CS 390 – Lecture 35
No Silver Bullet


Fred Brooks

- Project manager of IBM OS/360
- 1999 ACM Turing Award winner
- Founder (1964), former chair (20 years) of UNC-Chapel Hill Computer Science Department
- Currently, Kenan Professor of Computer Science at UNC-CH

The Software Werewolf

- Software is like a werewolf – it looks normal until the moon comes out and it turns into a monster
  - Missed deadlines
  - Blown budgets
  - Buggy software
- Everyone wants a silver bullet to kill the monster

There Is No Silver Bullet!

“... no single software engineering development will produce an order of magnitude improvement in programming productivity within ten years...”

Essential vs. Accidental Difficulties

- Following Aristotle, difficulties are either **essential** or **accidental**
- Essential: constituting or being part of the essence of something; inherent
- Accidental: Of or relating to a property, factor, or attribute that is not essential
  - Note: This does not mean “by chance”

Essential vs. Accidental Difficulties (2)

- Applied to software development, are difficulties:
- Essential – a fundamental quality of software development, OR
- Accidental – a problem of today’s software production methods
Past Breakthroughs Solve Accidental Difficulties

- High-level languages: eliminates much of accidental complexity by matching the level of the abstract program
- Time-sharing systems: preserves immediacy of code to its result
- Integrated development environments: allows programs to work together as conceptualized
  - Libraries, file formats, pipes & filters

Analysis of Difficulties

- To get 10x improvement
  - Accidental difficulties would have to be 9/10 of the overall problem
  - Tools and techniques would have to reduce these difficulties to 0
- Brooks’ conclusion
  - First is not true (probably much less)
  - Second is highly unlikely to happen

Concept vs. Representation

"I believe the hard part of building software to be the specification, design, and testing of this conceptual construct, not the labor of representing it and testing the fidelity of the representation."

Essence of Software

- What makes writing software hard?
  - Complexity
  - Conformity
  - Changeability
  - Invisibility

Complexity

- No two parts are alike
- Huge numbers of states
  - "...orders of magnitude more states than computers do..."
- Module interactions scale nonlinearly with size
- Cannot know the whole domain, process, or system

Complexity (2)

- Cannot abstract away the complexity because it is essential to the operation of the program
  - Physics models work because the complex details are not concerned with the essence of the domain
- Complexity comes from the tight interrelationships between heterogeneous artifacts
  - Specifications, documentation, code, test cases, etc.
Consequences of Complexity

- Communication overhead: cost overruns, schedule delays, personnel turnover
- Large number of states: unreliability
- Complex function: poor usability
- Complex structure: poor maintainability, security risks

Conformity

- Software must conform to arbitrary limitations imposed by humans (e.g. business process rules)
- Software arrives late in system design
- Software viewed as most changeable
- It is hard to plan for arbitrary changes that will occur late in development

Changeability

- Software is easier to change than hardware
- Asked to change during development
- Asked to change after deployment (post delivery maintenance); rare for manufactured things
- People underestimate the difficulties of change

Invisibility

- Code is invisible and unvisualizable
- Structure is extremely complex
- Structure is hidden
- There is only the external input/output view

Discussion

- Are Brooks’ claims true for the last 20 years? Next 50 years?
- Are the characteristics given really unique to software?
- Are there other essential properties?
- Have there been other accidental difficulties solved?
- Has there been significant progress since 1987?

No “Silver Bullet” in 1987-1997

- Languages like Ada: Remaining benefit small
- OOP: Good, but essential complexity remains
- AI: Not generally applicable; expert systems need an expert
- Automatic programming: Not generally applicable
- Graphical programming: Unvisualizable
- Program verification: Specification is still hard
- Environments & tools: Remaining benefit small
- Workstations: Faster compiles, but that is all
Is There Any Hope?

- Compare to medicine
  - Throw out simple, fast fixes for demon-possession, four humours
  - Apply persistent effort to slowly eradicate disease
- Compare to chemistry
  - Throw out alchemy
  - Spend years to understand atoms, then learn to synthesize gold

Yes! There Is Hope

- “A disciplined, consistent effort to develop, propagate, and exploit these innovations should indeed yield an order-of-magnitude improvement. There is no royal road, but there is a road.”

Things to Try

- Reuse: “buy, don’t build”
- Requirements refinement and rapid prototyping
- Incremental development: “grow, don’t build”
- Better SE training: identification and nurturing of great designers

“No Silver Bullet’ Refired”


Reflections on “No Silver Bullet”

- Lots of controversy and rebuttals in 1987
  - But no dramatic effects by 1995!
- Clarified “essential” vs. “accidental” for software
  - Essential: mental crafting of the conceptual construct
  - Accidental: implementing that concept
- Brooks’ believed that rendering the code is down to half or less of total effort
  - And no one ever claimed it to be 9/10!

Reflections on “No Silver Bullet” (2)

- NSB is not pessimistic (“there is a road”. But so what if it is!
  - Einstein said we cannot go faster than the speed of light
  - Goedel said some things cannot be computed
  - Do not waste time searching for quick solutions
Essential Difficulties not Hopeless

- **Complexity**
  - Can we create order from chaos? Are there "laws of software" like the 3 laws of thermodynamics
  - Simplify the implementation: hierarchical abstraction, incremental construction

- **Invisibility**
  - "...one needs multiple diagrams, each covering some distinct aspect..."

Brooks on Brooks

- "Buy, don't build"
  - Underestimated customizability

- **Reuse**
  - Need organizational support
  - Need good design and good documentation
  - Must address the "vocabulary problem"

- **Object-orientation**
  - Very promising, but we need to scale up from low-level ADTs like stacks
  - Need to pay large up-front training costs

Summary

- No one advance will give 10x improvement
  - All accidental difficulties have been solved
  - No one advance will address essential difficulties
- But maybe better exploit what we have learned
- ...or just roll up our sleeves and get to work
- Refired: We are making incremental progress, and that is still all we can hope for

Reading for Wednesday

- Link to "No Silver Bullet Reloaded Retrospective OOPSLA Panel Summary" on course webpage
- Continue the discussion of the difficulties of developing software