

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

My current area of research is Requirements Engineering (RE).

Focus is on *how* to get requirements for a computer-based system (CBS).

Among other things, we are interested in

- *why* a method of getting at requirements works when it does, and
- *why* a method of getting at requirements fails when it does.

Outline of Talk

- Definitions
- Economics
- No Silver Bullet
- FMs & Requirements
- Second Time Phenomenon
- Importance of Ignorance

Outline, Cont'd

- DeMillo, Lipton, and Perlis
- Hawthorne Effect
- Brooks's Law
- Failed Experiment Proving FMs' Effectiveness
- 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.

Flip Flop

I seem to flip flop on issues!

After I have spent some slides demolishing an idea, I will seemingly flip and advocate a variation of the same idea.

After I have spent some slides advocating an idea, I will seemingly flop and demolish a variation of the same idea.

Flop Flip

Thus, if you violently disagree with me at some point, please hang on! I might just agree with you in the end.

Of course, if by the end of the talk you still disagree with me, then grill me!

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!

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) [$P \leftrightarrow C$ means "partial \leftrightarrow complete"].

Verification

- 1 $P \leftrightarrow C$ formal specification of requirements
- 2 $P \leftrightarrow C$ verification of consistency and basic requirements satisfaction of requirements specification
- 3 $P \leftrightarrow C$ formal specification of design
- 4 $P \leftrightarrow C$ verification of consistency of requirements and design specifications

Verification, Cont'd

- 5 $P \leftrightarrow C$ formal specification of code
- 6 $P \leftrightarrow C$ verification of consistency of (requirements), design, and code specifications
- 7 C code
- 8 $P \leftrightarrow C$ verification of consistency of (requirements, design, and) code specifications and the code

Intensive Study of Problem

- 1 **C** study of one difficult aspect of requirements
 e.g., security, safety,
- 2 **C** study of one difficult aspect of design
- 3 **C** study of one difficult aspect of code

Cost Factors

Applying FMs drives costs up as high as:

- 2 fold with just levels V1 and V7,
- 2 fold with just level IS1 and V7,
- 5 fold if just level IS2 and V7,
- 10 fold with just levels V1–V4 and V7,
- and even higher if you go further.

Cost Factors, Cont'd

These rules of thumb were used by my boss as initial guesses in making contract-winning estimates that left a profit.

They work on total project costs, including for documentation, tests, etc.

Obviously, at different places, the data may be different.

For this talk, the scale of things is more important than the the exact numbers.

Cost Factors, Cont'd

These costs are not necessarily bad.

It is possible that doing levels V1–V4 saves a lot on level V7, on the coding itself, on the testing, and on later maintenance.

There is some evidence that this is so.

Other Directions in FMs

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.

Refutation

Given level 1 specifications

- type checking in spec
- cross reference checking in spec
- model checking of spec to find errors

Cost of Other Directions

These cost that of levels V1 and V7 above plus only 5–50% for the refutation, i.e., to 2.05–2.50 fold, which is cheaper than with full verification.

Economic Realities

For most software, it is 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.

Simplifications for Analysis

David Notkin makes the point about model checking:

Sometimes it is necessary to make simplifying assumptions to get a tractable model that can be checked.

Dilemma of Simplification

Dilemma:

- Without simplifications, cannot analyze and might overlook critical problems
- With simplifications, might overlook critical problems

A Cost Issue

In the end, it is an issue of costs.

Which problems cost more?

the ones overlooked by lack of analysis
OR
the ones overlooked by simplification

Nu?!

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 are levels V8 and V8, 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 (more later).

Important Fact

Remember that a program itself is a formal specification.

The programming language is a formally defined language with precise semantics just like Z, in fact, even more so than Z, which purposely leaves some things undefined.

One could not *prove* the consistency of specifications and code if code were not formal!

Programming as FM

Programming itself is a FM in the sense that writing a formal specification is a FM!

Remember that programming is building a theory from the programming language and library of abstractions (the ground) up, just like making new mathematics.

No Silver Bullet (NSB)

Fred Brooks says:

“There’s no silver bullet!”

He classifies software difficulties into two groups.

- Essence
- Accidents

Essence

- The *essence* of building software is devising the conceptual construct itself.
- This is very hard.
 - arbitrary complexity
 - conformity to given world
 - changes and changeability
 - invisibility

Accidents

- Most productivity gain came from fixing *accidents*
 - really awkward assembly language (HLLs)
 - severe time and space constraints (big & fast computers)
 - long batch turnaround time (TSOSs and PCs)

More Accidents

- clerical tasks for which tools are helpful (make, rcs, xref, spell, grep, fmt)
- drudgery of programming user interfaces (X, GUI libs, Java)

The Essence is Tough!

- However, the essence has resisted attack!

We have the same sense of being overwhelmed by the immensity of the problem and the seemingly endless details to take care of,

and we produce the same kind of poorly designed software that makes the same kind of stupid mistakes

as 35 years ago!

Another View

Another way to describe the essence is “requirements”, not specifications, which are just a statement of requirements, but the requirements themselves.

FMs just do not help identify requirements, i.e., do not help us crack the essence.

Theory Building

How many of you have developed a complete theory from the ground up in order to prove a bunch of theorems?

Do you recall the difficulties you had in

1. deciding which true statements of the domain should be
 - definitions,
 - axioms, and
 - theorems?

Theory Building, Cont'd

2. deciding the wording of theorems?
3. deciding the order in which to state and prove theorems?
4. carrying out proofs of the theorems?

Recall how iterative the process was, how as you were carrying out one instance of Step 4, you had to stop and go back and redo Steps 1, 2, and 3.

Theory Building, Cont'd

Well, if you are like me, the difficulties of 1, 2, and 3 together exceeds the difficulty of 4.

1, 2, and 3 discover the essence of the theory, deciding *what* to prove.

To do 4, you have to know *how* to prove theorems.

1, 2, and 3 are the RE stage of theory building.

4 is the implementation stage.

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!

Reality, Cont'd

von Neumann and Morgenstern (*Theory of Games*) say,

“There’s no point to using exact methods where there’s no clarity in the concepts and issues to which they are to be applied.”

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:

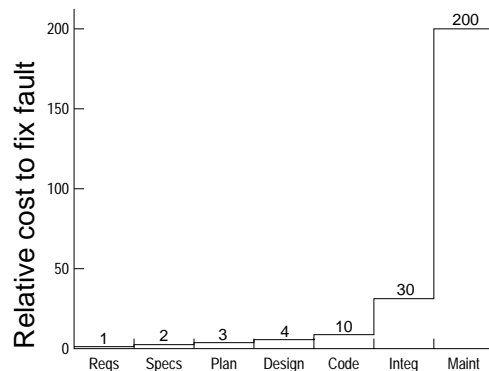
Use of them increases the correctness of the specifications.

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.



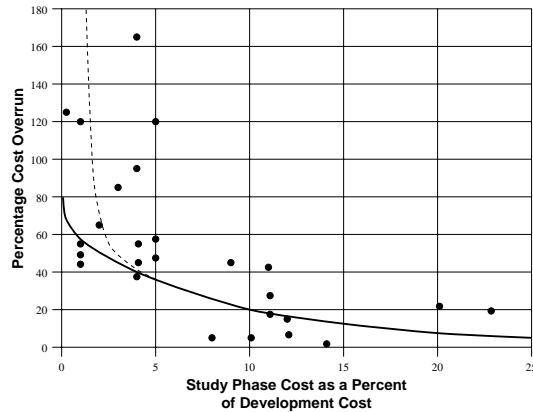
Phase in which fault is detected and fixed

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



Project Costs, Cont'd

The study, performed by W. Gruhl at NASA HQ includes such projects as

- Hubble Space Telescope
- TDRSS
- Gamma Ray Obs 1978
- Gamma Ray Obs 1982
- SeaSat
- Pioneer Venus
- Voyager

Some Advantages of Project Delay

Arnis Daugulis reports a significant increase in the quality of the requirements for a power plant control system as a result of delays in start of production caused by shortage of funds.

They had the luxury of time to do two revisions of the requirements.

He shudders to think of the failure that would have resulted had they started to implement the first requirements on time.

Other Benefits of FMs

In addition, formal specifications pay off in making it easier

- to generate test cases, and
- to write other documentation.

Changes Inevitable

Another reason FMs do not help identify requirements is that requirements always change—it's inherent in the software—and formalization requires freezing the requirements long enough to write the specification and carry out the verifications.

E-type Software

Meir Lehman identifies concept of E-type system

It is a system that solves a problem or implements an application in some *real world* domain.

Once installed, an E-type system becomes inextricably part of the application domain, so that it ends up altering its own requirements.

E-type Software, Example

- Consider a bank that exercises an *option* to automate its process and then discovers that it can handle more customers.
- It promotes and gets new customers, easily handled by the new system but beyond the capacity of the manual way.
- It cannot back out of automation.
- The requirements of the system have changed!

E-type Software, Cont'd

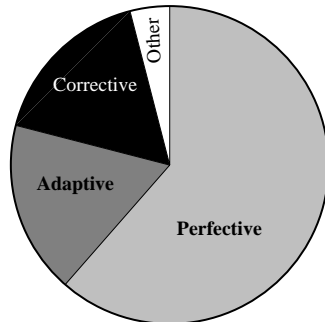
Daily use of a system causes an irresistible ambition to improve it as users begin to suggest improvements.

Who is not familiar with that, from either end?

Formalization of the requirements does nothing to make the details of these kinds of changes more predictable.

E-type Software, Cont'd

In fact, data show that most maintenance is *not* corrective, but for dealing with E-type pressures!



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

In the 20th anniversary release of the *Mythical Man Month*, Brooks admitted, “This [throw the first one away] I now perceive to be wrong”.

However, it is wrong not because you don't do better the second time, but because it implies the use of the waterfall model, which is understood as too simplistic.

Second Time, Cont'd

He favors a more incremental approach, including prototyping, which itself causes the production implementation to be the second formal time on the prototyped aspects.

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.

Euripedes said:

"Second thoughts are always wiser"

Importance of Ignorance

Based on:

“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!

Dilbert’s Ph.B.

I am reminded of a Dilbert cartoon, in which Wally tells his Ph.B. that the Ph.B.’s connection to the network is broken and says, “Uh-oh, it’s a ‘Token Ring’ LAN. That means the token fell out and it’s in this room someplace.”

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.

More on Smartness

Smart =

- is good at getting to the bottom of things,
- can sniff out inconsistencies in different people's tacit assumptions,
- understands abstraction and how computer-based systems are built,
- is not afraid to ask so-called stupid questions to expose all tacit assumptions, etc.

Recommendations

Each requirements engineering team needs

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

Ignorance A Point of View?

Scott Adams has written a book titled *Dilbert: When did Ignorance become a point of view?*

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!

The Optimist

From the Berlitz Italian Book:

The head of an important firm, looking at an application, is astonished when he notices that the applicant, though lacking experience, asks for a high salary.

Rather puzzled, he asks him, “Doesn’t it seem to you that you are asking for an excessive salary, considering the little experience you have?”

The Optimist, Cont’d

“On the contrary,” replies the applicant. “Work performed by one who knows nothing about it is harder and should be better paid.”

The Hidden Essence

Don Batory puts it another way:

Ignorance helps you see the hidden essence of the problem, that the domain experts are hopelessly beyond.

The ignoramus’s mind has not been poisoned by expertise.

Expert Theorem Provers

J. Strother Moore, one of the co-authors of the Boyer-Moore Theorem Prover (BMP), observes the behaviors of people guiding the BMP in a failing attempt to prove a non-theorem.

The author of the theorem persists in fruitlessly trying different approaches much longer than another person, who is seeing the non-theorem for the first time.

Ignorant Theorem Provers

The ignorant prover not only more quickly accepts that the non-theorem cannot be proved, but also more quickly sees why the non-theorem cannot be proved. He or she more quickly finds a correct theorem.

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’s 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.

An Implication

An attempt to train non-mathematically mature domain experts to apply formal methods in their domain is not likely to succeed.

Here, you have “dumb” experts, dumb in the sense of mathematically naive.

You need smart ignoramuses.

Another Implication

I believe that automatically generating code from a specification compromises the second-time benefits of writing the code manually from the specification.

IV&V

Arndt von Staa observes that:

Independent Verification and Validation teams are smart ignoramuses with respect to the domain areas of the SWICBS that they verify and validate.

Proper Context for FMs

So maybe we have identified the proper context for FMs.

It's place is in the highly *social* process of requirements engineering.

This reminds me of the 1979 DeMillo, Lipton, and Perlis paper, “Social Processes and Proofs of Theorems and Programs”.

DeMillo, Lipton & Perlis

They observed that mathematical proofs work because of the social processes in and around them that help to insure that only correct theorems get published (and even then they are not all correct).

They argued that the proofs required by FMs applied to programming are generally carried out by grunt mathematicians working alone and without the benefit of social interaction.

Social Interaction

Why? because, unlike publishable proofs in mathematics, proofs about programs are quite simply and frankly *boring*.

Proofs without social processes are not trustworthy enough for the needs of FMs.

Bored grunt mathematicians make mistakes!

Theorem Provers

The verification community has actually solved this problem, by replacing grunt mathematicians by theorem proving programs that never get bored and, once they have matured as programs, never make mistakes.

Theorem Provers

Today, no one advocating FMs would even suggest that humans would do any verification, except of the most interesting theorems, e.g., in an intensive study approach.

But that is not my point here.

Inspections

The use of formal inspections is an attempt to inject

- social processes (inspections are fun!)
- IV&V with smart ignoramuses

into the development of a SWICBS.

Another Possible Explanation

Ric Hehner offered another possible explanation for the FMs success stories, at least those that involve a FMs evangelist joining a real project in an experimental application of the FM.

Hawthorne Effect

A study in the 1930s in Hawthorne, Illinois discovered that the act of merely studying individual behavior can impact it.

The participants in an experimental application of FMs may try harder and succeed simply because they are in an experiment.

Burkinshaw's Observation

About 6 years after publishing the "Importance of Ignorance" paper, I was told about the quotation of P. Burkinshaw, an attendee of the Second NATO Conference on Software Engineering in Rome in 1969 (Buxton and Randell, 1969)

Burkinshaw's Observation, Cont'd

Get some intelligent ignoramus to read through your documentation and try the system; he will find many "holes" where essential information has been omitted. Unfortunately intelligent people don't stay ignorant too long, so ignorance becomes a rather precious resource.

Burkinshaw's Observation, Cont'd

Interestingly, this quotation was Burkinshaw's *only* recorded entry in the proceedings.

Thus, he had discovered the importance of ignorance long before I had.

But there is more to Burkinshaw's quotation, one more sentence, in fact.

Burkinshaw's Observation, Cont'd

Suitable late entrants to the project are sometimes useful here.

Brooks's Law

This additional sentence calls to mind Brooks's Law (Brooks, 1975; Brooks, 1995).

Adding manpower to a late software project makes it later.

Brooks's Law, Cont'd

Fred Brooks explains:

- the lines of communication within a team grows quadratically with team size,
- the number of hours available for work grows linearly with team size.

Brooks's Law, Cont'd

Consequently, for any project, at some number n of team members,

hours needed for additional communication required by an additional team member

\geq

hours that the additional team member adds to hours available for work on the project.

Brooks's Law, Cont'd

Brooks's law suggests that most large CBS development projects are already staffed beyond n .

Smart Ignorance and Brooks's Law

Burkinshaw's observation suggests a way that the bad effects of Brooks's law may be made an advantage.

Yes, you will pay a delay, but you will get the benefits of ignorance.

So you have to decide which is higher,

- the cost of the delay or
- the value of the ignorance.

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

Attempt to empirically prove the effectiveness of FMs in producing quality software.

FMs vs. No FMs in Development

They arranged two groups of teams of university students

Each team in group number

- 1 learned FMs and used them in a term-long project to develop a program**
- 2 did not learn 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!

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 whether FMs, and not properties of the subjects, are the cause of the difference.

Also, in a university, it's not considered legitimate to force people to take a course as heavy as and as advanced 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

My Message, Cont'd

Thanks to Jan Rutten, a part-time FMologist at CWI for asking this question.

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.**

