No Silver Bullet

CSCI 5828: Foundations of Software Engineering
Lecture 02 — 08/27/2015

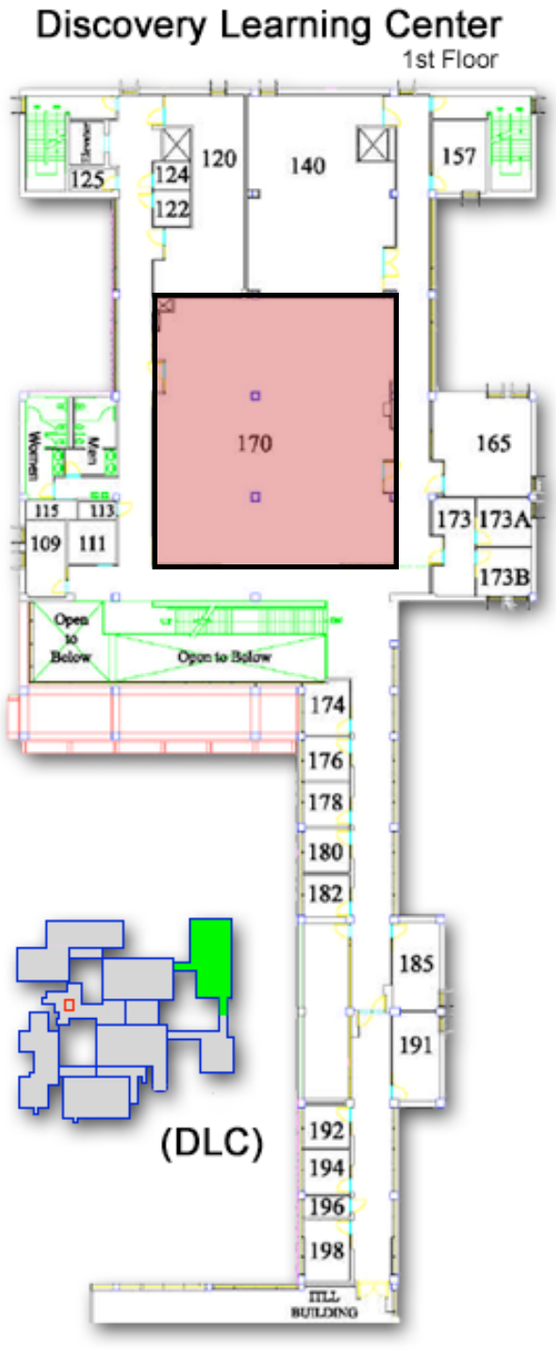
Getting my Act Together

- Two Announcements
  - **First:** in Lecture 1, I had a slide that "announced' my office hours as Fridays from 2 PM to 3 PM
  - **Second:** I also confidently proclaimed that the midterm was going to be held on Tuesday of Week 8: October 13th

- BOTH OF THESE ANNOUNCEMENTS WERE **WRONG** (sigh)
  - So, let's try this again

Office Hours

- Mondays, 3 PM from 4 PM
- Wednesdays, 12 PM to 1 PM
  - DLC 170 (shown in red on right)
- Please send me e-mail to let me know you plan to stop by
  - There WILL be some days where I cannot hold the office hours as planned
    - So, if you e-mail me beforehand I can let you know if it will work for sure



Midterm

- **The midterm will be held on Tuesday, October 20th**
  - BBA students will need to work with BBA to identify a person to proctor their midterm exam; You will have from October 20th to October 27th (one week) to take your exam and have it sent to me by your proctor
- This is Week 9 of the semester
  - It turns out that I am gone for a trip to the Grace Hopper conference during week 8.
  - That means that I will be pre-taping week 8's lectures during week 7. They will run at the regular class times during week 8 while I'm gone.
- Questions?

Lecture Goals

- Introduce thesis of Fred Brook's No Silver Bullet
  - Classic essay by Fred Brooks discussing "Why is SE so hard?"

Image © Bryan M. Mathers; Used via a CC License

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"

- **NOTE:** From this statement you can infer the **definition** of a "silver bullet":
  - *A single technique or technology that by itself can deliver one order-of-magnitude improvement to some aspect of software development.*
- Note: one order of magnitude is the same as saying a 10x improvement

Why? Essence and Accidents

- Brooks divides the problems facing software engineering into two categories
  - **essence**: difficulties inherent, or intrinsic, 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 software development 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, both the number **and types** of parts increase exponentially
  - the latter increase is the most significant

Complexity (II)

- You can't abstract away the complexity of the application domain. Consider:
  - air traffic control, international banking, avionics software
- 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 (I)

- A lot of complexity facing software engineers is **arbitrary**
  - Consider designing a software system to support an existing business process when 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

Conformity (II)

- Other instances of conformity
  - Adapting to a pre-existing environment
    - such as integrating with legacy systems
    - and if the environment changes (for whatever reason), you can bet that software will be asked to change in response
  - Implementing regulations or rules that may change from year to year
  - Dealing with a change in vendor imposed by your customer
- Main Point: **It 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 (I)

- 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

Changeability (II)

- 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
    - Contrast with more tangible domains
      - Imagine asking for a new layout of a house after the foundation has been poured

Invisibility (I)

- Software is, by its nature, invisible and intangible; 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
    - UML contains 13 different diagram types (!)
        - to model class structure, object relationships, activities, event handling, software architecture, deployment, packages, etc.
    - The notations of the different types almost never appear in the same diagram
        - they really do document 13 different aspects of the software system!

Invisibility (II)

- Hard to get both a “big picture” view as well as details
  - 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, Take 2

- Brooks reflects on No Silver Bullet‡, 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
- Others misunderstood what Brooks calls “obscure writing”
  - e.g., “accidental” 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

‡ This reflection appears in The Mythical Man-Month, 20th Anniversary Edition

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 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; he said that **no single technique could produce an order of magnitude increase by itself**
  - He argues instead that **several techniques in tandem could achieve it 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 improvement over 40 years is hardly magical…”

Quiz Yourself

- Essence or Accident?
  - A bug in a financial system is discovered that came from a conflict in state/federal regulations on one type of transaction
  - A program developed in two weeks using a whiz bang new application framework is unable to handle multiple threads since the framework is not thread safe
  - A new version of a compiler generates code that crashes on 32-bit architectures; the previous version did not
  - A fickle customer submits 10 change requests per week after receiving the first usable version of a software system

Coming Up Next

- Lecture 2a: Git, Github, and Markdown

- Homework 1 is due by the **start** of Lecture 4 (next Thursday)
  - See class website for details