CECS 343 SWE – Final Pre-view II Wk #16 vLect 5/6

Final Pre-view II
Final: CSULB CECS-343 on Wed 5/13 7:15-9:15pm online

<Review posted async video lecture material>
*** Read the Paper ***
"No Silver Bullet: Essence and Accidents of Software Engineering" 1986
Fred Brooks, 34th Turing (IBM; also taught w/ Dave Parnas)

Sources (2):
1. https://www.cs.unc.edu/techreports/86-020.pdf
2. The Mythical Man-Month: Essays on Software Engineering
   o- Brooks, 2ed. 1999

Overview Sections 1 & 2

(What to watch for in the paper)

Pbm: Doing S/W Project & Avoiding **FAILURE**
Keys:
Essential vs Accidental Difficulties
The Irreducible Essence

Essential vs Accidental Difficulties
== Details ==
Accidental:
Tools/Materials (T/M) Issues
& Mistakes in Mapping Arch/Concepts (from Pbm Domain) into T/M

Essence:
Conceptual Issues
& Mistakes in Mapping "True Pbm" to "Reqts & Model/Arch/Design"

Conceptual Parts
- o- **Declarative** (no timing/sequence)
  - o-- Major data + Modularity + linkages
  - o-- Vastly easier w/out timing
- o- **Procedural** (sequence + timing)
  - o-- Algorithms/Fcns + Data flow
  - o-- Flow + 'Feedback' influences state

Conceptual Steps
1. **User Spec**: (Actually) What User Needs
2. **Design**: S/W to Deliver Needs
3. **Validation**: Will Design Deliver Needs?

(*) All BEFORE (ramp up staffing) S/W is Built!
(& Before Detailed-Design, etc.)
But Needed, else high failure risk

3'. **State of the Art, Today**: Negligible Validation
  o- Does the Design SEEM okay? (Subjective)
(State of the Art:
  o- Large % Projects abandoned
    or non-useful on delivery
    or massive overrun on predictions)

Irreducible Essence Keys
1. **Complexity**:
  o- Scales **exponentially** with project size

o2. **Conformity**: Pgm not isolated.
  o- Must **work with outside** stuff

o3. **Changeability**:
  o- Initial Pbm Understanding is **Never Correct**

o4. **Invisibility**/Emergence(unforeseen & significant)
  o- Many Interlocking Concepts
  o- Many data Flows, with Feedback // Looping (which can cause "chaos")
  o- Many simplified Visualizations (but they are interlocking in complex ways)
  (*) S/W Too Complex to Visualize

  Quote: "As soon as we attempt to **diagram** ...
    usually **not even planar**, much less hierarchical."

  Traditional Sol'n: **Model/Abstraction**
    o- AKA **Simplify by ignoring parts & hope those parts are not important**

Pbm: Can't find bugs **in ignored parts**
(*) S/W inherently un-visualizable

==== Sections 3 & 4 Brief Overview ====
3. **Break-thrus on Accidental** Difficulties
o- There HAVE been some
(And today, more can still be done)
o- But they don't address the Essence!
Recall: Accidental = Tools/Materials
These Break-thrus = Better T/M
(*) Like writing with a chisel on stone;
Hard to write a lot, Typos expensive;
o- Pen & Paper is easier

4. **Hopes for the Silver** (from 1986)
o- a "Silver Bullet" is a Killer Sol'n

(*) Why Need? To prevent Failed Projects
o- Large & Medium scale
o- To lesser extent, Small also

New Cutting-edge Tech -- Could it Help?

A quick look at each, & why likely not
Most are still applicable today

No Silver Bullet
Part II: Break-thrus & Hopes
Sections 3 & 4 Details

3. Break-thrus on Accident Difficulties
Recall:
Accidental == Tools/Materials Limits
& Mistakes Mapping Concepts into T/M

Here are a few, and how they helped:

3.1 **High-Level Languages HLLs** (1960's & 70's)
o- Pro: 5x dev speedup
o-- Improves Concept vs T/M mapping
o- Pro: Better abstractions, automates small stuff
vs ASM & memory layouts by hand
o- Con: New accidental stuff for mistakes
EX: Recursion (seems hard to learn)
EX: Iterators (iffy w/o immutables)
Q: Replaced what low-level parts
with what high-level parts?
Q: Why is "esoteric constructs" significant?

3.2 **Time-Sharing** == 1 (virtual) computer per dev'r
- o- Pro: 5x dev speedup
- o- Pro: Shorter feedback loop on typos/bugs
  vs Submit punched-card decks & DMV-waiting
- o- Con: Negligible !

Q: What is most serious effect of "interruption"?

3.3 **(IDEs) Unified Programming Environments**
- o- Auto-help: ID & find very simple details
- o- Like the Web as a quick ref quide
- o- Con: Complicated snakes & knobs, as always

Q: What Accidental do IDEs address?

(*) But these don't address the Essence

---

4. **Hopes for the Silver**
- o- a "Silver Bullet" is a Killer Sol'n:
  "Order-of-magnitude (10x) Gains
  in S/W Productivity and Quality"
- o- Brooks didn' think they would work out

(*) After 35 years, none of these helped much

Q: So what's the point of this section today?

A: Most of these are Still Touted as being Hopes

4.1 **High-Level Language HLLs Advances**

Mostly the ADA lang (1980) [looked "good", results not so much]
- o- Pro: strict type-safe abstractions, modules, hier structuring
- o- Cons:
  - o-- Strict type-safe doesn't scale well
  - o-- Over-rich feature set, hard to learn
  - o-- Minor "improves" over prior HLL langs

4.2 **OOP & other Technical Fads**
- o- Pro: Easy to build **bags** (wraps vars & fcns)
  & **hier replaces**/speeds cut-n-paste duping
- o- Con: tall class **hier is hard to change safely**
  & **mom-hatting hides bad** RT bugs

(Recall **SOLID/D**; esp. OCP & Liskov)

Q: How are ADTs related to "storage structure"?

Q: How are ADTs related to Hier Types?

Q: What is the issue with "large amounts of syntactic material"?

Q: What is "itself essential"?

Q: What is the "unnecessary underbrush"?

4.3 **AI**
o- Idea is intelligent help w coding
o- Maybe even help with avoiding RT bugs
o- Automate Verification Test?
(None of it happened; and still a long way off)
Q: What is AI-1? AI-2?
Q: Who is "Parnas"?
Q: What does Parnas say about "once we see how the program works"?

4.4 **Expert Systems (AI sub-field)**
(AKA **RBS**, Rule-Based System)
(NB, RBS by Newell & Simon: Turing & Nobel)
o- Idea is intelligent help w coding
o-- Includes advice, explains its reasoning
o- Cvt SME brains to RBS rules
o- Used today in simple repeatable domains
o- Best hope: prompt/check newbie Devrs
Q: What are the two "clear advantages" offered?
Q: What is the "most important advance offered", per Brooks?
Q: What is "the essential prerequisite" for building an expert system?
Q: How does Brooks characterize "the best" versus "the average" S/W Engr'g practice?

4.5 **"Automatic" Programming** (AKA **Formal Methods** 1)
o- Write Spec & push button --> Full Code
o- Like std HLL, just more complex/abstract
(*) Parnas: "**Glamorous, not meaningful**"
o- Doesn't Validate Reqts, just Verifies
(Today, Can't scale up; tiny pbms only)
EX: O.S. kernels, simpler applied math
Con: Works for pbms w few "knobs", too neat
Q: Who is "Parnas"?
Q: What has automatic programming "always been"?

4.6 **Graphical Pgmg**
o- Write diagrams & push button --> Code
o-- Simple easy neat domain, but still hard
o-- Viz hard; RT bugs very hard
Q: When do pgmrs "draw flowcharts"?
Q: What is the issue with "screens"?
Q: What is the "airplane-seat" metaphor?
Q: How do diagrams relate to "S/W elephant"?

4.7 **Pgm Verif** (AKA **Formal Mds** 2/3)
o- Verif Model before Build/Test
o- Verifs are hard work for small pgms
Con: Doesn't scale up
Con: Verif mech is a new source of bugs
(*) Con: Doesn't Validate Reqts
o- (Circular) requires rough model before pgm?
o-- But, you get verified model
Q: What takes up "much" of the S/W dev effort?
Q: What might verif "reduce", per Brooks?
Q: What is the "hardest part" of S/W dev?
Q: Why is the section's last sentence
a promotion for Agile?

4.8 **Environments (IDEs) and Tools**
o- Easiest stuff already done
(Today, 35 years and small progress)

4.9 **Workstations**
o- Amazingly better than Timesharing
o- Today: have laptops
& 2-,3-head screens
& million times faster/mem bigger
But NO DENT in the Essence

Next Episode: III. Promising Attacks
@@----------

Promising Attacks on Essence
Section 5 Overview

5. **Promising Attacks on the Essence**
Recall: What is the Essence?
Adequately Creating (for the User Pbm): Reqts + Model/Arch
So that the Complex Conceptual Structures (== Model/Arch)
Simultaneously Satisfies
1. Reqts (Verification)
2. User needs/wants (Validation)
(*) **All BEFORE we "staff up"** to Design/Code/Integrate/Deploy + Test/Fix

Why Attack the Essence?
(*) To **Avoid Prediction/Estimate Failure**
To create **great designs**:
o- faster, smaller, simpler, cleaner, and produced with less effort

Look for 5 Keys:
1. Validate (for Users) Reqts/Model BEFORE Build
2. Accurate up-front project Estimates (likely a pipe-dream)
3. Speed up Dev -- for Concepts
4. Avoid RT bugs & Big Integration pbms (main time-sink)
5. Easily handle Reqts Changes

---

5. Promising Attacks on Essence
=== Details ===
5. Intro
o- Essence = "the formulation of these complex conceptual structures"
Q: What is the "productivity equation"?

(5.1-2 Recap)
@@
5.1 **Buy versus Build (COTS)**
(*) Key: Assemble from **Reliable** Parts
Thus eliminate 85-**95**% dev time (~1/10th-size of **new code**)
It's why OOP *Reuse* is big -- but Reliable?
| Pro: Small debug time;
| Delivery immediate;
| Much cheaper than dev effort cost
| Better documentation
| (NB Pkg'g = 3x dev cost for In-house use)
| Con: Is there Applicable COTS?
| Q: Why did COTS become big in the '80s?
| Q: Name one new ('80s) "generalized" COTS S/W
| package type that stands out as a "dramatic
| exception" to Brooks' comment that these
| hadn't changed much from the '60s.

o5.2 **Reqts Refinement (and Adjustment) & Rapid Prototyping**
[Ever Fancier "slideshows" including interactive, but not real]
(*) Key: Expands Reqts->Model->Proto->Show User
o-- some Dev time, but still w/ small staff
o- Still BEFORE staff-up Design+Code+Integ...
o- **Gives up on 1-pass Reqts->Model->Validate**
| Q: What is the "hardest single part" of S/W dev?
| A: Deciding precisely what to build.
| o- Combo of Reqts & Model/Arch
| Q: What is the "most important function" a S/W
| designer/architect does for its client?
| A: Reqts Iterative Extraction and Refinement.
| o- Sounds a bit like Agile?
| (*) Cust/user doesn't know precisely and in detail what is needed
| o- Must allow for extensive iteration of Design-and-Show users
| o- And STILL not precise and detailed reqts
| without several rounds of Build-and-Try-It
| (*)* Iterative Rapid Prototyping to get reqts
| Def: Prototype Simulates Inportant I/Fs and does Main Fcns
| w/o Quality reqts (H/W, speed/size, frills, errors)
| (*) So cust/user can verify it's what's needed
| Q: **Spec up front? (Big-Bang Style)**
| **Brooks: "fundamentally wrong"**
| (*)** Cust viewpoint:
| Cannot Spec (RFQ), get bids, do contract,
| get it built and installed & it works well

o5.3 **Incremental Dev: Grow, not Build, S/W**
[Sounds like Agile !!]
(*) Key: S/W too complex; cuz
Fundamentally CAN'T be spec'd in advance
o- In Nature, really complex things are Grown
o- Grown by Incremental Development
o-- Seq of Rapid (Usable) Prototypes, each doing more
o-- We learn from Pgm mistakes and correct
Mistakes in: Reqts, Model, Design
Revealed by Testing (per spec & per user)
cuz Perfect-The-First-Time Never Works

Q: Who is "**Harlan Mills**"?
Q: What should a system "first be made to do"?
Q: What kind of "growing of software" is done?
Q: How does "morale" play a part?
Q: How does "Growing S/W" solve the Essence
Pbm BEFORE we build the full system?
(Because "AFTER" isn't the big question)

o5.4 **Great Designers**
[We give up and claim we need the 1% great people; so grow them.]
(*) Premise: Great Designs (AKA Model/Arch)
avoid Reqts+Model/Arch mistakes
Con: True: but requires "great designers"
o- Good Design Practices can be taught
o- Good vs Poor: Poor designs are unsound
o- **Diff between Great and Average designer:**
10x (up to 1000x) in debug/fixup time
(*) Average designer produces "Good Design"
(A sound design is "good enough" for awhile)
o- Brooks says: need to spend a pile of $$$ to
o1. **Find good candidates**
o2. Mentor them (Mentor is "great designer")
[Mentoring because it **largely can't be adequately written down**]
o3. Pay them a lot more
Cons:
1. How to tell if one is great?
2. Who wants a primadona?
(AKA do humble great designers exist?)
Q: What is the "central question in how to improve the S/W art"?
Q: What are the characteristics of designs from the "best designers"?

<Review posted vid-lect docs, 5 docs>
343-vlect-200415-Dsgn-Pats-Recur.pdf
343-vlect-200420-Quality+Testing.pdf
343-vlect-200422-Testing-Strategies+.pdf
343-vlect-200427-Proj-Plan-Ctrl.pdf
343-vlect-200429-Project-Plan+Metrics.pdf
[NOX]
o32.4 SW Process + Metrics
o32.4.2 Estab Baseline
o32.5 Metrics for Small Shop
o32.4.3 Metrics Collection, Calc, Eval
McConnell-2006 p 58 “the Black Art”
(*)* One Feature ...

xx