16.355

Software Engineering Concepts

Prof. Nancy Leveson

Fall 2005

http://sunnyday.mit.edu/16.355

Outline

- Is There a Problem?
- Why is Software Engineering Hard?
- Syllabus and Class Description

Is there a problem?

- Examples:
  - AAS (FAA Advanced Automation System)
  - FBI CIC
  - IRS Modernization Program
  - C−17
  - Ariane 5

- Head of AF Systems Command: ‘‘Software is the achilles heel of weapons development"

- 7 out of every 10 major weapons development programs are encountering software problems and the rate is increasing.

Some "Data" (Myths?)

- Development of large applications in excess of 5000 function points (~500,000 LOC) is one of the most risky business undertakings in the modern world (Capers Jones)
- Risks of cancellation or major delays rise rapidly as overall application size increases (Capers Jones):
  - 65% of large systems (over 1,000,000 LOC) are cancelled before completion
  - 50% for systems exceeding half million LOC
  - 25 % for those over 100,000 LOC
- Failure or cancellation rate of large software systems is over 20% (Capers Jones)

More "Data" (Myths?)

- After surveying 8,000 IT projects, Standish Group reported about 30% of all projects were cancelled.
- Average cancelled project in U.S. is about a year behind schedule and has consumed 200% of expected budget (Capers Jones).
- Work on cancelled projects comprises about 15% of total U.S. software efforts, amounting to as much much as $14 billion in 1993 dollars (Capers Jones).

And Yet More

- Of completed projects, 2/3 experience schedule delays and cost overruns (Capers Jones)

  [bad estimates?]

- 2/3 of completed projects experience low reliability and quality problems in first year of deployment (Jones).

- Software errors in fielded systems typically range from 0.5 to 3.0 occurrences per 1000 lines of code Bell Labs survey).

- Civilian software: at least 100 English words produced for every source code statement.

  Military: about 400 words (Capers Jones)

Have you ever been on a project where the software was never finished or used?

What were some of the problems?

Death March Projects

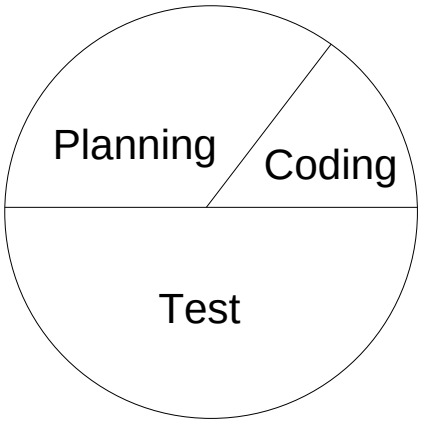
- Feature (scope) creep
- Thrashing
- Integration problems
- Overwriting source code
- Constant re−estimation
- Redesign and rewriting during test
- No documentation of design decisions
- Etc.

Types of Problem Projects (Yourdan)

- Mission Impossible
  Likely to succeed, happy workers
- Ugly
  Likely to succeed, unhappy workers
- Kamikaze
  Unlikely to succeed, happy workers
- Suicide
  Unlikely to succeed, unhappy workers

Understanding the Problem

Development Costs

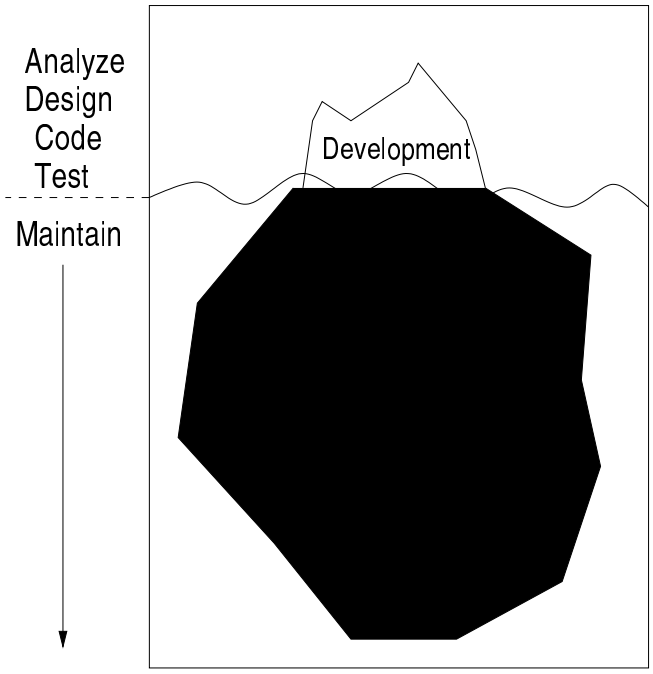


1/3 planning

1/6 coding

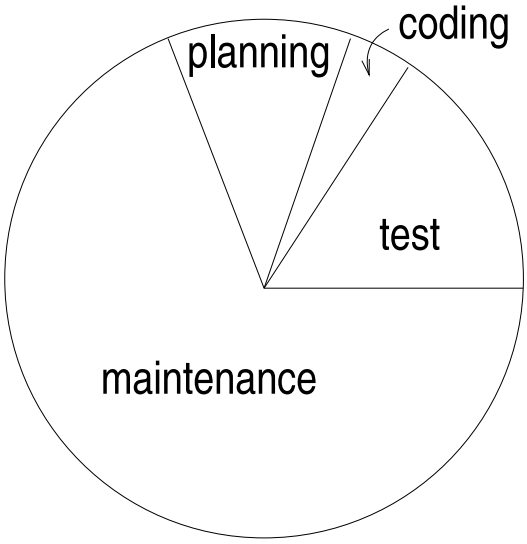
1/4 component test

1/4 system test



Development costs are only the tip of the iceberg.

Understanding the Problem (2)



Software Maintenance:

20% error correction
20% adaptation
60% enhancements

Most fielded software errors stem from requirements not code

Software Evolution (Maintenance)

- Belady and Lehman's Laws:
  - Software will continually change.
  - Software will become increasingly unstructured as it is changed.
- Leveson's Law:
  - Introducing computers will not reduce personnel numbers or costs.

Are Things Improving?

- Is software improving at a slower rate than hardware?

  "Software expands to fill the available memory" (Parkinson)

  "Software is getting slower more rapidly than hardware becomes faster" (Reiser)

- Expectations are changing

Is software engineering more difficult than hardware engineering?

Why or why not?

.

Why is software engineering hard?

- "Curse of flexibility"
- Organized complexity
- Intangibility
- Lack of historical usage information
- Large discrete state spaces

The Computer Revolution

- Design separated from physical representation; design became a completely abstract concept.

- Machines that were physically impossible or impractical to build become feasible.

- Design can be changed without retooling or manufacturing.

- Emphasis on steps to be achieved without worrying about how steps will be realized physically.

The Curse of Flexibility

- "Software is the resting place of afterthoughts."
- No physical constraints
  - To enforce discipline on design, construction and modification
  - To control complexity
- So flexible that start working with it before fully understanding what need to do
- The untrained can get partial success.

  "Scaling up is hard to do"
- *‘‘And they looked upon the software and saw that it was good. But they just had to add one other feature ...’’*

What is Complexity?

The underlying factor is intellectual manageability

1. A "simple" system has a small number of unknowns in its interactions within the system and with its environment.

2. A system becomes intellectually unmanageable when the level of interactions reaches the point where they cannot be thoroughly
   - planned
   - understood
   - anticipated
   - guarded against

Ways to Cope with Complexity

- Analytic Reduction (Descartes)
  - Divide system into distinct parts for analysis purposes.
  - Examine the parts separately.

- Three important assumptions:
  1. The division into parts will not distort the phenomenon being studied.
  2. Components are the same when examined singly as when playing their part in the whole.
  3. Principles governing the assembling of the components into the whole are themselves straightforward.

Ways to Cope with Complexity (con't.)

- Statistics
  - Treat as a structureless mass with interchangeable parts.
  - Use Law of Large Numbers to describe behavior in terms of averages.
- Assumes components sufficiently regular and random in their behavior that they can be studied statistically.

What about software?

- Too complex for complete analysis:
  - Separation into non−interacting subsystems distorts the results.
  - The most important properties are emergent.
- Too organized for statistics
  - Too much underlying structure that distorts the statistics.

"Organized Complexity" (Weinberg)

Other Factors

- Large discrete state spaces
  - Continuous vs. discrete math
  - Lacks repetitive structure found in computer circuitry
  - Cannot test exhaustively
- Intangibility
  - Invisible interfaces
  - Hard to experiment with and manage
  - Transient hardware faults vs. software errors
  - Hard to diagnose problems

And One More

- No historical usage information

  to allow measurement, evaluation, and improvement of standard designs over time.

  - Always specially constructed.
  - Usually doing new things.

Class Objectives

1. Students will be able to evaluate software engineering techniques and approaches.

"It is important that students bring a certain ragamuffin barefoot irreverance to their studies. They are here not to worship what is known, but to question it."

Jacob Bronowski, The Ascent of Man

"The developed theories ...have rarely been subjected to empirical testing, and so their value remains unknown. They provide zealots with opportunities to market a rash of seminars and courses and to flood the literature with papers advocating the new technologies. When the theories are subjected to testing, what little evidence has been obtained sometimes suggests that the claimed benefits, in fact, may not exist.

Vessey and Weber

Religious approach to SE (vs. scientific):

- Accept on faith (because sounds right)
- Gurus (Jackson, Yourdan, etc.)
- Sects (OO, Ada vs. Modula 2)

Arguments:

- Proof by vigorous handwaving.
- Unsupported hypotheses.
- False analogies.

Hawthorne Effect

Class Objectives

2. Students will be able to exercise professional judgement in selecting an approach for a particular project based on an understanding of:

   How the present state of software practice came about

   What was tried in the past

   What worked and what did not work

   Why

- Required Background
- Assignments
  - No programming
  - Reading summaries:
    - Main ideas or themes
    - Critical evaluation
    - Any additional thoughts
  - Some additional short assignments

Reading: Both classic papers and new ones

> *I would like to see computer science teaching set deliberately in a historical framework.... The absence of this element in their training causes people to approach every problem from first principles. They are apt to propose solutions that have been found wanting in the past. Instead of standing on the shoulders of their precursors, they try to go it alone.*
>
> *Maurice Wilkes*