Formal Methods in the 21'st Century
An Assessment of Today — Predictions for the Future

Dines Bjørner
Technical University of Denmark
bjorner@gmail.com
April 1998

ABSTRACT
It has been practice, during software engineering related conferences in the last almost 20 years, to discuss and debate the rôle of formal techniques: specification and calculi, in software engineering. So also ICSE'98 will feature such a panel. We outline a possible basis for a discussion, cum debate (?), around the topic, but take as our departure point a recent statement made by Tony Hoare.

1 A Motto?

In preparation for a World Congress on Formal Methods[1] Tony Hoare expressed:

- **Maturity:** Use of a formal method is no longer an adventure; it is becoming routine.
- **Convergence:** The choice of a formal method or tool is no longer controversial: they are chosen in relation to their purpose and they are increasingly used in effective combination.
- **Cumulative progress:** Promise of yet further benefit is obtained by accumulation of tools, libraries, theories, and case studies based on the work of scientists from many schools which were earlier considered as competitors.

The present panel will consider this "triptych" and discuss, possibly debate it:

- Are Hoare's statements applicable in your country, in your region, on your continent?
- How do we achieve Cumulative Progress?

2 Formal Methods

Some definitions and delineations may be in order:

[1]World Congress on Formal Methods (in the design of computing systems) to be held in Toulouse, France, 20–24 September 1999 and to be co-sponsored by ACM, AMAST, EATCS, ETAPS, FME [Formal Methods Europe], IEEE CS, IFIP, etc. See appendix B.

- Method:

  By a method we understand a set of principles for selecting and applying techniques and tools in order effectively to analyse and synthesise an effective artifact: here software.

- Formal Method:

  By a Formal Method we understand a set for Formal Specification and Design Calculi techniques.

  When we say 'formal method' we thus do not imply that the principles and all techniques and tools are formalised or even formalisable.

  A design calculi is usually a set of proof or specification transformation rules.

- Formal Specification:

  A Formal Specification is expressed in a Formal Language.

  It expresses, in a comprehensive and consistent way, a model of whatever it describes: A Model of an Application Domain (void of any reference to possible computing support), a Requirements (void of any reference as to how to implement these requirements), a Software Architecture (emphasising external, but concrete, implementation-biased, interfaces); a Program Organisations (detailing internal interfaces); etc.

- Formal Language:

  A Formal Language is presented by specifying, at a meta-level, (i) a syntax, (ii) a possibly model-oriented, for example denotational or structural operational semantics, and (iii) a proof system.

- Design Calculi:

  (i+ii) allows tools to be developed — tools that support the input, editing and syntactic and abstract interpretation processing of specifications. (iii) allows tools to be developed — tools that support reasoning (automated theorem proving, mechanically assisted proofs) about specifications.

- Formal Development:

  A development is formal iff all specifications are formal: all proof obligations have been identified and dutifully formally proved using the proof system.

- Rigorous Development:

  A development is rigorous if all specifications have been expressed in the formal specification language and if relevant theorems of correctness and other crucial properties have been formally expressed — but where proofs have at most been conducted in a way where the reasoning processes have referred to model-theoretic properties of the modelled domain, requirements or software — as the case may be.

- Systematic Development:

  A development is rigorous if all specifications have been expressed in the formal specification language and if reasonable relations between steps of development have been informally reasoned.

We refer to [1]. Here Jackson points out that professionalising software engineering seemingly requires a high degree of specialisation — *to which I add:* — and that within each highly specialised software engineering (viz.: compiler development, reactive/real-time & safety critical systems, workpiece systems, etc.) an "own" (set of) method(s) need evolve with their own distinct uses of formal methods, i.e. formal specification and design calculi techniques and tools.

3 Domain / Requirements / Software

Software Design: In order to design software we need requirements. *Requirements:* In order to express functional requirements we need clarify the domain.

- *Domain Engineering* embodies, it seems, systems identification: Recognising which are the relevant phenomena in the domain; capturing these; analysing and validating emerging domain models.

- *Requirements Engineering:*

  Requirements reside in the domain. Traditionally requirements express functional and non-functional properties.

- *Software Design Engineering:*

  Software design is based on requirements:

  - Software Architecture — restricted to reflecting only the external interface behaviour — implements functional behaviours (?)
  - Program Organisation — that part of the more generally understood concept of software architecture which (additionally) reflects internal interface behaviour — additionally implements non-functional requirements.

To what extent are formal methods relevant in each of these phases, stages, steps?

4 Regional/Continental Differences?

It seems to the chairman of this ICSE'98 Panel on Formal Methods that one can identify at least two, albeit seemingly related differences of approach between US and European "Formal Methods":

- One Step vs. Indefinite # of Steps:

  In the US there seems to be a predominant number of researchers and practitioners of so-called Formal Methods who primarily, or only, see formal development as that of formally specifying a piece of software and proving that software correct wrt. specification.

  In Europe, notably Northern Europe, there seems to be a predominant number of researchers and practitioners of so-called Formal Methods who primarily see development as a many stage refinement of abstract specifications towards concrete, eventually executable specifications.

- Tools vs. Specifications:

  In the US, perhaps as a consequence of the previous view, if correct, tools represent the essence of formal methods.

  In Northern Europe the essence of formal methods is that of specification: of problem domain, requirements and software design analysis.

- Executable Specifications vs. Intellectual Documents:

  And, perhaps again as a derived consequence of the above views, it seems that many "US" software engineering researchers and practitioners demand that specifications be executable, whereas their Norther European counterparts, in consequence of "confessing" to specifications as analytical, intellectual documents of knowledge do not expect executability.

Possible discussion/debate items could be:

- Has the panel chairman gone nuts?
- Is there a difference between the US and Europe?
- Is Australia like Europe or the US?
- Where in the Spectrum is Japan Located?

5 Formal Method Myths & Commandments!

A background also for the panel are the following publications [2, 3, 4]. See references at end of this panel statement.

5.1 Seven Myths of Formal Methods

In [2] Anthony Hall lists and dispels the following seven "Myths":

1. *Formal Methods can Guarantee that Software is Perfect*
2. *Formal Methods are all about Program Proving*
3. *Formal Methods are only Useful for Safety-Critical Systems*
4. *Formal Methods Require highly trained Mathematicians*
5. *Formal Methods Increase the Cost of Development*
6. *Formal Methods are Unacceptable to Users*
7. *Formal Methods are Not Used on Real, Large-Scale Software*

5.2 Seven More Myths of Formal Methods

In [3] Jonathan P. Bowen and Michael G. Hinchey continue dispelling myths:

8. *Formal Methods Delay the Development Process*
9. *Formal Methods are Not Supported by Tools*
10. *Formal Methods mean Forsaking Traditional Engineering Design Methods*
11. *Formal Methods only Apply to Software*
12. *Formal Methods are Not Required*
13. *Formal Methods are Not Supported*
14. *Formal Methods people always use Formal Methods*

5.3 Ten Commandments of Formal Methods

In [4] Jonathan P. Bowen and Michael G. Hinchey suggests ten rules of software engineering conduct:

I. *Thou shalt choose an appropriate notation*

II. *Thou shalt formalise but not over-formalise*

III. *Thou shalt estimate costs*

IV. *Thou shalt shall have a formal methods guru on call*

V. *Thou shalt not abandon thy traditional development methods*

VI. *Thou shalt document sufficiently*

VII. *Thou shalt not compromise thy quality standards*

VIII. *Thou shalt not be dogmatic*

IX. *Thou shalt test, test, and test again*

X. *Thou shalt reuse*

6 Panel Discussion

The panel will also discuss and debate:

- Adequacy and representability of Tony Hoare's statement, section 1
- Are the section 2 characterisations of formal methods adequate. and if not: what is missing — or is the focus entirely wrong?
- Are there geographical differences, wrt. formal methods (section 4), and if so Why?
- Relevance of 7+7 Myths and 10 Commandments, section 5
- What, if so deemed, must we do to secure continuous progress into the next century?

APPENDICES

A Web References to 'Formal Methods'

We refer to the following overall www reference to a repository on applied work on 'Formal Methods':

`http://www.csr.ncl.ac.uk/-projects/FME/InfRes/applications/methods.html`

Enjoy the surfing!

B FM'99: World Congress on Formal Methods

Please also surf to:

`http://www.it.dtu.dk/~db/fm99/cfp.ps`

Here you will find a call for paper on a world congress on formal methods. FM'99, in addition to a 10 stream foundations & methodology and industrial applications **technical symposium** will also feature a **tools fair and application forum**, a set of formal methods **user group meetings**, and **industry tutorials** on formal software development for railway systems, avionics, telecommunications, hardware, and many other application areas.

REFERENCES

[1] Michael Jackson. Problems, methods and specialisation. *Software Engineering Journal*, pages 249–255, November 1994. .

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

[3] J.P. Bowen and M. Hinchey. Seven More Myths of Formal Methods. Technical Report PRG–TR–7–94, Oxford Univ., Programming Research Group, Wolfson Bldg., Parks Road, Oxford OX1 3QD, UK, June 1994. Shorter version published in LNCS Springer Verlag FME'94 Symposium Proceedings.

[4] J.P. Bowen and M. Hinchey. Ten Commandments of Formal Methods. Technical report, Oxford Univ., Programming Research Group, Wolfson Bldg., Parks Road, Oxford OX1 3QD, UK, 1995.