No Silver Bullet

Kenneth M. Anderson
University of Colorado, Boulder

CSCI 5828 — Supplement to Lecture 2 — 01/22/2008

Lecture Goals

- Introduce thesis of Fred Brook’s No Silver Bullet
  - Classic essay by Fred Brooks discussing “Why is SE so hard?”
  - Available at link below:
    - No Silver Bullet in IEEE Digitial Library

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
  - first, the accidents of software engineering would have to account for 90% of the overall effort
  - second, tools would have to reduce accidental problems to zero
- Brooks doesn't believe that the former is true…
  - and the latter is nigh impossible because each new tool or technique solves some problems **while introducing others**

The Essence

- Brooks divides the essence into four subcategories
  - complexity
  - conformity
  - changeability
  - invisibility

- Lets consider each in turn

Complexity (I)

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

Complexity (II)

- You can't abstract away the complexity of the application domain. Consider:
  - air traffic control
  - international banking
  - flight software for space craft
- These domains are intrinsically complex and this complexity will appear in the software system as designers attempt to model the domain
  - Complexity also comes from the numerous and tight relationships between heterogeneous software artifacts such as specs, docs, code, test cases, etc.

Complexity (III)

- 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

Conformity

- A significant portion of the complexity facing software engineers is **arbitrary**
  - Consider designing a software system for an existing business process and a new VP arrives at the company
  - The VP decides to “make a mark” on the company and changes the business process
  - Our system must now conform to the (from our perspective) arbitrary changes imposed by the VP
- Other instances of conformity
  - Having to integrate with a non-standard module interface
  - Adapting to a pre-existing environment
    - if env. changes, you can bet that software will be asked to change
- Main Point: Its is almost impossible to plan for arbitrary change; instead, you just have to wait for it to occur and deal with it when it happens

Changeability

- Software is constantly asked to change
    - Other things are too, however, manufactured things are rarely changed after they have been created
        - instead, changes appear in later models
            - automobiles are recalled only infrequently
            - buildings are expensive to remodel
- With software, the pressure to change is greater
    - in a project, it is functionality that is often asked to change and software EQUALS functionality (plus its malleable)
    - clients of a software project often don't understand enough about software to understand when a change request requires significant rework of an existing system

Invisibility

- Software is by its nature invisible; and it is difficult to design graphical displays of software that convey meaning to developers
- Contrast to blueprints: here geometry can be used to identify problems and help optimize the use of space
- But with software, its difficult to reduce it to diagrams
- Hard to get both a “big picture” view as well as a set of detailed views
  - Hard to convey just one issue on a single diagram; instead multiple concerns crowd and/or clutter the diagram hindering understanding
  - 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
  - OO Analysis, Design, Programming
  - …

Promising Attacks on the Essence

- Buy vs. Build
  - Don't develop software when you can avoid it
- Rapid Prototyping
  - Use to clarify requirements
- Incremental Development
  - don't build software, grow it
- Great designers
  - Be on the look out for them, when you find them, don't let go!

No Silver Bullet Refired

- Brooks reflects on the “No Silver Bullet” paper, ten years later
  - Lots of people have argued that their methodology, technique, or tool is the silver bullet for software engineering
    - If so, they didn't meet the deadline of 10 years or the target of a 10 times improvement in the production of software
- Other people misunderstood what Brooks calls “obscure writing”
  - e.g., when he said “accidental”, he did not mean “occurring by chance”;
    - instead, he meant that the use of technique A for benefit B unfortunately introduced problem C into the process of software development

The Size of Accidental Effort

- Some people misunderstood his point with the 90% figure
  - Brooks doesn't actually think that accidental effort is 90% of the job
    - its much smaller than that
- As a result, reducing it to zero (which is effectively 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
- Brooks states:
  - We will surely make substantial progress over the next 40 years; an order of magnitude over 40 years is hardly magical…

Coming Up Next

- Lecture 3: Introduction to Concurrency
    - Chapter 1 of Magee and Kramer