CS 510/17

Software Engineering

Xiangyu Zhang

The Nature of Software...

- Software is easy to reproduce
  - Cost is in its *development*
    - in other engineering products, manufacturing is the costly stage
- Software development is mainly a human activity
  - Hard to automate, labor intensive
- Software is intangible
  - Hard to understand development effort
- The demand of scaling up is pressing

The Nature of Software ...

- Untrained people can hack something together
  - Quality problems are hard to notice.
- Software is easy to modify
  - People make changes without fully understanding it
- Software *deteriorates* by having its design changed:
    - erroneously, or
    - in ways that were not anticipated, thus making it complex

The Nature of Software

- Resulting facts.
  - Much software has poor design and is getting worse
  - Demand for software is high and rising
  - We are in a perpetual 'software crisis'
  - We have to learn to 'engineer' software

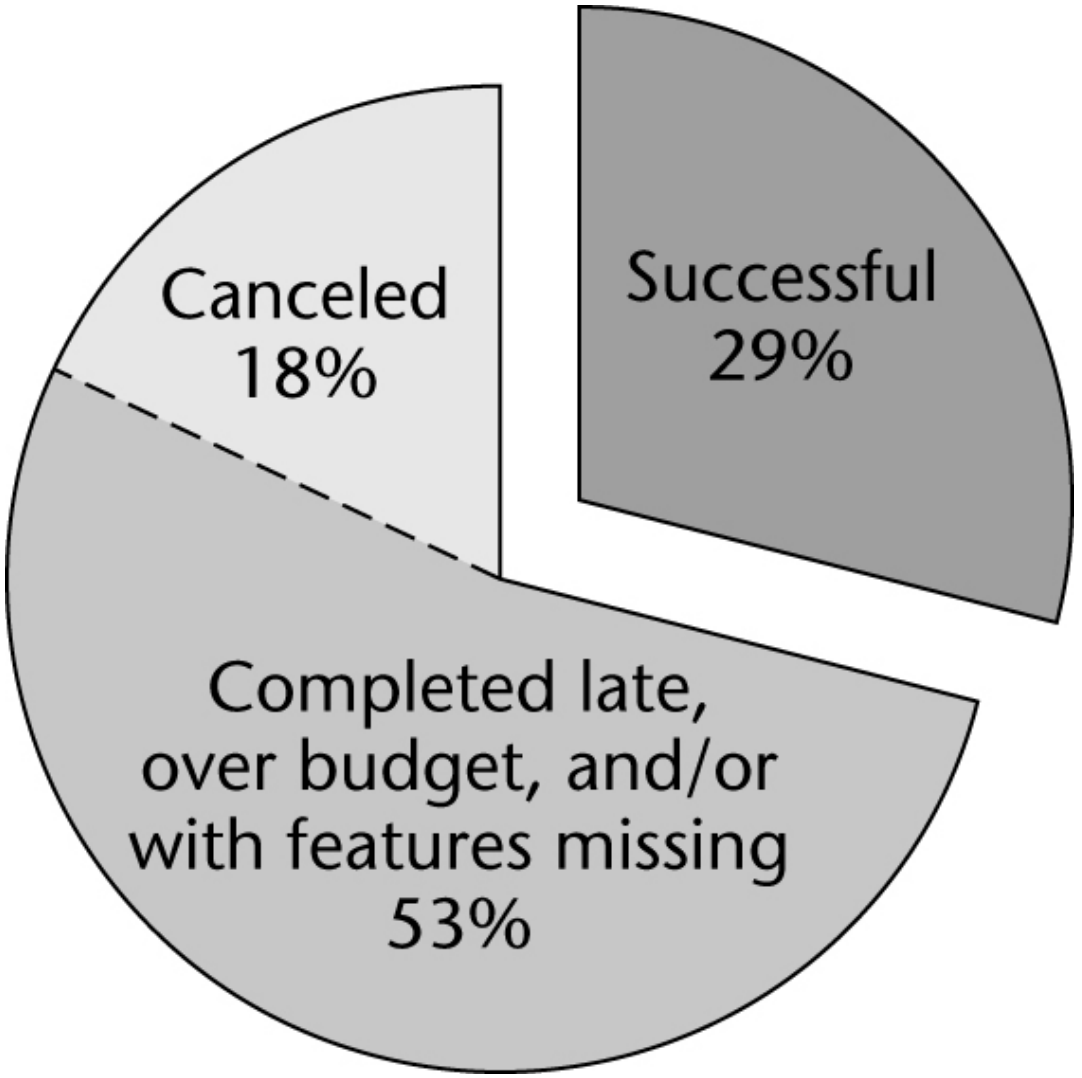
An Article

- Frequently Forgotten Fundamental Facts about Software Engineering
  by Robert L. Glass

  http://www.computer.org/portal/web/buildyourcareer/fa035
  - **Complexity:** For every 10-percent increase in problem complexity, there is a 100-percent increase in the software solution's complexity.
  - **People:** Good programmers are up to 30 times better than mediocre programmers, according to "individual differences" research. Given that their pay is never commensurate, they are the biggest bargains in the software field.
  - **Tools and techniques:** Most software tool and technique improvements account for about a 5- to 30-percent increase in productivity and quality. But at one time or another, most of these improvements have been claimed by someone to have "order of magnitude" (factor of 10) benefits. Hype is the plague on the house of software.
  - **Reliability:** Error detection and removal accounts for roughly 40 percent of development costs. Thus it is the most important phase of the development life cycle.

Standish Group Data

- Data on 9236 projects completed in 2004



–Figure 1.1

Cutter Consortium Data

- 2002 survey of information technology organizations
  - 78% have been involved in disputes ending in litigation
- For the organizations that entered into litigation:
  - In 67% of the disputes, the functionality of the information system as delivered did not meet up to the claims of the developers
  - In 56% of the disputes, the promised delivery date slipped several times
  - In 45% of the disputes, the defects were so severe that the information system was unusable

Implications

- The software crisis has not been solved.
- Dropping CS enrollment (now increasing)

What is SE - Historical Aspects

- 1968 NATO Conference, Garmisch, Germany
- Aim: To solve the *software crisis*
- Software is delivered
  - Late
  - Over budget
  - With residual faults
  - Missing features

What is SE?

- Software engineering
  - A discipline whose aims are
    - The production of fault-free software **(dependability)**,
    - Delivered on time and within budget **(productivity)**,
    - That satisfies the client's needs **(usability)**,
    - Furthermore, the software must be easy to modify when the client's needs change **(maintainability)**.
- Extremely broad
  - Computer science (PL, networking, OS, databases…), management, economics, etc.

What should be Taught in a Graduate Level SE Course?

- Hard to answer
  - Professor A: testing.
  - Professor X (my colleague in another university): textbooks.
  - Professor D (leader in the field): requirement analysis, modeling, nice plan but unsatisfactory implementation.
  - Recruiters from Amazon: agile programming and SOA
  - No conclusion.
- The two aspects
  - Religion: waterfall vs. XP, design patterns, architectures.
  - analytic.
    - Machine tractable, provable, such as model checking, program analysis, verification, testing

CS510 Content

- Algorithmic software engineering (hard)
  - Machine tractable and technical
    - Program analysis, model checking, software verification, testing.
  - The majority of the course.
    - The idea is to first teach the technical part so that you can design and work on your project.
  - Understanding and hands-on experience
- Bible part (not covered)
  - Project management
  - Design patterns
  - Requirements Analysis
  - Software metrics
  - UML

Pros and Cons

- Cons
  - May not be the course that your expect
  - Challenging
    - New concepts
    - Lot of coding, x86 machine code, java byte code, large analysis infrastructure
  - A lot of material not covered by any textbook
- Pros
  - Prepare yourself with the state-of-the-art automated techniques
  - Synergy with your research
  - Self-satisfaction

Grading Policy

- Projects 55% (3 tiny projects, 1 term project)
- Final 20%,
- Midterm 10%,
- Homework 15%,
- A short paper presentation (5%, replaceable with a tiny project)
- Qualifier: final

http://xyz-wiki.cs.purdue.edu/cs510-17s/doku.php?id=home

More Details about Course Conduction

- Three small projects: (25%, tentative)
  - Using Valgrind memcheck to debug a given faulty program without knowing the fault. (5%)
  - Using LLVM to generate call-graphs. (15%)
  - Using Klee to perform dynamic test generation. (5%)
- Term project (30%)
  - Using VALGRIND to build a provenance tracking tool.
    - (One person) a provenance tracking tool.
    - (Two persons) a provenance tracking tool that handles both data and control dependences.

More Details about Course Conduction

- (Goal) the three tiny projects and most home work assignments will occur in the first two months of the semester.
- The last month or more will focus on the large project
- Each one (??) will have one short paper presentation (15 minutes). One presentation scheduled for each week on Thursday. More will be scheduled towards the end.
  - The short one happens every Thursday. Papers are selected from ICSE14-16 and FSE14-16.
- I will occasionally post challenges, mostly about devising (not implementing) solutions for realistic challenges.
  - Chances to earn extra credits.

Books Used

- Michael Huth, Mark Ryan, Logic in Computer Science – Modelling and Reasoning about Systems.
- Aditya P. Mathur, Foundation of Software Testing.
- Alfred V. Aho, Monical S. Lam, Ravi Sethi, Jeffrey D. Ullman, Compilers Principles, Techniques, & Tools.

Suggested Readings

- Gamma, Helm, Johnson, Vlissides, *Design Pattern — Elements of Reusable Object-Oriented Software*, AW, 1995.
- Beck, *Extreme Programming explained — Embrace Change*, AW 1999.
- Fowler, *Refactoring: Improving the Design of Existing Code*, AW 1999.
- Edmund Clarke, Orna Grumberg, Model Checking, The MIT Press.Hanne Ris-Nielson,

Academic Integrity

All work that you submit in this course must be your own. Unauthorized group efforts are considered *academic dishonesty*.

You may discuss homework in a general way with others, but you may not consult any one else's written work. You are guilty of academic dishonesty if:

- You examine another's solution to a programming assignment (PA)
- You allow another student to examine your solution to a PA
- You fail to take reasonable care to prevent another student from examining your solution to a PA and that student does examine your solution.

Automatic tools will be used to compare your solution to that of every other current or past student. Don't con yourself into thinking you can hide your collaboration. The risk of getting caught is too high, and the standard penalty is way too high.

Questions?