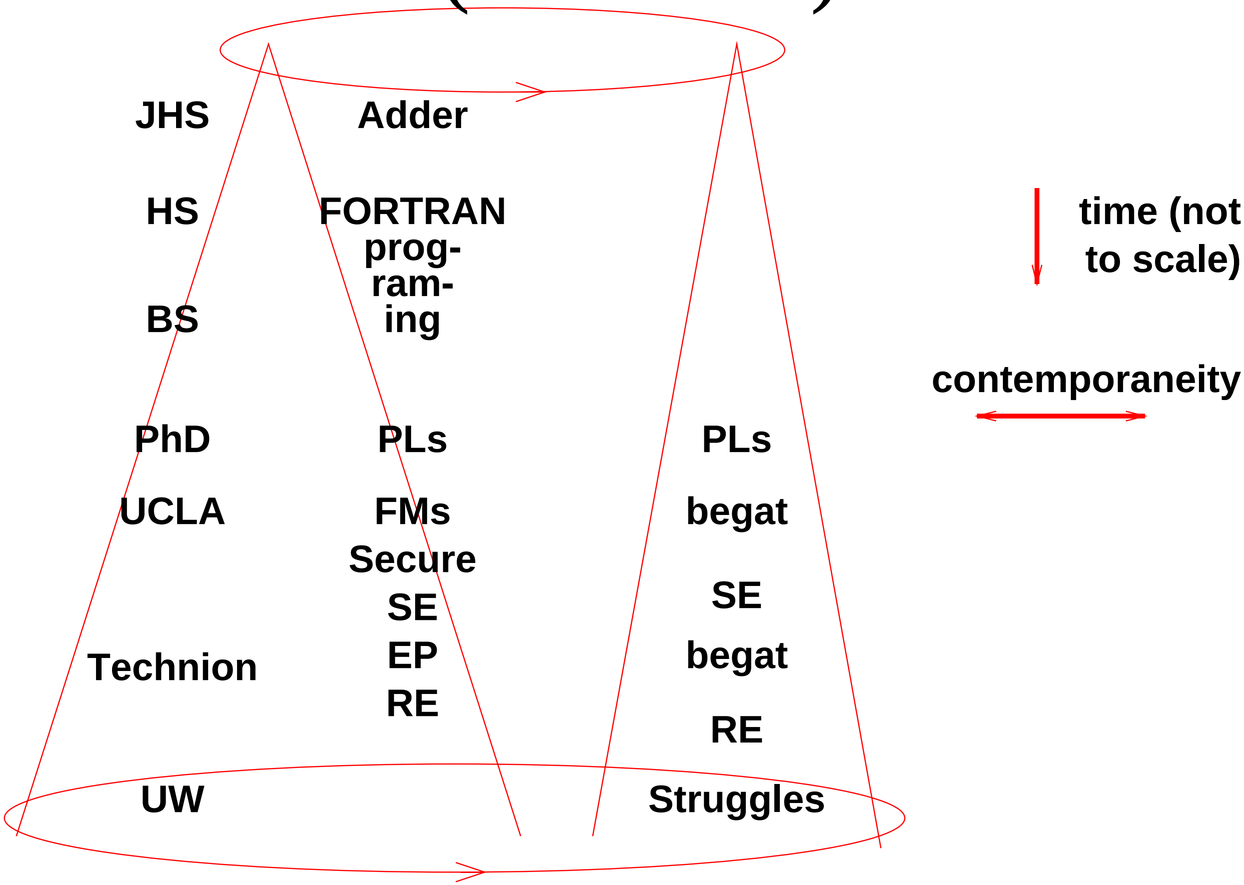
The Prehistory and History of RE (+ SE) as Seen by Me: How My Interest in FMs Helped to Move Me to RE

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

Outline (Pictorial)



Foreword

FM

Please note that I *believed* in FMs.

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

A long time ago …

Foreword, Cont'd

I worked for a company, SDC, that sold FM technology and applied FM to clients' system development problems, including for secure operating systems.

I did some fundamental work on the underlying theory.

Vocabulary

CS = Computer Science

CBS = Computer-Based System

SW = Software

PL = Programming Language

FM = Formal Method

SE = Software Engineering

EP = Electronic Publishing

RE = Requirements Engineering

More Terminology

We 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.

Overall Focus

We will see that my focus has always been on writing correct and good SW, even while I have been in many different, SW-related fields.

My progression through PLs, FMs, Security, SE, and finally RE, has been to follow what I thought would help most to achieve that focus.

That is, when I specialized or shifted fields, it was because I thought the field I was in was not getting to the root of the problem.

Origin of These Slides

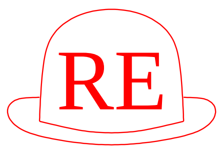
These slides are an enhancement of slides prepared for a keynote at a 2017 workshop celebrating the 40th anniversary of the birth of RE in 1977.

We

In the following, at any time, …

“We” = all the people in whatever field I was in at the time.

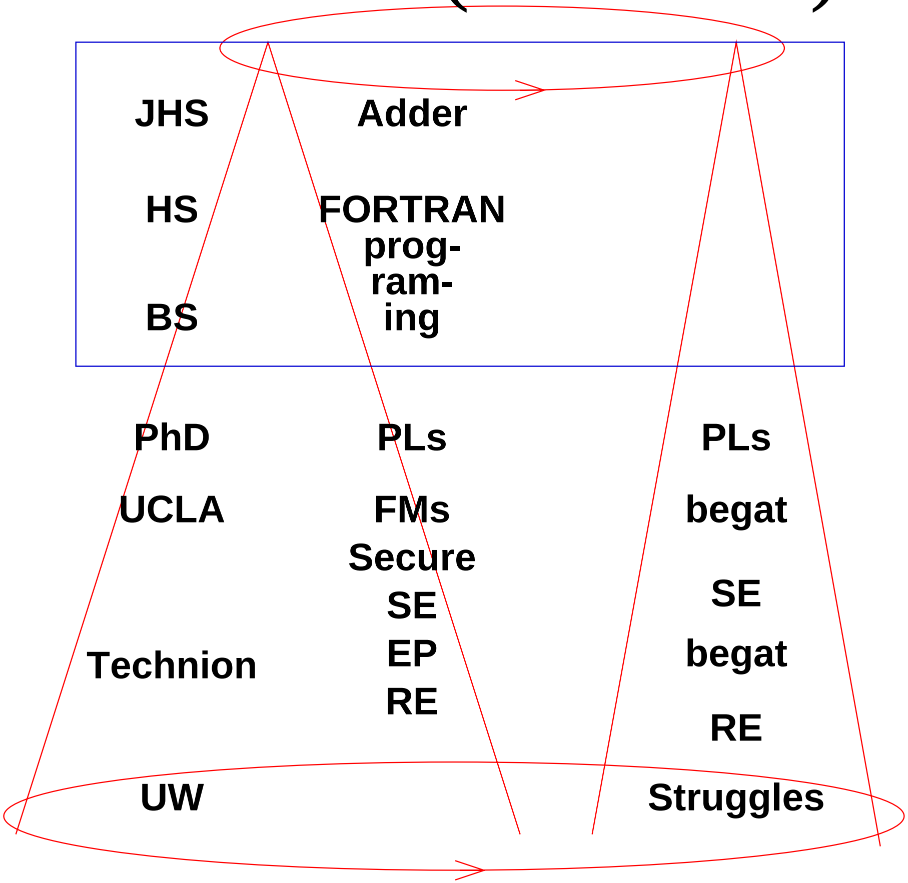
So it is context dependent.

I use hats, e.g., 

, in the upper right hand corner of a slide to name the current context.


1960s

Outline (Pictorial)



My 1960s Start in Computing

In the beginning, in junior and senior high schools, I

- **built a relay computer, an adder, in 1962, age 14, for a junior high school science fair,**
- **learned to program in FORTRAN in the summer of 1965, age 17, at an NSF SSTP at IIT in Chicago (Ed Reingold was my dorm counselor!),**

My Start, Cont'd

- **wrote my first real-life application, Operation Shadchan, a party 1-1 matching program based on the questionnaire of Operation Match, a 1-*n* dating program, in the Spring of 1966, age 17, for my synagogue's youth group's annual party,**

My Start, Cont'd

In college (university), I

- **studied pure math from 1966–1969, at RPI, an engineering school, to get a B.S., not a B.E. as most of my class mates,**
- **programmed statistical and curve-fitting SW for the Chemistry Dept. at RPI, to make spending money (I wrote FORTRAN from formulae they gave me.),**

My Start, Cont'd

- **programmed payroll applications in RPG for a service bureau in Troy, NY (home of RPI) in the Summer of 1969, to make money to go to grad school.**

SOTP BIAFIUIW

Through all this, I did seat-of-the-pants build-it-and-fix-it-until-it-works (SOTP BIAFIUIW) SW development, …

simultaneous RE, design, and coding, …

not really understanding the distinction between RE, design, and coding, …

SOTP BIAFIUIW, Cont'd

thinking that all of it were just parts of programming, …

probably like a whole lot of programmers, even professionals, did.

Grad School

Later, in grad school, I

- **started grad school at Brown in 1969 as a pure Math PhD student (Never mind an MS; that's for people who want to *work* for a living.),**

- **took Measure Theory from Herbert Federer, who literally wrote the textbook, and discovered that I had promoted myself to my level of incompetence (the Peter Principle) in math,**

Grad School, Cont'd

- **did a lateral transformation to take computer science courses in the Applied Math department down the street,**

- **fell in love with PLs when I took Peter Wegner's course, PLs, Information Structures, and Machine Organization (PLISMO), from the book he wrote from his PhD thesis, and**

Grad School, Cont'd

- **ended up getting my PhD in 1973 from Peter on**

  the design of and the formal specification of Oregano, an improvement over Algol 68 and over Basel; …

  it was designed to be more orthogonal than either by keeping the architecture of its implementation firmly in mind; …

  that architecture became the basis for its operational VDL formal specification.

CS Journals in Early 1970s

PL

At that time, there were only 3 journals in CS,

CACM, monthly,

JACM, quarterly, and

_CR_, quarterly.

So, I read at least the abstract of _every_ paper published in CS journals for a few years.

CS People in Early 1970s

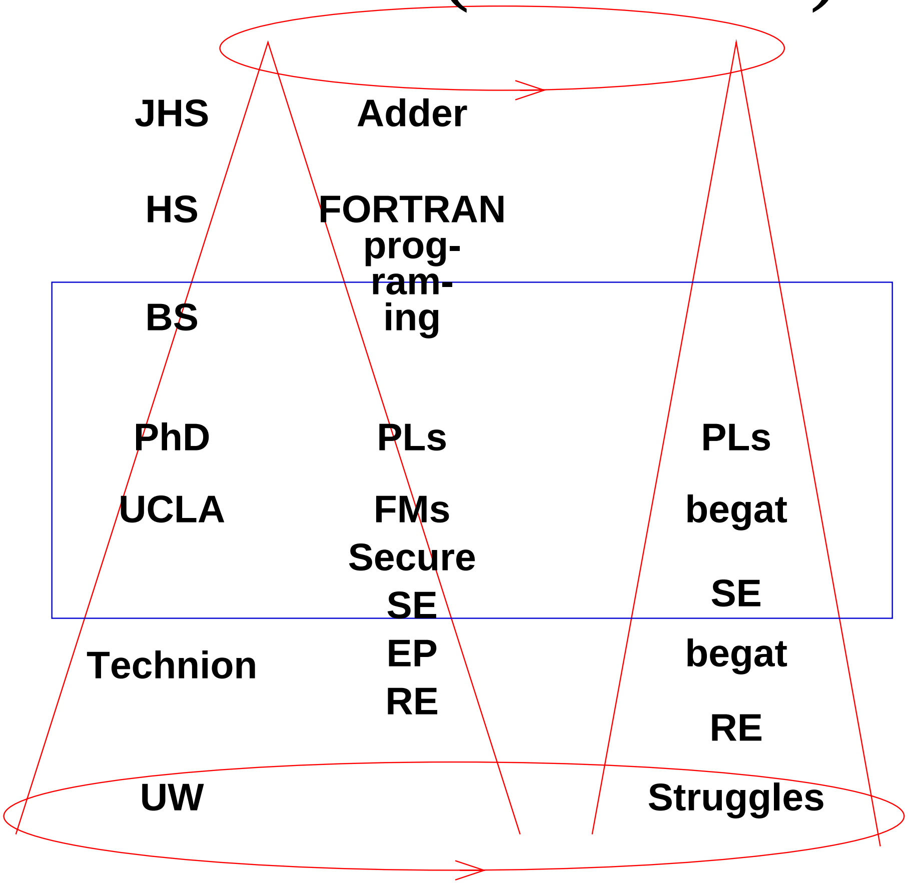
Also, the number of people in CS in the early 1970s was small enough that any person could know just about everybody in his or her field and many in other fields.

And most of the pioneers were still alive.

So, I met just about *everybody*, …


1970s

Outline (Pictorial)



Assistant Professor at UCLA

I started as an assistant professor in 1972 at UCLA, where the ARPAnet that later became the Internet, was happening.

I started off in the field of PLs.

SIGPLAN was the *biggest* SIG of the ACM at the time.

We all knew how difficult it was to write correct SW that does what its client wants.

PL Research in Early 1970s

The overarching concern of PL research in the early 1970s was:

- **to design a PL in which people would write correct and good SW, and**

- **to try to design a PL in which it was difficult, even impossible, to write bad SW**

Mission Impossible

But of course, that is impossible

We realized that you could easily write *really* atrocious SW in even the most structured PL

…

At one meeting, someone (I forgot whom) came up to the blackboard & showed us the following goto-free structured program:

Atrocious SW

```
for i from 1 to 4 do
    case i in
        1: S1,
        2: S2,
        3: S3
        out S4
    esac
od
```

which, of course, is equivalent to

S1; S2; S3; S4 ☹

My PL Research

My own PL research was in

- **making PLs more orthogonal,**
- **adding features to PLs in an orthogonal way**
- **operational formal semantics of PLs and their features.**

My PL Research, Cont'd

I ended up being involved with the Algol 68 committee from 1972 through the early 80s.

SARA

All this time at UCLA, I was a member of Jerry Estrin's SARA group.

SARA was a multi-notation system design language, a competitor of SA and PSL/PSA, and …

a FM based on data and control flow diagrams, and

a precursor of UML.

SARA, Cont'd

SARA was implemented with textual input but line-printer graphic display of models so that it could be used over ARPAnet.

SARA provided analysis tools to verify well-formedness and mutual consistency of models, to run simulations, etc., like PSA for PSL.

SARA, Cont'd

Several of my PhD students built pieces of, analyzed parts of, or applied SARA for their theses.

It was in connection to this research that I met some of the authors of the papers of the papers in the January 1977 issue of *TSE*, …

e.g., Doug Ross, John Brackett, Dan Teichroew, and Mack Alford.

SARA, Cont'd

The irony of all this SARA work is that …

while other things I did feel to me as having used what became RE thinking or having facilitated my realization of the importance of RE and its activities, …

this SARA work did nothing of the sort.

SARA, Cont'd

In fact, I will admit to being *totally* surprised that the organizers of this 40th anniversary workshop thought that the collection of papers in the January 1977 *TSE* marked the birth of RE.

To me, the work they did is more technical and notational, than attacking the fundamentals of RE, but that's my viewpoint.

SARA, an Aside

You see, …

All of this work assumed that the requirements were GIVEN to you by the client on a silver platter, and the hard part was the specification and the analysis. It was only years later that we began to realize that getting the requirements to start with was the HARD part.

January 1977 *TSE*

RE

Two of the articles have “RE” in their titles:

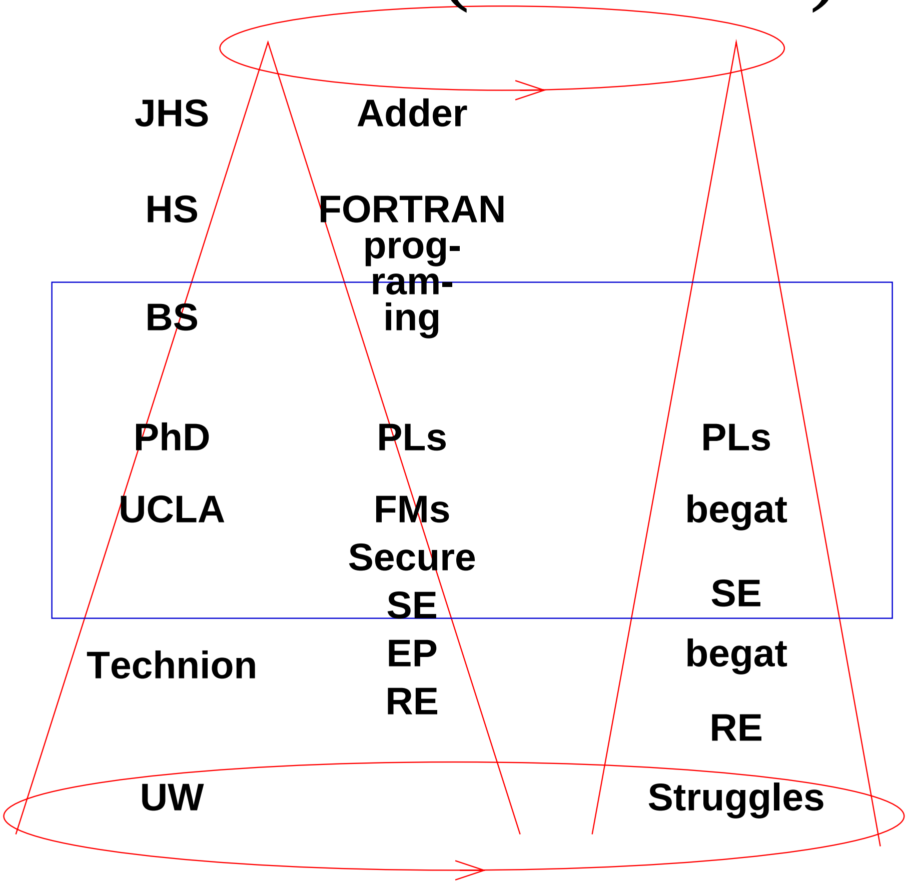
- **“An Extendable Approach to Computer-Aided Software Requirements Engineering”**
- **“A Requirements Engineering Methodology for Real-Timc Processing Requirements”**

January 1977 *TSE*, cont'd

But the articles consider RE to be the process of arriving at consistent, complete requirements specifications from the requirements the client gives to the engineers.

None of the articles deals with the *HARD* part of RE.

Outline (Pictorial)



Mid ’70s Foment in PL Area

In the mean time, in the PL field, we realized that the key to getting better SW was *not* to improve PLs, *but* to improve the *process* of SW development.

1968 NATO Meeting

SE

The 1968 NATO meeting had already suggested in response to the SW crisis (bad and badder and badderer SW is being produced as the need for SW is growing) that *maybe*

- **we should be systematic and science based and**
- **we should be *engineering* our SW,**

just like bridge builders engineer their bridges based on the laws of physics.

1968 NATO Meeting Report

“SE” was used only in the report title and in other meta-text, …

not in any participant’s article.

The field did not exist yet.

Birth of SE field

Thus, was born the field of SE, initially populated with PL people who realized

- **that the PL used in programming has little or no effect on the quality of the SW programmed with it, and**

- **that programmers' behavior had a far bigger impact on the quality of SW they produced than the PLs they used.**

Switching to SE

SE

So I, like a whole bunch of other PL people, ended up switching in the mid to late 1970s to SE.

We tried during the 1970s and 1980s (when ICSE met only every 18 months) to find methods, possibly assisted by math, to develop correct SW meeting its client’s needs.

Morphing of Fields

For these switchers, …

- **the study of PLs morphed to the study of SW development methods, and …**

- **formal semantics for PLs morphed to FMs of SW development.**

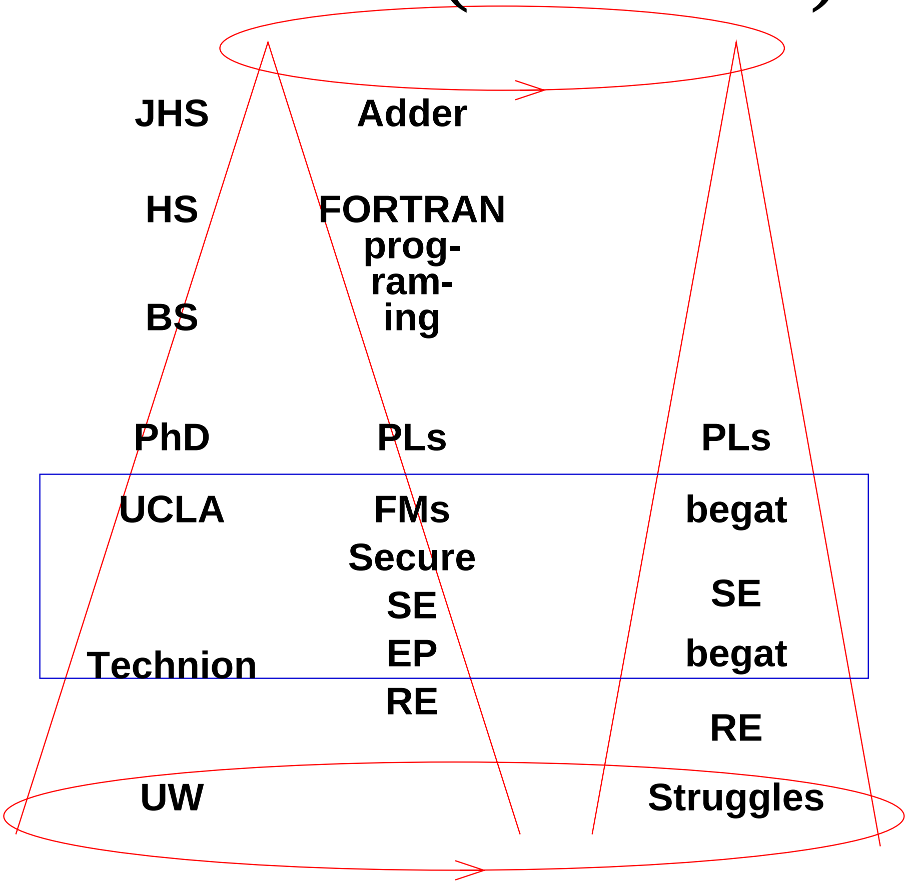
My Sojourn into Security

In the early 1980s, as a result of supervising several people doing formal methods, and in particular Richard Kemmerer who did (1) a formal specification of the kernel of the UCLA secure UNIX and (2) a formal verification of that the kernel met the specification of security, …

I got involved in the security community.


1980s

Outline (Pictorial)



Security, Cont'd

I consulted for the Formal Development Method (FDM) group of SDC ($\rightarrow$ UNiSYS) that was working on secure operating systems, e.g., Blacker.

I ended up publishing a paper in *IEEE TSE* showing how the theorems that the group's verifier proved about an Ina Jo formal specification of a system were sufficient to prove that the system, if implemented as specified, would meet the specified criteria.

Security, Cont'd

From all this 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 Sojourn into EP

While I was doing this SE and FM stuff, I made a parallel diversion in the mid 1980s through mid 1990s into Electronic Publishing (EP):

Built EP SW

I got to design and build SW for multi-lingual and multi-directional word processing.

Beginning My Move to RE

RE

During this time, in 1981, I published a paper with Orna Berry about how I managed to do the best job ever in specifying software that she had to write, in a domain that I knew nothing about.

I agreed to do this job *only* because I was married to her at the time!

Beginning My Move, Cont'd

RE

In retrospect, I consider this to be my first RE paper.

It's certainly one of the very earliest on the elicitation aspect of RE.

Ignorance Hiding

She had to write some programs that played statistical games with experimental data.

I got my lowest Math grade in the undergrad Probability and Statistics class, a B, (it ruined my perfect Math GPA.) because I had *no* intuition for probability.

So, I was ignorant in the statistics domain.

Ignorance Hiding, Cont'd

To be able to hide my ignorance so I could work effectively with the requirements as she expressed them to me, …

I made the experimental data an ADT, with each magic function that I did not understand, e.g., standard deviation or standard error, being a method of the ADT. I knew that the client understood what they mean and how to implement them. So I worked with this ADT with its methods taken as primitive.

Ignorance Hiding, Cont'd

SE

I thought and claimed in this paper that this ignorance hiding technique was the basis of the success …

as well as my ability to nudge the client to give information

and to do strong-type checking on natural language sentences.
(Using the same verb with different numbers and kinds of direct objects in different sentences is a type error.)

Importance of Ignorance

By 1994, I figured out that the reason for the success was not the ignorance hiding, but the very ignorance!

Importance of …, Cont'd

So in 1994, I published "The Importance of Ignorance in RE" claiming that every RE team for a CBS requires along with domain (of the CBS) experts at least one smart ignoramus of the domain, who will

- **provide out-of-the-box thinking that leads to creative ideas, and**
- **ask questions that expose tacit assumptions.**

Empirical Validation

In 2013–2015, my PhD student, Ali Niknafs, conducted controlled experiments to empirically validate that

for the task of brainstorming for requirement ideas, …

among 3-person teams consisting of only computer scientists or software engineers, …

Empirical Validation, Cont'd

RE

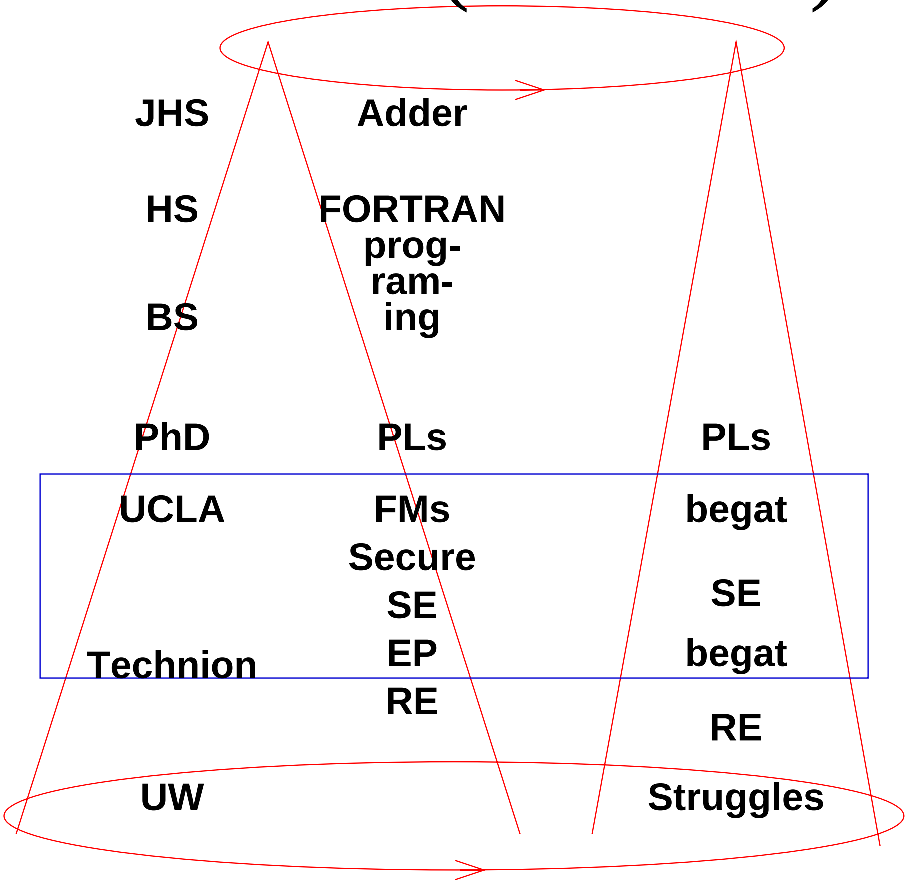
the teams with *one or two members ignorant* in the domain …

generated *more and better requirement ideas* …

than teams consisting of …

> ***only ignorants* of the domain or …**
>
> ***only awares* of the domain.**

Outline (Pictorial)



The Birth of the RE Field

After a while, in the mid 1980s, a subset of the SE people began to notice that SE methods and FMs do not really solve the problem of ensuring the production of quality SW.

- **They address mainly development and not determining requirements.**
- **They don’t scale well, particularly FMs: For some funny reason, FM people did not use FMs when building tools to help do FMs. (More later.)**

A Realization

Then, a subset of the SE field came to the realization that the real problem plaguing CBS development was that we did not understand the requirements of the CBS we are building.

A Realization, Cont'd

Brooks, in 1975, had said it well:

"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."

Even a FMs Person Got it

RE

Even an initial-algebras, FMs person, Joe Goguen, came to this realization.

He ended up being a keynoter at the first RE conference in 1993.

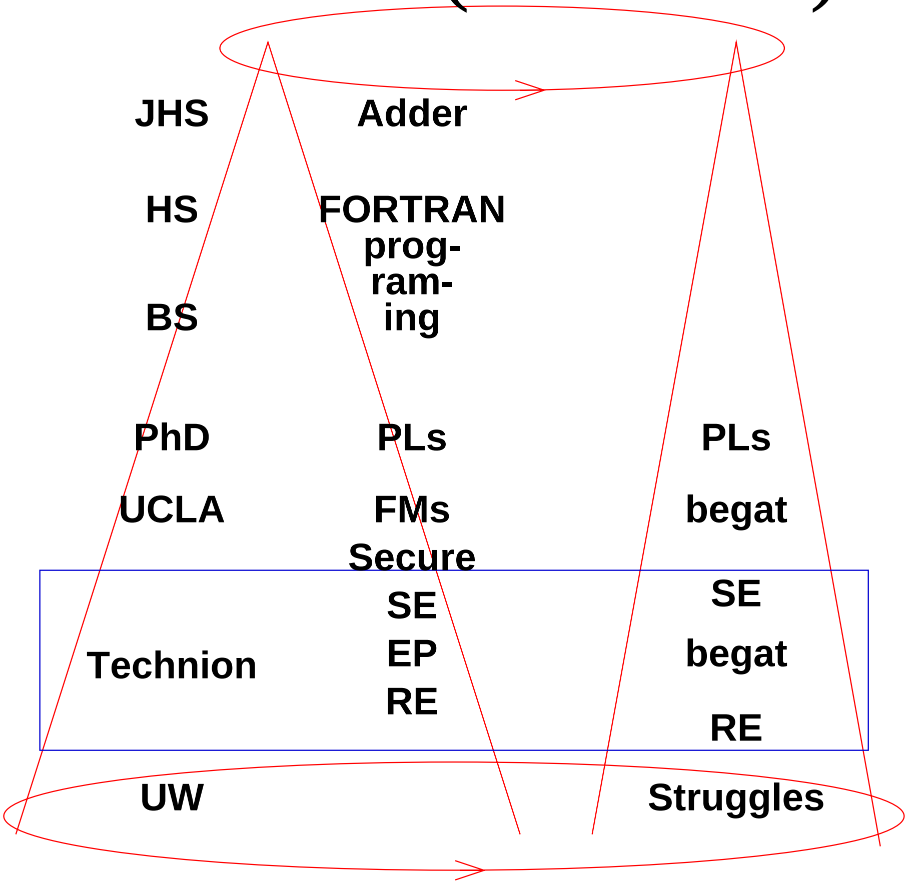
The next slide has a 1994 quote from Joe, not from the keynote, but from a draft of a paper for the book on *Requirements Engineering: Social and Technical Issues* that he was writing with Marina Jirotka.

Surprising Goguen Quote

It is not quite accurate to say that requirements are in the minds of clients; it would be more accurate to say that they are in the social system of the client organization. They have to be invented, not captured or elicited, and that invention has to be a cooperative venture involving the client, the users, and the developers. The difficulties are mainly social, political, and cultural, and not technical.


1990s

Outline (Pictorial)



A Realization, Cont'd

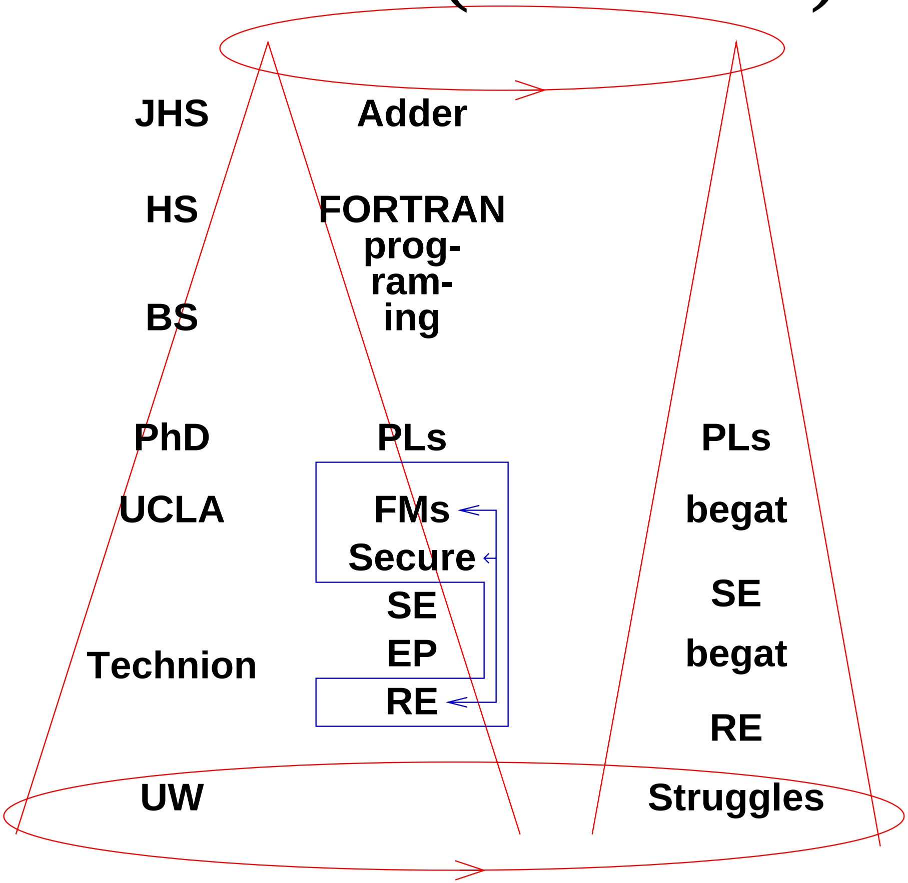
This subset of the SE folk formed the RE field,

1. **by piggybacking on the nearly annual International Workshop on Software Specification and Design (IWSSD) in the mid to late 1980s and early 1990s,**
2. **from 1993, in two alternating conferences, ISRE and ICRE, that later merged into one (RE),**
3. **from 1994, in an annual working conference, REFSQ,**
4. **from 1996, in a flagship journal, REJ.**


2000s

Fast Forward

Outline (Pictorial)



More About FM Part of My History

I explore this part in greater depth.

First, what I noticed as it was happening.

Then, explaining some of it more formally.

Viewing FM from an RE lens!

Motivation to Write These Slides

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’s all the same as in the 1970s and 1980s.

Never Change, Cont'd

In my opinion, FMs will never be adopted by large numbers of CBS developers. Why?

Yes, there *have been* and there *are* breakthroughs in FMs, but these are not the only technological breakthroughs that affect programming.

Never Change, Cont'd

With each tech breakthrough, all those CBSs that were too difficult to build without the breakthrough get built almost overnight!

This tech breakthrough could be a FM! e.g., Finite State Machine Specs

Then What?

Then what's left?

CBSs that are even *more* difficult to build!

We are left in the state that existed before the latest breakthrough, needing still more breakthroughs to tackle the CBSs at the current frontier.

Then What? Cont'd

The problem with FMs is that because they are not the only breakthroughs, the gap between FMs and the difficult CBSs at the frontier gets bigger and bigger.

No technology, and in particular FMs, will ever catch up.

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 CBS as part of a team**

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 FDM suite of FM tools**
- **bi-directional editor**
- **tri-directional formatter**
- **letter stretching bi-directional formatter**

Never Actually *Used* FMs

FM

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

even for the proof updater for the FDM suite of FM tools.

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

Never *Used* FMs, Cont'd

Neither did Val Schorre and John Scheid in developing the other tools for the FDM suite, including a verification condition generator (VCG) for Ina Jo specs, and an interactive theorem prover (ITP).

(They did use Val's compiler-compiler to deal with the syntax.)

Never *Used* FMs, Cont'd

Note that these tools *were* used in production applications of the FDM to building some half dozen verifiably secure systems at SDC for the US DOD and NSA.

Never *Used* FMs, Cont'd

Apparently, neither did other developers of FM tools (at least the ones I knew).

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.

FMs For Only Small Programs

FM

So, FMs 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

FM

Tony Hoare said (I think in late 1970s through 1980s),

“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.

Matchmaker

I did this while I was in HS, long before I knew about FMs.

Later, it proved to be a variation of the stable marriage problem, with a 50-factor bi-directional attractiveness function, based on questionnaire answers.

In retrospect, the central formal model would have accounted for less than 5% of the code.

Matchmaker, Cont'd

The rest of the code deals with

- **incorrectly filled questionnaires,**
- **the complexities of having a mix of absolute criteria and do-the-best-that-you-can criteria, and**
- **having to deal with too-picky people who did not get matched by the algorithm, but still had to be matched for the party they paid for.**

Bi-Di Formatting and Editing

Algorithm for basic bi-di reformatting after line-breaking text as if it's uni-directional is 8 lines long, assuming existence of a function that reverses the text of its argument.

But this algorithm accounts for less than

- **5% of my ffortid ("ditroff" spelled backwards)**
- **1% of the Unicode bi-di algorithm**

Back to the FDM ITP

In retrospect, I can see why FMs were not used to develop the ITP.

The central, formal part of the ITP was a small fraction of its code.

Back to the FDM ITP, Cont'd

The rest dealt with

implementing the really nice interaction with the user (the person trying to prove a theorem)

managing the current proof, including keeping track of what had been proved in a way that made it easy for a user to apply any of it at any time, …

and this part is tough to formalize.

What vs. How Specifications

Many times, it is much easier to express an algorithm to do something than to give an algorithm-independent description of what the something is:

- **industrial processes**
- **exceptions to a central algorithm**
- **New York bagels (chewiness vs boil-then-bake)**

Lessons Learned from FMs

Even as I was observing these difficulties in the application of FMs, …

I learned some important lessons from the FM work that did not need FMs *per se* to be utilized.

Fundamental Lesson of FMs

FMs applied to Security taught me the fundamental essence of RE:

The only way to ensure that a constructed CBS will have any of a whole class of desirable properties (e.g., security, reliability, robustness, safety, survivability) that must permeate the CBS's entire behavior is to require the property from the very beginning; it cannot be added to the implementation as an after thought.

No Brainer of RE

This essence leads directly to the idea that you need to understand the requirements of a CBS that you are going to build before you can build it.

This is really a no-brainer

No Brainer, Cont'd

because, ultimately, it is impossible to write the next line of code that you are going to write without knowing what the line of code is supposed to do, i.e., …

without knowing the line's requirements.

Nu?

Failings of FMs

Even as FMs applied to Security taught me the fundamental essence 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.**

We explore why.

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]**

RE

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

RE

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]"

Hoare on Small Programs

RE

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 approximation.

80–90% of the code = exceptional details.

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.

Formal Model Still Useful

Hoare says,

> **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.**

An Example

Hoare adds,

> **A large space in the grammar of a language is taken up by irregular verbs. But without a model of what is regular, every verb would be irregular.**

FMs Not Doing What RE Needs

RE concerns validation more than verification, …

but FMs deal with …

Verification, but ...

FMs have the power to put

verifying the correctness of a CBS implemention 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,

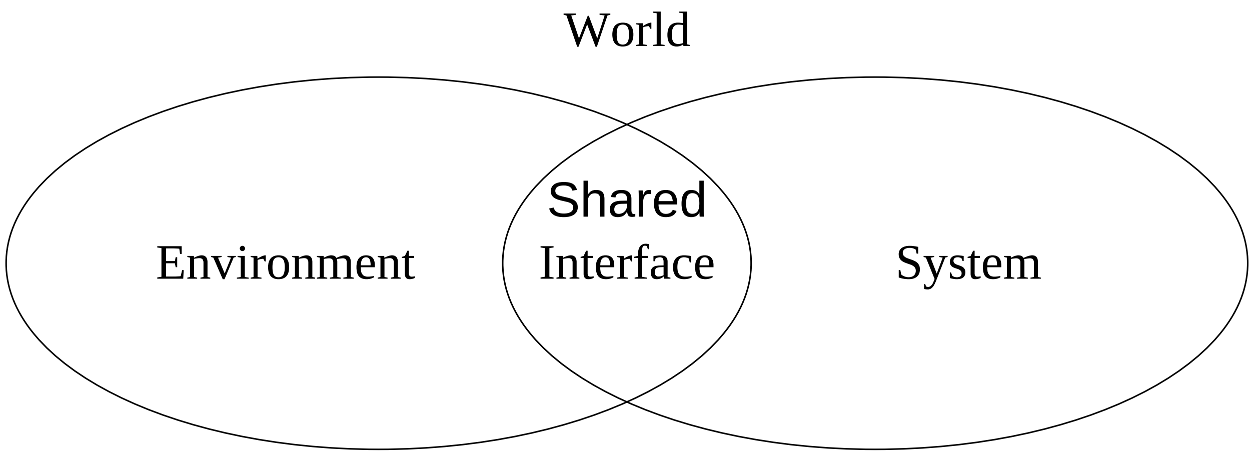
or the RE Reference Model.

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 in*formal,

and the boundaries of Env are *hopelessly fuzzy*:

Butterfly in Rio $\rightarrow$ 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 the program or any formal specification.**

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

If program uses machine learning, then *S* is effectively informal, and its truth is dependent on the learning set in ways that defy formalization.

Formal vs. Informal

Michael Jackson [1995] once said:

“Requirements engineering is where the informal meets the formal.”

- **Raw ideas: informal**
- **Code: formal**

Informal Meets Formal

Informality is unavoidable.

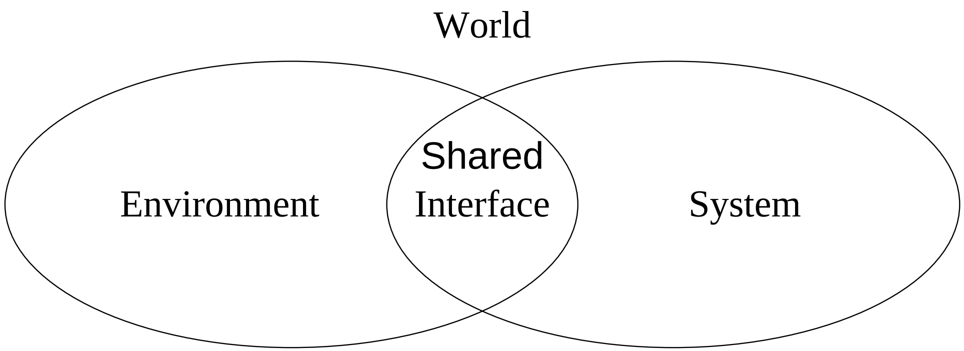
Meeting Point is Unavoidable

There is no way to go from ideas to code without determining requirements for the code from the ideas.

That is, no programmer can write code without knowing what the code is to do, even if he or she has to decide what the code is to do on the spot.

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?

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*.

Two Types of Requirements

There are two types of requirements:

1. **scope determining**
2. **scope determined**

E.g., for a pocket calculator with $+$**,** $-$**,** $\times$**,** $\div$**,**

1. $\ln$ **and** x^y**, are scope-determining requirements.**
2. **“that** $d \neq 0$ **in** $n \div d$**” is a scope-determined requirement.**

Difference Between Types

A pocket calculator without one particular scope determining requirement is just a less useful and less attractive calculator.

A pocket calculator without one particular scope determined requirement is a flawed calculator, which will give the wrong result or fail for some inputs.

FMs and the Two Types

FMs help discover scope-determined requirements.

FMs offer little help discovering scope-determining requirements, …

because each scope-determining requirement is independent of the others.

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

ZJVF and the Two Types

RE

In terms of ZJVF and the World, generally,

- **each assumption, *a* in *D*, or**
- **each entity, *e*, in the Env that affects or is affected by the Sys,**

gives rise to its own scope determining requirement, *r*,

namely, that addressing *a* or dealing with *e*.

ZJVF and the Two Types

If no stakeholder thinks of *a* or *e*, …

then no stakeholder will naturally think of *r*.

Hard to Think of These

It's hard to think of these *a*s and *e*s, …

because the boundary of Env is fuzzy.

So even if you could list every *a* and *e* in Env, …

you're never sure that nothing from (World – Env) can come into play.

Value of RE Reference Model

The RE RM has become extremely valuable as a …

lightweight, informal version of a FM …

that is able to answer many questions that come up during RE for a CBS.

Value of RE RM, Cont'd

The RE RM is used to help

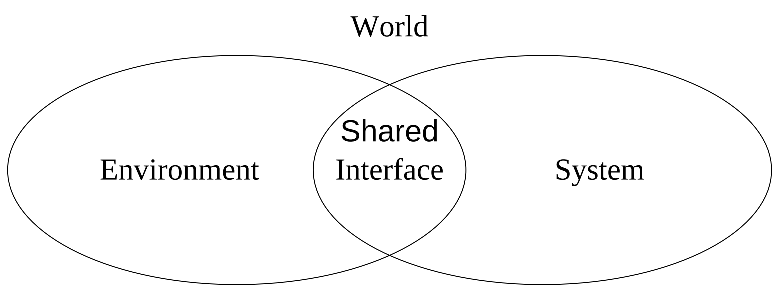
- **partition the World, i.e., to decide for each of Env, Intf, and Sys, what is in it and is not, …**

  sometimes to shuffle an entity among Env, Intf, and Sys

Value of RE RM, Cont'd

- **decide *What* vs. *How*:**

  ***What* is in the vocabulary of Env**
  ***S* is in the vocabulary of Intf**
  ***R* is in the vocabulary of Env**
  ***How* is in the vocabulary of Sys–Intf**



Value of RE RM, Cont'd

- **permanently tolerate an inconsistency *I* between *R* and *S* and the World,**

  by lying in *D* that *I* is not a problem, …

  e.g., for the Airplane CBS, permanently tolerate that a bird's meeting an airplane in the air can crash the airplane, by lying in *D* that there are no birds in the air.

Value of RE RM, Cont'd

The RE RM is a major focus in the RE course at the University of Waterloo.

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 a 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.

But there are some fundamental differences between a program and a math model, as it's usually done.

Math Model vs. Program

Each is a model of the real world.

Different audience:

- **math model read by smart human; can deal with “YUWIM”**
- **program read by dumb computer; cannot deal with “YUWIM”**

Math vs. Program, Cont'd

Because of difference in audience,

- **math model can get away with simplifications and approximations for tractability;**

- **program must deal with every detail, with *no approximation*, or else program fails at exception conditions, e.g., plane crashes.**

Fickas on Outliers

Steve Fickas once said,

“Sciences ignore outliers.”

But, robust software cannot.

Central Math Model in Code

FM

In a program based on a mathematical model of some real-world phenomenon, …

the mathematical model amounts to 20% of the code, and the code to deal with the outliers, the approximations, the exceptions, etc. amounts to 80% of the code.

Code as Math Model

So, code is a much more complete mathematical model than most mathematical models produced by mathematicians or scientists.

Even then, as we saw with the World Model and the ZJVF, it cannot be a *perfect* model.

The Inevitable Pain of CBS Development

The inevitable pain of CBS development arises from the fact that every CBS that is *used* in the real world has to be updated in order to respond to the changes in its requirements that result from its being used.

Inevitable Pain, Cont'd

Even if the initial development of the CBS from its first requirements spec is systematic by some method, possibly a FM, …

subsequent updates are SOTP patching jobs.

I have never heard of anyone throwing out the current version of the CBS in order to apply its development method to the updated requirements spec.

Inevitable Pain, Cont'd

Instead, he or she makes in situ modifications of the requirements spec, the code, and all other artifacts in between, …

to try to make all look like they are the result of having applied the same development method to the modified requirements spec.

A Lie

That *never* quite works (☹).

- **This updating is very difficult because it is akin to lying perfectly consistently, which is very hard to do.**

- **The lie is making *all* artifacts appear as if they were produced during an application of the development method to produce the current version from scratch!**

A Lie, Cont'd

Change is relentless, and therefore, lying is perennial!

Change is Relentless

RE

Why is change in a CBS relentless? Because of changes in the CBS's requirements:

- **We did not understand the CBS's requirements to begin with.**
- **We made mistakes in expressing what we understood.**
- **We deployed the CBS into the real world, giving rise to the Lehman feedback loop that changes the CBS's own requirements!**

What *Does* Work?

Good people, not good methods!

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 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.

Flawed Experiment

FM

"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.

L

FMs vs. No FMs

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 the flaw in the experiment (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.

No Control

Also, there was *no* control over whether the FM teams actually *used* FMs in the development.

Might be that the FM teams took advantage of skills, e.g., abstracting, logical thinking, etc., used in FMs, to improve their programming without actually doing any FM.

Not enough information to know.

Alternative Explanation

Berry and Tichy offered an 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 …, 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!

Don't be too hard on Sobel and Clark!

L

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”.

Lesson Learned

Good FMers make good programmers.

So if you’re managing a SW development, hire FMers to be your programmers!

My Message to FMers

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 & complete requirements specs, where 75–85% of the errors occur:

- **FMs applied to make the specs more correct, i.e., to eliminate errors of commission & discover missing scope determin*ed* requirements**
- **FMer applied to make the specs more complete, i.e., to eliminate errors of omission & discover new scope determin*ing* requirements**