Formal Methods (FMs)
The Very Idea*
Some Thoughts

Daniel M. Berry, University of Waterloo
Formerly at: Technion, GMD-FIRST

***with apologies to James H. Fetzer**

Outline of Talk

- **Definitions**
- **Economics**
- **Requirements Engineering**
- **FMs & Requirements**
- **Second Time Phenomenon**
- **Importance of Ignorance**
- **Conclusions**

Where I am going?

You will see that I am generally in favor of FMs, but there are serious problems of which we must be aware.

I will show some unconventional ideas as to why FMs are successful when they are.

I will suggest that FMs help the most when the applier is most ignorant about the problem domain.

Hang on! It should be controversial and fun!

Foreword

Please note that I *believe* in FMs.

I use them.

I have even worked for a company that sells FM technology and applies FM to clients' system development problems, including for secure operating systems.

I did some fundamental work on the underlying theory a long time ago.

Definitions

SWICBS=Software-Intensive–Computer-Based System

Actually, I've never heard of a CBS that is not SWI, but the emphasis is important.

The most flexible part of a CBS is its SW; thus, it is the SW that gets changed every time!

RE=Requirements Engineering

Definitions

What is a FM?

For the purpose of this talk, I am trying to include in the realm of FMs anything anyone working in FM claims is a FM.

There are many levels of formality and completeness (of application, not the normal formal sense of the word).

Three Classes of FM

- **Verification**
- **Intensive Study of Key Problem**
- **Refutation**

Verification

- **formal specification requirements**
- **formal specification of design**
- **formal specification of code**
- **code**

Verification of consistency of formal requirements

Verification of equivalence between items

Intensive Study of Key Problem

Intensive study of one difficult aspect of requirements

e.g., security, safety.

- **formal specification of aspect-relevent requirements**
- **code**

Verification that the aspect-relevant code satisfies the formal specification

Refutation

Instead of trying to prove that the SWICBS meets its requirements, try to refute the claim that it does.

Cheaper, because all that is needed is one counter example.

- **model checking**

Cost Factors

Applying FMs drives costs up as high as:

- **2 fold with just formal spec and code**
- **2 fold with just intensive study and code**
- **3 fold with formal spec, model checking, and code**
- **5 fold with formal spec, consistency verification, and code**
- **10 fold with formal spec, consistency verification, code, and code varification**

Cost Factors, Cont'd

These costs are not necessarily bad.

There is some evidence that doing the verifications saves a lot on the coding itself, on the testing, and on later maintenance.

Economic Realities

For most software, FMs are just not worth the cost; you can get more than acceptable quality by inspection for up to 15% more and absolutely superb results by just doing the software twice at the cost of about 100% more.

However, for highly safety- and security-critical systems, for which the cost of failure is death or is considered very high, FMs are necessary and worth the cost.

Most Errors Introduced During Requirements Specification-1

Boehm [1981]: At TRW, 54% of all errors were detected after coding and unit test; and, 65-85% of these errors were allocatable to the requirements, design, and documentation stages rather than the coding stage, which accounted for only 25% of the errors.

Requirements Errors, Cont'd.

In many cases, erroneous behavior is actually required.

In other cases, no behavior is required, but what happens is not right.

Usefulness of Verification

So, it is not clear how useful is full code verification, the *most* expensive, if only 25% or fewer of the errors are introduced during development (and they are probably the easiest to fix).

Usefulness of Verification, Cont'd

It seems that it's more cost effective to spend 15% more than development costs (i.e., 115%) for development with inspections than to spend 10 fold for development with verification, just to eliminate the relatively few coding errors.

Therefore, the focus of FMs must be on requirements.

RE is Difficult

Fred Brooks says:

"The hardest single part of building a software system is deciding precisely what to build.... No other part of the work so cripples the resulting system if it is done wrong. No other part is more difficult to rectify later."

Formal Methods Myth:

Some FM evangelists claim:

If* only *you had written a formal specification of the system, you wouldn't be having these problems

Mathematical precision in the derivation of software eliminates imprecision

Reality

Yes, formal specifications are extremely useful in identifying inconsistencies in requirements specifications, especially if one carries out some minimal proofs of consistency and constraint or invariant preservation,

just as writing a program for the specification!

FMs do *not* find all gaps in understanding!

Reality, Cont'd

As Gordon and Bieman observe, omissions of functions are difficult to recognize in formal specifications,

... just as they are in programs!

Preservation of Difficulty

Indeed, Oded Sudarsky has pointed out the phenomenon of *preservation of difficulty*. Specifically, difficulties caused by lack of understanding of the real world situation are not eliminated by use of formal methods; instead the misunderstanding gets formalized into the specifications, and may even be harder to recognize simply because formal definitions are harder to read by the clients.

Bubbles in Wall Paper

Sudarsky adds that formal specification methods just shift the difficulty from the implementation phase to the specification phase. The "air-bubble-under-wallpaper" metaphor applies here; you press on the bubble in one place, and it pops up somewhere else.

One Saving Grace

Lest, you think I am totally against formal methods, they *do* have one positive effect, and it's a BIG one:

Use of them increases the correctness of the specifications.

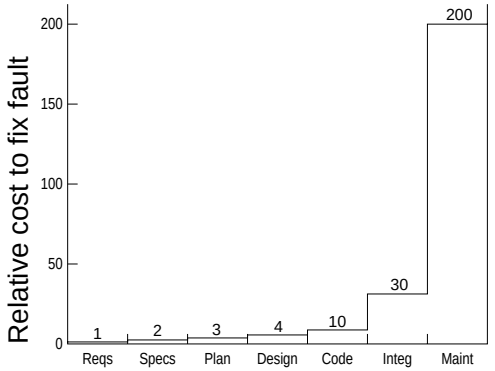
Therefore, you find more errors of commission at specification time than without them, saving considerable money for each bug found earlier rather than later.

Error Repair Costs

Remember: the cost to repair an error goes up dramatically as project moves towards completion and beyond ...

The next slide shows how dramatically this cost goes up.

Error Repair Costs, Cont'd



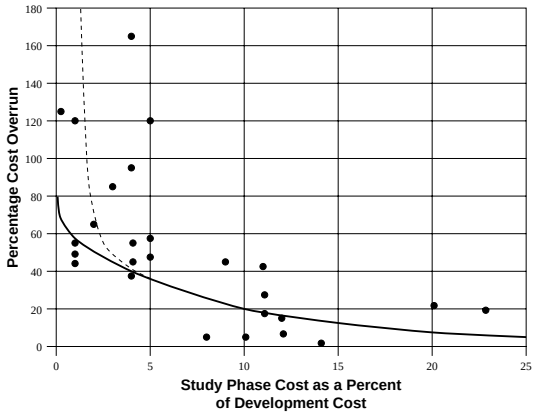
Phase in which fault is detected and fixed

RE & Project Costs

The next slide shows the benefits of spending a significant percentage of development costs on studying the requirements.

It is a graph by Kevin Forsberg and Harold Mooz relating percentage cost overrun to study phase cost as a percentage of development cost in 25 NASA projects.

Project Costs, Cont'd



Errors of Omission

But, if FMs are not so helpful to find errors of omission, what *is* helpful?

Errors of Omission, Cont'd

Having lots of smart people thinking, brainstorming, and talking about the requirements!

And you know? FMologists are pretty smart people.

So maybe having FMologists is more important than doing FMs.

Second Time Phenomenon

"Specification and Prototyping: Some Thoughts on Why They Are Successful"

{Daniel M. Berry, Jeannette M. Wing}
Proceedings of TAPSOFT Conference
pp. 117–128, Berlin, March 1985

Second Time, Cont'd

We believe that formal methods work, but *not* because of any inherent property of formal methods as opposed to just plain programming (which is really also a formal method).

Rather, because of the second time phenomenon, which is:

Second Time, Cont'd

If you do anything a second time around you do better, because you have learned from your mistakes the first time around.

Indeed, Fred Brooks says:

"*Plan* to throw one [the first one] away; you *will* anyway!"

In other words, you cannot get it right until the second time.

Second Time, Cont'd

If you write a formal specification and then you write code, you've done the problem formally two times.

Of course, the code will be better than if you had not done the formal specification.

It's the second time!

Two Formal Times

Note that doing it informally the first time and then writing code does not have the same effect.

It's too easy to handwave and overlook details and thus fail to find the mistakes from which you learn.

It's gotta be two *formal* developments, specifications *or* code, for the two-time phenomenon to work.

Requirements Centered

Observe how this is all requirements centered.

You are not going to fix implementation errors the second time around:

- **not the same implementation**
- **even if it were the same, you can introduce *new* errors in the rewrite**

The focus of the redoing is on understanding the essence and eliminating requirement errors.

"Importance of Ignorance in Requirements Engineering"

Daniel M. Berry
Journal of Systems and Software
28:2, 179–184, February, 1995

An RE Experience

In 1995, I was called in as a consultant to help a start-up write requirements for a new multi-port Ethernet switching hub.

I protested that I knew nothing about networking and Ethernet beyond nearly daily use of telnet, ftp, and netfind.

Earlier in my life, I had worried that the ether in Ethernet cables might evaporate!

An Experience, Cont'd

Despite my ignorance, I did a superb job, in fact, better than I normally do in my areas of expertise.

Empirical Observation

I noticed that I and my ex-wife did our best requirements engineering on projects in which we were most ignorant in the domain.

Ignorance is the Key

By being ignorant of the application area, I was able to avoid falling into the tacit assumption tarpit!

Ignorance is the Key, Cont'd

It was clear to me that the main problem preventing the engineers at the start-up from coming together to write a requirements document was that

- **all were using the same vocabulary in slightly different ways,**
- **none was aware of any other's tacit assumptions, and**
- **each was wallowing deep in his own pit.**

Ignorance is the Key, Cont'd

My lack of assumptions forced me

- **to ferret out these assumptions and**
- **to regard the ever so slight differences in the uses of some terms as inconsistencies.**

Need Ignorance

Our conclusion is that every requirements engineering team requires a person who is ignorant in the application domain, the ignoramus of the team, who is not afraid to ask questions that show his or her ignorance, and who *will* ask questions about anything that is not entirely clear.

Still Need Experts

We are not claiming that expertise is not needed.

***Au contraire*, you cannot get the material in which to find inconsistencies without the experts.**

Ignorance, Not Stupidity!

We are not claiming that the ignoramus is stupid.

***Au contraire*, he or she must be an expert in general software system structures and must be smart enough to catch inconsistencies in statements made by experts in fields other than his or her own.**

Recommendations

Each requirements engineering team needs

- **at least one domain expert, usually supplied by the customer**
- **at least one smart ignoramus**

Resumes of the Future

Resumes of future software engineers will have a section proudly listing all areas of ignorance.

This is the only section of the resume that shrinks over time!

The software engineer will charge fees according to the degree of ignorance: the more ignorance, the higher the fee!

Success Stories of FMs

The typical success story describes a FM person convincing a project to apply some particular FM.

The deal is that the FM person joins the team and either does or leads the formalization effort.

Success Stories, Cont'd

The reported experience shows the FM person slowly learning the domain from the experts by asking lots of questions and making lots of mistakes.

The end result is that the application of the FM found many significant problems earlier and the whole development was cheaper, faster, etc. than expected.

Failure Stories of FMs

I have not seen any.

Mathematicians as Ignoramuses

Martin Feather of JPL on Importance of Ignorance Paper:

I have often wondered about the success stories of applications of formal methods. Should these successes be attributed to the formal methods themselves, or rather to the intelligence and capabilities of the proponents of those methods?

Mathematicians, Cont'd

Typically, proponents of any not-yet-popularised approach must be skilled practitioners and evangelists to [bring the approach] to our attention. Formal methods proponents seem to have the additional characteristic of being particularly adept at getting to the heart of any problem, abstracting from extraneous details, carefully organizing their whole approach to problem solving, etc.

Mathematicians, Cont'd

Surely, the involvement of such people would be beneficial to almost any project, whether or not they applied "formal methods." Daniel Berry's contribution to the February 1995 Controversy Corner, "The Importance of Ignorance in Requirements Engineering," provides further explanation as to why this might be so.

Mathematicians, Cont'd

In that column, Berry expounded upon the beneficial effects of involving a "smart ignoramus" in the process of requirements engineering. Berry argued that the "ignoramus" aspect (ignorance of the problem domain) was advantageous because it tended to lead to the elicitation of tacit assumptions.

Mathematicians, Cont'd

He also recommended that "smart" comprise (at least) "information hiding, and strong typing ... attuned to spotting inconsistencies ... a good memory ... a good sense of language...," so as to be able to effectively conduct the requirements process.

Mathematicians, Cont'd

Formal methods people are usually mathematically inclined. They have, presumably, spent a good deal of time studying mathematics. This ensures they meet both of Berry' criteria. Mastery of a non-trivial amount of mathematics ensures their capacity and willingness to deal with abstractions, reason in a rigorous manner, etc., in other words to meet many of the characteristics of Berry's "smartness" criterium.

Mathematicians, Cont'd

Further, during the time they spent studying mathematics, they were avoiding learning about non-mathematics problem domains, hence they are likely to also belong in Berry's "ignoramus" category. Thus a background in formal methods serves as a strong filter, letting through only those who would be an asset to requirements engineering.

Real Value of FMs

Perhaps the real value of FMs is that they attract really good people, the formal methodologist, who is good at dealing with abstractions, who is good at modeling, etc., the smart ignoramus, into working on the development of your SWICBS.

Managers know that the success of a SWICBS development project depends more on personnel issues than on technological issues.

Failed Experiment

"Formal Methods Application: An Empirical Tale of Software Development", by Ann E. K. Sobel and Michael R. Clarkson, *IEEE Transactions on Software Engineering* 28:3, 157–161, March 2002

FMs vs. No FMs in Development

They arranged two groups of teams of university students

Each team in group:

1. **learned FMs and used them in a term-long project to develop a (common) program**
2. **not learned FMs and did term-long project to develop same program**

Results

1. **100% of programs produced by FM teams passed all of a set of 6 test cases.**
2. **only 45.5% of programs produced by nonFM teams passed all of same set of test cases.**

Wow!!

Conclusions

Sobel and Clarkson's Conclusions:

Since teams did not differ by all sorts of academic measures, the successes were due to the use of FMs

Wrong!

Walter Tichy and I independently spotted flaw in the reasoning (We ended up writing a joint note).

Voluntary Selection!

Only students who had voluntarily taken an optional course on FMs were in FMs teams.

NonFM teams consisted of only students who had *not* taken this FMs course.

Alternative Explanation

Berry and Tichy offered alternative theory for results:

The reason for the success was presence of the people who were interested in, and presumably skilled in, in FMs, abstract thinking, etc.

They program better naturally! course.

Alternative Explanation, Cont'd

The teams consisting of FMs users, whose programs passed all the tests, were just plainly and simply *better programmers* than the teams not containing any FMs users, whose programs did not pass all the tests.

No surprise there!

It's Hard to Experiment

It's really hard to devise a proper controlled experiment that can test that FMs are the cause of the difference, simply because in a University, it's not considered legitimate to force people to take a course as heavy as FMs.

My Message to FMologists

Forget about proving programs, i.e., code, correct; it's not cost effective:

- **it increases development cost by an order of magnitude;**
- **only 15–25% of all errors are introduced by coding; and**
- **numerous experiments show that inspection does a good job of eliminating coding errors for only 15% overhead.**

My Message, Cont'd

Focus on getting correct and complete requirements specifications, where 75–85% of the errors occur:

- **FMs applied to make the specifications more correct, i.e., to eliminate errors of commission**
- **FMologist applied to make the specifications more complete, i.e., to eliminate errors of omission**

Singers vs. Songs

Farhad Arbab of CWI reminded me of a famous line,

"It's the singer, not the song!",

and said

"It's the FMologist, not the FM!"

Conclusion

It is my belief that FMs work when they work, not so much because of formality, but rather because of

1. **what is learned when applying FMs, that can be applied in the next round of development and**
2. **the nature of the people who willingly and enthusiastically apply FMs.**

Formal versus Informal Requirements Specifications

a Discussion Panel

Daniel M. Berry's Position

Discussion is Somewhat Academic

If we *implement* a system, we write a formal specification (FS) for it, even if our requirements specification is informal.

The implementing code is formal.

Code is a Formal Spec

Whatever benefits or drawbacks we ascribe to a FS, applies also to code, e.g.

⊕ **unambiguous**
⊕ **writing it exposes flaws in reasoning**

⊖ **hard to read by clients/users**

Formality Works Both Ways

It both helps and hinders in getting at What issues:

⊖ **harder to read to many; so it is less helpful in promoting creativity**
⊖ **concern for formality can detract from our looking for things overlooked**

⊕ **formality makes handwaving our way to false beliefs of correctness harder**

Informal Meets Formal

In any CBS development the informal meets the formal somewhere:

- **when a FS is written from informal ideas**
- **•••**
- **when code is written from an informal spec**

Informal Meets Formal, Cont'd

imprecise, natural language expression of fuzzy, incomplete, ill-formed ideas

↓

precise, complete, and consistent specification of something approximating the ideas

Translation: Informal→Formal

Ambiguity in the informal expression of the ideas

before the ↓ can lead to

an incorrect formalization

even though the formal statement is unambiguous, complete, and consistent.

***Subconscious disambiguation*!**

Problems of RE

- **Figuring out what the requirements are**
- **Writing specifications for the requirements**
- **Validating specifications with client/users**
- **Understanding specified requirements**

Figuring out what reqs are

- **Understanding the real world (RW)**
- **Identifying all about the RW that is relevant**
- **Getting complete set of functions (no errors of omission)**
- **Getting them consistent with real world (no errors of commission)**
- **Getting them right**
  - **Right functionality**
  - **Right user interface**
  - **Right resilience**

Writing specs for reqs

- **Completeness (no errors of omission)**
- **Consistency (no errors of commission)**
- **Nonambiguous**

Validating specs with client/users

- **Does it say all what they want?**
- **Does it say only what they want?**
- **Do they really understand what it says?**

Understanding specs

- **Will implementers implement all the requirements?**
- **Will implementers implement only the requirements?**
- **Will implementers find all their questions answered?**

Formal vs. Informal Specs

Let's go back over the problems and see where the kind of specification you write makes a difference.

- ○ **No difference**
- ◑ **Some difference**
- ❍ **Some difference, maybe negative**
- ● **Lots of difference**

Figuring out what reqs are

- Understanding the real world (RW)
- Identifying all about the RW that is relevant
- Getting complete set of functions (no errors of omission)
- Getting them consistent with real world (no errors of commission)
- Getting them right
  - Right functionality
  - Right user interface
  - Right resilience

Writing specs for reqs

- Completeness (no errors of omission)
- Consistency (no errors of commission)
- Nonambiguous

Validating specs with client/users

- Does it say all what they want?
- Does it say only what they want?
- Do they really understand what it says?

Understanding specs

- Will implementers implement all the requirements?
- Will implementers implement only the requirements?
- Will implementers find all their questions answered?

Problems of RE

- **Figuring out what the requirements are**
- **Writing specifications for the requirements**
- **Validating specifications with client/users**
- **Understanding specified requirements**

Formal vs. Informal Specs

I think you will see that most of RE's tough problems are somewhat independent of the form of the specification.

The effect is only partial in many cases, and in some cases it may be negative.

I think that the toughest problem is figuring out *what* the requirements are, and that activity is totally independent of the form of the specification.

Tripping Up PhD Candidates

My favorite way to trip up a FMs PhD candidate at his or her defense is to find a mssing requirement in the nontrivial example the thesis invariably uses as an extended case study.

Tripping, Cont'd

Always works:

- **The student is embarrassed that the FM failed to find the missing requirement.**
- **The student is embarrassed that he or she failed to notice the missing requirement.**
- **The student has more work to do before filing the thesis.**
- **It's *so* easy to find such an overlooked requirement.**

Tripping, Cont'd

Sometimes, I find also an issue in the example for which common sense played more of a role in noticing than did the FM, and I make the student discuss that issue in a revision of the thesis.

This too is *so* easy to find.