

The Industrial Take-up of Formal Methods in Safety-Critical and Other Areas: A Perspective

Jonathan Bowen¹ and Victoria Stavridou²

¹ Oxford University Computing Laboratory, Programming Research Group,
11 Keble Road, Oxford OX1 3QD, UK. Email: <Jonathan.Bowen@comlab.ox.ac.uk>

² Department of Computer Science, Royal Holloway, University of London,
Egham, Surrey TW20 0EX, UK. Email: <victoria@dcs.rhbnc.ac.uk>

Abstract. Formal methods may be at the crossroads of acceptance by a wider industrial community. In order for the techniques to become widely used, the gap between theorists and practitioners must be bridged effectively. In particular, safety-critical systems offer an application area where formal methods may be engaged usefully to the benefit of all. This paper discusses some of the issues concerned with the general acceptance of formal methods and concludes with a summary of the current position and how the formal methods community could proceed to improve matters in the future.

“To err is human but to really foul things up requires a computer.”
Farmers’ Almanac for 1978 (1977) ‘Capsules of Wisdom’

1 Introduction

The software used in computers has become progressively more complex as the size of computers has increased and their price has decreased [36]. Unfortunately software development techniques have not kept pace with the rate of software production and improvements in hardware. Errors in software are renowned and software manufacturers have in general issued their products with outrageous disclaimers that would not be acceptable in any other more established industrial engineering sector.

It has been suggested that formal methods are a possible solution to help reduce errors in software. Sceptics claim that the methods are infeasible for any realistically sized problem. Sensible proponents recommend that they should be applied selectively where they can be used to advantage. More controversially, it has been claimed that formal methods, despite their apparent added complexity in the design process, can actually *reduce* the overall cost of software. The reasoning is that while the cost of the specification and design of the software is increased, this is a small part of the total cost, and time spent in testing and maintenance may be considerably reduced. If formal methods are used, many more errors should be eliminated earlier in the design process and subsequent changes should be easier because the software is better documented and understood.

2 Technology Transfer Problems

The following extract from the BBC television program *Arena* broadcast in the UK during October 1990 graphically illustrates the publicly demonstrated gap between much of the computing and electronics industry, and the formal methods community, in the context of safety-critical systems; these, arguably, have the most potential benefit to gain from the use of formal methods [4].

Narrator: [On Formal Methods] *“...this concentration on a relatively immature science has been criticized as impractical.”*

Phil Bennett, IEE: *“Well we do face the problem today that we are putting in ever increasing numbers of these systems which we need to assess. The engineers have to use what tools are available to them today and tools which they understand. Unfortunately the mathematical base of formal methods is such that most engineers that are in safety-critical systems do not have the familiarity to make full benefit of them.”*

Martyn Thomas, Chairman, Praxis plc: *“If you can't write down a mathematical description of the behaviour of the system you are designing then you don't understand it. If the mathematics is not advanced enough to support your ability to write it down, what it actually means is that there is no mechanism whereby you can write down precisely that behaviour. If that is the case, what are you doing entrusting people's lives to that system because by definition you don't understand how it's going to behave under all circumstances? ... The fact that we can build over-complex safety-critical systems is no excuse for doing so.”*

This repartee is typical not only of the substantial technology transfer problems, but also of the debate between the “reformist” (pro “real world”) and the “radical” (pro formal methods) camps in software engineering [42].

2.1 Misconceptions and barriers

Unfortunately formal methods is sometimes misunderstood and relevant terms are even misused in industry (at least, in the eyes of the formal methods community). For example, the following two alternative definitions for *formal specification* are taken from a glossary issued by the IEEE [25]:

1. *A specification written and approved in accordance with established standards.*
2. *A specification written in a formal notation, often for use in proof of correctness.*

The meaning of “formal notation” is not elaborated further in the glossary, although “proof of correctness” is defined in general terms.

Some confuse formal methods with “structured methods”. While research is underway to link the two and provide a formal basis to structured methods (e.g.,

see [26]), the two communities have, at least until now, been sharply divided apart from a few notable exceptions. Many so-called formal “methods” have concentrated on notations and/or tools and have not addressed how they should be slotted into existing industrial best practice. On the other hand, structured methods provide techniques for developing software from requirements to code, normally using diagrams to document the design. While the data structures are often well defined (and easily formalized), the relationships between these structures are often left more hazy and are only defined using informal text (natural language).

Industry has been understandably reluctant to use formal methods since they have been largely untried in practice. There are many methods being touted around the market place and formal methods are just one form of them. When trying out any of these new techniques for the first time, the cost of failure could be prohibitive and the initial cost of training is likely to be very high. For formal methods in particular, few engineers, programmers and managers currently have the skills to apply the techniques beneficially (although many have the ability).

Unfortunately, software adds so much complexity to a system that with today’s formal techniques and mechanical tools, it is intractable to analyze all but the simplest systems exhaustively. In addition, the normal concept of tolerance in engineering cannot be applied to software. Merely changing one bit in the object code of a program may have a catastrophic and unpredictable effect. However, software provides such versatility that it is the only viable means of developing many products.

Formal methods have been a topic of research for many years in the theoretical computer science community. However they are still a relatively novel concept for most people in the computing industry. While industrial research laboratories are investigating formal methods, there are not many examples of the use of formal methods in real commercial projects. 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. [20] is an excellent article that helps to dispel some of the unfounded notions and beliefs about formal methods.

Up until quite recently it has widely been considered infeasible to use formal techniques to verify software in an industrial setting. Now that a number of case studies and examples of real use are available, formal methods are becoming more acceptable in some industrial circles [19, 22, 24]. Some of the most notable of these are mentioned in [9], particularly those where a quantitative indication of the benefits gained have been published.

2.2 Modes of use

Formal methods may be characterized at a number of levels of usage and these provide different levels of assurance for the resulting software that is developed. This is sometimes misunderstood by antagonists (and even enthusiasts) who assume that using formal methods means that *everything* has to be proved correct.

In fact much current industrial use of formal methods involves no, or minimal, proofs [3].

At a basic level, formal methods may simply be used for a high-level specification of the system to be designed (e.g., using the Z notation). The next level of usage is to apply formal methods to the development process (e.g., VDM), using a set of rules or a design calculus that allows stepwise refinement of the operations and data structures in the specification to an efficiently executable program. 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.

Mechanical theorem provers such as HOL and the Boyer-Moore system have been used to verify significant implementations, 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 great improvements will need to be made in the usability of these tools before they can be widely accepted in the computing industry. Tools are now becoming commercially available (e.g., the B tool and Lambda) but there is still little interest in industry at large. Eventually commercial pressures should improve these and other similar 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’, whilst improving, are areas that require considerable further research and development effort.

2.3 Cost considerations

The prerequisite for industrial uptake of formal techniques is a formalism which can adequately deal with the pertinent aspects of computer-based systems. However, the existence of such a formalism is not sufficient; the relevant technology must also be able to address the problems of the industry by integrating with currently used techniques [44], and must do so in a way that is commercially advantageous.

It should be noted that despite the mathematical basis of formal methods, errors are still possible because of the fallibility of humans and, for mechanical verification, computers. However formal methods have been demonstrated to reduce errors (and even costs and time to market) if used appropriately [24, 30]. In general though, formal *development* does increase costs [10]. For example, at the 1992 Z Users Meeting Andrew Bradley of British Aerospace reported the following typical productivity figures for different development approaches in terms of lines of code (LOC) per man year:

Non-safety critical code	1400–1600 LOC/m.y.
Normal safety critical code	700–800 LOC/m.y.
Full formal development	200–400 LOC/m.y.

Even if the use of formal methods incurs higher development costs, this is unlikely to be the predominant factor. The critical considerations to a greater or

lesser extent (depending on market growth rates) are development speed and final product cost. Is it, therefore, evident that formal methods can deliver cheaper products rapidly? Given the current technology, the over-zealous use of formal methods can easily slow down rather than speed up the process, although the reverse is also possible if formal methods are used selectively. It is however the case that in specialized markets such as the high integrity sector, other factors such as product quality may be the overriding concern. A further consideration must be whether formal methods can enhance product quality, and even company prestige.

3 Industrial-scale Usage

As has previously been mentioned, the take up of formal methods is not yet great in industry, but their use has normally been successful when they have been applied appropriately [41]. Some companies have managed to specialize in providing formal methods expertise (e.g., CLInc in the US, ORA in Canada and Praxis in the UK), although such examples are exceptional. A recent international investigation of the use of formal methods in industry [13, 14] provides a view of the current situation by comparing some significant projects which have made serious use of such techniques.

[9] provides a survey of selected projects and companies that have used formal methods in the design of safety-critical systems and [1] gives an overall view of this industrial sector in the UK. In critical systems, reliability and safety are paramount. Extra cost involved in the use of formal methods is acceptable, and the use of mechanization for formal proofs may be worthwhile for critical sections of the software. In other cases, the total cost and time to market is of highest importance. For such projects, formal methods should be used more selectively, perhaps only using rigorous proofs or just specification alone. Formal documentation of key components may provide significant benefits to the development of many industrial software-based systems without excessive and sometimes demonstrably decreased overall cost (e.g., see [22, 24, 30]).

3.1 Application areas and techniques

Formal methods are applicable in a wide variety of contexts to both software and hardware. They are useful at a number of levels of abstraction in the development process from requirements capture, through to specification, design, coding, compilation and the underlying digital hardware itself. Some research projects are specification investigating the formal relationships between these different levels [5, 8] which are all important to avoid errors. An example of a suggested overall approach to project organization using formal methods is provided by [37].

The *Cleanroom* approach is a technique that could easily incorporate the use of existing formal notations to produce highly reliable software by means of non execution-based program development [16]. This technique has been applied

very successfully using rigorous software development techniques with a proven track record of reducing errors by a significant factor, in both safety-critical and non-critical applications. The programs are developed separately using informal (often just mental) proofs before they are certified (rather than tested). If too many errors are found, the process rather than the program must be changed. The pragmatic view is that real programs are too large to be formally proved correct, so they must be written correctly in the first place! The possibility of combining Cleanroom techniques and formal methods is now being investigated [34].

There is considerable research into object-oriented extensions of existing formal notations such as Z and VDM [40] and the subject is under active discussion in both communities. Object-oriented techniques have had considerable success in their take-up by industry, and such research may eventually lead to a practical method combining the two techniques. However there are currently a large number of different dialects and some rationalization needs to occur before industry is likely to embrace any of the notations to a large degree.

An important but often neglected 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 [7]. Formal specification alone has proved beneficial in practice in many cases [3]. Such use allows the possibility of formal development subsequently as experience is gained.

The *human-computer interface* (HCI) is an increasingly important component of most software-based systems. Errors often occur due to misunderstandings caused by poorly constructed interfaces [27]. Formalizing an HCI in a realistic and useful manner is a difficult task, but progress is being made in categorizing features of interfaces that may help to ensure their reliability in the future. There seems to be considerable scope for further research in this area, which also spans many other disparate disciplines, particularly with application to safety-critical systems where human errors can easily cause death and injury [21].

4 Motivation for the Use of Formal Methods

4.1 Standards

Up until relatively recently there have been few standards concerned specifically with formal notations and methods. Formal notations are eschewed in many software-related standards for describing *semantics*, although BNF-style descriptions are universally accepted for describing *syntax*. The case for the use of formal notations in standards is now mounting as formalisms become increasingly understood and accepted by the relevant readership [6]. Hopefully this will produce more precise and less ambiguous standards in the future, although there is still considerable debate on the subject and widely differing views across different countries [15]. Formal notations themselves have now reached the level of maturity that some of them are being standardized (e.g., LOTOS, VDM and Z).

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*). Many emerging safety-related standards are at the discussion stage [43]. A major impetus has already been provided in the UK by promulgation of the MoD interim standard 00-55 [2], which mandates the use of formal methods and languages with sound formal semantics.

It is important that standards should not be prescriptive, or that parts that are should be clearly separated and marked as such. Goals should be set and the onus should be on the software supplier that their methods 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. Some guidance is worthwhile, but is likely to date quickly. As a result, it may be best to include it as a separate document or appendix so that it can be updated more frequently to reflect the latest available techniques and best practice. For example, 00-55 includes a separate guidance section.

4.2 Legislation

Governmental 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 from 1st January 1993 [32]. 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. A maximum penalty of three months in jail or a large fine are possible. Suppliers will have to demonstrate that they are using best working practice, which could include, for example, the use of formal methods.

However, care should be taken in not overstating the effectiveness of formal methods. In particular, the term *formal proof* has been used quite loosely sometimes, and this has even led to litigation in the law courts over the VIPER microprocessor, although the case was ended before a court ruling was pronounced [29]. If extravagant claims are made, it is quite possible that a similar case could occur again. 00-55 differentiates between *formal proof* and *rigorous argument*, preferring the former, but sometimes accepting the latter with a correspondingly lower level of design assurance. Definitions in such standards could affect court rulings in the future.

4.3 Education and certification

Most modern comprehensive standard text books on software engineering now include a section on formal methods. Many computing science courses, especially

in Europe, are now including a significant portion of basic relevant mathematical training (e.g., discrete mathematics such as set theory and predicate logic). In this respect, education in the US seems to be lagging behind.

[35] discusses the accreditation of software engineers by profession 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 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 can be applied. This is gradually being alleviated as the necessary mathematics is being taught increasingly in computing science curricula. In the past has been necessary for companies to provide their own training or seek specialist help, although formal methods courses are now quite widely available from both industry and academia in some countries (e.g., for the UK, see [33]). It appears that Europe is leading the US and the rest of the world in this particular battle, and in the use of formal methods in general, so this may be a good sign for the long term development and reliability of software emanating from Europe.

Some standards and draft standards are now recognizing the problems and recommending that appropriate personnel should be used, especially 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, but there are possible drawbacks as well as benefits by introducing such a ‘closed shop’ since suitably able and qualified engineers may be inappropriately excluded (and vice versa).

4.4 Bridging the gap

Technology transfer is often fraught with difficulties and is inevitably – and rightly – a lengthy process. Problems at any stage can lead to overall failure [11]. A technology such as formal methods should be well established before it is applied, especially in critical applications where safety is paramount. Awareness of the benefits of formal methods must be publicized to a wide selection of both technical and non-technical people, especially outside the formal methods community (e.g., as in [39]), and the possibilities and limitations of the techniques available must be well understood by the relevant personnel to avoid costly mistakes.

Unfortunately, the rapid advances and reduction in cost of computers in recent years has meant that time is not on our side. However, formal techniques are now sufficiently advanced that they should be considered for selective use in software development, provided the problems of education can be overcome. It is likely that there will be a skills shortage in this area for the foreseeable future and significant difficulties remain to be overcome [12].

Software standards, especially those concerning safety, are likely to provide a motivating force for the use of formal methods, and it is vital that sensible and realistic approaches are suggested in emerging and future standards. 00-55 [2] seems to provide such an example at present and is recommended as guidance for other proposed standards in this area.

5 Conclusions

We conclude with a subjective account of the state of the art and current issues in the formal specification and verification of computer-based systems, both in software and hardware.

5.1 What we have today

Many specification notations and models. although there is scope for improvement, especially in the area of requirements capture. In the words of C.A.R. Hoare [23] models are like seeds scattered in the wind. Most will perish but others will root and flourish. The more seeds the better. A question often asked is “which is the best notation/system to use?” It seems to us that the choice of notation for specifying systems containing both hardware and software are often similar. Although trends come and go, in the end, the answer must relate to the characteristics of the specific product being developed and the background of the individuals involved.

Well developed theorem proving tools. The differences between the quality of such tools produced in the mid-1980s with their counterparts that are being developed today are very marked, particularly through the provision of much improved user interfaces. However further improvements will still need to be made for widespread acceptance in mainstream industry and today’s research prototypes will take significant time, resources and, importantly, appropriate marketing to become tomorrow’s industrial-strength tools.

Some impressive verification results. Proving the correctness of simple but realistic systems based on small microprocessors is no longer an issue as the work on SACEM [19] and at CLInc [18, 31], for example, illustrates. SACEM is notable for its size, involving around 80 man years effort for a system that is designed for actual use. The work at CLInc is impressive since it considers the linked verification of various levels of abstraction of the system for both software and hardware on a verified microprocessor that has actually been fabricated. However, systems based on widely used programming languages and microprocessors are still problematic and we can only deal with some of their aspects.

Despite these achievements, there are some very real problems hindering widespread use of formal methods. Some of these are articulated elsewhere [13, 14]. We summarize the issues by pointing out that formal methods deployment in the large remains an esoteric, risky and potentially very costly activity whose impact on processes and products of the computer industry

has never been properly evaluated. A scientifically sound and objective study of these issues is sorely missing.

5.2 Issues for the future

We must distinguish between the issues relating to technology and those relating to research. By technological issues we mean the problems concerned with 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 microprocessor-based system in a formal methods laboratory does not imply that a multi-billion dollar manufacturer will have its designers use a theorem prover. Suitable technology must be produced before the process is enabled and as with any other endeavour the user (not the research) community must be the driving force.

Technology issues

Industrial quality tools which are well engineered and have suitable user interfaces. As mentioned earlier, the user interface issue has been the prime beneficiary of theorem proving developments from 1985 onwards. Much remains to be done in improving the efficacy of theorem proving tools by, for instance, incorporating fast decision procedures (such as BDDs) and graph reduction packages where appropriate. Producing industrial quality software of any kind requires levels of funding which are not normally available to research workers. On the other hand, before industry can be convinced to sponsor the development of quality tools they must be persuaded that they have a use for them; but they will not consider such tools useful unless they are well engineered and hence there is a “chicken and egg” situation. The answer must be a level of synergy between the two camps.

Interface to best existing practice. Formal methods are a supplemental *not* a replacement technology. They must therefore work in harmony with existing methods and tools, the purpose of the integration being to strengthen rather than replace existing technology. Happily, there has been a definite trend in this direction by many verification workers. This is happening in two complementary ways. Firstly, the rôle of formal methods as a supplemental technology is being explored by a number of researchers. Secondly, efforts to interface to current technology are exemplified by some investigations into the semantics of industrially used notations and the subsequent provision of formal reasoning frameworks.

So, the verification community has, in the past 5 years or so, gone some way in producing results which enable the transformation of research into technology although much remains to be done in terms of technology transfer. Meetings with a good mix of industry and academia are to be encouraged, such as SAFECOMP [17] and those organized by the UK Safety-Critical Systems Club [38].

Research issues

A concern, separate from ways of transforming research results into technology fit for purpose, is the direction of further research in verification. We believe that much remains to be done in terms of notations and models. Research projects such as the European ESPRIT **ProCoS** project [5] and the UK **safemos** project [8] are attempting to address such issues.

However, a most pressing question is the shape of the future theorem proving activity itself, whether it relates to software, hardware or both. We strongly believe that theorem proving methods have a great deal in common with software engineering and future breakthroughs in the area are likely to originate from recognizing and exploiting this fact.

In our view *proofs are programs* and the evolution of theorem proving closely resembles the evolution of programming practice. Currently, complete low-level proofs look and read like machine code programs with all the consequences for the process of producing such proofs. Proof tactics that largely characterize the writing and execution of proofs closely resemble assembler macros. It seems to us that theorem proving will have to make the difficult transition from an activity of the select few to an engineering discipline in much the same way that software engineering has evolved from the days of machine/assembly coding of the 1950s and the 1960s.

The crucial prerequisite of the evolution of software engineering has been the concept of *abstraction*. Abstraction provided by operating systems, high level programming languages and their compilers, specification languages and the associated machinery such as editors, debuggers, configuration managers and so on. We believe that discovering ways of harnessing abstraction in theorem proving will be exciting and empowering.

Liskov and Guttag argue that abstraction is 2-dimensional [28]. Firstly, it can be obtained via *parameterization*, e.g., the use of procedure parameters so that a piece of code is usable on large, perhaps polymorphic data sets. Secondly, abstraction by *specification* is used to hide details by concentrating on what needs to be achieved by a piece of code rather than describing ways of achieving what is required. Abstraction by parameterization is, to some degree, already exploited in theorem proving through the logical framework ideas of including the object logic as a parameter of a theorem prover rather than “hard-wiring” it into the system. Abstraction by specification, however, is largely unresearched. We believe that such abstraction is urgently in need of research and holds great promise for the future, if the experience of software engineering is to be heeded. Proof specification languages and proof refinement, automatic or not, are aims well worth striving towards. Once these methods have been understood and developed, the theorem provers of today will become the proof compilers of tomorrow.

Acknowledgements

Jonathan Bowen is funded by the collaborative UK Information Engineering Directorate **safemos** project (IED3/1/1036).

References

1. *Safety related computer controlled systems market study*. A review for the Department of Trade and Industry by Coopers & Lybrand in association with SRD-AEA Technology and Benchmark Research (HMSO, London, 1992)
2. *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)
3. Barden, R., Stepney, S., 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
4. Barroca, L., McDermid, J.: Formal methods: use and relevance for the development of safety critical systems. *The Computer Journal* **35** 6 (December 1992)
5. Bjørner, D.: Trusted computing systems: the ProCoS experience. *Proc. 14th International Conference on Software Engineering (ICSE)*, Melbourne, Australia (11–14 May 1992)
6. Blyth, D., Bolddyreff, C., Ruggles, C., Tetteh-Lartey, N.: The case for formal methods in standards. *IEEE Software* (September 1990) 65–67
7. Bowen, J.P.: Formal specification in Z as a design and documentation tool. *Second IEE/BCS Conference, Software Engineering 88*, Conference Publication No. 290 (July 1988) pp. 164–168
8. Bowen, J.P.: Towards verified systems (Elsevier, Real-time Safety-critical Systems Series, 1993) In preparation
9. Bowen, J.P., Stavridou, V.: Safety-critical systems, formal methods and standards. Technical Report PRG-TR-5-92, Programming Research Group, Oxford University Computing Laboratory, UK (1992) Revised version to appear in the *Software Engineering Journal*
10. Bowen, J.P., Stavridou, V.: Formal methods and software safety. In [17] (1992) pp. 93–98
11. Buxton, J.N., Malcolm, R.: Software technology transfer. *Software Engineering Journal* **6** 1 (January 1991) 17–23
12. 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)
13. Craigen, D., Gerhart, S., Ralston, T.J.: An international survey of industrial applications of formal methods. Atomic Energy Control Board of Canada, U.S. National Institute of Standards and Technology, and U.S. Naval Research Laboratories (1993) To appear
14. Craigen, D., Gerhart, S., Ralston, T.J.: Formal methods reality check: industrial usage. In *Formal Methods Europe Symposium (FME'93)* (Springer-Verlag, LNCS, 1993) In this volume
15. Deransart, P.: Prolog standardisation: the usefulness of a formal specification, on `comp.lang.prolog`, `comp.specification` and `comp.software-eng` electronic USENET newsgroups (October 1992)
16. Dyer, M.: *The Cleanroom approach to quality software development* (Wiley Series in Software Engineering Practice, 1992)
17. 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)

18. Good, D.I., Young, W.D.: Mathematical methods for digital system development. In Prehn, S., Toetenel, W.J. (eds.): *VDM '91, Formal Software Development Methods*, Volume 2: Tutorials (Springer-Verlag, LNCS 552, 1991) pp. 406–430
19. Guiho, G., Hennebert, C.: SACEM software validation. *Proc. 12th International Conference on Software Engineering (ICSE)* (IEEE Computer Society Press, March 1990) pp. 186–191
20. Hall, J.A.: Seven myths of formal methods. *IEEE Software* (September 1990) 11–19
21. 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
22. Hill, J.V.: Software development methods in practice. *Proc. COMPASS '91: 6th Annual Conference on Computer Assurance* (1991)
23. Hoare, C.A.R.: Let's make models. In Baeten, J.C.M., Klop, J.W. (eds.): *Proc. CONCUR '90* (Springer-Verlag, LNCS 458, 1990)
24. Houston, I., King, S.: CICS project report: experiences and results from the use of Z in IBM. In Prehn, S., Toetenel, W.J. (eds.): *VDM '91, Formal Software Development Methods* (Springer-Verlag, LNCS 551, 1991) pp. 588–603
25. IEEE standard glossary of software engineering terminology. In *IEEE Software Engineering Standards Collection* (Elsevier Applied Science, 1991)
26. Josephs, M.B., Redmund-Pyle, D.: Entity-relationship models expressed in Z: a synthesis of structured and formal methods, Technical Report PRG-TR-20-91, Programming Research Group, Oxford University Computing Laboratory, UK (July 1991)
27. Learmount, D.: Airline safety review: human factors. *Flight International* 142 4238 (22–28 July 1992) 30–33
28. Liskov, B., Guttag, J.: *Abstraction and Specification in Program Development* (MIT Press, 1986)
29. MacKenzie, D.: Computers, formal proof, and the law courts. *Notices of the American Mathematical Society* 39 9 (November 1992) 1066–1069
30. May, D., Barrett, G., Shepherd, D.: Designing chips that work. In Hoare, C.A.R., Gordon, M.J.C. (eds.): *Mechanized reasoning and hardware design* (Prentice Hall International Series in Computer Science, 1992) pp. 3–19
31. Moore, J.S. *et al.*, Special issue on system verification. *Journal of Automated Reasoning* 5 4 (1989) 409–530
32. Neesham, C.: Safe conduct. *Computing* (12 November 1992) 18–20
33. Nicholls, J.E.: A survey of Z courses in the UK. In Nicholls, J.E. (ed.), *Z User Workshop, Oxford 1990* (Springer-Verlag, Workshops in Computing, 1991) pp. 343–350
34. Normington, G.: Cleanroom and Z. In Bowen, J.P., Nicholls, J.E. (eds.), *Z User Workshop, London 1992* (Springer-Verlag, Workshops in Computing, 1993) To appear
35. Pyle, I.: Software engineers and the IEE. *Software Engineering Journal* 1 2 (March 1986) 66–68
36. Potocki de Montalk, J.P.: Computer software in civil aircraft. *Microprocessors and Microsystems*. In Cullyer, W.J. (ed.): Special issue on safety critical systems (1993) To appear
37. Ravn, A.P., Stavridou, V.: Project organisation. In Bjørner, D., Langmaack, H., Hoare, C.A.R.: *Provably Correct Systems*, chapter 9, part 1, ESPRIT BRA 3104 ProCoS Technical Report (1992) Available from Department of Computer Science, DTH, Lyngby, Denmark

38. Redmill, F., Anderson, T.: Safety-critical systems – current issues, techniques and standards (Chapman and Hall, 1993)
39. Stein, R.M.: Safety by formal design. *BYTE* (August 1992) p. 157
40. Stepney, S., Barden, R., Cooper, D. (eds.): Object orientation in Z (Springer-Verlag, Workshops in Computing, 1992)
41. Thomas, M.C.: The industrial use of formal methods. *Microprocessors and Microsystems*. In Cullyer, W.J. (ed.): Special issue on safety critical systems (1993) To appear
42. 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)
43. Wallace, D.R., Kuhn, D.R., Ippolito, L.M.: An analysis of selected software safety standards. *IEEE AES Magazine* (August 1992) 3–14
44. Wing, J.M., Zaremski, A.M.: Unintrusive ways to integrate formal specifications in practice. In Prehn, S., Toetenel, W.J. (eds.), *VDM '91, Formal Software Development Methods* (Springer-Verlag, LNCS 551, 1991) pp. 547–569