Technical Report

UCAM-CL-TR-357
ISSN 1476-2986

Number 357

Seven more myths of formal methods

Jonathan P. Bowen, Michael G. Hinchey

December 1994

ISSN 1476-2986

Seven More Myths of Formal Methods*

Jonathan P. Bowen
Oxford University Computing Laboratory
Programming Research Group
Wolfson Building, Parks Road, Oxford OX1 3QD, UK.
Email: `Jonathan.Bowen@comlab.ox.ac.uk`
URL: `http://www.comlab.ox.ac.uk/oucl/people/jonathan.bowen.html`

Michael G. Hinchey
University of Cambridge Computer Laboratory,
New Museums Site, Pembroke Street, Cambridge CB2 3QG, UK.
Email: `Mike.Hinchey@cl.cam.ac.uk`
URL: `http://www.cl.cam.ac.uk/users/mgh1001/`

Abstract

For whatever reason, formal methods remain one of the more contentious techniques in industrial software engineering. Despite great increases in the number of organizations and projects applying 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 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.

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.

Mathematician's Delight (1943) by W. W. Sawyer

*This paper will appear in *IEEE Software*

1 Introduction

Formal Methods continue to grow in popularity; growing numbers of delegates at conferences such as the Formal Methods Europe symposia and the Z User Meetings are indicative of this. Perhaps more interesting in this respect is the increasing number of papers devoted to formal methods and more formal approaches that surface at conferences with which one would not normally associate formal methods.

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 [4]. 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 and project managers into that latter category.

2 Hall's Original Seven Myths

In a seminal article [5], 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, five years later, these and other misconceptions still abound.

Formal methods are, unfortunately, often the subject of extreme hyperbole or deep criticism in many of the 'popular press' science journals. We should point out, that we believe the former to be more detrimental in the long run. 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.

Indeed even basic terms such as 'formal specification' are likely to be confused. For example, the following alternative definitions are taken from 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.

The latter is the accepted sense by those concerned with formal methods, but the former may have more widespread acceptance in industrial circles. A search of the abbreviation CSP on 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. 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.

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 [6] 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).

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 the Sizewell-B nuclear power plant in the UK, IBM's CICS system, the Darlington Nuclear Facility in Canada, and the most recent Airbus aircraft (all of which are reported in [6]) 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 we believe 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 [6].

Myth 8. *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. If one follows the old adage, "estimate the cost and then double it", one is still likely to underestimate. 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, 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 (see, for example, [2, 3, 6]) 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, data from these projects is of limited use; comparisons with more conventional developments and applications in more process control-like domains 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.

Myth 9. *Formal Methods are not supported by tools.*

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 [6], 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.

A number of formal methods incorporate tool support as part of the method itself. In this category we naturally include specification languages with executable subsets (such as OBJ), and formal methods that incorporate theorem provers as a key component—e.g., Larch (with LP, the Larch Prover), HOL, Nqthm and PVS.

A range of basic tools are now widely available, many of them in the public domain. For example, for support using the Z notation, ZTC is a PC and Sun-based type-checker freely available electronically via anonymous FTP for non-commercial purposes, 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, running under Microsoft Windows, 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. The Mural system provides support for the construction of VDM specifications and refinements; internal consistency of specifications is verified by generation of proof obligations, which may be discharged using the proof assistant. FDR from Formal Systems Europe is a model-checker and refinement-checker for CSP. ProofPower, a tool available from ICL on which considerable development effort has been expended, 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 seems to be too slow, but particular successes have been reported with HOL and EVES.

We expect that in the future more emphasis will be placed on IFDSEs (Integrated Formal Development Support Environments), which will support most stages of formal development, from initial functional specifications, through design specifications and refinement and will also provide support for specification animation, proof of properties and proofs of correctness. 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. VDM-SL specifications are entered in ASCII; 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. Finally, a pretty-printer uses the ASCII input to generate VDM-SL specifications in LaTeX format.

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 10. *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 they provide support for a formal notation, or *formal specification language*, and some form of deductive apparatus, or *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 the way in which a process is to be conducted. In the context of system engineering, a method is defined to consist of: (1) an underlying *model* of development, (2) a *language*, or *languages*, (3) defined, ordered steps, and (4) guidance for applying these in a coherent manner. (This definition is modified from [8].)

Clearly many so-called formal methods do not address all of these issues. While they 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, 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, the model proposed by Royce in 1970 (now commonly referred to as the 'waterfall' model), which only allows for limited re-iteration of phases of the life cycle. Yet 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 [5] 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.

Traditional 'structured' methods are of course severely limited in that they offer limited means of reasoning about the validity of a specification, or whether certain requirements are mutually exclusive. The former is often only discovered post-implementation; the latter, during implementation. Formal methods, of course, enable 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 *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. The result is that effectively two views of the system are presented, allowing different developers to concentrate on those particular aspects that are of interest to them.

It has been suggested that such an integrated approach allows a structured design to be used as a basis for insights into the construction of a formal specification. This idea is quite a contentious issue. A number of people have cited this as a disadvantage of the technique and something that should not be encouraged. The view is that an approach that allows a structured design to guide the development of a formal specification severely restricts levels of abstraction and goes against many of the principles of formal specification techniques. On the other hand, there is a very valid argument that taking such an approach is often easier for those unskilled in the techniques of formal specification to follow, and can aid in the management of size and complexity, and provide a means of structuring specifications [9].

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. 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. In fact, one project being undertaken for British Aerospace involves a development using BAe's traditional development methods and formal 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 to ISO 9000. The aim of the dual development technique in this project is to investigate how formal methods might fit better into existing development practices.

More integrated approaches 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. These all augur much potential, but unlike the parallel approach have yet to be applied to realistic systems.

Myth 11. *Formal Methods only apply to software.*

Formal methods can equally well be applied to hardware design as to software development. 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. A collection of invited papers written by experts in the field covers the applications outlined here and others in more detail [7].

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*, a list 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. In this case the hardware compiled each time need not be separately proven correct, thus reducing the proof burden considerably. For instance, a microprocessor could be compiled into hardware by describing the microprocessor as a interpreter written in a high-level language. Additions and changes to the instruction set could easily be made by editing the interpreter and recompiling the hardware with no additional proof of correctness required.

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 12. *Formal Methods are not required.*

Most professionals involved with software engineering have, at some point or other, heard the argument that formal methods are not required. This is a mistruth; while there are occasions where formal methods would be 'over-kill', there are many 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 due to the catastrophic consequences of a system failure. (See for example the presentation of the formal specification of an algorithm to determine the result in a single transferable voting system in [6].) 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, but have strongly recommended and in the future may mandate the use of formal methods in certain classes of applications [1, 2].

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. 00-55 sets forth guidelines and requirements; the requirements 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. 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 report on software for computers in the safety systems of nuclear power stations in conjunction with David Parnas at McMaster University. Ontario-Hydro have developed a number of standards and procedures within the framework set by AECB and further procedures are under development. Some 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 strongly advisable in certain classes of applications, and are likely to be required in an increasing number of cases in the future [1].

Myth 13. *Formal Methods are not supported.*

If media attention is anything to go by, interest in formal methods has grown phenomenally, if from a small base. Along with 'object-orientation' and a few other keywords, it has quickly become one of the great 'buzz-words' in the computer industry.

Long gone are the days when lone researchers worked on developing appropriate notations and calculi. The development of the more popular formal methods owes much to the contributions of a great number of people, not just their 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 degree of 'trade-off' between the expressiveness of a language and the levels of abstraction that it supports. Making a language more expressive does indeed facilitate briefer and more elegant specifications, but can make reasoning more difficult.

LOTOS was standardized in 1989, and draft ISO standards for both Z and VDM have been proposed [1]. These standards set forth a number of sound constructs and their associated formal semantics, making it easier to read other people's specifications (assuming, that is, that the relevant users will conform to these 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 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`) 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 an OBJ FORUM (contact `objforum-request@comlab.ox.ac.uk`) are also in operation.

Z FORUM has spawned `comp.specification.z`, a gatewayed electronic newsgroup which is read regularly by around 45,000 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, etc., as does the `comp.software-eng` newsgroup.

A recently-established mailing list at University of Idaho (contact `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 addressing educational issues (to subscribe, contact `zugeis-request@comlab.ox.ac.uk`) has also been established. In addition, the newsletter of the IEEE Technical Segment Committee on the Engineering of Complex Computer Systems (contact `ieee-tsc-eccs-request@cl.cam.ac.uk`) addresses issues related to formal methods and formal methods education.

Electronically accessible anonymous FTP archives for Z (including an on-line and regularly revised comprehensive bibliography) and other formal methods exist on the global Internet computer network. The global World Wide Web (WWW) electronic hypertext system, which is rapidly becoming very popular 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, Nqthm and PVS for downloading on the network:

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

A plethora of formal methods books are now available, and most reputable computer science publishers carry a title on the Z notation, or on discrete mathematics looking suspiciously like Z. The proceedings of various symposia and workshops offer invaluable reading on up-to-date developments in formal methods. The proceedings of the Formal Methods Europe symposia (and their predecessors, the VDM symposia) are available in the Springer-Verlag *Lecture Notes in Computer Science* series, while the proceedings of the Refinement Workshops and the last five Z User Meetings have been published in the Springer-Verlag *Workshops in Computing* series. Both of these series contain the proceedings of many other interesting colloquia, workshops and conferences on formal methods.

Formal methods are not quite so popular in the US, although they are gaining momentum. While 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. Perhaps the *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 run jointly by Formal Methods Europe and the British Computer Society 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 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 (for example, see [10]. 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 (e.g., [3]), 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, a recent NSF-sponsored workshop aims to establish a curriculum for teaching formal methods in US undergraduate programmes. One would hope that 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. Popular Z courses are run by Logica Cambridge Limited, Praxis, Formal Systems (Europe) Ltd., as well as by the Oxford University Computing Laboratory. In fact, as much as 70% of all industrially-based formal methods courses focus on the Z notation. Formal Systems (Europe) Ltd. also run 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 run an industrially-based formal methods course using VDM and VDM^{++}.

Once upon a time, as all good stories start, formal development might have been a lone activity, a lone struggle, but certainly one can no longer argue that formal methods are not supported!

Myth 14. *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. In many cases, the user interface is to be configurable, and various colour combinations will highlight certain conditions (red denoting an undesirable situation, etc.). The great difficulty is, however, in determining how the user interface should 'look-and-feel'. Certain functionality might be more conducive to particular interfaces, but other requirements might preclude that. It is very difficult to reason about user interfaces; 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, 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 (mentioned earlier), which is often cited as a major application of formal methods (and resulted in IBM and OUCL being jointly awarded a UK Queen's Award for Technological Achievement in 1992), 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 necessary, 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 [5]. 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. (Both of these projects made use of Z.) In addition, they have been successfully employed in defining reusable software architectures, 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 Toolbox, formal methods have often not been used extensively in the *development* of the formal methods support tools described in Myth 9 (above). HOL is addressing this issue with the addition of a formally developed proof *checker* which simply confirms that a proof script generated by HOL using the small number of basic axioms does indeed apply them correctly.

4 Conclusion

The question arises as to how the technology transfer process from formal methods research to practice can be facilitated. 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. A forthcoming collection of papers [6] 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 allow the formal development of systems from requirements through specification, program and hardware in a unified framework. In addition, an associated **ProCoS-WG** Working Group of 24 industrial *and* academic partners has been set up for three years. Meetings of the project and Working Group are held jointly to allow a free flow of ideas and comments in both directions. It is hoped that a more industrially oriented collaborative project on the application of some of the ideas developed in **ProCoS** will result in due course.

As usual, it should be stressed that formal methods are not a panacea, but one approach among 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* [5] which made this paper possible.

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

References

[1] Bowen, J.P. & Hinchey, M.G.: Formal Methods and Safety-Critical Standards. *Computer* 27(8): 68–71, August 1994.

[2] Bowen, J.P. & Stavridou, V.: Safety-Critical Systems, Formal Methods and Standards. *Software Engineering Journal*, 8(4):189–209, July 1993.

[3] Gerhart, S., Craigen, D. & Ralston, T.: Experience with Formal Methods in Critical Systems. *IEEE Software*, 11(1):21–28, January 1994.

[4] Gibbs, W.W.: Software's Chronic Crisis. *Scientific American*, 271(3):86–95, September 1994.

[5] Hall, J.A.: Seven Myths of Formal Methods. *IEEE Software*, 7(5):11–19, September 1990.

[6] Hinchey, M.G. & Bowen, J.P., editors: *Applications of Formal Methods*. Prentice Hall International Series in Computer Science, 1995.

[7] Hoare, C.A.R. & Gordon, M.J.C., editors: *Mechanized Reasoning and Hardware Design*. Prentice Hall International Series in Computer Science, 1992.

[8] Kronlöf, K., editor: *Method Integration: Concepts and Case Studies*. John Wiley & Sons, Series in Software Based Systems, 1993.

[9] 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.

[10] Wing, J.M.: A Specifier's Introduction to Formal Methods. *Computer*, 23(9):8–24, September 1990.