., and TOETENEL, W.J. (Eds.): 'VDM '91, Formal Software Development Methods' (Springer-Verlag, *Lecture Notes in Computer Science*, 1991) **551**, pp. 571–587

[10] BENNETT, P.A.: 'Safety' *in* McDERMID, J.A. (Ed.): 'Software Engineer's Reference Book', chapter 60 (Butterworth-Heinemann Ltd., Oxford, 1991)

[11] BJØRNER, D.: 'A ProCoS project description: ESPRIT BRA 3104', *Bulletin of the EATCS*, 1989, **39**, pp. 60–73

[12] BLOOMFIELD, R.E., FROOME, P.K.D., and MONAHAN, B.Q.: 'Formal methods in the production and assessment of safety critical software', *Reliability Engineering & System Safety*, **32**, (1), 1989, pp. 51–66 (Also in [89].)

[13] BLYTH, D., BOLDDYREFF, C., RUGGLES, C., and TETTEH-LARTEY, N.: 'The case for formal methods in standards', *IEEE Software*, September 1990, pp. 65–67

[14] BOEBERT, W.E.: 'Formal verification of embedded software', *ACM SIGSOFT Software Engineering Notes*, July 1980, **5**, (3), pp. 41–42

[15] BOEHM, B.: 'Software risk management tutorial'. TRW-ACM Seminar, April 1988

[16] BOWEN, J.P., and BREUER, P.T.: *in* VAN ZUYLEN, H. (Ed.): 'The REDO Compendium of Reverse Engineering for Software Maintenance', chapter 9 (John Wiley, 1992)

[17] BOWEN, J.P., and STAVRIDOU, V.: 'Formal methods and software safety', in [47], 1992, pp. 93–98

[18] BOWEN, J.P., and STAVRIDOU, V.: 'The industrial take-up of formal methods in safety-critical and other areas: a perspective', *in* 'Formal Methods Europe Symposium (FME'93)', Odense Technical College, Denmark, 19–23 April 1993 (Springer-Verlag, *Lecture Notes in Computer Science*, 1993) To appear

[19] BOYER, R.S., and MOORE, J.S.: 'A computational logic handbook' (Academic Press, Boston, 1988)

[20] BROCK, B., and HUNT, W.A.: 'Report on the formal specification and partial verification of the VIPER microprocessor'. Technical Report No. 46, Computational Logic Inc., Austin, Texas, USA, January 1990

[21] BROWN, M.J.D.: 'Rationale for the development of the UK defence standards for safety-critical computer software'. Proc. COMPASS '90, Washington DC, USA, June 1990

[22] BURNS, A.: 'The HCI component of dependable real-time systems', *Software Engineering Journal*, July 1991, **6**, (4), pp. 168–174

[23] BUTLER, R.W., and FINELLI, G.B.: 'The infeasibility of experimental quantification of life-critical software reliability'. Proc. ACM SIGSOFT '91 Conference on Software for Critical Systems, *Software Engineering Notes*, ACM Press, December 1991, **16**, (5), pp. 66–76

[24] BUTH, B., BUTH, K-H., FRÄNZLE, M., VON KARGER, B., LAKHNECHE, Y., LANGMAACK, H., and MÜLLER-OLM, M.: 'Provably correct compiler development and implementation', *in* 'Compiler Construction '92', 4th International Conference, Paderborn, Germany (Springer-Verlag, *Lecture Notes in Computer Science*, 1992) **641**

[25] BUXTON, J.N., and MALCOLM, R.: 'Software technology transfer', *Software Engineering Journal*, January 1991, **6**, (1), pp. 17–23

[26] CANNING, A.: 'Assessment at the requirements stage of a project'. Presented at '2nd Safety Critical Systems Club Meeting', Beaconsfield, UK, October 1991 (Available from Advanced Software Department, ERA Technology Ltd, Cleeve Rd, Leatherhead KT22 7SA, UK.)

[27] CHAPRONT, P.: 'Vital coded processor and safety related software design', in [47], 1992, pp. 141–145

[28] CHARETTE, R.N.: 'Applications strategies for risk analysis' (McGraw Hill, Software Engineering Series, 1990)

[29] CLUTTERBUCK, D.L., and CARRÉ, B.A.: 'The verification of low-level code', *Software Engineering Journal*, May 1988, **3**, (3), pp. 97–111

[30] COHEN, B., and PITT, D.H.: 'The identification and discharge of proof obligations' *in* 'Testing Large Software Systems', Wolverhampton Polytechnic, UK, 1990

[31] COHN, A.J.: 'A proof of correctness of the Viper microprocessor: the first level' *in* 'VLSI Specification, Verification and Synthesis' (Kluwer Academic Publishers, 1988)

[32] COHN, A.J.: 'Correctness properties of the Viper block model: the second level'. Proc. 2nd Banff Workshop on Hardware Verification (Springer-Verlag, 1988)

[33] COHN, A.J.: 'The notion of proof in hardware verification', *Journal of Automated Reasoning*, May 1989, **5**, pp. 127–139

[34] COLEMAN, D.: 'The technology transfer of formal methods: what's going wrong?'. Proc. 12th ICSE Workshop on Industrial Use of Formal Methods, Nice, France, March 1990

[35] CRAIG, I.: 'The formal specification of advanced AI architectures' (Ellis Horwood, AI Series, 1991)

[36] CRAIGEN, D. (Ed.): 'Formal methods for trustworthy computer systems (FM89)' (Springer-Verlag, Workshops in Computing, 1990)

[37] CULLYER, W.J.: 'Hardware integrity', *Aeronautical Journal of the Royal Aeronautical Society*, September 1985, pp. 263–268

[38] CULLYER, W.J.: 'High integrity computing' *in* JOSEPH, M. (Ed.): 'Formal Techniques in Real-time and Fault-tolerant Systems' (Springer-Verlag, *Lecture Notes in Computer Science*, 1988) **331**, pp. 1–35

[39] CULLYER, W.J., and PYGOTT, C.H.: 'Application of formal methods to the VIPER microprocessor' *in* 'IEE Proceedings, Part E, Computers and Digital Techniques' May 1987, **134**, (3), pp. 133–141

[40] CURZON, P.: 'Of what use is a verified compiler specification?', Technical Report No. 274, Computer Laboratory, University of Cambridge, UK, 1992

[41] CYRUS, J.L., BLEDSOE, J.D., and HARRY, P.D.: 'Formal specification and structured design in software development', *Hewlett-Packard Journal*, December 1991, pp. 51–58

[42] DAVIES, J.: 'Specification and proof in real-time systems'. Technical Monograph PRG-93, Programming Research Group, Oxford University Computing Laboratory, April 1991

[43] DE CHAMPEAUX, D. *et al.* 'Formal techniques for OO software development'. OOPSLA'91 Conference in Object-Oriented Programming Systems, Languages, and Applications, *SIGPLAN Notices*, ACM Press, November 1991, **26**, (11), pp. 166–170

[44] 'Safety related computer controlled systems market study', Review for the Department of Trade and Industry by Coopers & Lybrand (HMSO, London, 1992)

[45] DYER, M.: 'The Cleanroom approach to quality software development' (Wiley Series in Software Engineering Practice, 1992)

[46] FENTON, N., and LITTLEWOOD, B.: 'Evaluating software engineering standards and methods'. Proc. 2èmes Rencontres Qualiteé Logiciel & Eurometrics '91, March 1991, pp. 333–340

[47] FREY, H.H. (Ed.).: 'Safety of computer control systems 1992 (SAFECOMP'92)', Computer Systems in Safety-critical Applications, Proc. IFAC Symposium, Zürich, Switzerland, 28–30 October 1992 (Pergamon Press, 1992)

[48] GLASS, R.L.: 'Software vs. hardware errors', *IEEE Computer*, December 1980, **23**, (12)

[49] GOGUEN, J., and WINKLER, T.: 'Introducing OBJ3'. Technical Report SRI-CSL-88-9, SRI International, Menlo Park, California, USA, August 1988

[50] GOLDSACK, S.J., and FINKELSTEIN, A.C.W.: 'Requirements engineering for real-time systems', *Software Engineering Journal*, May 1991, **6**, (3), pp. 101–115

[51] GOOD, D.I., and YOUNG, W.D.: 'Mathematical methods for digital system development' *in* PREHN, S., and TOETENEL, W.J. (Eds.): 'VDM '91, Formal Software Development Methods', Volume 2: Tutorials (Springer-Verlag, *Lecture Notes in Computer Science*, 1991) **552**, pp. 406–430

[52] GORDON, M.J.C.: 'HOL: A proof generating system for Higher-Order Logic' *in* BIRTWISTLE, G., and SUBRAMANYAM, P.A. (Eds.): 'VLSI Specification, Verification and Synthesis' (Kluwer, 1988) pp. 73–128

[53] GRIES, D.: 'Influences (or lack thereof) of formalism in teaching programming and software engineering' *in* DIJKSTRA, E.W. (Ed.): 'Formal Development of Programs and Proofs', chapter 18 (Addison Wesley, University of Texas at Austin Year of Programming Series, 1990) pp. 229–236

[54] GUIHO, G., and HENNEBERT, C.: 'SACEM software validation'. Proc. 12th International Conference on Software Engineering (IEEE Computer Society Press, March 1990) pp. 186–191

[55] HALANG, W.A., and KRÄMER, B.: 'Achieving high integrity of process control software by graphical design and formal verification', *Software Engineering Journal*, January 1992, **7**, (1), pp. 53–64

[56] HALL, J.A.: 'Seven myths of formal methods', *IEEE Software*, September 1990, pp. 11–19

[57] HALL, P.A.V.: 'Software development standards', *Software Engineering Journal*, May 1989, **4**, (3), pp. 143–147

[58] HAMMER, W.: 'Handbook of system and product safety' (Prentice-Hall Inc., Englewood Cliffs, New Jersey, USA, 1972)

[59] HANSEN, K.M., RAVN, A.P., and RISCHEL, H.: 'Specifying and verifying requirements of real-time systems'. Proc. ACM SIGSOFT '91 Conference on Software for Critical Systems, *Software Engineering Notes*, ACM Press, December 1991, **16**, (5), pp. 44–54

[60] HARRISON, M.D.: 'Engineering human error tolerant software' *in* NICHOLLS, J.E. (Ed.): 'Z User Workshop, York 1991' (Springer-Verlag, Workshops in Computing, 1992) pp. 191–204

[61] HELPS, K.A.: 'Some verification tools and methods for airborne safety-critical software', *Software Engineering Journal*, November 1986, **1**, (6), pp. 248–253

[62] HILL, J.V.: 'The development of high reliability software – RR&A's experience for safety critical systems'. Second IEE/BCS Conference, Software Engineering 88, Conference Publication No. 290, July 1988, pp. 169–172

[63] HILL, J.V.: 'Software development methods in practice'. Proc. 6th Annual Conference on Computer Assurance (COMPASS), 1991

[64] HOARE, C.A.R.: 'Algebra and models' *in* HE Jifeng, and OLDEROG, E.R. (Eds.): 'Interfaces between Languages for Concurrent Systems', chapter 1, Volume II, ESPRIT BRA 3104, Provably Correct Systems, ProCoS, Draft Final Deliverable, October 1991

[65] HOARE, C.A.R., and GORDON, M.J.C. (Eds.) 'Mechanized reasoning and hardware design' (Prentice Hall International Series in Computer Science, UK, 1992)

[66] HOARE, C.A.R., HE Jifeng, BOWEN, J.P., and PANDYA, P.K.: 'An algebraic approach to verifiable compiling specification and prototyping of the ProCoS level 0 programming language', *in* DIRECTORATE-GENERAL OF THE COMMISSION OF THE EUROPEAN COMMUNITIES (Ed.): 'ESPRIT '90 Conference Proceedings', Brussels (Kluwer Academic Publishers B.V., 1990) pp. 804–818

[67] HOUSTON, I., and KING, S.: 'CICS project report: experiences and results from the use of Z in IBM' *in* PREHN, S., and TOETENEL, W.J. (Eds.): 'VDM '91, Formal Software Development Methods' (Springer-Verlag, *Lecture Notes in Computer Science*, 1991) **551**, pp. 588–603

[68] HUMPHREY, W.S., KITSON, D.H., and CASSE, T.C.: 'The state of software engineering practice: a preliminary report'. Proc. 11th International Conference on Software Engineering, Pittsburgh, USA, May 1989, pp. 277–288

[69] 'Safety-related systems: A professional brief for the engineer'. The Institution of Electrical Engineers, Savoy Place, London WB2R 0BR, UK, January 1992

[70] IYER, R.K., and VERLARDI, P.: 'Hardware-related software errors: measurement and analysis', *IEEE Transactions on Software Engineering*, February 1985, **SE-11**, (2)

[71] JACKY, J.: 'Formal specifications for a clinical cyclotron control system' *in* MORICONI, M. (Ed.): 'Proc. ACM SIGSOFT International Workshop on Formal Methods in Software Development', *Software Engineering Notes*, ACM Press, September 1990, **15**, (4), pp. 45–54

[72] JACKY, J.: 'Safety-critical computing: hazards, practices, standards and regulation', *in* DUNLOP, C., and KLING, R. (Eds.): 'Computerization and controversy', chapter 5 (Academic Press, 1991) pp. 612–631

[73] JACKY, J.: 'Verification, analysis and synthesis of safety interlocks'. Technical Report 91-04-01, Department of Radiation Oncology RC-08, University of Washington, Seattle, WA 98195, USA, April 1991

[74] JAFFE, M.S., LEVESON, N.G., HEIMDAHL, M.P., and MELHART, B.E.: 'Software requirements analysis for real-time process-control systems', *IEEE Transactions on Software Engineering*, March 1991, **SE-17**, (3), pp. 241–258

[75] JOANNOU, P.K., HARAUZ, J., TREMAINE, D.R., ICHIYEN, N. and CLARK, A.B.: 'The Canadian nuclear industry's initiative in real-time software engineering'. Ontario Hydro, 700 University Avenue, Toronto, Ontario M5G 1X6, Canada, 1991

[76] JONES, C.B.: 'Systematic software development using VDM', 2nd edition (Prentice Hall International Series in Computer Science, 1990)

[77] KANDEL, A., and AVNI, E.: 'Engineering risk and hazard assessment', Volume I (CRC Press, Boca Raton, Florida, USA, 1988)

[78] KNIGHT, J.C., and LEVESON, N.G.: 'A reply to the criticisms of the Knight & Leveson experiment', *ACM SIGSOFT Software Engineering Notes*, January 1990, **15**, (1), pp. 25–35

[79] KNIGHT, J.C., and KIENZLE, D.M.: 'Preliminary experience using Z to specify a safety-critical system', *in* 'Z User Workshop, London 1992' (Springer-Verlag, Workshops in Computing, 1993) To appear

[80] KOPETZ, H., ZAINLINGER, R., FOHLER, G., KANTZ, H., and PUSCHNER, P.: 'The design of real-time systems: from specification to implementation and verification', *Software Engineering Journal*, May 1991, **6**, (3), pp. 73–82

[81] LADEAU, B.R., and FREEMAN, C.: 'Using formal specification for product development', *Hewlett-Packard Journal*, December 1991, pp. 62–66

[82] LAPRIE, J.C.: 'Dependability: a unifying concept for reliable computing and fault tolerance' *in* ANDERSON, T. (Ed.): 'Dependability of Resilient Computers', chapter 1 (Blackwell Scientific Publications, Oxford, 1989) pp. 1–28

[83] LAPRIE, J.C. (Ed.): 'Dependability: basic concepts and terminology' (Springer-Verlag, 1991)

[84] LEVESON, N.G.: 'Software safety: why, what and how', *ACM Computing Surveys*, June 1986, **18**, (2), pp. 125–163

[85] LEVESON, N.G.: 'Software safety in embedded computer systems', *Communications of the ACM*, February 1991, **34**, (2), pp. 34–46

[86] LEVESON, N.G., and TURNER, C.T.: 'An investigation of the Therac-25 accidents', UCI Technical Report #92-108 (& University of Washington TR #92-11-05), Information and Computer Science Dept., University of California, Irvine, CA 92717, USA, 1992

[87] LINDSAY, P.A.: 'A survey of mechanical support for formal reasoning', *Software Engineering Journal*, 1988, **3**, (1), pp. 3–27

[88] LITTLEWOOD, B.: 'The need for evidence from disparate sources to evaluate software safety', Proc. Safety-critical Systems Symposium, Bristol, UK, February 1993 (Springer-Verlag, 1993) To appear

[89] LITTLEWOOD, B., and MILLER, D. (Eds.): 'Software reliability and safety' (Elsevier Applied Science, London and New York, 1991) (Reprinted from *Reliability Engineering & System Safety*, **32**, (1)–2, 1989.)

[90] LITTLEWOOD, B., and STRIGINI, L.: 'The risks of software', *Scientific American*, November 1992, **267**, (5), pp. 38–43

[91] MACKENZIE, D.: 'The fangs of the VIPER', *Nature*, 8 August 1991, **352**, pp. 467–468

[92] MACKENZIE, D.: 'Negotiating arithmetic, constructing proof: the sociology of mathematics and information technology', Programme on Information & Communication Technologies, Working Paper Series, No. 38, Research Centre for Social Sciences, University of Edinburgh, 56 George Square, Edinburgh EH8 9JU, UK, November 1991

[93] MAHONY, B., and HAYES, I.J.: 'A case-study in timed refinement: a mine pump', *IEEE Transactions on Software Engineering*, September 1992, **18**, (9), pp. 817–826

[94] MALCOLM, R.: 'Safety critical systems research programme: technical workplan for the second phase' *in* REDMILL, F. (Ed.): 'Safety Systems: The Safety-Critical Systems Club Newsletter', Centre for Software Reliability, University of Newcastle upon Tyne, UK, January 1992, **1**, (2), pp. 1–3

[95] MALER, O, MANNA, Z., and PNUELI, A.: 'From timed to hybrid systems' *in* DE BAKKER, J.W., HUIZING, C., DE ROEVER, W.-P., and ROZENBERG, W. (Eds.): 'Real-Time: Theory in Practice, REX Workshop' (Springer-Verlag, *Lecture Notes in Computer Science*, 1992) **600**, pp. 447–484

[96] MANNA, Z., and PNUELI, A.: 'The temporal logic of reactive and concurrent systems: specification' (Springer-Verlag, 1992)

[97] MAY, D.: 'Use of formal methods by a silicon manufacturer' *in* HOARE, C.A.R. (Ed.): 'Developments in Concurrency and Communication', chapter 4 (Addison-Wesley, University of Texas at Austin Year of Programming Series, 1990) pp. 107–129

[98] MAYGER, E.M., and FOURMAN, M.P.: 'Integration of formal methods with system design'. Proc. Conference on Very Large Scale Integration (VLSI '91), Edinburgh, UK, 1991, pp. 3a.2.1–3a.2.11

[99] McDERMID, J.A.: 'Formal methods: use and relevance for the development of safety critical systems' *in* BENNETT, P.A.: 'Safety Aspects of Computer Control' (Butterworth-Heinemann, 1991)

[100] MOORE, J.S. *et al.*, 'Special issue on system verification', *Journal of Automated Reasoning*, 1989, **5**, (4), pp. 409–530

[101] MOSER, L.E., and MELLIAR-SMITH, P.M.: 'Formal verification of safety-critical systems', *Software — Practice and Experience*, August 1990, **20**, (8), pp. 799–821

[102] MUKHERJEE, P., and STAVRIDOU, V.: 'The formal specification of safety requirements for the storage of explosives'. Technical Report No. DITC 185/91, National Physical Laboratory, Teddington, Middlesex TW11 0LW, UK, August 1991

[103] MYERS, W.: 'Can software for the strategic defense initiative ever be error-free?', *IEEE Computer*, November 1986, **19**, (11)

[104] 'Peer review of a formal verification/design proof methodology'. NASA Conference Publication 2377, July 1983

[105] NATSUME, T., and HASEGAWA, Y.: 'A view on computer systems and their safety in Japan', in [47], 1992, pp. 45–49

[106] NEESHAM, C.: 'Safe conduct', *Computing*, 12 November 1992, pp. 18–20

[107] NEUMANN, P.G. (Ed.): 'Subsection on certification of professionals', *ACM SIGSOFT Software Engineering Notes*, January 1991, **16**, (1), pp. 24–32

[108] NEUMANN, P.G.: 'Illustrative risks to the public in the use of computer systems and related technology', *ACM SIGSOFT Software Engineering Notes*, January 1992, **16**, (1), pp. 23–32

[109] NORMINGTON, G: 'Cleanroom and Z', *in* 'Z User Workshop, London 1992' (Springer-Verlag, Workshops in Computing, 1993) To appear

[110] OSTROFF, J.S.: 'Formal methods for the specification and design of real-time safety critical systems', *Journal of Systems and Software*, 1992, **18**, (1), pp. 33–60

[111] PAGE, I., and LUK, W.: 'Compiling Occam into field-programmable gate arrays' *in* MOORE, W., and LUK, W. (Eds.): 'FPGAs', Oxford Workshop on Field Programmable Logic and Applications (Abingdon EE&CS Books, 15 Harcourt Way, Abingdon OX14 1NV, UK, 1991) pp. 271–283

[112] PALFREMAN, J., and SWADE, D.: 'The dream machine' (BBC Books, London, 1991)

[113] PARNAS, D.L., VON SCHOUWEN, A.J., and SHU PO KWAN 'Evaluation of safety-critical software', *Communications of the ACM*, June 1990, **33**, (6), pp. 636–648

[114] PARNAS, D.L., ASMIS, G.J.K., and MADEY, J.: 'Assessment of safety-critical software in nuclear power plants', *Nuclear Safety*, April–June 1991, **32**, (2), pp. 189–198

[115] PARNAS, D.L., and MADEY, J.: 'Functional documentation for computer systems engineering'. Version 2, CRL Report No. 237, TRIO, Communications Research Laboratory, Faculty of Engineering, McMaster University, Hamilton, Ontario, Canada L8S 4K1, September 1991

[116] PASQUINE, A., and RIZZO, A.: 'Risk perceptions and acceptance of computers in critical applications', in [47], 1992, pp. 293–298

[117] PELAEZ, E.: 'A gift from Pandora's box: the software crisis'. PhD Thesis, Edinburgh University, UK, 1988

[118] PROBERT, P.J., DJIAN, D., and HUOSHENG HU: 'Transputer architectures for sensing in a robot controller: formal methods for design', *Concurrency: Practice and Experience*, August 1991, **3**, (4), pp. 283–292

[119] PYLE, I.: 'Software engineers and the IEE', *Software Engineering Journal*, March 1986, **1**, (2), pp. 66–68

[120] RALSTON, T.J.: 'Preliminary report on the international study on industrial experience with formal methods', *in* 'COMPASS '92: 7th Annual Conference on Computer Assurance', Gaithersburg, Maryland, USA, 15–18 June 1992.

[121] RAVN, A.P., and RISCHEL, H.: 'Requirements capture for embedded real-time systems'. Proc. IMACS-MCTS Symposium, Lille, France, Volume 2, May 1991, pp. 147–152

[122] RAVN, A.P., and STAVRIDOU, V.: 'Project organisation', *in* RAVN, A.P. (Ed.): 'Embedded, Real-time Computing Systems', chapter 8, Volume I, ESPRIT BRA 3104, Provably Correct Systems, ProCoS, Draft Final Deliverable, October 1991

[123] READE, C., and FROOME, P.: 'Formal methods for reliability' *in* ROOK, P. (Ed.): 'Software Reliability Handbook', chapter 3 (Elsevier Applied Science, 1990) pp. 51–81

[124] REASON, J.: 'Human error' (Cambridge University Press, UK, 1990)

[125] 'Risk: analysis, perception and management'. The Royal Society, 6 Carlton House Terrace, London SW1Y 5AG, UK, 1992

[126] RUSHBY, J., and WHITEHURST, R.A.: 'Formal verification of AI software'. Contractor Report 181827, NASA Langley Research Center, Hampton, Virginia, USA, February 1989

[127] RUSHBY, J.: 'Formal specification and verification of a fault-masking and transient-recovery model for digital flight control systems'. Technical Report SRI-CSL-91-3, SRI International, Menlo Park, California, USA, January 1991 (Also available as NASA Contractor Report 4384.)

[128] RUSHBY, J., VON HENKE, F., and OWRE., S.: 'An introduction to formal specification and verification using EHDM'. Technical Report SRI-CSL-91-02, SRI International, Menlo Park, California, USA, February 1991

[129] RUSHBY, J., and VON HENKE, F.: 'Formal verification of algorithms for critical systems'. Proc. ACM SIGSOFT 91 Conference on Software for Critical Systems, *Software Engineering Notes*, ACM Press, December 1991, **16**, (5), pp. 1–15

[130] SCHOLEFIELD, D.J.: 'The formal development of real-time systems: a review'. Technical Report YCS 145, Dept. of Computer Science, University of York, UK, 1990

[131] SELBY, R.W., BASILI, V.R., and BAKER, F.T.: 'Cleanroom software development: an empirical evaluation', *IEEE Transactions on Software Engineering*, September 1987, **SE-13**, (9), pp. 1027–1037

[132] SENNETT, C.T.: 'High-integrity software' (Pitman Computer Systems Series, 1989)

[133] SHOSTAK, R.E., SCHWARTZ, R., MELLIAR-SMITH, P.M.: 'STP: a mechanized logic for specification and verification' *in* '6th International Conference on Automated Deduction (CADE-6)' (Springer-Verlag, *Lecture Notes in Computer Science*, 1982) **138**

[134] SMITH, C.L.: 'Digital control of industrial processes', *ACM Computing Surveys*, 1970, **2**, (3), pp. 211–241

[135] SMITH, D.J., and WOOD, K.B.: 'Engineering Quality Software: a review of current practices, standards and guidelines including new methods and development tools', 2nd edition (Elsevier Applied Science, 1989)

[136] SOMMERVILLE, I.: 'Software engineering', 3rd edition (Addison Wesley, 1989)

[137] 'Special issue on reliability', *IEEE Spectrum*, October 1981, **18**, (10)

[138] SPIVEY, J.M.: 'Specifying a real-time kernel', *IEEE Software*, September 1990, pp. 21–28

[139] SPIVEY, J.M.: 'The Z notation: a reference manual', 2nd edition (Prentice Hall International Series in Computer Science, 1992)

[140] SRIVAS, M., and BICKFORD, M.: 'Verification of the FtCayuga fault-tolerant microprocessor system, vol 1: a case study in theorem prover-based verification'. Contractor Report 4381, NASA Langley Research Centre, Hampton, Virginia, USA, July 1991 (Work performed by ORA corporation.)

[141] STEIN, R.M.: 'Safety by formal design', *BYTE*, August 1992, p. 157

[142] STEIN, R.M.: 'Software safety' *in* 'Real-time Multicomputer Software Systems', chapter 5 (Ellis-Horwood, 1992) pp. 109–133

[143] STEPNEY, S., BARDEN, R., and COOPER, D. (Eds.): 'Object orientation in Z' (Springer-Verlag, Workshops in Computing, 1992)

[144] SWADE, D.: 'Charles Babbage and his calculating engines' (Science Museum, London, UK, 1991)

[145] THOMAS, M.C.: 'The future of formal methods' *in* BOWEN, J.P. (Ed.): 'Proc. 3rd Annual Z Users Meeting', Oxford University Computing Laboratory, UK, December 1988, pp. 1–3

[146] THOMAS, M.C.: 'Development methods for trusted computer systems', *Formal Aspects of Computing*, 1989, **1**, pp. 5–18

[147] TIERNEY, M.: 'The evolution of Def Stan 00-55 and 00-56: an intensification of the "formal methods debate" in the UK'. Proc. Workshop on Policy Issues in Systems and Software Development, Science Policy Research Unit, Brighton, UK, July 1991

[148] TIERNEY, M.: 'Some implications of Def Stan 00-55 on the software engineering labour process in safety critical developments'. Research Centre for Social Sciences, Edinburgh University, 1991

[149] VON NEUMANN, J.: 'Probabilistic logics and synthesis of reliable organisms from unreliable components' *in* 'Collected Works', Volume 5 (Pergamon Press, 1961)

[150] WALDINGER, R.J., and STICKEL, M.E.: 'Proving properties of rule-based systems'. Proc. 7th Conference on Artificial Intelligence Applications, IEEE Computer Society, February 1991, pp. 81–88

[151] WALLACE, D.R., KUHN, D.R., and CHERNIAVSKY, J.C.: 'Report of the NIST workshop of standards for the assurance of high integrity software'. NIST Special Publication 500-190, Computer Systems Laboratory, National Institute of Standards and Technology, Gaithersburg, MD 20899, USA, August 1991 (Available from the Superintendent of Documents, Government, U.S. Printing Office, Washington, DC 20402, USA.)

[152] WALLACE, D.R., KUHN, D.R., and IPPOLITO, L.M.: 'An analysis of selected software safety standards', *IEEE AES Magazine*, August 1992, pp. 3–14

[153] WARD, W.T.: 'Calculating the real cost of software defects', *Hewlett-Packard Journal*, October 1991, pp. 55–58

[154] WEBB, J.T.: 'The role of verification and validation tools in the production of critical software', *in* INCE, D. (Ed.): 'Software Quality and Reliability: Tools and Methods', Unicorn Applied Info Technology reports 6, chapter 4, (Chapman & Hall, London, 1991) pp. 33–41,

[155] WENSLEY, J. *et al.* 'SIFT: design and analysis of a fault-tolerant computer for aircraft control', *Proc. IEEE*, 1978, **60**, (10), pp. 1240–1254

[156] WIRTH, N.: 'Towards a discipline of real-time programming', *Communications of the ACM*, August 1977, **20**, (8), pp. 577–583

[157] WICHMANN, B.A. (Ed.): 'Software in safety-related systems' (Wiley, 1992) Also published by BCS

[158] WRIGHT, C.L., and ZAWILSKI, A.J.: 'Existing and emerging standards for software safety'. The MITRE Corporation, Center for Advanced Aviation System Development, 7525 Colshire Drive, McLean, Virginia 22102-3481, USA, MP-91W00028, June 1991 (Presented at the IEEE Fourth Software Engineering Standards Application Workshop, San Diego, California, USA, 20–24 May 1991.)

[159] XILINX, Inc.: 'The programmable gate array data book'. San Jose, California, USA, 1991

[160] YOULL, D.P.: 'Study of the training and education needed in support of Def Stan 00-55'. Cranfield IT Institute Ltd, UK, September 1988 (Can also be found as an appendix of the April 1989 00-55 draft.)

[161] ZHOU ChaoChen, HOARE, C.A.R., and RAVN, A.P.: 'A calculus of durations', *Information Processing Letters*, 1991, **40**, (5), pp. 269–276

Hazard	Safety measure	Cost (in 1985)
Car crashes	Mandatory seat belts	\$500
Cervical cancer	Screening	\$75,000
Radiation	Reduce exposure & leaks	\$75,000,000
Toxic air	Reduce emissions	\$5,880,000,000

Table 1: Cost of saving a life.

Application	Notation	Specification	Verification	Machine Support
Aviation	STP/EHDM	•	•	•
	Spectool/Clio	•	•	•
Railway systems	B	•	•	•
Nuclear power plants	VDM	•		•
Medical systems	HP-SL & Z	•		
Ammunition control	VDM	•	•	•
Embedded micros	HOL	•	•	•

Table 2: Applications of formal methods to safety-critical systems.

Approach	Specification & Analysis	Develop-ment	Testing	Redevelop-ment	Validation & Correction	Total
Development without formal methods or diversity applied	·07	·12	·36	·48	·38	1·31
Development with formal methods and no diversity	·14	·12	·36	0	·38	1·0
Development with two channel diversity and no formal methods	·07	·24	·72	·96	0	1·99

Table 3: Comparison of cost-effectiveness of approaches at Rolls-Royce and Associates.

Project	**KNCSS**	**Test Hours**	**Defect Density**
This project	16.1	20.3	0.06
Project X	81.8	61.2	0.20

Table 4: Comparison of some Hewlett-Packard project metrics.

Country	Body	Sector	Name	FMs content	FMs mandated	Status (Jan. 1992)	Year
US	DoD	Defence	MIL-STD-882B MIL-STD-882C	No Likely	N/A ?	Standard Draft	1985 1992
US	RTCA	Aviation	DO-178A DO-178B	No Yes	N/A ?	Guideline Draft	1985 1992
Europe	IEC	Nuclear	IEC880	No	N/A	Standard	1986
UK	HSE	Generic	PES	No	N/A	Guideline	1987
Europe	IEC	Generic	IEC65A WG9 IEC65A 122	Yes Yes	No No	Draft Proposed	1989 1991
Europe	ESA	Space	PSS-05-0	Yes	Yes	Standard	1991
UK	MoD	Defence	00-55	Yes	Yes	Interim	1991
US	IEEE	Generic	P1228	No	No	Draft	1992
UK	RIA	Railway	–	Yes	Yes	Draft	1991
Canada	AECB	Nuclear	–	Yes	?	Draft	1991

Table 5: Summary of software-related standards.