What is a formal method, (and what is an informal method) ?

Les Hatton
Oakwood Computing*

March 4, 2005

Abstract

This short position paper traces a very personal view of formal methods in the period 1982-1997.

From Proceedings of the 12th International conference on Computer Assurance, 1997, Gaithersburg, MD, USA.

As this is a position paper, I will use the first person to describe my own experiences in formal methods all the way from outright belief in the power of mathematics in the early '80s, to a measurement-tempered and rather cautious optimism in the late '90s. Its a very personal and probably narrow view but there may be interesting lessons here for others and it will help to explain my reservations about formal methods.

In the early '80s, I managed a million line Fortran development in geophysical signal processing from the ground up. I had never done this before and like so much of our development in software engineering, I had to learn on the job. It is no wonder that so many of our projects fail. I read and read, trying desperately to convert myself from a mathematician / geophysicist into a software engineer. It didn't take me long to realise that for all the complexity of the Navier-Stokes fluid equations and solving coupled non-linear partial differential equations as I did when studying tornadoes as a research student, at least I generally knew what I was doing or could find somebody who did. I don't think I have ever known what I was doing in software engineering at the start of a project and in some memorable occasions, at the end also.

During this period, and like many others, I was strongly influenced by the pioneering work of Tony Hoare at Oxford. I became a strong advocate of formal methods believing that they were truly the way forward. For 2 or 3 years, I worked with such technologies and used them in practice. The high point of this was appearing in a BBC television program around 1985 on mathematical methods in software. Alonog with some very famous companies like IBM, and

*lesh@oakcomp.co.uk

of course, Tony, we took turns presenting our experiences. When our turn came, we had to stand in front of a camera with a white board full of suitably arcane mathematical symbols and in spite of all genetic odds, (I have a forehead like a ski slope, complete with moguls), look intelligent while Vivaldi's "Four Seasons", the international standard music for popular science thundered away in the background. As myself and a colleague stood gazing with steely eyes at the camera, gripping our marker pens with mathematical fervour, the cameraman said "There's an error in that". You could have heard a pin drop. He was right, I had an "and" instead of an "or" in one expression. The cameraman hadn't a clue about the application area we were working in, but he was doing a part-time Ph.D. in maths (the swine) and recognised the logical inconsistency in the propositional calculus we were using to specify the problem.

There followed a long embarrassed pause. The producer generously offered to edit it out as he said the looks on our faces were worth a million dollars but he had a deep sympathy for human frailty. I am very proud to report that after much soul-searching, we left it in, because it made the point about formal methods far better than any of the somewhat artificially engineered situations we were presenting. I still have a copy of the video tape and I do still look a complete idiot, an experience which has prepared me well for a career in software engineering, because I don't think you can function as a software engineer until you have really screwed something up at least once. I have been fortunate to have really screwed things up several times.

The net result of these experiences was that I knew that formal methods were highly beneficial in some circumstances but that they seemed to be only practically useable in certain circumstances and the symptom for this was a nagging feeling that you "couldn't get your mind round a problem". I think I know why 13 years later when studying the astonishing behaviour of the defect density curve with component size distribution in software systems, (its U-shaped, [3]).

So, from around 1987 onwards, I became increasingly more interested in why software systems seem to be so intractable, although they generally work OK. In all of this time, I feel though that formal methods has made little progress. I believe that this is a classic symptom of engineering systems when you solve the wrong problem first.

I saw this in its full glory for the first time when I studied an air-traffic control system with Shari Pfleeger in 1993/1994, ([5]). Here was the ideal case, a company with considerable experience with such systems and a management led drive to use them as effectively as possible. The resulting system is very good indeed, but provided very stong evidence that:-

1. Formal methods, in this ccase in the form of specifications, only seemed to have a benefit when accompanied by other technologies, in this case a comprehensive and objectively defined test plan.

2. the improvement in defect density wasn't that great, in this case just about a factor of 3 reduction, rather than the order of magnitude I was

expecting.

At about the same time, I attended a paper on a formally-specified Cobol parser written in C, [4]. In this case, the specification was written in a set-theoretic language called Z, which always reminds me of the effects of letting a flock of birds with inky feet loose in a paper factory, such is its notational obscurity. During this presentation, the speaker made the extraordinary statement that there were far more defects in the Z than there were in the resulting C.

So, where do I stand today ? Well, I believe we have far worse problems in software systems than those solved by formal methods. Its is as if formal methods affects the design, but we then cloud that design in a fog during implementation. Only if you blow the fog away with something such as 'thorough' objectively guided testing, can you see that a formal design is superior to an informal design.

This brings me nicely to the title of this position paper. It might be much more appropriate only to ask the questions "What is an informal method ?". Well, there can be no such thing. All programmers operate by logical manipulation in some representation in their own reasoning system. Historically, things in mathematics only get written down when things get tough or you are required to justify your reasoning for some reason, perhaps because the reader is not as fluent as the writer, or you are taking a maths exam or something. To me, we use formal methods all the time - programmers do not randomly drag symbols out of thin air and write them down, (although you could be excused for thinking this with some PC software). This may be why there is really not that much difference according to data like that of [5]. It may also explain the observation that some programmers naturally produce very high reliability code without writing anything down. They naturally reason in the programming language itself. After all, if you remove the poorly defined parts of a language by enforcing a suitable subset, you are left with a formal representation language, [1], [2].

I have no doubt that formal methods in their written form will improve things, but I don't believe by very much and not until we solve other worse problems first, and measurement will play a crucial role in identifying these.

References

[1] Hatton, Les (1995) *Computer programming languages and safety-related systems*, Proceedings of 3rd Safety-Critical Systems Symposium, ed. F.Redmill and T.Anderson, Springer-Verlag

[2] Hatton, Les (1995) *Safer C: developing for high-integrity and safety-critical systems*, ISBN 0-07-707640-0, McGraw-Hill

[3] Hatton, Les (1997) *Re-examining the fault density - component size connection*, IEEE Software, 14(2), (March/April)

[4] Ostrolenk G., Southworth M. et. al (1994) *Cost-effective evaluation of a COBOL parser using an operational profile*, CSR'94, Dublin, Ireland

[5] Pfleeger S.L., Hatton L. (1997) *Investigating the influence of formal methods*, IEEE Computer, 30(2), p.33-43, February