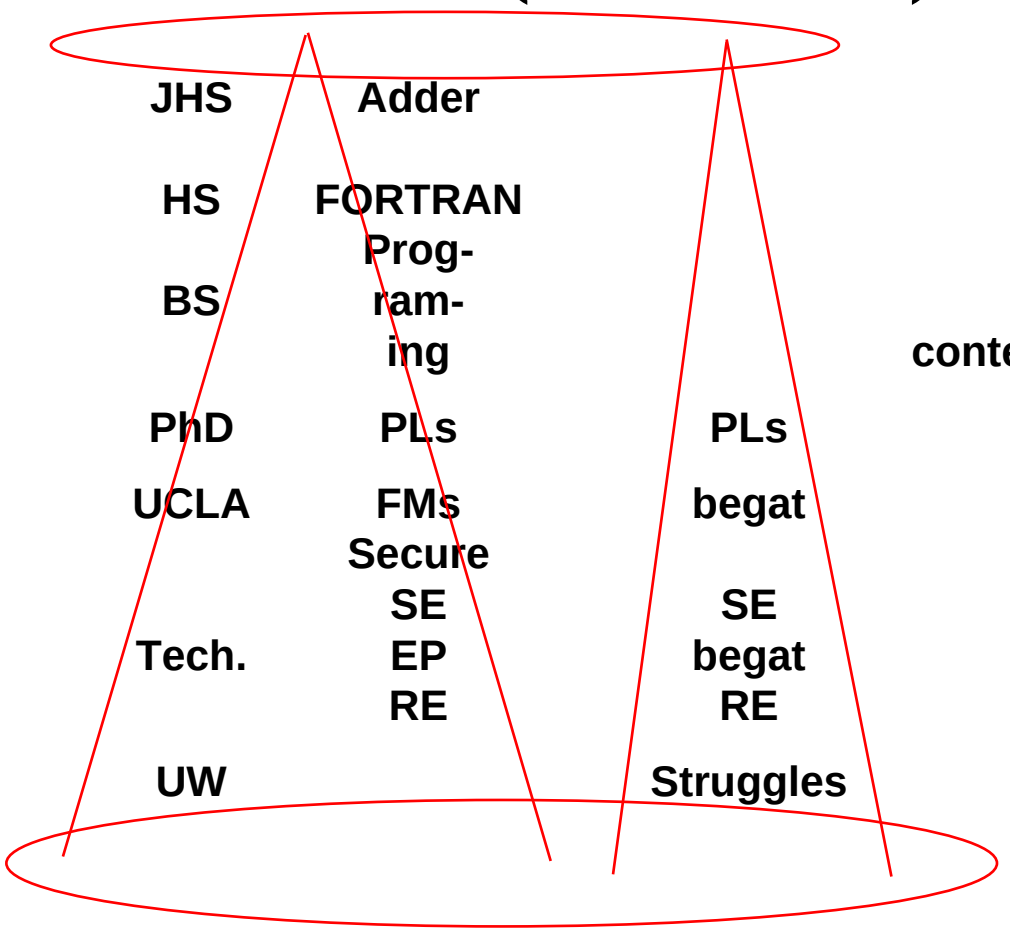
The Prehistory and History of RE (+ SE) as Seen by Me

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

Outline (Pictorial)



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

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.

The workshop organizers had identified the January 1977 issue of *IEEE TSE* as marking the birth of RE.

IEEE TRANSACTIONS ON

SOFTWARE ENGINEERING

JANUARY 1977 VOLUME SE-3 NUMBER 1

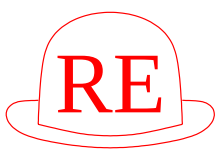
A PUBLICATION OF THE IEEE COMPUTER SOCIETY

We

In the following, at any time, …

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

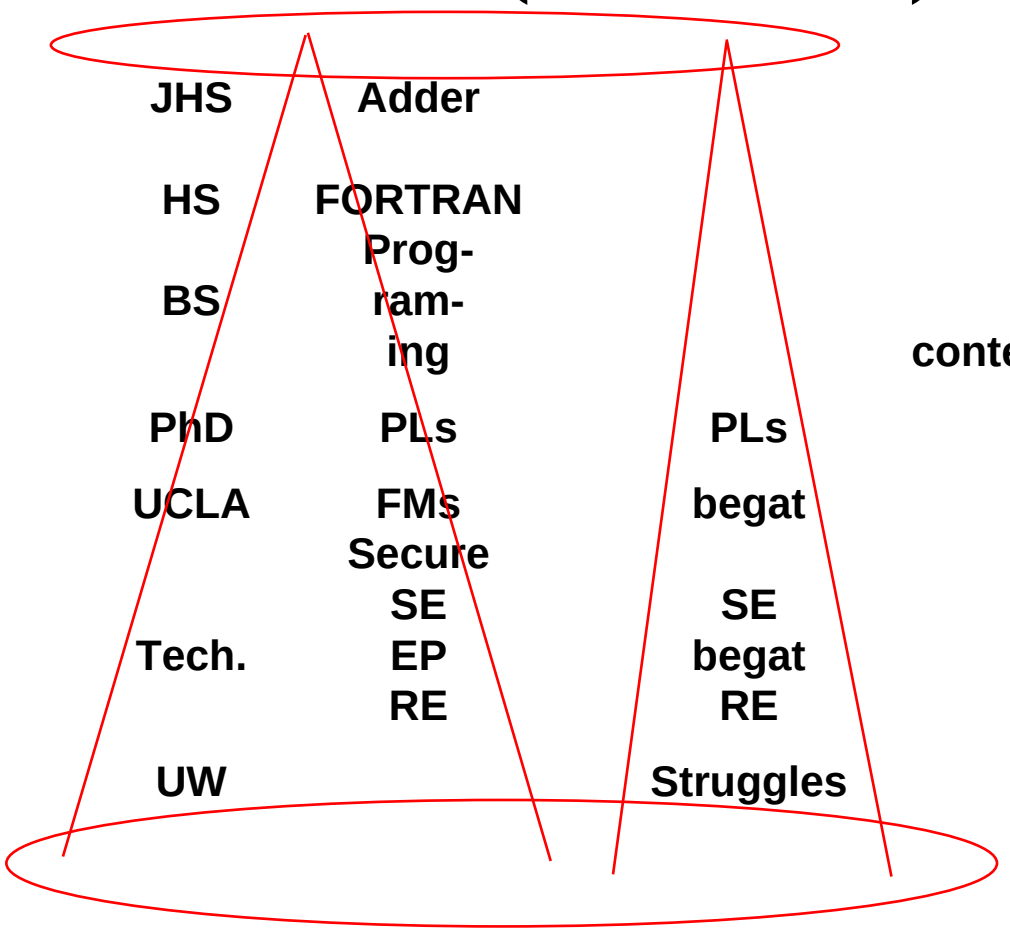
So it is context dependent.

**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, 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, Con'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,**
- **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,**

My Start, Con'd

- **programmed statistical and curve-fitting SW for the Chemistry Dept. at RPI, to make spending money (I wrote FORTRAN from formulae they gave me.),**
- **joined ACM in 1967 (member # 10*****), and**
- **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, 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

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.

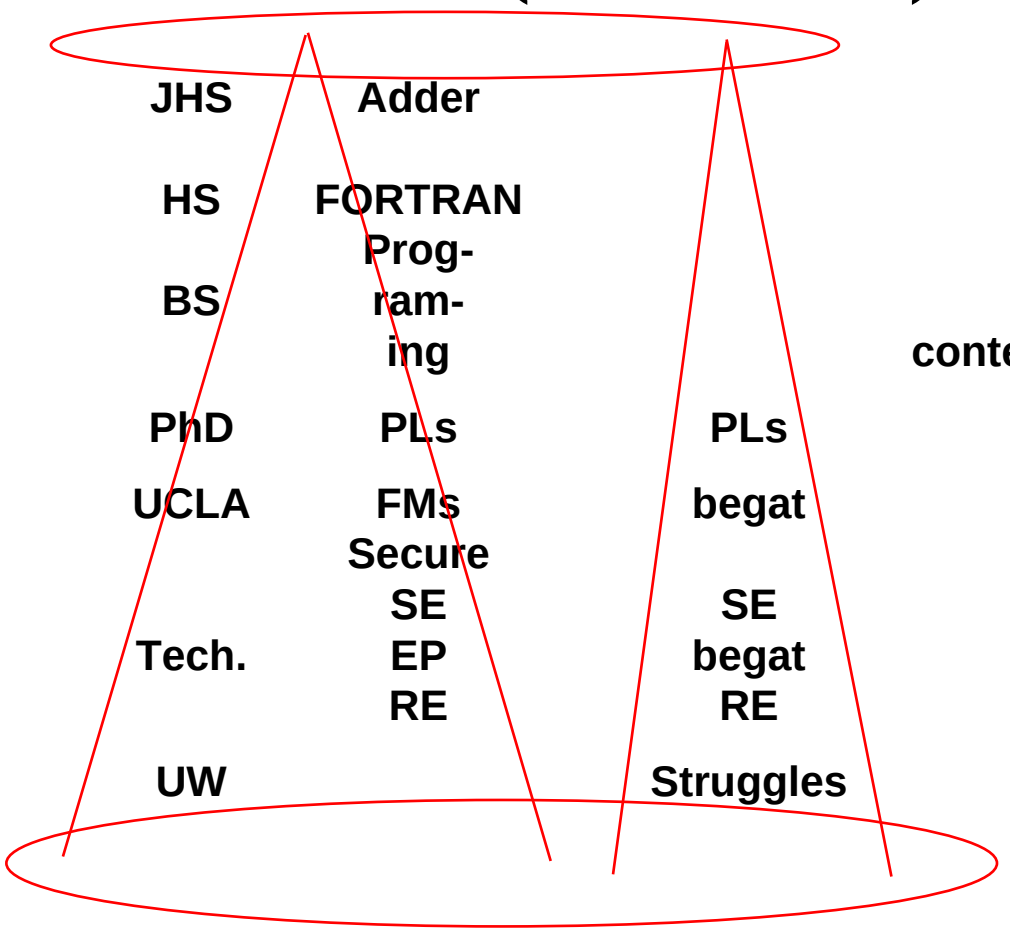
And most of the pioneers were still alive.

So, I met just about *everybody*, …

including the authors of *IEEE TSE* January 1977, at conferences or even in LA while at UCLA.


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.

Early Signs of RE Thinking

Note my own RE orientation of trying to fit a new feature into the existing language in the cleanest way, exploring it thoroughly before beginning to implement it.

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 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 workship 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*

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 Reqcuirements”**

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.

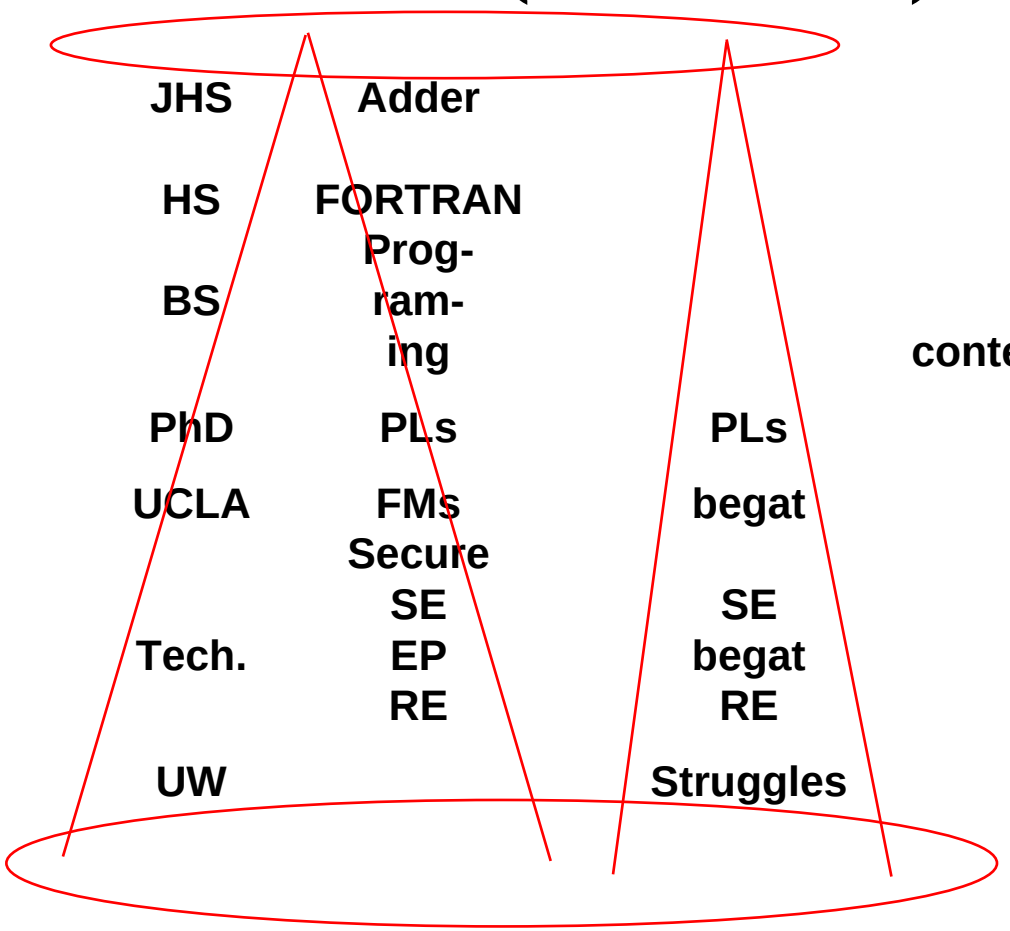
None of the articles deals with the *HARD* part of RE.

E-mail

In 1977, I started using e-mail as a replacement for the difficult-to-use telephone to connect with most of my acquaintances, who were CSers!

In 1980s, I started a campaign to convince my non-CS friends and my family to do the same.

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

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

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.**

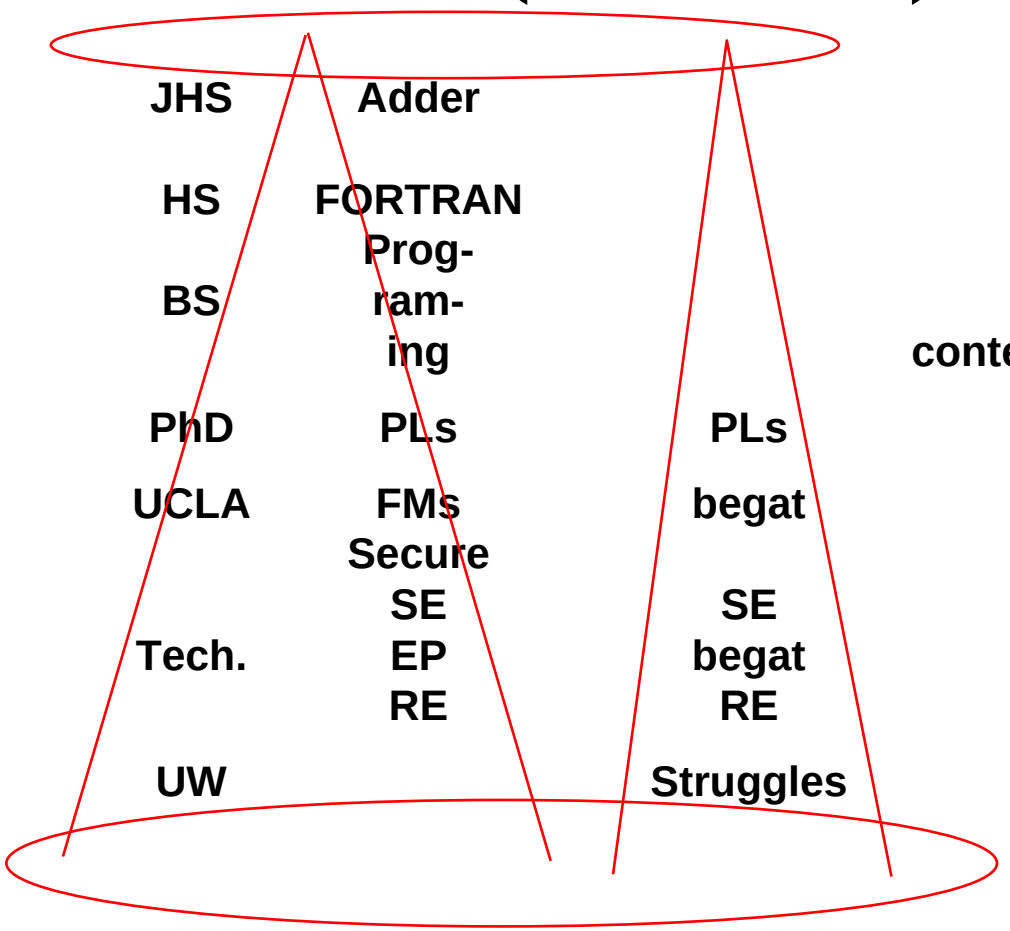
My Sojourn into Security

In the early 1980s, as a result of superivising 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 that was working on secure operating systems, in particular Blacker.

I ended up publishing a paper in *IEEE TSE* showing how the theorems that the group's verifer 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).

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

I tried to find the most orthogonal way to integrate the new features, using the least leaky user abstractions.

RE Orientation Even in EP

Note the RE orientation here

- **in the concern for orthogonality and**
- **in finding the least leaky user abstractions.**

These make the new features easier to use because they suffer no surprising exceptions.

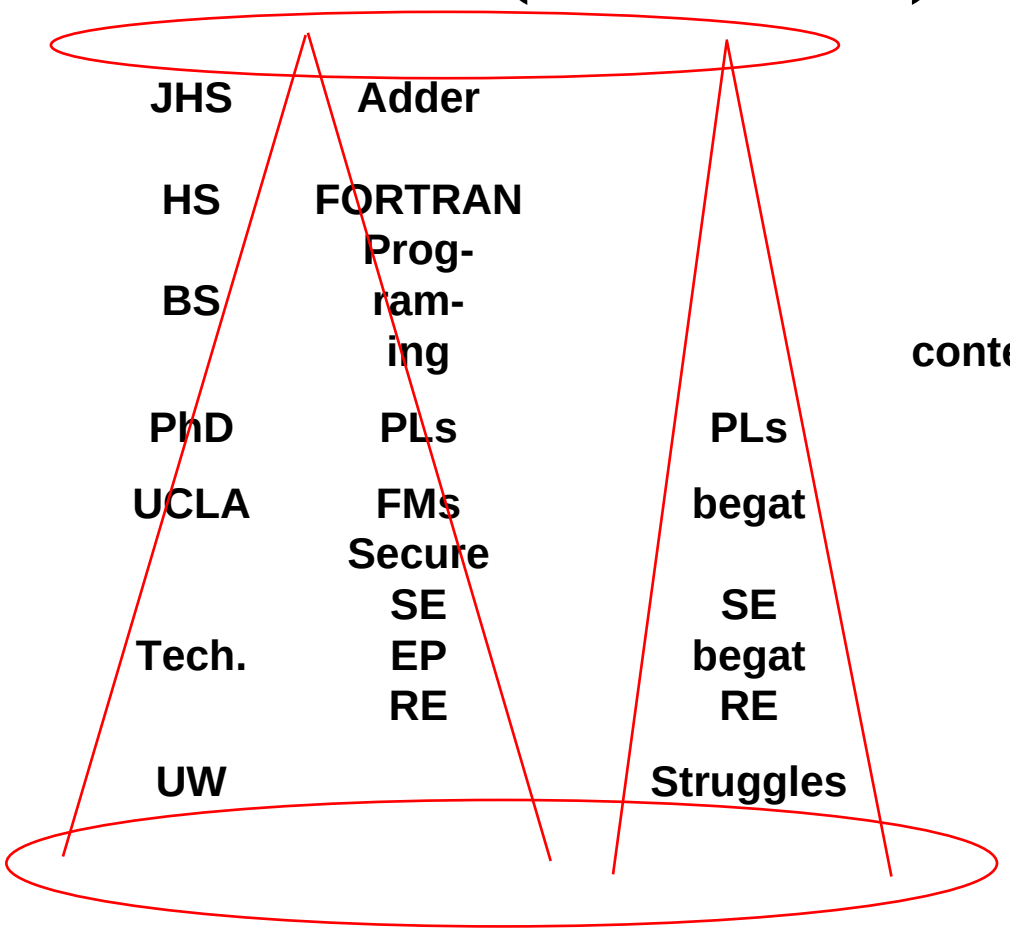
Quit Unicode Effort over RE

I quit the Unicode bidirectional working group over a requirement issue.

- **A majority wanted only one period in the whole character set, with contextual determination of an instance's writing direction and override for exceptions.**
- **I and a few others wanted one period per writing direction, with explicit specification of an instance's writing direction.**

Choice has MAJOR impact on users' actions.

Outline (Pictorial)



Beginning My Move to 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

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

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

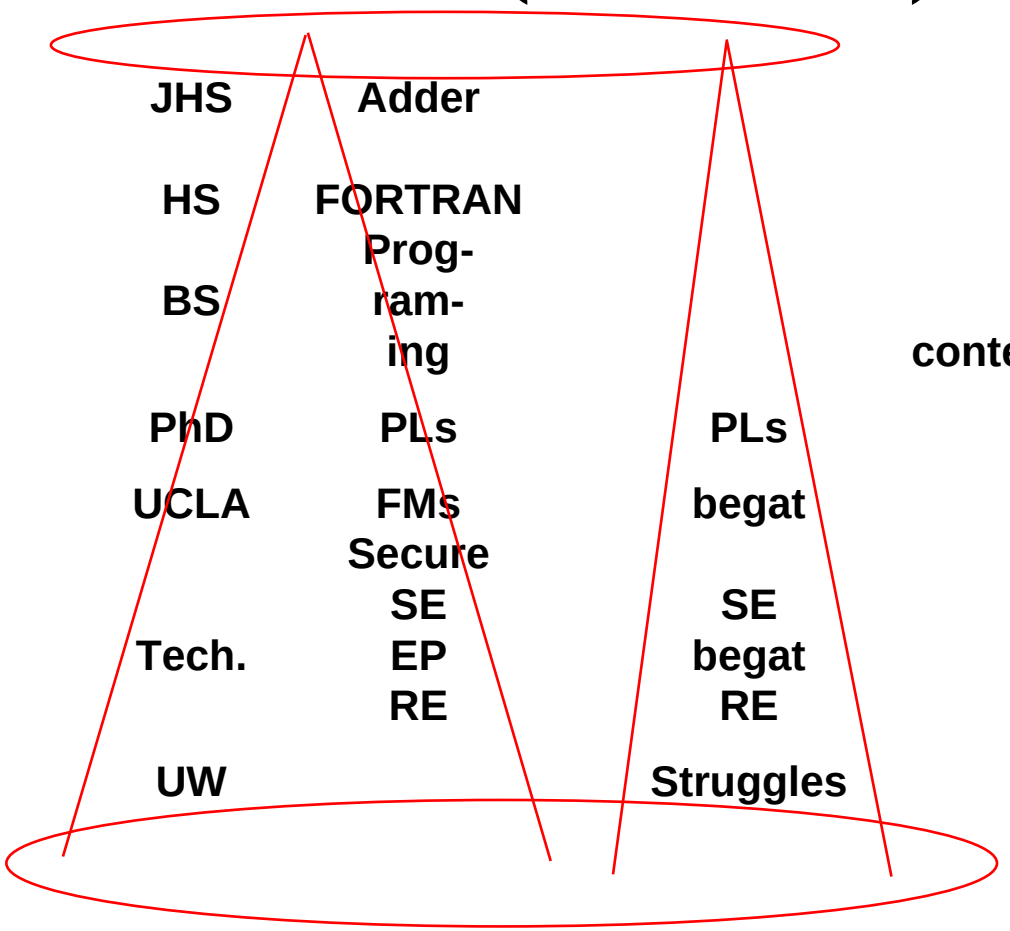
the teams with *one or two members ignorant* in the domain …

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 don’t scale well, particularly FMs: For some funny reason, FM people did not use FMs when building tools to help do FMs.**

Birth of RE Field, Cont'd

- **A method works well only the first time on any CBS. After that, when the CBS must be updated, e.g., for requirements changes, the artifacts produced by the method must be updated to be consistent with the changes.**

Birth of RE Field, Cont'd

- **This updating is 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 method to produce the current version from scratch!**

- **Change is relentless, and therefore, lying is perennial!**

Change is Relentless

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!**

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."

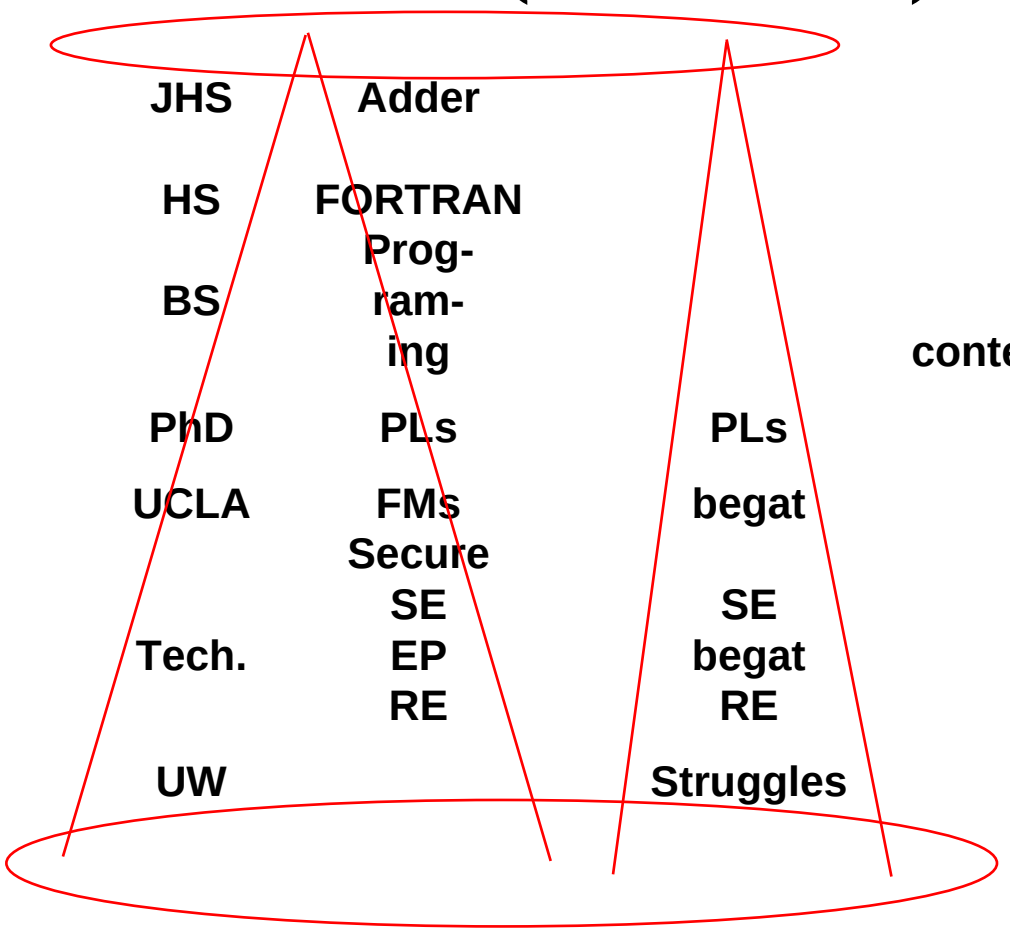
Even a FMs Person Got it

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

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


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.**

IWSSD's Modus Operandi

- **Each workshop had its designated CBS, e.g., meeting scheuler, library, elevator.**
- **An exemplar natural-language specification of it was distributed prior to the workshop.**
- **Each paper's authors should apply its method or tool to the exemplar.**

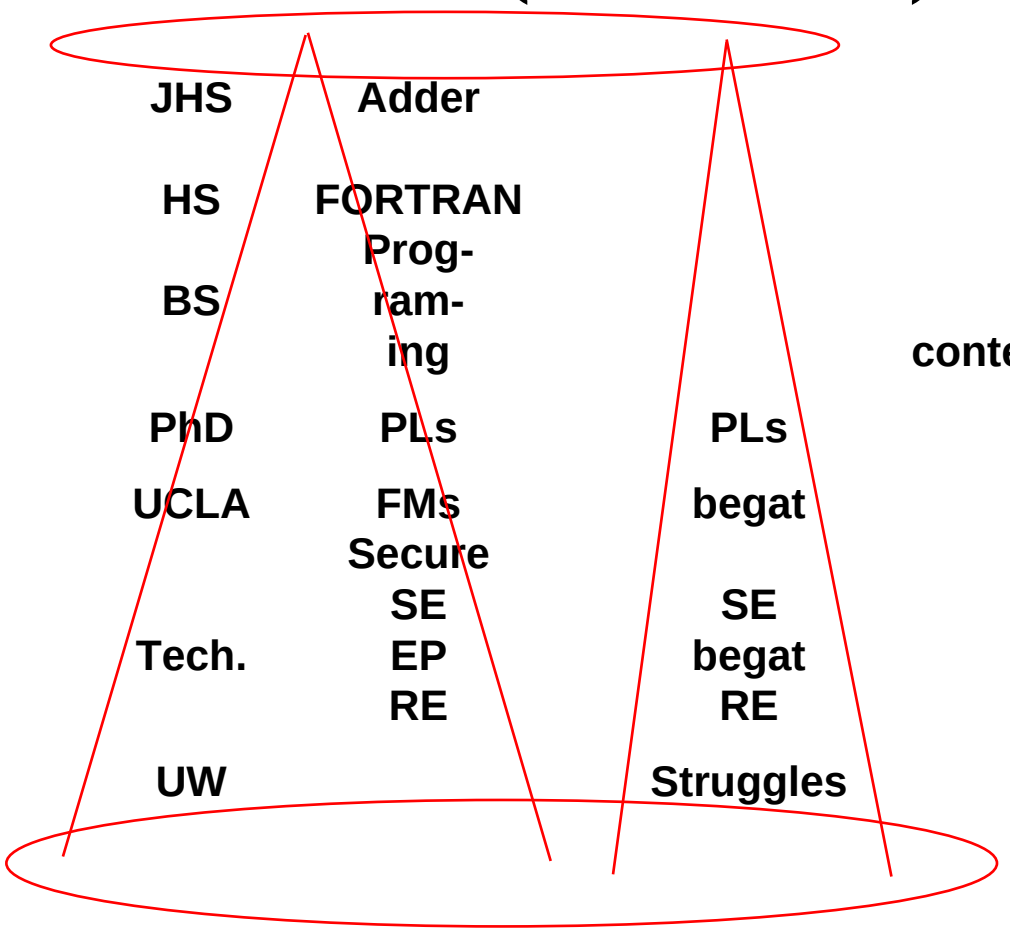
IWSSD Not an RE Conference

The workshop, as a whole, was still assuming that the requirements were given.

But we made a regular special session at the worksop dealing with elicitation!


2000s

Outline (Pictorial)



HCI from Graphics

I believe that the same forces that created RE out of SE, …

a realization that the hard part of development problem at hand was figuring out *what* to develop, …

created HCI out of Computer Graphics.

RE Now

Even within RE, there has been a lot of concern for …

technology: notation, methods, tools, FMs, etc. …

as well as for …

the human side: elicitation, creativity, emotions, politics, psychology, sociology, etc.

Both Are Important

Both technology and the human side are important.

Both need to be studied thoroughly and should be the subject of research.

A Continuous Struggle

However, I find a continuous struggle within RE that mirrors the decades-long struggle that created RE from PLs via SE.

The struggle is that:

- **As CSers, we *love* technology. We like to think that technology can solve *all* problems.**
- **But, we discover that it doesn't, sometimes to our surprise.**

Struggle Within PL Field

For example, we thought that designing the perfect PL would improve SW development.

They did, …

but not anywhere nearly enough.

PL Field Struggle, Cont'd

The problem was that the process of making the SW has a bigger impact than the PL on the eventual quality of the SW.

So we invented SE to focus on the actual process of developing SW.

Struggle Within SE Field

We thought that methods and tools applied to the actual programming would improve SW development.

They did, …

but not anywhere nearly enough.

SE Field Struggle, Cont'd

The problem was that following the best methods was useless if we did not know what to build, and …

the available methods had no effect on getting that knowledge.

So we invented RE to focus on the process of deciding what to build.

Struggle Within RE Field

It's always a tension between technology, methods, tools, etc.

and a human thing, e.g. how do we *humans* develop, how do we *humans* find out how to build.

As in SE, both are essential, …

but within the RE field, this tension continues.

Even From the Beginning

I remember in the early 90s, when we were piggy backing on the IWSSD conference in the Requirements Elicitation Track, when many a paper offered a new method, occasionally with a tool, for analyzing requirements specifications.

We were trying to accumulate a collection of exemplar specifications that could be used to test any such method or any such tool in a standard, comparable way.

Exemplar Specs

I was going along with this, when all of a sudden it hit me:

These examplars are too late!

They represent the *polished output* of the process that we were concerned about, namely requirements elicitation.

Exemplar Specs, Cont'd

Each exemplar needs to be a collection of the horrendously incomplete, inconsistent, sloppy *initial* documents that are produced by members of a customer's organization when they first decide to build any system.

These are unpolished RFPs, vision documents, etc.

Exemplar Specs, Cont'd

Not everyone agreed with me, and some agreed only partially.

But about 5 years later, I saw this:

Automated Software Engineering 4, 419–438 (1997)

Requirements and Specification Exemplars

MARTIN S. FEATHER feather@jpl.nasa.gov
Jet Propulsion Laboratory, Pasadena

STEPHEN FICKAS fickas@cs.uoregon.edu
Computer Science Department, University of Oregon

ANTHONY FINKELSTEIN acwf@cs.city.ac.uk
Department of Computer Science, City University

AXEL VAN LAMSWEERDE avl@info.ucl.ac.be
Département d'Ingénierie Informatique, Université Catholique de Louvain

Abstract. Specification exemplars are familiar to most software engineering researchers. For instance, many will have encountered the well known library and lift problem statements, and will have seen one or more published specifications. Exemplars may serve several purposes: to drive and communicate individual research advances; to establish research agendas and to compare and contrast alternative approaches; and, ultimately, to lead to advances in software development practices.

Because of their prevalence in the literature, exemplars are worth critical study. In this paper we consider the purposes that exemplars may serve, and explore the incompatibilities inherent in trying to serve several of them at once. Researchers should therefore be clear about what successfully handling an exemplar demonstrates. We go on to examine the use of exemplars not only for writing specifications (an end product of requirements engineering), but also for the requirements engineering process itself. In particular, requirements for good requirements exemplars are suggested and ways of obtaining such exemplars are discussed.

....

Acknowledgment

Thanks to Dan Berry, Robert Darimont, Philippe Massonet, Bashar Nuseibeh, David Till and the anonymous reviewers for their help, guidance and suggestions.

Struggle Over Technology

You find RE researchers developing techniques, methods, and tools, i.e., technology.

Often this technology is being developed to assist in doing a task that people do not like to do, e.g., tracing.

Struggle Over …, Cont'd

The main reason a person doesn't like to do such a task, is that the beneficiary of the task is someone else down stream, and …

the person who has the knowledge to do the task gains nothing from doing the task [Arkley & Riddle] other than a pain in the tukhis, …

mainly because he or she already has the knowledge.

I.e., there is no incentive to do the task, *even* if there is assistive technology.

Struggle Over …, Cont'd

But, if people have no incentive to apply the technology,

- **in the interest of being more agile,**
- **because the technology is too cumbersome, or**
- **they don't even see the value of the doing what the technology helps them do,**

then, the technology is not going to be applied, no matter how good it is.

Struggle Over …, Cont'd

Unfortunately, many technology developers are failing to consider this human aspect.

(Note that this is all independent of NIH (not invented here).)

Another Struggle

RE in practice involves a lot of talking with people and asking them questions.

Yet, you find an attitude that just talking with people and asking them questions isn't sexy enough to be the subject of good RE research.

RE is Very Inclusive

To me, RE includes *anything*, I repeat, *anything*, that can be shown to …

improve the process by which we determine the requirements of a CBS and …

that leads, downstream, to a better CBS …

as a result of what is done to determine the CBS's requirements.

Very Inclusive, Cont'd

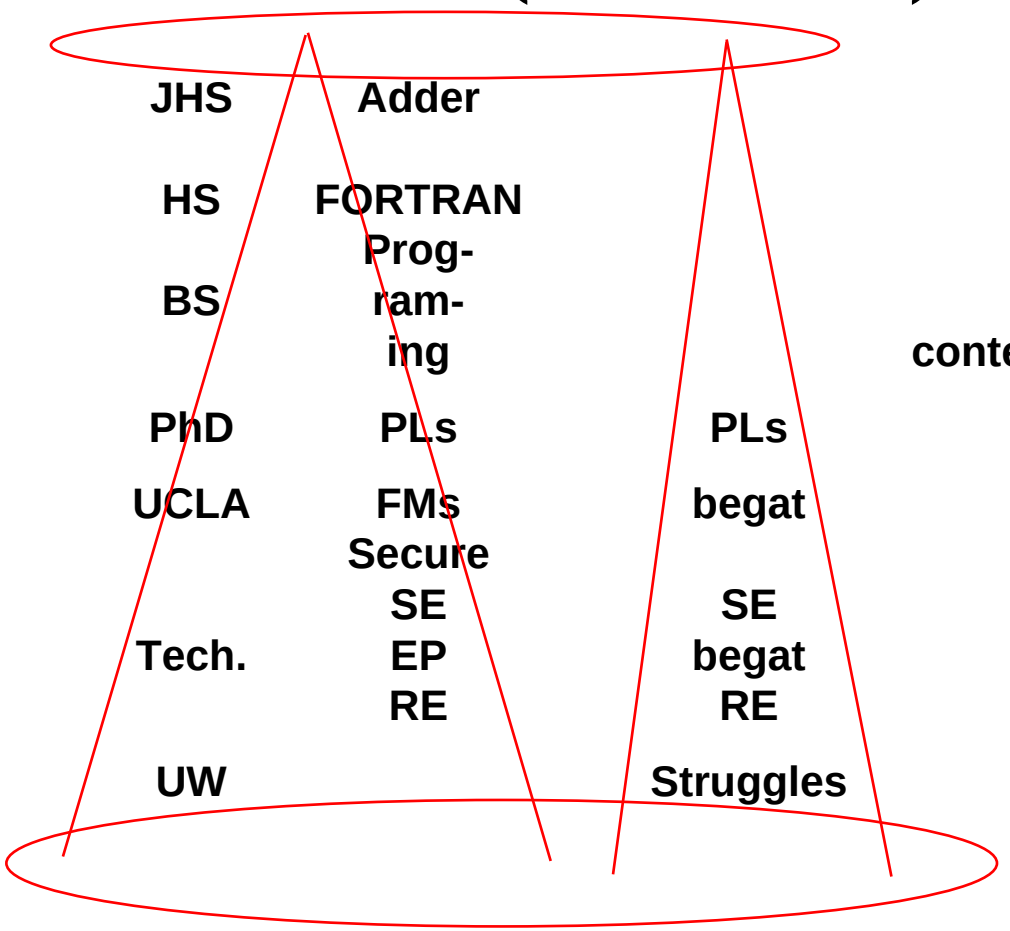
I don't care *what* the *anything* is —

technology, psychology, sociology, management, role playing, fun and games, and even feeding your client milk and cookies before eliciting requirements

— so long as it has an empirically demonstrable positive effect on the requirements gathering and on the eventual CBS!


2010s

Outline (Pictorial)



RE Everywhere

I see RE problems and lessons …

walking down the street, …

everywhere!

(The ambiguity of who is walking is intended.)

E.g., in house building, house remodeling, NY bagels, a synagogue's kitchen, the atrium of UW's Davis Centre, Waterloo Region's light rail, U.S. income tax instruction booklet, and even Biblical passages.

In Today's World!

In today's world, everything, especially SW development, is multi-disciplinary.

At Google, requirements elicitation teams have people from multiple disciplines, experts in the CBS's domain, engineers, lawyers, psychologists, sociologists, HCI experts, UX experts, and even SW engineers, ☺ …

- **to gather what is needed for the CBS, and**
- **to gather *new, out of the box* ideas.**

Conclusion

I have been in computing in one way or another since 1963 and have been programming since 1965.

While I have been in a whole gamut of CS fields and have picked up understandings of other CS fields, …

often, by supervising a graduate student who picked his or her own topic and taught it to me.

Conclusion, Cont'd

I see now that I have always been heading towards my current field, RE, …

because, in retrospect, no matter what X I was building, the hardest problem that demanded most of my attention was "What is really required of X?", i.e., "What are X's requirements?"

Lessons I Have Learned:

- **importance of talking with customers and users,**
- **importance of domain ignorance in RE,**
- **security, robustness, user interfaces, etc. have to be *required* into a CBS from the beginning,**
- **importance of knowing what the CBS is to do, as much and as early as possible,**
- **RE is everywhere, and**
- **every RE rule has exceptions.**

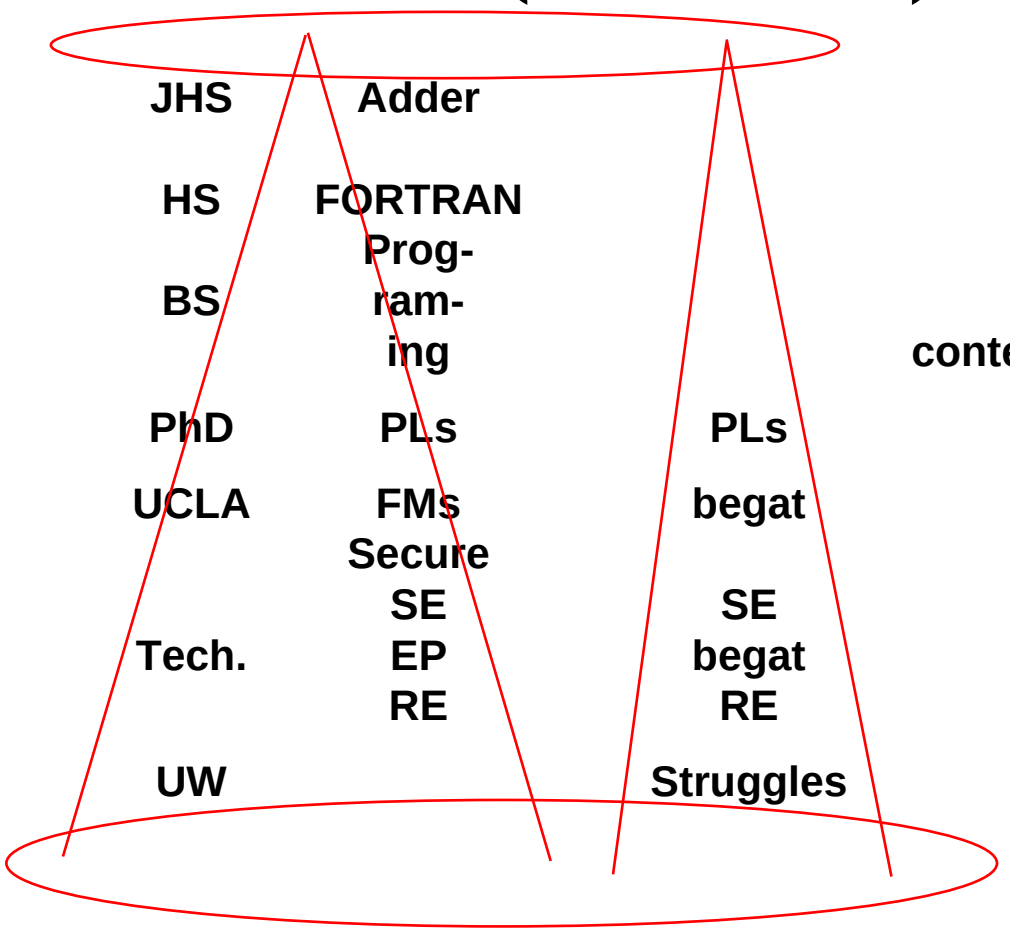
Take Away

My main take away message is very simple:

The RE field includes whatever helps do RE in real life.

And I intentionally left off “for CBSs” in the previous sentence.

Outline (Pictorial)



SE History Appendix

Continuing the SE Thread from the point of the RE split in early 1990s:

Losing Faith

In the early 1990s, I began to lose faith in the traditional here's-a-new-method-or-tool-that-will-solve-all-your-SW-development-problems SE research, …

and even in the here's-a-new-method-or-tool-that-will-solve-some-of-your-SW-develop-ment-problems reseach, …

i.e., neat, cool solutions to solve not-very-real or non-existent problems that we thought *had* to exist.

Losing Faith, Cont'd

Each new method or tool was YAM or YAT that was really no better than any of the preceding Ms or Ts for the same purpose.

Of course, each such M or T worked better than the others for the developers of the M or T!

I was just as guilty as everyone else!

Paying Attention to …

I began to pay more attention to the practitioners that were attending and were saying, in essence, that we researchers were not *listening* to what practitioners were saying were the *real* problems that they faced.

People like Barry Boehm, Fred Brooks, Bill Curtis, Tom DeMarco, Tim Lister, etc.

Paying Attention, Cont'd

And they were saying that the hard problems were

- ***people* problems and**
- **just understanding the problems that a system to be developed was being asked to solve.**

Technology, i.e., methods and tools, did not really address these problems.

To Empirical SE

In 1998, Walter Tichy, of RCS fame, asked in *IEEE Computer* "Should computer scientists experiment more?" and answered in the positive, especially in SE:

To Empirical SE, Cont'd

Computer scientists and practitioners defend their lack of experimentation with a wide range of arguments. Some arguments suggest that experimentation is inappropriate, too difficult, useless, and even harmful. This article discusses several such arguments to illustrate the importance of experimentation for computer science.

He argued that we need to put the *science* into CS.

Importance of Empirical SE

Experimentation is needed to test the validity of long-held, folklore-, belief-, and even theory-supported claims of method and tool effectiveness, e.g.,

"… the famous Knight-and-Leveson experiment … [about] the failure probabilities of multi-version programs. Conventional theory [Avizienis et al.] predicted that the failure probability of a multi-version program was the product of the failure probabilities of the individual versions.

Importance of …, Cont'd

"[The experiment showed that] the failure probabilities of real multi-version programs were significantly higher [than predicted]. … the experiment falsified the basic assumption of conventional theory, namely that faults in program versions are statistically independent."

Very Satisfying Result

Personally, this was a very satisfying result, because I had been the outvoted dissenting examiner of a PhD thesis of one of Avizienis's students that reported as "promising" the results of an experiment run in my SE class in which most three-version programs were more incorrect on test data than the least correct individual program.

Conundrum

The basic conundrum of experimental SE:

An experiment about an SE method or tool that is small enough to be well controlled and repeated enough to be statistically valid (internally valid) …

is *too small* to be realistic and generalizable to real-life SW development (externally valid).

Conundrum, Cont'd

How the hell do you conduct a realistic, but controlled experiment to compare the effectiveness of waterfall and agile development approaches?

Conundrum, Cont'd

How realistic is a two-hour fully controlled experiment comparing 20 3-person teams doing waterfall to 20 3-person teams doing agile on the same programming problem with the same programming language with random assignment of people to teams, etc.

(to make sure that the only difference between the teams are the approaches used)?

Conundrum, Cont'd

How generalizable is a comparison between two multi-year industrial developments of systems, one using waterfall and one using agile methods?

(The languages, the systems developed, the people, the team sizes, etc. may all be different in the two developments.)

Experiments on Inspections

In the mid to late 1990s, there were lots of experiments to assess the costs and benefits, e.g., compared to testing, of many variations of Mike Fagan's code inspections.

These experiments, conducted by Vic Basili, John Knight, Dewayne Perry, Aadm Porter, Harvey Siy, Larry Votta, etc. systematically tested all sorts of variations, e.g., team sizes, durations of steps, checklists or not, real or virtual meeting, synchronous or not, etc.

Doubly Valid Experiments

These experiments were internally *and* externally valid because of the *lucky* accident that …

a real-life inspection meeting lasts about two hours, …

and is thus experiment sized!

Essentially, no other real-life SE process is experiment sized.

Acknowledgements

Thanks to Jo Atlee, Nancy Day, Shahram Esmaeilsabzali, Mike Godfrey, Dana Mohaplova, Derek Rayside, and Vicky Sakhnini for their comments on dry runs of this talk.