Seven More Myths of Formal Methods

JONATHAN P. BOWEN, *Oxford University*
MICHAEL G. HINCHEY, *University of Cambridge*

◆ *New myths about formal methods are gaining tacit acceptance both outside and inside the system-development community. The authors address and dispel these myths based on their observations of industrial projects.*

> *Mathematicians first used the sign $\sqrt{-1}$ without in the least knowing what it could mean, because it shortened work and led to correct results. People naturally tried to find out why this happened and what $\sqrt{-1}$ really meant. After two hundred years they succeeded.*
>
> — *W. W. Sawyer, Mathematician's Delight, 1943.*

In 1990, Anthony Hall published a seminal article that listed and dispelled seven myths about the nature and application of formal methods.[1] Today — five years and many successful applications later — formal methods remain one of the most contentious areas of software-engineering practice.

In essence, a formal method is a mathematically based technique for describing a system. Using formal methods, people can systematically specify, develop, and verify a system. However, as we show in the box on page 37, basic definitions of formal methods and related terms are somewhat confused.

What *is* clear is that despite 25 years of use, few people understand exactly what formal methods are or how they are applied.[2] Many nonformalists seem to believe that formal methods are merely an academic exercise — a form of mental masturbation that has no relation to real-world problems. The media's portrayal of formal methods does little to help the situation. In many "popular press" science journals, formal methods are subjected to either deep criticism or, worse, extreme hyperbole.

Many of Hall's myths were — and we believe to a certain extent still are — propagated by the media. Fortunately, today these myths are held more by the public and the computer-science community at large than by system developers. It is our concern, however, that new myths are being propagated, and more alarmingly, are receiving a certain tacit acceptance from the system-development community. We reexamine Hall's

0740-7459/94/$04.00 © 1994 IEEE

myths in the box on this page and, following his lead, we address and dispel seven new myths about formal methods.

MYTH 8

♦ *Formal methods delay the development process.*

Several formal-methods projects have run notoriously over schedule. However, to assume this is a problem inherent in formal methods is irrational. These projects were delayed not because formal-methods specialists lacked ability, but because they lacked experience in determining how long development should take.

Estimating project cost is a major headache for any development team. If you follow the old adage, "estimate the cost and then double it," you're still likely to underestimate. Determining 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 Barry Boehm's Cocomo model,[3] which weights various factors according to the organization's history of system development. Herein is 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 and familiarity with the problem. Even with traditional development methods, this information is not always available. Historical information about projects that used formal development techniques is likely to be even more scarce, because we have not yet applied formal methods to a sufficient number of projects. Surveys of formal development[4,5] and highlights of successes, failures, hindrances, and so on, will eventually provide us with the information we require.

Many of the much-publicized formal-methods projects have been in very specialized domains, producing data that is of limited use. Future work with more conventional developments and applications in domains such as process control will likely provide more useful data.

Despite these difficulties, there have been some very successful formal-methods projects in which development time was significantly reduced. The Inmos T800 floating-point unit chip, produced using Z and the Occam Transformation System, was finished 12 months ahead of schedule, and the application of Z (and more recently B) to IBM's CICS system resulted in a 9 percent savings in development costs.

MYTH 9

♦ *Formal methods lack tools.*

Just as in the late 1970s and early 1980s, when CASE and computer-aided structured-programming tools were seen as a way to increase programmer productivity and reduce "bugs," tool support is now seen as a way to increase productivity and accuracy in formal development. Many projects place great emphasis on tool support.[5] This is by no means coincidental, but rather follows a trend that we expect will result in integrated workbenches to support formal specification, just as CASE workbenches support system development using more traditional structured methods.

Several formal methods incorporate tool support within the method itself. In this category are specification languages with executable subsets (such as OBJ) and formal methods that incorporate theorem provers as a key component, such as Larch (with the Larch Prover), Nqthm (successor to the Boyer-Moore prover), and higher order logic (supported by HOL and more recently, the

HALL'S MYTHS REVISITED

In 1990, Hall articulated and dispelled the following myths about formal methods,

♦ Myth 1: *Formal methods can guarantee that software is perfect.*
♦ Myth 2: *Formal methods are all about program proving.*
♦ Myth 3: *Formal methods are only useful for safety-critical systems.*
♦ Myth 4: *Formal methods require highly trained mathematicians.*
♦ Myth 5: *Formal methods increase the cost of development.*
♦ Myth 6: *Formal methods are unacceptable to users.*
♦ Myth 7: *Formal methods are not used on real, large-scale software.*

Myths that formal methods can guarantee perfect software and eliminate the need for testing (Myth 1) are not only ludicrous, but can have serious ramifications in system development if naive users of formal methods take them seriously.

Although claims that formal methods are all about proving programs correct (Myth 2) and are only useful in safety-critical systems (Myth 3) are untrue, they are not quite so detrimental. A number of successful applications in non-safety-critical domains have helped to clarify these points.

The derivation of many simple formal specifications of complex problems, and the successful development of several 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 nonspecialists in system development with formal methods has ruled out the myth that formal methods are unacceptable to users (Myth 6). The successful application of formal methods to several 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).

PVS Prototype Verification System).

Many basic tools are widely available today. For example, Z is supported by ZTC, a PC- and Sun-based type-checking system available via anonymous file-transfer protocol for noncommercial purposes, and by Fuzz, a commercial type-checker that also runs under Unix and DOS. More integrated packages that support typesetting and specification integrity checking include Logica Cambridge's Formaliser (for Microsoft Windows), Imperial Software Technology's Zola (which also incorporates a tactical proof system), and York Software Engineering's Cadiz (a tool suite for Z that now supports the refinement to Ada code). The Mural system, developed at University of Manchester, supports the construction of VDM specifications and refinements; using the proof assistant, users can generate proof obligations to verify the internal consistency of specifications. FDR, from Formal Systems Europe, is a model- and refinement-checker for CSP (communicat ing sequential processes). CRI (Computer Resources International) produces an associated toolset for the Raise development method (Rigorous Approach to Industrial Software Engineering), which is a more comprehensive successor to VDM. Finally, ICL's ProofPower uses higher order logic to support specification and verification in Z.

INTEGRATING FORMAL AND STRUCTURED METHODS CAN OFFER FULL CYCLE SUPPORT.

Perhaps motivated by the ProofPower approach, much attention has been focused on tailoring various "generic" theorem provers for use with model-based specification languages like Z. Although an implementation in OBJ seems to be too slow, success has been reported with HOL and EVES, a toolset based on Zermelo-Fraenkel set theory.

In the future, we expect more emphasis to be placed on integrated formal-development support environments, which are intended to support most formal-development stages, from initial functional specifications through design specifications and refinement. These environments will also support specification animation, proof of properties, and proofs of correctness. Such toolkits will be integrated so that, like integrated programming-support environments, they will support both version control and configuration management and development by larger teams. They will also facilitate more harmonious development by addressing all of the development-process activities. Such environments do not as yet exist, but several toolkits represent steps in the right direction.

IFAD's VDM-SL Toolbox supports formal development in VDM-SL and includes, as you might expect, standard type checkers and static semantics checkers. Developers enter VDM-SL specifications in ASCII. An interpreter supports all of the executable constructs of VDM-SL, allowing a form of animation and specification "testing." The executed specifications can be debugged using an integrated debugger, and testing information is automatically generated. Finally, a pretty-printer uses the ASCII input to generate VDM-SL specifications in LaTex format.

The B-Toolkit, from B-Core, is a set of integrated tools that augment Abrial's B-Method and the associated B-Tool for formal software development by addressing industrial needs in the development process. Many believe that B and the B-Method represent the next generation of formal methods; if this is true, then B and similar toolkits will certainly form the basis of future formal-development environments.

MYTH 10

♦ *Formal methods replace traditional engineering design methods.*

One of the major criticisms of formal methods is that they are not so much "methods" as formal systems. Although they provide support for a formal notation (formal specification language), and some form of deductive apparatus (proof system), they fail to support many of the methodological aspects of the more traditional structured-development methods.

In the context of an engineering discipline, a *method* describes how a process is to be conducted. In the context of system engineering, a method consists of an underlying development model; a language or languages; defined, ordered steps; and guidance for applying these in a coherent manner.[6]

Many so-called formal methods do not address all of these issues. Although they support some of the design principles of more traditional methods — such as top-down design and stepwise refinement — they place little emphasis on the underlying development model and provide little guidance as to how development should proceed. Structured-development methods, using a model such as Boehm's spiral model, on the other hand, generally support all stages of the system life cycle from requirements elicitation through postimplementation maintenance. In general, these underlying models recognize the iterative nature of system development. However, many formal development methods assume that specification is followed by design and then by implementation, in strict sequence. This is an unrealistic view of development — every developer of complex systems must revisit both the requirements and the specification at much later stages in development.

Although Hall disputes the myths that formal methods are unacceptable to users and require significant mathematical ability, more traditional design methods excel at requirements elicitation and interaction with users. They offer notations that can be understood by nonspecialists and serve as the basis for a contract.

Traditional structured methods are severely limited because they offer few ways to reason about the validity of a specification or whether certain requirements are mutually exclusive. The former is often only discovered after implementation; the latter, during implementation. Formal methods, of course, allow the possibility of reasoning about requirements, their completeness, and their interactions.

Indeed, instead of formal methods replacing traditional engineering-design methods, a major area for research is the *integration* of structured and formal methods. Such an integration leads to a "true" development method that fully supports the software life cycle and allows developers to use more formal techniques in the specification and design phases, supporting refinement to executable code and proof-of properties. The result is that two views of the system are presented, letting developers concentrate on aspects that interest them.

Some people suggest that this integrated approach lets structured design serve as a basis for insights into the formal specification. This idea is clearly controversial. Opponents argue that an approach that allows a structured design to guide formal-specification development severely restricts levels of abstraction and goes against many principles of formal-specification techniques. Proponents of integration argue that the approach is easier for users unskilled in formal-specification techniques, that it aids in size and complexity management, and that it provides a way to structure specifications.[7]

Approaches to method integration vary from running structured and formal methods in parallel, to formally specifying transformations from structured-method notations to formal-specification languages.

Much success has been reported using the former technique. The problem, however, is that because the two methods are being addressed by different personnel, the likelihood that benefits will be highlighted is low. In many cases, the two development teams do not adequately interact. For example, there is a project underway at British Aerospace using traditional and formal development methods in parallel. The two development teams are not permitted to communicate, and the formal approach will be subject to the same standards reviews, which are certified against ISO 9000. The project's aim is to investigate how formal methods might better fit into current development practices.

More integrated approaches to integration include the translation of SSADM (Structured Systems Analysis and Design Methodology) into Z as part of the SAZ project; the integration of Yourdon Modern Structured Analysis and Z in a more formalized manner, and the integration of various structured notations with VDM and CSP. Although these approaches may have great potential, unlike the parallel approach they have yet to be applied to realistic systems.

MYTH 11

♦ *Formal methods only apply to software.*

Formal methods can be applied equally well to hardware design and software development. Indeed, this is one of the motivations of the HOL theorem prover that was used to verify parts of the Viper microprocessor. Other theorem-proving systems that have been applied to hardware verification include the Boyer-Moore, Esterel, Nuprl, 2OBJ, Occam Transformation System, and Veritas proof tools. Model checking is also important in checking hardware designs if the state space is small enough (and techniques like Binary Decision Diagrams handle an impressive number of states). Perhaps the most convincing and complete hardware-verification exercise is Computational Logic's FM9001 microprocessor, which has been verified down to a gate-level netlist representation using the Boyer-Moore theorem prover. (A netlist is a list of component gates and their interactions.)

Inmos provides two examples of real-world industrial use. The T800 transputer floating-point unit has been verified by starting with a formalized Z specification of the IEEE floating-point standard. The Occam Transformation System was then used 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

DEFINING FORMAL METHODS

Highly publicized accounts of formal-methods application to a number of well-known systems, such as the Sizewell-B nuclear power plant in the UK, IBM's CICS system, and the most recent Airbus aircraft, have helped bring the industrial application of formal methods to a wider audience.

However, even basic terms such as "formal specification" are still likely to be confusing. For example, the following alternative definitions are given in a glossary issued by the IEEE:

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.

Although the latter is accepted in the formal-methods community, the former may have more widespread acceptance in industrial circles. A search of the abbreviation CSP in an online acronym database cited "Commercial Subroutine Package," "CompuCom Speed Protocol," and "Control Switching Point," but not "Communicating Sequential Processes" — which would be the likely choice of people working with formal methods. Finally, 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!

Besides ambiguity in the basic terminology, the formal notations themselves can be confusing to practitioners not trained in their use, and as a result the uninitiated might find it easier to ignore them than to investigate further.

FORMAL METHODS RESOURCES

There are several electronic distribution lists on formal methods and related topics, including

- ♦ Z Forum (zforum-request@comlab.ox.ac.uk),
- ♦ VDM Forum (vdm-forum-request@mailbase.ac.uk),
- ♦ Larch Interest Group (larch-interest-request@src.dec.com), and
- ♦ OBJ Forum (objforum-request@comlab.ox.ac.uk).

Z Forum has spawned comp.specification.z, an electronic newsgroup that is read regularly by about 30,000 people worldwide. A newsgroup devoted to specification in general, comp.specification, regularly generates discussions on formal methods, as well as the more traditional structured methods, object-oriented design, and so on, as does the comp.software-eng newsgroup.

A recently established mailing list at University of Idaho (formal-methods-request@cs.uidaho.edu) addresses formal methods in general, rather than any specific notation, and a new mailing list run by the Z User Group addresses educational issues (zugeis-request@comlab.ox.ac.uk). In addition, the newsletter of the IEEE Technical Segment Committee on the Engineering of Complex Computer Systems (ieee-tsc-eccs-request@cl.cam.ac.uk) addresses issues related to formal methods and formal-methods education.

There are also anonymous FTP archives for Z (including an online and regularly revised comprehensive bibliography). The global World Wide Web electronic hypertext system, which is rapidly becoming very popular, also provides support for formal methods. A useful starting point is http://www.comlab.ox.ac.uk/archive/formal-methods.html which provides pointers to other electronic archives concerned with formal methods and lets you download tools such as HOL and PVS.

Periodicals. The proceedings of the Formal Methods Europe symposiums (and their predecessors, the VDM symposium) are available in Springer-Verlag's *Lecture Notes in Computer Science* series, while the proceedings of the Refinement Workshops and the last five Z User Meetings have been published in Springer-Verlag's *Workshops in Computing* series. Both of these series contain the proceedings of many other interesting colloquiums, workshops, and conferences on formal methods.

Although papers on formal methods are becoming well-established at a number of US conferences, there is as yet no regular conference in the US devoted to formal methods. The Workshop on Industrial-Strength Formal Specification Techniques may represent a step in that direction (see the report on pp. 106-107). Although formal methods are gaining momentum in the US, the main journals and publications devoted to formal methods are based in Europe — and in the UK, specifically.

These include *Formal Aspects of Computing*, *Formal Methods in System Design* and the *FACS Europe* newsletter run by Formal Methods Europe and the British Computer Society's Special Interest Group on Formal Aspects of Computing Science, among 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 to run special issues on the subject.

As far as we know, there are no US journals devoted specifically to formal methods, although some of the highly respected journals, such as *IEEE Transactions on Software Engineering* and *Journal of the ACM*, and popular periodicals, such as *Computer*, *IEEE Software*, and *Communications of the ACM*, regularly publish relevant articles. *IEEE TSE*, *Computer*, and *IEEE Software* coordinated successful special issues on formal methods in 1990. In January 1994, an *IEEE Software* special issue on safety-critical systems devoted considerable attention to formal methods, as has a newly launched journal, *High Integrity Systems*.

Courses. Popular Z courses are run by Logica Cambridge, Praxis, Formal Systems (Europe), and Oxford University Computing Laboratory. About 70 percent of all industrially based formal-methods courses focus on the Z notation. Formal Systems also runs a CSP course and a CSP with Z course, both of which have been given in the US as well as the UK. IFAD in Denmark offers an industrially based formal-methods course using VDM and VDM++.

checked for correctness. (A collection of papers by experts in the field covers these applications in more detail.[8])

A more recent approach to hardware development is *hardware compilation*. This allows a high-level program to be compiled directly into a netlist of simple components and their interconnections. If required, Field Programmable Gate Arrays allows this to be done entirely as a software process, since these devices let the circuit be configured according to the static RAM contents within the chip (this route is particularly useful for rapid prototyping).

It is also possible to prove that the compilation process itself correct. In this case, the burden of proof is reduced considerably because there is no need to prove the hardware correct with each separate compilation. For example, a microprocessor could be compiled into hardware by describing the microprocessor as an interpreter written in a high-level language. Additions and changes to the instruction set can be made easily by editing the interpreter and recompiling the hardware with no additional proof-of-correctness required.

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

MYTH 12

♦ *Formal methods are unnecessary.* At some point or another, most of us have heard the argument that formal methods are not required. This is untrue. Although there are occasions in which formal methods are in a sense "overkill," in other situations they are very desirable. In fact, the use of formal methods is recommended in any system where correctness is of concern. This clearly applies to safety- and security-critical systems, but it also applies to systems in which you need (or want) to ensure that the system will avoid the catastrophic consequences of a failure.

Sometimes formal methods are not only desirable, but required. Many standards bodies have not only used formal specification languages in making their own standards unambiguous, but have mandated or strongly recommended the use of formal methods in certain classes of applications.[9,10]

The International Electrotechnical Commission specifically mentions temporal logic and several formal methods (CCS, CSP, HOL, LOTOS, OBJ, VDM, and Z) in the development of safety-critical systems. The European Space Agency suggests that VDM or Z, augmented with natural-language descriptions, should be used to specify safety-critical system requirements. It also advocates proof-of-correctness, a review process, and the use of a formal proof before testing. The UK Ministry of Defence draft Interim Defence Standards 00-55 and 00-56 mandate the extensive use of formal methods. The draft standard 00-55 sets forth guidelines and requirements that 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 proofs and rigorous (but informal) correctness proofs, as well as more conventional static and dynamic analysis. The draft standard 00-56 deals with the classification and hazard analysis of the software and electronic components of defense equipment, and also mandates the use of formal methods.

Canada's Atomic Energy Control Board has commissioned, in conjunction with David Parnas at McMaster University, a proposed standard for software in the safety systems of nuclear-power stations. Ontario Hydro has developed a number of standards and procedures within the framework set by AECB, and more procedures are under development. Standards and procedures developed by Canadian licensees mandate the use of formal methods and, together with 00-55, are among the farthest reaching at the moment.

Whether or not you believe that formal methods are necessary in system development, you cannot deny that they are indeed *required* in certain classes of applications and are likely to be required more often in the future.[9]

STANDARDS ARE POINTLESS IF THEY DON'T REFLECT THE OPINIONS OF ACTIVE USERS.

MYTH 13

♦ *Formal methods are not supported.*

Once upon a time (as all good stories start) formal development might have been a solitary activity, a lone struggle. Today, however, support for formal methods is indisputable. If media attention is anything to go by, interest in formal methods has grown phenomenally, albeit from a small base. Along with object orientation, formal methods have quickly become great buzzwords in the computer industry. Long gone are the days when lone researchers worked on developing appropriate notations and calculi. The development of more popular formal methods owes much to the contributions of many people beyond the method originators. In many cases, researchers and practitioners extended the languages to support their particular needs, adding useful (though sometimes unsound) operators and data structures and extending the languages with module structures and object-oriented concepts.

There is a certain trade-off between the expressiveness of a language and the levels of abstraction that it supports. Making a language more expressive facilitates briefer and more elegant specifications, but it can also make reasoning more difficult. LOTOS was standardized in 1989, and the International Organization for Standardization has proposed draft standards for both Z and VDM.[9] These standards set forth sound constructs and their associated formal semantics, making it easier to read other people's specifications (assuming, of course, that they conform to the standards).

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 several 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 distribution lists, books, periodicals, and conferences. We list some examples of each in the box on page 38.

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, a recent NSF-sponsored workshop sought to establish a curriculum for teaching formal methods in US undergraduate programs. We hope this will become a regular event, and 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.

MYTH 14

♦ *Formal-methods people always use formal methods.*

There is widespread belief that proponents of formal methods apply them in all aspects of system development. This could not be further from the truth. Even the most fervent supporters of formal methods recognize that other approaches are sometimes better.

In user-interface design, for example, it is very difficult for the developer to determine, and thus formalize, the exact requirements of human-computer interaction at the outset of a project. In many cases, the user interface must be configurable, with various color combinations highlighting certain conditions (such as red to denote an undesirable situation). The great difficulty, however, is in determining how the user interface should look and feel. The appropriateness of a particular interface is a subjective matter and not really amenable to formal investigation. Although there have been several (somewhat successful) approaches to formal specification in user interfaces,[11] in general conformance testing here falls in the domain of informal reasoning.

There are many other areas in which, although possible, formalization is impractical because of resources, time, or money. Most successful formal-methods projects involve the application of formal methods to critical portions of system development. Only rarely are formal methods alone applied to all aspects of system development. Even within IBM's-CICS project — which is often cited as a major successful application of formal methods — only about one-tenth of the entire system was actually subjected to formal techniques (although this still involved hundreds of thousands of lines of code and thousands of pages of specifications). Clearly (with appropriate apologies to Einstein), *system development should be as formal as necessary, but not more formal.*

Formal methods have been used to develop a number of support tools for conventional development methods, such as the SSADM CASE tool described by Hall. Formal methods have also been used to help redevelop a reverse engineering and analysis toolset for Cobol at Lloyd's Register. Both of these projects used Z, which was also used in defining reusable software architectures and greatly simplified the decomposition of function into components and the protocols of interaction between components.

To the best of our knowledge, however, formal methods have not been used extensively to develop the formal-methods support tools described in Myth 9. Exceptions to this are the VDM-SL Toolbox and the addition of a formally developed proof checker to HOL.

How can the technology-transfer process from formal-methods research to practice be facilitated? To start with, more *real* links between industry and academia are required, and the successful use of formal methods must be better publicized. We have edited a forthcoming collection of papers[5] that will play its part by describing the use of formal methods at an industrially useful scale.

More research is required to further develop the use of formal methods. For example, ProCoS, the ESPRIT basic research project on provably correct systems, is investigating theoretical underpinnings and techniques to allow the formal development of systems in a unified framework — from requirements to specification, program, and hardware. In addition, a ProCoS Working Group of 24 industrial and academic partners has been established. Joint meetings between the project and working groups over the next three years allows a free flow of ideas. The hope is that some of these ideas will be used in a more industrially oriented collaborative project in the future.

Formal methods are not a panacea, but one approach among many that can help to improve system reliability. However, to quote from a BBC radio interview with Bev Littlewood of the Centre for Software Reliability at City University in London,

"*. . . 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!*" ◆

ACKNOWLEDGMENTS

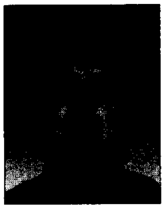
We thank Anthony Hall for inspiring this article by authoring the "Seven Myths of Formal Methods." Jonathan Bowen is funded by UK Engineering and Physical Sciences Research Council (EPSRC) grant GR/J15186. Mike Hinchey is funded by ICL.

REFERENCES

1. J.A. Hall, "Seven Myths of Formal Methods," *IEEE Software*, Sept. 1990, pp 11-19.
2. W.W. Gibbs, "Software's Chronic Crisis," *Scientific American*, Sept. 1994, pp. 86-95.
3. B.W. Boehm, *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, N.J., 1981.
4. S.L. Gerhart, D. Craigen, and T. Ralston, "Experience with Formal Methods in Critical Systems," *IEEE Software*, Jan. 1994, pp. 21-28.
5. *Applications of Formal Methods*, M.G. Hinchey and J.P. Bowen, eds., Fall 1995, Prentice-Hall, Hemel Hempstead, UK, to appear; http://www.comlab.ox.ac.uk./archive/formal-methods/afm-book.html.
6. *Method Integration: Concepts and Case Studies*, K. Kronlöf, ed., John Wiley & Sons, New York, 1993.
7. L.T. Semmens, R.B. France, and T.W.G. Docker, "Integrating Structured Analysis and Formal Specification Techniques," *The Computer J.*, Dec. 1992, pp. 600-610.
8. *Mechanized Reasoning and Hardware Design*, C.A.R. Hoare and M.J.C. Gordon, eds., Prentice-Hall, Englewood Cliffs, N.J., 1992.
9. J.P. Bowen, "Formal Methods in Safety-Critical Standards," *Proc. 1993 Software Engineering Standards Symp.*, IEEE CS Press, Los Alamitos, Calif., 1993, pp. 168-177.
10. J.P. Bowen and V. Stavridou, "Safety-Critical Systems, Formal Methods and Standards," *Software Engineering J.*, July 1993, pp. 189-209.
11. A. Dix, *Formal Methods for Interactive Systems*, Academic Press, San Diego, Calif., 1991.

Jonathan Bowen is a senior researcher at the Oxford University Computing Laboratory. He has worked in the field of computing in both industry and academia since 1977. He currently manages the ESPRIT ProCoS-WG Working Group of 24 European partners and is working in the area of provably correct hardware/software codesign. His interests include formal specification, Z, provably correct compilation, rapid prototyping using logic programming, decompilation, hardware compilation, safety-critical systems, and online museums.

Bowen received an MA in engineering science from Oxford University. He won the 1994 IEE Charles Babbage Premium award. He chairs the Z User Group, is conference chair for the ZUM'95 international conference of Z users, and is a member of the IEEE Computer Society, ACM, and Euromicro.



Mike Hinchey is a researcher with the University of Cambridge Computer Laboratory and a professor in the Real-Time Computing Laboratory at New Jersey Institute of Technology. His research interests include formal specification, formal methods, real-time systems, concurrency, method integration, CASE, and visual programming and environments. He has published widely on various aspects of software engineering and is the author or editor of several books on software development with formal methods.

Hinchey received a BSc in computer science from University of Limerick, Ireland, an MSc in computation from Oxford University, and a PhD in computer science from University of Cambridge. He is treasurer of the Z User Group, program chair for the ZUM'95 international conference of Z users, and editor of the newsletter of the IEEE Computer Society's Technical Segment Committee on Engineering of Complex Computer Systems. He is a member of the IEEE, ACM, AMS, and an associate fellow of the Institute of Mathematics.

Address questions about this article to Bowen at Oxford University Computing Laboratory, Wolfson Building, Parks Rd., Oxford OX1 3QD, UK; Jonathan.Bowen@comlab.ox.ac.uk;http://www.comlab.ox.ac.uk/oucl/people/jonathan.bowen.html or to Hinchey at University of Cambridge Computer Laboratory, New Museums Site, Pembroke St., Cambridge CB23QG, UK; Mike.Hinchey@cl.cam.ac.uk; http://www.cl.cam.ac.uk/users/mgh1001/