

Unpanel: How Useful Are Formal Methods in Requirements Engineering?

Daniel M. Berry
University of Waterloo, Canada
dberry@uwaterloo.ca

Vocabulary

CBS = Computer-Based System

SW = Software

PL = Programming Language

FM = Formal Method

RE = Requirements Engineering

More Terminology

I talk about methods, approaches, artifacts, and tools as technology that help us develop CBSs. I use “method” to stand for all of them so I don’t have to keep saying “method, approach, artifact, or tool” in one breath.

We

In the following, at any time, ...

“We” = all the people in the field in whose viewpoint is the current slide.

So “we” is context dependent.

Every slide is in the RE field except those in the FM field, marked with  in the upper right hand corner.



Foreword

























Please note that I *believed* in FMs.

I used them and still occasionally still use lightweight versions of them.









A long time ago ...

Look at the titles of my early (late '70s thru '80s) papers and students' PhD titles.





1985





- [j21]     Daniel M. Berry:
A Denotational Semantics for Shared-Memory Parallelism and Nondeterminism. Acta Informatica 21: 599-627 (1985)
- [j20]     Shahrzade Mazaher, Daniel M. Berry:
Deriving a Compiler From an Operational Semantics Written in VDL. Comput. Lang. 10(2): 147-164 (1985)
- [j19]     Daniel M. Berry:
An Ina Jo® proof manager for the formal development method. ACM SIGSOFT Softw. Eng. Notes 10(4): 19-25 (1985)
- [j18]     Cary Buchman, Daniel M. Berry, Jakob Gonczarowski:
DITROFF/FFORTID, An Adaptation of the UNIX DITROFF for Formatting Bidirectional Text. ACM Trans. Inf. Syst. 3(4): 380-397 (1985)
- [j17]     Shaula Yemini, Daniel M. Berry:
A Modular Verifiable Exception-Handling Mechanism. ACM Trans. Program. Lang. Syst. 7(2): 214-243 (1985)
- [c11]     Daniel M. Berry, Jeannette M. Wing:
Specification and Prototyping: Some Thoughts on Why They Are Successful. TAPSOFT, Vol.2 1985: 117-128

1983

- [j16]     Nancy G. Leveson, Anthony I. Wasserman, Daniel M. Berry:
BASIS: A Behavioral Approach to the Specification of Information Systems. Inf. Syst. 8(1): 15-23 (1983)
- [j15]     Daniel M. Berry:
A New Methodology for Generating Test Cases for a Programming Language Compiler. ACM SIGPLAN Notices 18(2): 46-56 (1983)









1982

- [j14]     Daniel M. Berry, Carlo Ghezzi, Dino Mandrioli, Francesco Tisato:
Language Constructs for Real-Time Distributed Systems. Comput. Lang. 7(1): 11-20 (1982)





- [j28]     Christine Aguilera, Daniel M. Berry:
The use of a repeated phrase finder in requirements extraction. J. Syst. Softw. 13(3): 209-230 (1990)

[←] 1980 – 1989 

















1989

- [j27]     Kris K. Abe, Daniel M. Berry:
indx and findphrases, A System for Generating Indexes for Ditroll Documents. Softw. Pract. Exp. 19(1): 1-34 (1989)
- [c12]     Yoëlle S. Maarek, Daniel M. Berry:
The use of lexical affinities in requirements extraction. IWSSD 1989: 196-202

1988

- [j26]     Zeev Becker, Daniel M. Berry:
TRIROFF, an Adaptation of the Device-Independent TROFF for Formatting Tri-directional Text. Electron. Publ. 2(3): 119-142 (1988)

1987

- [j25]     Bijan Arbab, Daniel M. Berry:
Operational and Denotational Semantics of Prolog. J. Log. Program. 4(4): 309-329 (1987)
- [j24]     Daniel M. Berry, Nancy Yavne, Moshe Yavne:
Application of program design language tools to abbot's method of program design by informal natural language descriptions. J. Syst. Softw. 7(3): 221-247 (1987)
- [j23]     Shaula Yemini, Daniel M. Berry:
An Axiomatic Treatment of Exception Handling in an Expression-Oriented Language. ACM Trans. Program. Lang. Syst. 9(3): 390-407 (1987)
- [j22]     Daniel M. Berry:
Towards a Formal Basis for the Formal Development Method and the Ina Jo Specification Language. IEEE Trans. Software Eng. 13(2): 184-201 (1987)

Supervised Doctoral Dissertations

1. Meyers, Barbara F., "Design of a Language for an Associative Processor," 1975 (UCLA)
2. Schwartz, Richard L., "An Axiomatic Definition of ALGOL 68," 1978 (UCLA)
3. Erlinger, Michael, "Analysis of Retention Storage Management for Generalized Block Structure Languages," 1979 (UCLA)
4. Kemmerer, Richard A., "Verification of the UCLA Security Kernel: Abstract Model, Mapping, Theorem Generation, and Proof," Co-advised, 1979 (UCLA)
5. Leveson, Nancy G., "Applying Abstract Data Type Methodology to Data Base Systems," Co-advised, 1980 (UCLA)
6. Mishergghi, Shakib H., "An Investigation of The Architectural Requirements of SIMULA 67," 1980 (UCLA)
7. Penedo, Maria Heloisa, "The Use of a Module Interconnection Specification Capability in the synthesis of Reliable Software Systems," Co-advised, 1980 (UCLA)
8. Yemini, Shaula, "The Replacement Model for Modular, Verifiable Exception Handling," 1980 (UCLA)
9. Paolini, Paolo, "Abstract Data Types and Data Bases," 1981 (UCLA)
10. Schwabe, Daniel, "Formal Techniques for the Specification and Verification of Protocols," 1981 (UCLA)
11. Mazaher, Shahrzade, "An Approach to Compiler Correctness," 1981 (UCLA)
12. Burstin, Meir D., "Requirements Analysis of Large Software Systems," Co-advised, 1985 (Tel Aviv University)
13. Worley, Duane R., "A Methodology, Specification Language, and Automated Support Environment for Computer Aided Design Systems," 1986 (UCLA)
14. Krell, Eduardo A., "A SARA-based Ada Programming Support Environment," 1986 (UCLA)
15. Lor, Edward Kar-Wing, "A Requirement-Driven System Design Environment," 1988 (UCLA)



Foreword, Cont'd

I consulted for a company, SDC, that applied its FM to clients' critical-system development problems, including for secure operating systems.

I did and published some fundamental work on the underlying theory of SDC's FM.

Security Work

From all this security work and from its community that included such people as Peter Neumann, I learned a lesson that goes right to the essence of RE:

There is no way to add security to any CBS after it is built; the desired security must be *required from the beginning* so that security considerations permeate the entire development lifecycle.

My Motivation to Do this Unpanel

**I am occasionally asked to referee a FMs
paper, and**

I occasionally hear a FMs talk.

Motivation, Cont'd

I am struck by how little has changed from 1970s. I read or get a sense of:

- **Here's a new approach to formalize X . (X is the same as in 1970s)**
- **If only developers would listen to us!**
- **We're on the verge of a breakthrough that will convince developers to use FMs.**

It seems to be all the same as in the 1970s and 1980s.

Motivation, Cont'd

The use of LLMs and ChatGPT to convert requirements specification into code

reminds me of

the automatic programming craze in the '70s and early '80s, by Bob Balzer, *et al.*

Same technical difficulties.

Same cost issues.



Motivation, Cont'd

Don't get me wrong!

There *have* been advances in methods, e.g.,

**SAT provers,
refutation,
model checking,
domain-specific formalizations
alloy,
etc.**



Motivation, Cont'd

Some substantial SW has been developed, verified to meet their specs, e.g.,

**secure kernels of operating systems,
security applications,
network protocols,
secure voting applications,
etc.**



Unlike Some FMers

I was always writing software for real-world applications:

- **medium-sized CBSs by myself or with or by my students, and**
- **large-sized CBSs as part of their teams**



Such as

- **matchmaking for a party (before knew about FMs)**
- **tools for regression analysis for chemists (before knew about FMs)**
- **bi-directional formatter**
- ***proof updater for SDC's suite of FM tools***
- **bi-directional editor**
- **tri-directional formatter**
- **letter stretching bi-directional formatter**



Never Actually *Used* FMs

I never even *considered* using FMs to develop any *real* SW ...

even for the proof updater for SDC's suite of FM tools.

Knowing what I knew about developing these systems, I would have been crazy to.

But, I did use my FM-based skills of abstraction and modeling to my advantage.



Never *Used* FMs, Cont'd

Apparently, neither did other developers of FM tools (at least the ones I knew, including the developers of SDC's suite).

This seemed to be one of the dirty, dark secrets among FM tool builders.

No one in his right mind would consider *using* FMs to build these tools.

The perception was that it would just take too long, and they might never finish.



FM's For Only Small Programs

So, FM's could be used only for the development of *small* programs.

Operating system kernels and trusted system kernels *are* small programs.

So some FMers began a push to get all programs to be small!



Hoare on Small Programs

Tony Hoare said (in mid 1970s),

“Inside every large program is a small program struggling to get out.”

I got in to the habit of trying to identify the central algorithm, the small program, at the heart of each of my programs.

Having done so, still the program was messy and the programming was hard.

Failings of FMs

Even as FMs applied to Security taught me the fundamental essence and necessity of RE,

FMs have proved incapable of

- **dealing adequately with the kinds of CBSs that we need to build, and**
- **doing what we need to do in RE.**

I explore why here.

FMs Not Deal With CBSs That We Build

Let's see what Tony Hoare says.

Tony Hoare's Reversal

From Tony Hoare's Wikipedia page:

https://en.wikipedia.org/wiki/Tony_Hoare

For many years under his leadership his Oxford department worked on formal specification languages such as CSP and Z. These did not achieve the expected take-up by industry, and in 1995 Hoare was led to reflect upon the original assumptions:[24]

Tony Hoare's Reversal, Cont'd

“Ten years ago, researchers into formal methods (and I was the most mistaken among them) predicted that the programming world would embrace with gratitude every assistance promised by formalisation to solve the problems of reliability that arise when programs get large and more safety-critical. Programs have now got very large and very critical — well beyond the scale which can be comfortably tackled by formal methods.

Tony Hoare's Reversal, Cont'd

There have been many problems and failures, but these have nearly always been attributable to inadequate analysis of requirements or inadequate management control. *It has turned out that the world just does not suffer significantly from the kind of problem that our research was originally intended to solve. [Italics are mine]*

Tony Hoare's Reversal, Cont'd

[24] Hoare, C. A. R. (1996). “Unification of Theories: A Challenge for Computing Science”. Selected papers from the *11th Workshop on Specification of Abstract Data Types Joint with the 8th COMPASS Workshop on Recent Trends in Data Type Specification*. Springer-Verlag. pp. 49–57. ISBN 3-540-61629-2.

Hoare on Small Programs

Tony Hoare once said (in mid 1970s),

“Inside every large program is a small program struggling to get out.”

Later (in early 2000s) he added,

“the small program can be found inside the large one only by ignoring the exceptions.”

Now I Understand

Now I understand that what I was observing about the distribution of code is normal.

Distribution of Code

10–20% of the code = central algorithm.

80–90% of the code = exceptional details.

99.99% of execution time is spent in the central 10–20% of the code.

Distribution of Code

“99.99% of execution time is spent in the central 10–20% of the code.”

It’s hard to test the exceptional details code, the 80–90% of the code, because it gets executed less than 0.01% of the execution time, ...

and correcting a defect in this 80–90% of the code likely involves changes in related fragments scattered over the entire code.

Formal Model Still Useful

Hoare continues,

It is not the intention of this note to deprecate the value of mathematical modeling in addition to program design. Without a mathematical model, everything would be an exception.

FMs Not Doing What RE Needs

**RE concerns validation more than verification,
...**

but FMs deal with ...



Verification, ...

FMs have the power to put

**verifying the correctness of a CBS
implementation w.r.t. its specifications**

on a much firmer basis than is possible with

**testing the CBS w.r.t. its specifications with
well-chosen test data.**

..., but Not Validation

**However, this power does *very little* towards
validating the specifications w.r.t. its
customer's needs and wants,
i.e., its customer's requirements.**

And Here's Why

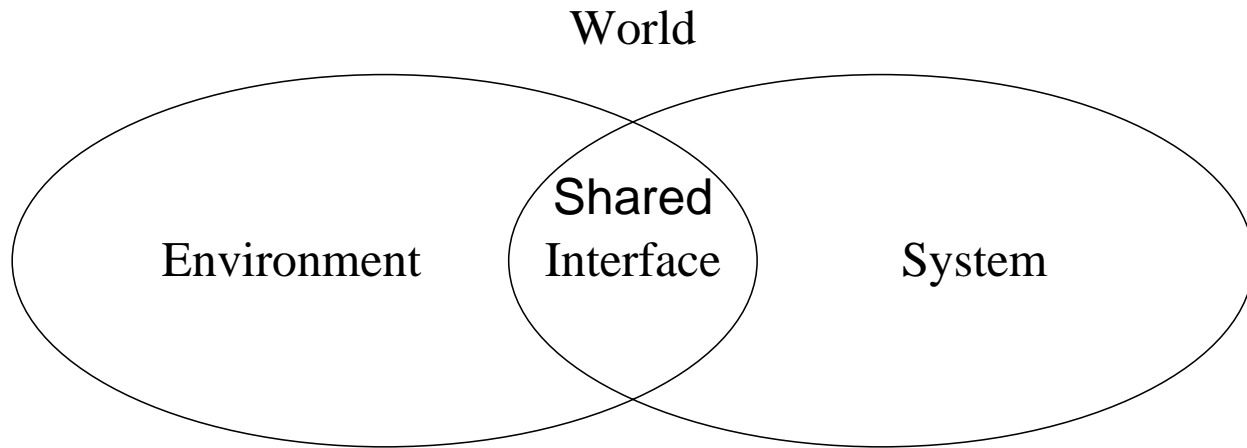
The next bunch of slides are about what has become known as the Reference Model for Requirements and Specifications by Gunter, Gunter, Jackson, and Zave, or the RE Reference Model.

The World and the CBS

The world in which a CBS operates is divided into

- **an Env, the environment affecting and affected by the CBS, and**
- **a Sys, the CBS itself, that intersect at their**
- **Intf, their Interface, and**
- **the rest of the world.**

The World and the CBS



Not Precise

While Sys, the CBS, is formal (mathematical),

**the rest of the world, including Env, is
*hopelessly informal,***

**and the boundaries of Env are *hopelessly
fuzzy:***

Butterfly in Rio → Golden Gate Bridge

So finding all details to not ignore is hard.

Famous Validation Formula

The informality has been made formal in the Zave–Jackson Validation Formula (ZJVF):

$$D, S \vdash R$$

D Domain Assumptions, in Env, informal

S System Spec, in Intf, can be formal

R Requirements, informal, in Env, informal

Truth of each of *D* and *R* in Env is *empirical*.

Sys Spec Formal?

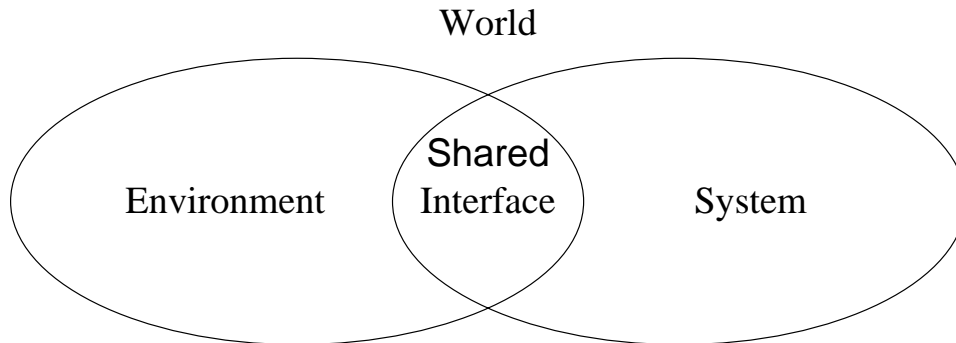
S is formal, if it is about a program written in a PL.

If program is molecular, then even S is informal, and its truth is empirical.

If program is the result of machine learning, then S is effectively informal, and its truth is empirical because the program's behavior depends on the learning data in ways that defy formalization.

Where Are the Exceptions?

From where is that 80–90% of the code = exceptional details?



From the Env, but not from the outside World!

But are we sure that it's not from the outside World?

Well ... um ...

Example: Airplane

Sys = airplane

Env = the sky

World = everything not relevant

Are the following in the Env:

- **flying bird?**
- **something in the hand of someone on the ground?**

The boundaries of Env are *hopelessly fuzzy*, making formalization of Env even harder.



FMs are Helpful When ...

FMs are helpful when the focus is on getting the central algorithm right, when the problems are entirely inside the Sys, e.g., for

- **network protocols,**
- **security kernels,**
- **velc.,**

in which, e.g., swapping the order of two steps could make a deadlock or a security leak.

These are CBSs for which D is identically true .

FMs are Not Helpful When ...

FMs are not helpful, and even give false hopes, when the focus is on getting the interaction of the Sys with the Env right, e.g., in

- **process control, e.g., on an aircraft,**
- **cyber-physical systems,**
- **velc.,**

in which the Sys's response to an unexpected input is failure.

Unexpected Input?

The problem with unexpected input from what was *thought* to be outside of Env:

“If no one happens to think of it, it just ain’t gonna be there in the spec.”

Now Your Turn!

OK,

I've had my initial say!

It's *your* turn now!!!!



What *Does* Work?

Good people, not good methods!

The remaining slides, not shown during the actual talk, capture my answer to one question.



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.



Real Value of FMs

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

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