Seven More Myths of Formal Methods:
Dispelling Industrial Prejudices

Jonathan P. Bowen[1] and Michael G. Hinchey[2]

[1] Oxford University Computing Laboratory, Programming Research Group
Wolfson Building, Parks Road, Oxford OX1 3QD, UK.
Email: Jonathan.Bowen@comlab.ox.ac.uk
[2] University of Cambridge Computer Laboratory
New Museums Site, Pembroke Street, Cambridge CB2 3QG, UK.
Email: Mike.Hinchey@cl.cam.ac.uk

Abstract. For whatever reason, formal methods remain one of the more contentious techniques in industrial software engineering. Despite some improvement in the uptake of formal methods, it is still the case that the vast majority of potential users of formal methods fail to become actual users. A paper by Hall in 1990 [31] examined a number of 'myths' concerning formal methods, assumed by some to be valid. This paper considers a few more beliefs held by many and presents some counter examples.

1 Introduction

Formal Methods continue to grow in popularity; growing numbers of delegates at conferences such as FME and ZUM are indicative of this. Unfortunately, as interest in formal methods increases, the number of misconceptions regarding formal methods continues to grow in tandem. While formal methods have been employed, to some extent, for over a quarter of a century, there are still very few people who understand *exactly* what formal methods are, and how they are applied in practice [3]. Many people completely misunderstand what constitutes a formal method, and how formal methods have been successfully employed in the development of complex systems. Of great concern is the fact that we must place many professional system developers into that latter category.

2 Hall's Original Seven Myths

In a seminal article [31], Hall highlights seven popular misconceptions, or 'myths' as he calls them, of formal methods, and attempts to dispel these by means of an example. Regretfully, four years later, these and other misconceptions still abound. Formal methods are unfortunately the subject of extreme hyperbole or deep criticism in many of the 'popular press' science journals. From the claims that the authors of such articles make, it is quite clear that they have little or no understanding of what formal methods are, nor how they have been applied in industry.

For example, an article in *The Independent on Sunday* of 13th October 1991 focused on fears for the reliability of the software running the Sizewell-B Nuclear

Reactor. This was one of the more objective articles run by the popular press, and proposed 'backfitting' the Sizewell-B software with formal methods as has been done at Darlington [45]; indeed work on this has now started [2]. An article published at the same time in *New Scientist* also advocated the use of formal methods in the Sizewell system. This, however, was a primary example of the over-enthusiasm of so-called 'experts'. It suggested that the Sizewell system could only be reliable if formal methods were employed in its development and that formal methods would eliminate all bugs and guarantee the safety of the system.

Even technical journals are not exempt; Barwise [4] reports on the controversy of proofs of correctness arising in fora such as *Communications of the ACM*. He concludes that mathematical proofs of computer systems are of limited applicability because of the gulf between the model derived and reality. Barwise sees this 'vision' as the main contribution of his paper. But, no formal methods specialist would ever claim that a system was absolutely correct, but rather that it was correct *with respect to its specification*. Such misunderstandings of formal methods are quite typical amongst their fervent critics.

Indeed even basic terms such as 'formal specification' are likely to be confused [15]. A search of the abbreviation CSP in an on-line acronym database gave the answers *Commercial Subroutine Package*, *CompuCom Speed Protocol* and *Control Switching Point*, but not the name *Communicating Sequential Processes* which would spring to the minds of many in the formal methods community.[3] Besides ambiguity in the basic terminology, the formal notations themselves can of course be confusing to practitioners not trained in their use and in general it is easier to ignore them than to investigate them further [17].

Myths that formal methods can guarantee perfect software and eliminate the need for testing (Myth 1 in Hall's paper) are not only ludicrous, but can have serious ramifications in system development if naïve users of formal methods take them seriously. Claims that formal methods are all about proving programs correct (Myth 2 in Hall's paper) and are only useful in safety-critical systems (Myth 3), while untrue, are not quite so detrimental, and a number of successful applications in non safety-critical domains have helped to clarify these points (see [35] for examples).

The derivation of a number of simple formal specifications of quite complex problems, and the successful development of a number of formal methods projects under budget have served to dispel the myths that the application of formal methods requires highly trained mathematicians (Myth 4) and increases development costs (Myth 5). The successful participation of end-users and other non-specialists in system development with formal methods has ruled out the myth that formal methods are unacceptable to users (Myth 6), while the successful application of formal methods to a number of large-scale complex systems, many of which have received much media attention, should put an end to beliefs that formal methods are not used on real large-scale systems (Myth 7).

[3] A search for VDM *did* reveal the term *Vienna Development Method*, but also *Virtual DOS Machine* and *Virtual Device Metafile* which may or may not be desirable bedfellows!

Many non-formalists seem to believe that formal methods are merely an academic exercise, a form of mental masturbation for academics that has no relation to real-world problems. Highly publicized accounts of the application of formal methods to a number of well-known systems, such as Sizewell-B [2], CICS [36], the Darlington Nuclear Facility [45] and Airbus [46], have helped to bring the industrial application of formal methods to a wider audience.

3 Seven More Myths

Many of Hall's myths were, and to a certain extent still are, propagated by the media, and are myths held by the public and the computer science community at large, rather than by specialist system developers. It is our concern, however, that many other myths are still being propagated, and more alarmingly, are receiving a certain degree of tacit acceptance from the system development community. We hope to dispel many of those myths here, by reference to a number of real-life industrial applications of various formal methods which have proven to be generally successful. Many of the examples cited here are discussed in greater detail in [35]. We include a significant bibliography of references in this paper to allow readers to follow up on any of the large number of topics covered in outline here if they so wish.

Myth 1. *Formal Methods delay the development process.*

A number of formal methods projects have run notoriously over schedule. The assumption that this is inherent in the nature of formal methods is a rather irrational deduction. Certainly these projects were not delayed due to the lack of ability of the formal methods specialists, but rather a lack of experience in determining how long development *should* take. That is to say, the projects were not necessarily delayed, but development time was severely underestimated.

Project cost estimation is a major headache for any development team. Determining project development time is equally difficult (in fact, the two are inevitably intertwined). A number of models have been developed to cover cost and development time estimation. Perhaps the most famous is Boehm's COCOMO model [5], which weights various factors according to the historical results of system development within the organization.

Here we have the crux of the problem. Any successful model of cost and development-time estimation must be based on historical information and details such as levels of experience, familiarity with the problem, etc. Even with traditional development methods, such information might not be available. Using formal development techniques historical information is likely to be even more scarce, as we have not yet applied formal methods to a sufficient number of projects on which to base trends and observations. Surveys of formal development [20, 21] and a highlighting of successes, failures, hindrances, etc., will eventually provide us with the levels of information we require.

Many of the much publicized formal methods projects are in very specialized domains, and domains that are unlikely to be addressed on a very regular basis.

As such, such data is of limited use; comparisons with more conventional developments [25, 32] and applications in more process control-like domains [19, 22, 47] are likely to provide more useful data.

In addition, working in such unfamiliar domains would naturally be expected to greatly increase the development time (if one follows a model à la COCOMO) as would working with methods that were (then) still pretty much in their infancy, with little or no tool support.

We draw the reader's attention to some very successful formal methods projects whereby the use of such methods reduced development time significantly. We include here the 12 month saving on the development of the Inmos T800 floating-point unit chip, and the application of Z and B to IBM's CICS system [36]. Both of these projects were independently audited and these claims to reduced lead-times over conventional development methods were confirmed.

Myth 2. *Formal Methods are not supported by tools.*

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 (e.g., using B [1] or RAISE tools [49]). Hand proofs or design inevitably lead to human errors occurring for all but the simplest systems.

Just as in the late '70s and early '80s, when CASE (Computer-Aided Software Engineering) and CASP (Computer-Aided Structured Programming) tools were seen as a means of increasing programmer productivity and reducing programming 'bugs', tool support is now seen as a means of increasing productivity and accuracy in formal development. Most of the projects discussed in [35], for example, place great emphasis on tool support. This is by no means coincidental, but rather follows a trend, which it is expected will eventually result in integrated workbenches to support formal specification, just as CASE workbenches support system development using more traditional structured methods [21].

A number of basic tools are now widely available, many of them in the public domain. For example, ZTC is a public-domain PC and Unix-based type-checking system for Z, and the commercialized *f*UZZ type-checker also runs under Unix. More integrated packages that support typesetting and integrity checking of specifications include Logica Cambridge's Formaliser, Imperial Software Technology's Zola, which also incorporates a tactical proof system, and CADiZ, a suite of tools for Z from York Software Engineering, which has recently been extended to support refinement to Ada code. FDR from Formal Systems Europe is a model-checker and refinement-checker for CSP. ProofPower, a tool available from ICL, uses Higher-Order Logic to support specification and verification in Z.

Perhaps motivated by the ProofPower approach, much attention has focused recently on tailoring various 'generic' theorem provers for use with model-based specification languages such as Z. An implementation in OBJ [41] seems to be too slow, but particular successes have been reported with HOL [12] and EVES [52].

We expect that in the future more emphasis will be placed on IFDSEs (In-

tegrated Formal Development Support Environments), which will support most stages of formal development. Such toolkits will be integrated in that, like IPSEs (Integrated Programming Support Environments), they will support version control and configuration management, and facilitate more harmonious developments by addressing all of the development process activities, and development by larger teams. Such IFDSEs do not as yet exist, but a number of toolkits certainly represent steps in the right direction.

IFAD's VDM-SL Toolbox is a set of tools which supports formal development in draft standard VDM-SL. As one might expect, standard type-checkers and static semantics checkers are supported. An interpreter supports all of the executable constructs of VDM-SL allowing a form of animation and specification 'testing'; the executed specifications may be debugged using an integrated debugger, and testing information is automatically generated.

The B-Toolkit from B-Core (UK) Ltd., is a set of integrated tools which augments Abrial's B-Method for formal software development by addressing industrial needs in the development process. Many believe B and the B-Method to be representative of the next generation of formal methods; if this is true, then the B-Toolkit, and other similar such toolkits, will certainly form the basis of future IFDSEs.

Myth 3. *Formal Methods mean forsaking traditional engineering design methods.*

One of the major criticisms of formal methods is that they are not so much 'methods' as formal systems. While formal methods support some of the design principles of more traditional methods, such as top-down design and stepwise refinement, there is very little emphasis on an underlying model that encompasses each of the stages of the system development life cycle, nor any guidance as to how development should proceed.

Structured development methods, using a model of development such as Boehm's 'spiral' model [6], on the other hand, generally support all stages of the system life cycle from requirements elicitation through to post-implementation maintenance. Their underlying models, in general, recognize the iterative nature of system development, and that system development is not a straightforward process as exemplified in, for example, Royce's 'waterfall' model [51]. Yet, in many senses, many formal development methods assume that specification is followed by design and implementation in strict sequence. This is an unrealistic view of software development, and every developer of complex systems has experienced the need to revisit both system requirements and the system specification at much later stages in development.

While Hall [31] disputes the myths that a high degree of mathematical ability is required to be comfortable with formal methods, and that formal methods are unacceptable to users, more traditional design methods do indeed excel at requirements elicitation and interaction with system procurers. They offer notations that can be understood by non-specialists and which can be offered as the basis for a contract.

Indeed, instead of formal methods replacing traditional engineering design

methods, a major area for current and future research is the integration of structured and formal methods. Such an integration leads to a 'true' method of development that fully supports the software life cycle, while admitting the use of more formal techniques at the specification and design phases, supporting refinement to executable code, and proof of properties.

Approaches to method integration vary from running both structured and formal methods in parallel, to formally specifying transformations from the notations of structured methods to formal specification languages. Much success has been reported using the former technique [24, 40]. The problem is however that as the two development methods are being addressed by different personnel, the likelihood that the benefits of the approach will be highlighted is low. In many cases, the two development teams do not adequately interact.

More integrated approaches include the translation of SSADM into Z, as part of the SAZ project [48], the integration of Yourdon and Z in a more formalized manner [53, 54], and the integration of various structured notations with VDM [39] and CSP [50]. These all augur much potential, but unlike the parallel approach have yet to be applied to realistic systems.

Myth 4. *Formal Methods only apply to software.*

Formal methods can equally well be applied to hardware design as to software development [37]. Indeed, this is one of the motivations of the HOL theorem prover which was used to verify parts of the Viper microprocessor. Other theorem proving systems which have been applied to the verification of hardware include the Boyer-Moore, Esterel, HOL, Nuprl, 2OBJ, Occam transformation system and Veritas proof tools. Model checking is also important in the checking of hardware designs if the state space is sufficiently small to make this feasible. However techniques such as Binary Decision Diagrams (BDDs) allow impressively large numbers of states to be handled.

Perhaps the most convincing and complete hardware verification exercise is the FM9001 microprocessor produced by Computational Logic Inc. in the US, and which has been verified down to a gate level netlist representation using the Boyer-Moore theorem prover. Two examples of real industrial use are provided by Inmos. The T800 Transputer floating-point unit has been verified by starting with a formalized Z specification of the IEEE floating-point standard, and using the Occam Transformation System to transform a high level program to the low level microcode by means of proven algebraic laws. More recently, parts of the new T9000 Transputer pipeline architecture have been formalized using CSP and checked for correctness. [37] contains a number of invited papers written by experts in the field and covers the applications outlined here in more detail.

A more recent approach to the development of hardware is *hardware compilation*. This allows a high-level program to be compiled directly into a *netlist* of simple components such as gates and latches together with there interconnections. The technology of *Field Programmable Gate Arrays* (FPGAs) allows this process to be undertaken entirely as a software process if required (which is particularly useful for rapid-prototyping) since these devices allow the circuit to be configured according to the contents of a static RAM within the chip.

It is possible to prove the compilation process itself correct [34]. In this case the hardware compiled each time need not be separately proven correct, thus reducing the proof burden considerably.

In the future, such an approach could allow the possibility of provably correct combined hardware/software co-design. A unified proof framework would facilitate the exploration of design trade-offs and interactions between hardware and software in a formal manner.

Myth 5. *Formal Methods are not required.*

We have all heard the argument that formal methods are not required. This is a mistruth; while there are occasions where formal methods are in a sense 'over-kill', there are situations where they are very desirable. In fact, the use of formal methods is recommended in any system where the issue of correctness is of concern.

This clearly applies to safety-critical and security-critical systems, but equally to systems which are not classified in these terms, but where one needs, or wishes, to ensure that the system operates correctly. (See for example [43] which presents the formal specification of an algorithm to determine the result in a single transferable voting system.) There are occasions however where formal methods are not only desirable, but positively required. A number of standards bodies have not only used formal specification languages in making their own standards unambiguous [58], but have mandated or strongly recommended the use of formal methods in certain classes of applications [7, 16].

The *International Electrotechnical Commission* specifically mentions a number of formal methods (CCS, CSP, HOL, LOTOS, OBJ, VDM, Z) and temporal logic in the development of safety-critical systems. The *European Space Agency* suggests that VDM or Z, augmented with natural language descriptions, should be used for specifying the requirements of safety-critical systems. It also advocates proof of correctness, a review process, and the use of formal proof in advance of testing.

The UK Ministry of Defence (MoD) draft Interim Defence Standards 00-55 and 00-56 mandate the extensive use of formal methods. The requirements of Standard 00-55 include the use of a formal notation in the specification of safety-critical components, and an analysis of such components for consistency and completeness. All safety-critical software must also be validated and verified; this includes formal proof and rigorous (but informal) correctness proofs, as well as more conventional static and dynamic analysis. Standard 00-56 deals with the classification and hazard analysis of the software and electronic components of defence equipment, and also mandates the use of formal methods.

The Atomic Energy Control Board (AECB) in Canada has commissioned a proposed standard for software for computers in the safety systems of nuclear power stations in conjunction with David Parnas at McMaster University. Standards and procedures developed by Canadian licensees mandate the use of formal methods, and together with 00-55 are still some of the few to go so far at the moment.

Whether or not one believes that formal methods are necessary in system

development, one cannot deny that they are indeed *required* in certain classes of applications, and are likely to be required in an increasing number of cases in the future [7].

Myth 6. *Formal Methods are not supported.*

Formal methods have been under development since the mid-1960s. But it is in the last decade that significant developments have evolved, and over the last few years interest in formal methods has grown phenomenally. Along with 'object-orientation' and a few other keywords, it has quickly become one of the great 'buzz-words' in the computer industry.

Many formal languages have been extended to support particular needs, with the addition of useful (though sometimes unsound) operators and data structures, as well as module structures and object-oriented concepts. There is a certain degree of 'trade-off' between the expressiveness of a language and the levels of abstraction that it supports [57]. Making a language more expressive does indeed facilitate briefer and more elegant specifications, but can make reasoning more difficult.

LOTOS was standardized in 1989 (ISO 8807), and draft international standards for both VDM and Z have been proposed under ISO/IEC JTC1/SC22 [7]. The X3J21 Technical Committee on Formal Description Techniques (FDT) is overseeing the VDM and Z standardization process and is also interested in future developments such as object-oriented extensions to FDTs and their possible standardization.

Obviously, a standard is pointless if it does not reflect the opinions of active users, and the developments that have evolved in formal methods. There are now a number of outlets for practitioners to discuss draft standards, and to seek advice and solutions to problems and difficulties from other practitioners. Chief among these outlets are various (especially electronic) distribution lists, such as the Z FORUM (contact `zforum-request@comlab.ox.ac.uk` or see the gatewayed `comp.specification.z` newsgroup) and the recently established VDM FORUM (contact `vdm-forum-request@mailbase.ac.uk`). A Larch interest group (contact `larch-interest-request@src.dec.com`) and a HOL information list (contact `info-hol-request@lal.cs.byu.edu`) are also in operation.

Electronically accessible anonymous FTP archives for Z (including an on-line and regularly revised comprehensive bibliography [8]) and other formal methods exist on the global Internet computer network. The global World Wide Web (WWW) electronic hypertext system also provides support for formal methods. A useful starting point is the following WWW page which provides pointers to other electronic archives concerned with formal methods throughout the world, including substantial publicly accessible tools such as HOL [28], the Larch Prover (LP) [30], Nqthm [18], OBJ [27] and PVS [44] for downloading on the network:

`http://www.comlab.ox.ac.uk/archive/formal-methods.html`

The proceedings of various symposia and workshops offer invaluable reading on current developments in formal methods; many of these (e.g., [13, 59]) are

available in the Springer Verlag *Lecture Notes in Computer Science* and *Workshops in Computing* series.

Formal methods are not quite so popular in the US, although they are gaining momentum there. There is as yet no regular conference in the US devoted to formal methods, but perhaps the forthcoming *Workshop on Industrial-strength Formal specification Techniques* (WIFT) represents a step in that direction.

Again the main journals and publications devoted to formal methods are based in Europe, and the UK specifically. These include *Formal Aspects of Computing*, *Formal Methods in System Design* and the *FACS Europe* newsletter, amongst others. *The Computer Journal*, *Software Engineering Journal* and *Information and Software Technology* regularly publish articles on, or related to, formal methods, and have run or plan a number of special issues on the subject.

In the US, as far as the authors are aware, there are no journals devoted specifically to formal methods, although some of the highly respected journals, such as *IEEE Transactions on Software Engineering* and the *Journal of the ACM*, and the popular periodicals such as *IEEE Computer*, *IEEE Software* and the *Communications of the ACM* regularly publish relevant articles. IEEE *TSE*, *Computer* and *Software* ran very successful coordinated special issues on formal methods in 1990. More recently, in January 1994 an *IEEE Software* special issue on safety-critical systems also devoted a not inconsiderable amount of attention to formal methods [38], as has a newly launched journal in this area entitled *High Integrity Systems*.

Formal methods (in particular Z, VDM, CSP and CCS) are taught in most UK undergraduate computer science courses. Although still quite uncommon in the US, an NSF-sponsored workshop aims to establish a curriculum for teaching formal methods in US undergraduate programmes. One would hope that this will help to establish formal methods as a regular component of US university curricula. A number of industrially-based courses are also available, and in general can be tailored to the client organization's needs.

Once upon a time, formal development might have been a lone activity, but certainly one can no longer argue that formal methods are not supported.

Myth 7. *Formal Methods people always use Formal Methods.*

There is widespread belief that the proponents of formal methods apply formal methods in all aspects of system development. This could not be further from the truth. Even the most fervent supporters of formal methods must recognize that there are certain aspects of system development for which formal methods are just not as good as other approaches.

In user-interface (UI) design, for example, it is very difficult to formalize exactly the requirements of the human-computer interaction. The appropriateness of a UI is a very subjective matter, and not really amenable to formal investigation. Although there have been a number of (somewhat successful) approaches to the formal specification of UIs [23], in general it is accepted that UI conformance testing lies in the domain of informal reasoning.

There are many other areas where, although possible, formalization is just not practical from a resource, time, or financial aspect. Most successful formal

methods projects involve the application of formal methods to critical portions of system development. Only rarely are formal methods, and formal methods alone, applied to all aspects of system development. Even within the CICS project [36], only about a tenth of the entire system was actually subjected to formal techniques (although this still involved 100,000s of lines of code and 1000s of pages of specifications). With appropriate apologies to Einstein for the following maxim:

System development should be as formal as possible, but not more formal.

What is perhaps surprising is that many tools to support formal development have not been developed using formal techniques. Formal methods have indeed been applied to the development of a number of support tools for conventional development methods, such as the SSADM CASE tool described by Hall [31]. They have also been used as part of the (re)development process in a reverse engineering and analysis tool-set for COBOL at Lloyd's Register [10]. In addition, they have been successfully employed in defining reusable software architectures [26], where the use of Z greatly simplified the decomposition of function into components, and the protocols of interaction between components.

To the best of our knowledge, however, with the exception of the VDM-SL Toolkit, formal methods have often not been used extensively in the *development* of the formal methods support tools described in Myth 2 (above). HOL is addressing this issue by the addition of a formally developed proof *checker*.

4 Conclusion

The question arises as to how the technology transfer process from formal methods research to practice can be facilitated [56]. More *real* links between industry and academia are required; and well publicized demonstrations of successful uses of formal methods are needed to disseminate the benefits of their use. [35] aims to play its part in this by providing a collection of descriptions of the use of formal methods at an industrially useful scale written by the experts involved.

More research is of course required to develop the use of formal methods. For example, the European ESPRIT Basic Research project **ProCoS** on "Provably Correct Systems" is investigating the theoretical underpinning and techniques to allowing the formal development of systems from requirements through specification, program and hardware in a unified framework [11]. In addition, an associated **ProCoS-WG** Working Group of 24 industrial *and* academic partners has been set up for the next three years as an integral part of the project's plans [9].

As usual, it should be stressed that formal methods are not a panacea, but one approach amongst many that can help to improve system reliability. However, to quote Prof. Bev Littlewood, Centre for Software Reliability, City University, London, on a programme broadcast on 19 October 1993 by BBC Radio 4, it should be noted that:

> *"... if you want to build systems with ultra-high reliability which provide very complex functionality and you want a guarantee that they are going to work with this very high reliability ...*

... you can't do it!"

Acknowledgements

The authors would like to thank Anthony Hall for the inspiration of his original paper on the *Seven Myths of Formal Methods* [31] which made this paper possible. The paper is itself based on a longer Technical Report [14].

Jonathan Bowen is funded by the UK Engineering and Physical Research Council (EPSRC) on grant no. GR/J15186.

Mike Hinchey is currently with University of Cambridge Computer Laboratory, and is a faculty member of the Real-Time Computing Laboratory, Department of Computer and Information Science, New Jersey Institute of Technology, USA.

References

1. Abrial, J.-R.: *Assigning Meanings to Programs*. Prentice Hall International Series in Computer Science, to appear.
2. Anderson, S. & Bruns, G.: The Formalization and Analysis of a Communication Protocol. In [35].
3. Austin, S. & Parkin, G.I.: *Formal Methods: A Survey*, National Physical Laboratory, Teddington, Middlesex TW11 0LW, UK, March 1993.
4. Barwise, J.: Mathematical Proofs of Computer System Correctness. Notices of the American Mathematical Society, 36(7):844–851, September 1989.
5. Boehm, B.W.: *Software Engineering Economics*, Prentice Hall, 1981.
6. Boehm, B.W.: A Spiral Model of Software Development and Maintenance. *IEEE Computer*, 21(5):61–72, May 1988.
7. Bowen, J.P.: Formal Methods in Safety-Critical Standards. In *Proc. 1993 Software Engineering Standards Symposium (SESS'93)*, Brighton, UK, IEEE Computer Society Press, 1993, pp 168–177.
8. Bowen, J.P.: Select Z Bibliography. In [13], pp 359–396. Also available as Oxford University Computing Laboratory Technical Report PRG-TR-8-94, June 1994.
9. Bowen, J.P. *et al.*: A ProCoS II Project Description: ESPRIT Basic Research project 7071, *Bulletin of the European Association for Theoretical Computer Science (EATCS)*, 50:128–137, June 1993.
10. Bowen, J.P., Breuer, P.T. & Lano, K.C. Formal Specifications in Software Maintenance: From code to Z^{++} and back again. *Information and Software Technology*, 35(11/12):679–690, November/December 1993.
11. Bowen, J.P., Fränzle, M., Olderog, E-R. & Ravn, A.P.: Developing Correct Systems. In *Proc. Fifth Euromicro Workshop on Real-Time Systems*, Oulu, Finland, 22–24 June 1993. IEEE Computer Society Press, pp 176–187.
12. Bowen, J.P. & Gordon, M.J.C.: Z and HOL. In [13], pp 141–167.
13. Bowen, J.P. & Hall, J.A., editors: *Z User Workshop, Cambridge 1994*. Springer-Verlag, Workshops in Computing, 1994.
14. Bowen, J.P. & Hinchey, M.G.: *Seven More Myths of Formal Methods*. Oxford University Computing Laboratory Technical Report PRG-TR-7-94, June 1994.
15. Bowen, J.P. & Stavridou, V.: The Industrial Take-up of Formal Methods in Safety-Critical and Other Areas: A Perspective. In [59], pp 183–195.

16. Bowen, J.P. & Stavridou, V.: Safety-Critical Systems, Formal Methods and Standards. *Software Engineering Journal*, 8(4):189–209, July 1993.
17. Bowen, J.P. & Stavridou, V.: Formal Methods: Epideictic or Apodeictic? *Software Engineering Journal*, 9(1):2, January 1994.
18. Boyer, R.S. & Moore, J.S.: *A Computational Logic Handbook*. Academic Press, 1988.
19. Coombes, A.C., Fitzgerald, J.S., McDermid, J.A., Saeed, A. & Spencer, L.: Formal Specification of an Aerospace System: The Attitude Monitor. In [35].
20. Craigen, D., Gerhart, S. & Ralston, T.: An International Survey of Industrial Applications of Formal Methods (Volume 1: Purpose, Approach, Analysis and Conclusions, Volume 2: Case Studies). Atomic Energy Control Board of Canada, U.S. National Institute of Standards and Technology, and U.S. Naval Research Laboratories, NIST GCR 93/626, 1993. Available from National Technical Information Service, 5285 Port Royal Road, Springfield, VA 22161, USA.
21. Craigen, D., Gerhart, S. & Ralston, T.: Applications of Formal Methods: Observations and Trends. In [35].
22. Dehbonei, B. & Mejia, F.: Formal Development of Safety-Critical Software Systems in Railways. In [35].
23. Dix, A.: *Formal Methods for Interactive Systems*. Academic Press, Computers and People Series, 1991.
24. Draper, C.: Practical Experiences of Z and SSADM. In Bowen, J.P. & Nicholls, J.E., editors: *Z User Workshop, London 1992*, Springer-Verlag, Workshops in Computing, 1993, pp 240–254.
25. Fitzgerald, J.S., Larsen, P.G., Brookes, T. & Magillian, P.: Developing a Security-Critical System using Formal and Conventional Methods. In [35].
26. Garlan, D. & Delisle, N.: Formal Development of a Software Architecture for a Family of Instrumentation Systems. In [35].
27. Goguen, J.A. & Winkler, T.: Introducing OBJ3. Technical Report SRI-CSL-88-9, Computer Science Laboratory, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, USA, August 1988.
28. Gordon, M.J.C. & Melham, T.F., editors: *Introduction to HOL: A theorem proving environment for higher order logic*. Cambridge University Press, 1993.
29. Guaspari, D., Seager, M. & Stillerman, M.: Specifying the Kernel of a Secure Distributed Operating System. In [35].
30. Guttag, J.V. & Horning, J.J: *Larch: Languages and Tools for Formal Specification*, Springer-Verlag, Texts and Monographs in Computer Science, 1993.
31. Hall, J.A.: Seven Myths of Formal Methods. *IEEE Software*, 7(5):11–19, September 1990.
32. Hamilton, V. & Quinn, K.F.: A Case Study in the Use of Z within a Safety-Critical Software System. In [35].
33. Haughton, H. & Lano, K.: Formal Development of Safety-Critical Medical Systems. In [35].
34. He Jifeng, Page, I. & Bowen, J.P.: Towards a Provably Correct Hardware Implementation of Occam. In Milne, G.J. & Pierre, L., editors: *Correct Hardware Design and Verification Methods*, Springer-Verlag, LNCS 683, 1993, pp 214–225.
35. Hinchey, M.G. & Bowen, J.P., editors: *Applications of Formal Methods*. Prentice Hall International Series in Computer Science, to appear 1995.
36. Hoare, J.: Formal Development of CICS with B. In [35].
37. Hoare, C.A.R. & Gordon, M.J.C., editors: *Mechanized Reasoning and Hardware Design*. Prentice Hall International Series in Computer Science, 1992.

38. Knight, J. & Littlewood, B., editors: Special issue on Safety-Critical Systems. *IEEE Software*, January 1994.
39. Larsen, P.G., Plat N. & Toetenel, H.: A Formal Semantics of Data Flow Diagrams. *Formal Aspects of Computing*, 6, 1994.
40. Leveson, N.G.: Software Safety in Embedded Computer Systems. *Communications of the ACM*, 34(2):34–46, February 1991.
41. Martin, A.: Encoding $\mathcal{W}$: A Logic for Z in 2OBJ. In [59], pp 462–481.
42. Mataga, P. & Zave, P.: Multiparadigm Specification of an AT&T Switching System. In [35].
43. Mukherjee, P. & Wichmann, B.A.: Formal Specification of the STV Algorithm. In [35].
44. Owre, S., Rushby, J.M. and Shankar, N.: PVS: A Prototype Verification System. In Kapur, D., editor: *Automated Deduction – CADE-11*, Springer-Verlag, LNAI 607, 1992, pp 748–752.
45. Parnas, D.L.: Using Mathematical Descriptions in the Inspection of Safety-Critical Software. In [35].
46. Peleska, J., Hamer, U. & Hoercher, H.-M.: The Airbus A330/340 Cabin Communication System – A Z Application. In [35].
47. Plat, N., Durr, E.H. & de Boer, M.: CombiCom: Tracking and Tracing Rail Traffic using VDM^{++}. In [35].
48. Polack, F. & Mander, K.C.: Software Quality Assurance using the SAZ Method. In [13], pp 230–249.
49. The RAISE Language Group: *The RAISE Specification Language*. Prentice Hall, BCS Practitioner Series, 1992.
50. Randell, G.P.: Data Flow Diagrams and CSP. DRA Memorandum 4520, Malvern, UK, February 1992.
51. Royce, W.W.: Managing the Development of Large Software Systems. In *Proc. WESTCON'70*, August 1970, reprinted in *Proc. 9th International Conference on Software Engineering*, IEEE Press, 1987.
52. Saaltink, M.: Z and Eves. In Nicholls, J.E., editor: *Z User Workshop, York 1991*, Springer-Verlag, Workshops in Computing, 1992, pp 233–242.
53. Semmens, L.T., France, R.B. & Docker, T.W.G.: Integrating Structured Analysis and Formal Specification Techniques. *The Computer Journal*, 36(6):600–610, December 1992.
54. Semmens, L.T. & Allen, P.M.: Using Yourdon and Z. In Nicholls, J.E., editor: *Z User Workshop, Oxford 1990*, Springer-Verlag, Workshops in Computing, 1991, pp 228–253.
55. Srivas, M., Miller, S. & Rushby, J.: Formal Verification of AAMP5: A Case Study in the Verification of a Commercial Microprocessor. In [35].
56. Weber-Wulff, D.: Selling Formal Methods to Industry. In [59], pp 671–678.
57. Wing, J.M: A Specifier's Introduction to Formal Methods. *IEEE Computer*, 23(9):8–24, September 1990.
58. Woodcock, J.C.P., Gardiner, P.H.B. & Hulance, J.R.: The Formal Specification in Z of Defence Standard 00-56. In [13], pp 9–28.
59. Woodcock, J.C.P. & Larsen, P.G., editors: *FME'93: Industrial-Strength Formal Methods*. Springer-Verlag, LNCS 670, 1993.
60. Young, W.D.: Verifying a Simple Real-Time System with Nqthm. In [35].

This article was processed using the LaTeX macro package with LLNCS style