CSE584: Software Engineering

Lecture 1 (September 28, 1998)

David Notkin
Dept. of Computer Science & Engineering
University of Washington
www.cs.washington.edu/education/courses/584/CurrentQtr/

Lecture 1, Outline

- Intent and overview of course
- Overview of course work
- Notkin's top 10 "insights"
- Software engineering overview
  - Stuff you already know, but it's important to lay it out so we are working from the same page
- Administrivia and slop
  - Except tonight, *Tuesdays* 6:30-9:30PM
  - No lecture, November 3
    - ACM SIGSOFT Symposium on the Foundations of Software Engineering

Introductions?

- Very useful for me
  - What do you do?
  - What do you want from the class?
  - What are the most serious software engineering problems you face?
- But time consuming
- I'll have you do this electronically

But I do want some basics

? What companies do you work for?
? What is your general responsibility?
  ? Development, testing, maintenance, other?
- Take a couple of minutes at each site to gather these data
  - Jake Cockrell, our TA, will handle the UW site
  - The person whose last name comes first alphabetically handles the other sites
- Announce when you're ready

Distance learning

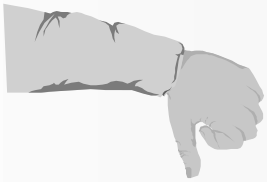
- This is my first try at teaching using distance learning
- So I'll need help in determining what works and what doesn't work
  - Be vocal about this (for immediate things, during class; for other things, by email)
- The whiteboard looks like a huge win
  - We will try to capture the images and put them on the web page

Interaction

- I like to have interaction students during class, especially 584
  - You have tons of key insights in your head
  - It's boring just listening to me
    - Especially in the evening & during a long class
- We have a new telephone-based audio system, which should help reduce lag
- Try just interrupting me; if that doesn't work, we'll try something else

Your undergraduate experience?

- How many of you took an undergraduate software engineering course?
- Did any of you think it was good?
- What, specifically, was particularly good or bad about it?



This is my guess about your answers

Intent of course

- Most of you have jobs engineering software
  - I don't (and I never really have)
- So, what can I teach you?
  - Convey the state-of-the-art
  - Better understand best and worst practices
  - Consider differences in software engineering of different kinds of software
- You provide the context and experience
- *Meeting and talking to each other is key*

Another key intent

- There is general agreement that
  - Research in software engineering doesn't have enough influence on industrial practice
  - And much in industry could improve
- Why is this true?
  - What can academia do to improve the situation?
  - What can industry do to improve the situation?

Possible impediments

- Lack of communication
  - Industry doesn't listen to academia
  - Academia doesn't understand industrial problems
- Academic tools often support languages not commonly used in industry
- In groups of 3 or 4, list some other possible impediments
  - In 5 minutes, a few groups will report their lists

Tichy's main impediment

- The lack of "experiments" in CS research
- I have lots of reactions, including
  - I don't think industry, as a rule, finds this to be a (the) major impediment
  - We do experimentation, in a different style
  - Evaluation is difficult in software engineering, so we must be creative
  - This is an example of science envy

Overview—five topics

- Design
- Evolution (maintenance, reverse engineering, reengineering)
- Requirements and specification
- Analyses and tools (static and dynamic)
- Quality assurance and testing
- Yes, there is some overlap

What's omitted? Lots

- Metrics and measurement
  - Some in QA
- CASE
  - Some in evolution and tools
- Software process
  - CMM, ISO 9000, etc.
- Specific methodologies
- What else?

Design (2 lectures)

- 1st lecture—classic topics
  - Information hiding
  - Layered systems
  - Event-based designs (implicit invocation)
- 2nd lecture—neo-modern design
  - Limitations of classic information hiding
  - Design patterns
  - Software architecture
  - Frameworks

Evolution (2 lectures)

- Why software must change
- How and why software structure degrades
- Approaches to reducing structural degradation
- Problem-program mapping
- Program understanding, comprehension, summarization

Requirements (2 lectures)

- Domain analysis
  - Use-case, collaborations, etc.
- Formal methods
  - State-based, algebraic, model-based
  - Model checking

Analyses and Tools (2 lectures)

- Static analyses
  - Type checkers
  - Extended type checkers
- Dynamic analyses
  - Profiling
  - Memory tools
  - Inferring invariants
    - Some cool research we've just started — I'd love your feedback on this (and love an alpha-tester even more — maybe a term project?)

Quality assurance (1 lecture)

- Verification vs. validation
- Testing
  - White box, black box, etc.
- Reliability
- Safety (maybe not, depending on overlap with 504)

Anything else?

Overview of course work

- Reports on the readings
- One web page for the class for each of the first four technical topics
  - Two students in charge of each of these 4 pages
  - This is a new approach, so I'm not 100% certain what to expect (or even what I expect)
- A final project (singly or in pairs)
- Details on the web --- clarifications through email, phone calls, etc.

Grading

- The web page lists the weights of the different parts of assigned work
- But I'll make you a deal
  - If you focus on the material and don't get compulsive about grading …
  - … then I will focus on the material and not get compulsive about grades

Notkin's Top 10 Observations

- About software engineering
  - With apologies and appreciation to many unnamed souls
- I'd appreciate help revising this list over the quarter

Number 0

- OK, I lied, there are 11 :-)
- Given it's my first distance learning course, I now get confused when you use the word, "makeup"

Thanks to:
http://members.aol.com/BoneEfx/index.html

Number 1

- We make a huge mistake by assuming similarity among software systems
  - Ex: Does (and should) the reliability of a nuclear power plant shutdown system tell us much about the reliability of an educational game program?
  - Ex: Does (and should) the design of a sorting algorithm tell us much about the design of an event-based GUI?
  - So, assume differences until proven otherwise

Number 2

- Intellectual tools still dominate mechanical tools in importance
  - How you think is more important than the notations, tools, etc. that you use
  - Ex: Information hiding is a key design principle
    - Interface mechanisms can enforce information hiding decisions but cannot help one make the decisions
  - Ex: The notion of design patterns is more important than languages that let you encode them

Number 3

- Analogies to other engineering disciplines are attractive but generally fall apart quickly
  - One key reason is because of the incredible rate of change in hardware and software technology
  - Another is that software seems to be constrained by few physical laws
  - But I'll make them anyway, I'm sure (and you will, too)
  - This is a variation on #1

Number 4

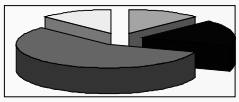
- It is often too easy to estimate the benefits of a "better" approach to engineering software without assessing its costs
  - "If only everyone only built software my way, it'd be great" is a common misrepresentation
    - Ex: The formal methods community is just starting to understand this
  - But at the same time, estimating the costs and the benefits is extremely hard, leaving us without a good way to figure out what to do

Number 5

- The properties that programming languages can ensure are distant from the properties we require software systems to have
  - Programming languages can help a lot, but they can't solve the "software engineering" problem
  - Ex: Contravariant type checking (such as in ML) has significant benefits, but regardless, it doesn't eliminate all errors in ML programs
    - And covariant typing, with its flaws, may be useful in some situations

Number 6

- The total software lifecycle cost will always be 100%
  - Software development and maintenance will always cost too much
  - Software managers will always bitch and moan
  - Software engineering researchers will always have jobs



Number 7

- Software engineering draws on mathematics, cognitive psychology, management, etc., but it is *engineering* and
  - not mathematics, nor cognitive psychology, nor management (nor etc.)
  - If somebody is talking about software without ever mentioning "software", run away

Number 8

- Tradeoffs are at the heart of software engineering, but we're not very good at it
  - Getting something for nothing is great, but it isn't usually possible
  - We almost always choose in favor of hard criteria (e.g., performance) over soft criteria (e.g., extensibility)
    - This makes sense, both practically and theoretically
    - Brooks' Golden Rule doesn't really work
    - But the situation leaves us up a creek to a large degree

Number 9

- It's always good to (re-)read anything written by Brooks, Jackson, and Parnas
  - Don't fall into Mark Twain's trap:
    - "A classic is something everyone wants to have read, but nobody wants to read."

Number 10

- Software engineering researchers should have a bit of the practitioner in them, and software engineering practitioners should have a bit of the researcher in them
  - At the end of the quarter, I hope that I'll have more understanding of practice, and you'll have more understanding of the research world

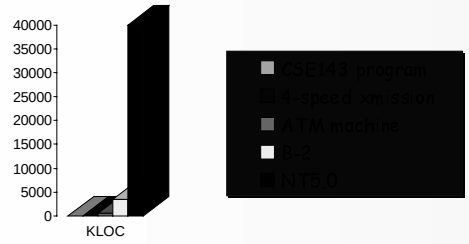
Software is critical to society

- Economically important
- Essential for running more enterprises
- Key part of most complex systems
- Essential for designing many engineering products

Sample code sizes

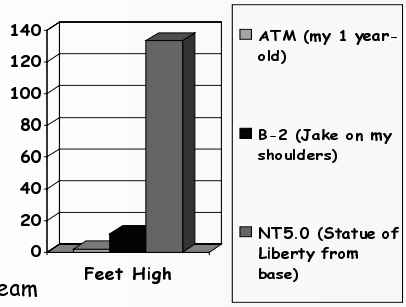
Bar code scanners	10-50KLOC
4-speed transmissions	20KLOC
ATC ground system	130KLOC
Teller machine	600KLOC
Call router	2.1MLOC
B-2 Stealth bomber	3.5MLOC
Seawolf submarine combat	3.6MLOC
Space shuttle	26MLOC+1MLOC/flight
NT5.0	40MLOC (w/scaffolding)

Relative sizes



Absolute sizes

- 50 lines per page
- Double sided
- 500 pages/ream (2 inches)



How I spend my time

- The Great Pyramid of Giza is 481′
- The Kingdome is 250′
- The Colossus of Rhodes is 110′
- The Eiffel Tower is 1033′
- The Graduate Reading Room in Suzzallo is 65′
- A 747 is 63′ to the top of the tail
- The Brooklyn Bridge is 135′ above the water
- Titanic's height from keel to bridge is 104′
- The EE/CSE building is about 90′

Delivered source lines per person

- Common estimates are that a person can deliver about 1000 source lines per year
  - Including documentation, scaffolding, etc.
- Obviously, most complex systems require many people to build
- Even an order of magnitude increase doesn't eliminate the need for coordination

Inherent & accidental complexity

- Brooks distinguishes these kinds of software complexity
  - We cannot hope to reduce the inherent complexity
  - We can hope to reduce the accidental complexity
- Some (much?) of the inherent complexity comes from the incredible breadth of software we build
- That said, it's not always easy to distinguish between these kinds of complexity

"The Software Crisis"

- We've been in the midst of a "software crisis" ever since the 1968 NATO meeting
  - **crisis** -- (1) an unstable situation of extreme danger or difficulty; (2) a crucial stage or turning point in the course of something [WordNet]
- We cannot produce or maintain high-quality software at reasonable price and on schedule
  - Gibb's *Scientific American* article
  - "Software systems are like cathedrals; first we build them and they we pray" —Redwine

Notkin's view—"mostly hogwash"

- Given the context, we do pretty well
  - We surely can, should and must improve
- Some so-called *software* "failures" are not
  - They are often management errors (Ariane, Denver airport, etc.)
  - Read comp.risks (far better than comp.software-eng)
- In some areas, we may indeed have a looming crisis
  - Safety-critical real-time embedded systems
  - Y2K?

Some "crisis" issues

- Relative cost of hardware/software
- Low productivity
- "Wrong" products
- Poor quality
  - Importance depends on the domain
- Constant maintenance
  - "If it doesn't change, it becomes useless"
- Technology transfer is slow

SE <> PL

Why is it hard?

- There is no single reason software engineering is hard—it's a "wicked problem"
- Lack of well-understood representations of software [Brooks] makes customer and engineer interactions hard
- Relatively young field
- Software intangibility is deceptive

Law XXIII, Norman Augustine [Wulf]

"Software is like entropy. It is difficult to grasp, weighs nothing, and obeys the second law of thermodynamics; i.e., it always increases."

Dominant discipline

- As the size of the software system grows, the key discipline changes [Stu Feldman, thru 10^7]
  - *Code Size* — *Discipline*
    - 10^3 — Mathematics
    - 10^4 — Science
    - 10^5 — Engineering
    - 10^6 — Social Science
    - 10^7 — Politics
    - 10^8 — ??

Is it engineering?