Lecture 7: No Silver Bullet

Kenneth M. Anderson

Foundations of Software Engineering

CSCI 5828 - Spring Semester, 1999

Today's Lecture

- Discuss the No Silver Bullet paper
- Brook's reflections on it after nine years

No Silver Bullet

"There is no single development, in either technology or management technique, which by itself promises even one order-of-magnitude improvement within a decade in productivity, in reliability, in simplicity."

-- Fred Brooks, 1986

i.e. There is no magical cure for the "software crisis"

Why? Essence and Accidents

- Brooks divides the problems facing software engineering into two categories
  - essence
    - difficulties inherent in the nature of software
  - accidents
    - difficulties related to the production of software
- Brooks argues that most techniques attack the accidents of software engineering

An Order of Magnitude

- In order to improve the development process by a factor of 10
  - the accidents of software engineering would have to account for 9/10ths of the overall effort
  - tools would have to reduce accidents to zero
- Brooks
  - doesn't believe the former is true and
  - the latter is highly unlikely, even if it was true

The Essence

- Brooks divides the essence into four subcategories
  - complexity
  - conformity
  - changeability
  - invisibility
- Lets consider each in turn

Complexity

- Software entities are amazingly complex
  - No two parts (above statements) are alike
    - Contrast with materials in other domains
  - They have a huge number of states
    - Brooks claims they have an order of magnitude more states than computers (e.g. hardware) do
  - As the size of the system increases, its parts increase exponentially

Complexity, continued

- Problem
  - You can't abstract away the complexity
    - Physics models work because they abstract away complex details that are not concerned with the essence of the domain; with software the complexity is part of the essence!
  - The complexity comes from the tight interrelationships between heterogeneous artifacts: specs, docs, code, test cases, etc.

Complexity, continued

- Problems resulting from complexity
  - difficult team communication
  - product flaws
  - cost overruns
  - schedule delays
  - personnel turnover (loss of knowledge)
  - unenumerated states (lots of them)
  - lack of extensibility (complexity of structure)
  - unanticipated states (security loopholes)
  - project overview is difficult (impedes conceptual integrity)

Conformity

- A significant portion of the complexity facing software engineers is arbitrary
  - Consider a system designed to support a particular business process
  - New VP arrives and changes the process
  - System must now conform to the (from our perspective) arbitrary changes imposed by the VP

Conformity, continued

- Other instances of conformity
  - Non-standard module or user interfaces
    - Arbitrary since each created by different people
      - not because a domain demanded a particular interface
  - Adapting to a pre-existing environment
    - May be difficult to change the environment
    - however if the environment changes, the software system is expected to adapt!
- It is difficult to plan for arbitrary change!

Changeability

- Software is constantly asked to change
  - Other things are too, however
    - manufactured things are rarely changed
      - the changes appear in later models
      - automobiles are recalled infrequently
      - buildings are expensive to remodel
- With software, the pressures are greater
  - software = functionality (plus its malleable)
    - functionality is what often needs to be changed!

Invisibility

- Software is invisible and unvisualizable
  - In contrast to things like blueprints
    - here geometry helps to identify problems and optimizations of space
  - Its hard to diagram software
    - We find that one diagram may consist of many overlapping graphs rather than just one
      - flow of control, flow of data, patterns of dependency, etc.
- This lack of visualization deprives the engineer from using the brain's powerful visual skills

What about X?

- Brooks argues that past breakthroughs solve accidental difficulties
  - High-level languages
  - Time-Sharing
  - Programming Environments
- New hopefuls
  - Ada, OO Programming, AI, expert systems, "automatic" programming, etc.

Promising Attacks on Essence

- Buy vs. Build
  - Don't develop software at all!
- Rapid Prototyping
  - Brooks buys in
- Incremental Development
  - grow, not build, software
- Great designers

No Silver Bullet Refired

- Brooks reflects on the "No Silver Bullet" paper, ten years later
  - Lots of people have argued that there methodology is the silver bullet
    - If so, they didn't meet the deadline of 10 years!
  - Other people misunderstood what Brooks calls "obscure writing"
    - For instance, when he said "accidental", he did not mean "occurring by chance"

The size of "accidental" effort

- Some people misunderstood his point with the "9/10ths" figure
  - Brooks doesn't actually think that accidental effort is 9/10th of the job
    - its much smaller than that
  - As a result, reducing it to zero (which is probably impossible) will not give you an order of magnitude improvement

Obtaining the Increase

- Some people interpreted Brooks as saying that the essence could never be attacked
  - That's not his point however; he said that no *single* technique could produce an order of magnitude increase by itself
- He argued that several techniques in tandem could achieve that goal but that requires industry-wide enforcement and discipline

Obtaining the Increase, continued

- Brooks states
  - "We will surely make substantial progress over the next 40 years; an order of magnitude over 40 years is hardly magical…"