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.
- Good engineer vs. bad engineer

http://www.computer.org/portal/web/buildyourcareer/fa035

Software is easy to modify

- People make changes without fully understanding it

Software does not 'wear out'

- It *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
Standish Group Data

Data on 9236 projects completed in 2004

- Canceled 18%
- Completed late, over budget, and/or with features missing 53%
- Successful 29%

-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.
  - Even worse - modern hardware vs. human capabilities
    - NSF cfp 2008
      - Compared to OS, ARCH, Networking, ...

- Dropping CS enrollment
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 (leader of the field in our department): 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
- The conclusion is that there is no conclusion.

Let's approach the question differently.

- expectations.
Expectation

Basic Knowledge about the field.
- In an undergraduate course
  - Software process, requirement analysis, UML, unit testing
- In a graduate level course
  - Design patterns, refactoring, software architecture, aspect oriented programming, and extreme programming.

For students interested in SE
- Be prepared to be a leader and a problem solver (not a coder).
- Acquaint yourself with techniques having large potential impact.
  - Story about Vista
  - Story about Amazon
  - White paper “Software for Dependable Systems: Sufficient Evidence?”
    - F22
    - Medical device software vulnerability
Expectations (continued)

- For students in other areas.
  - The goal is to achieve synergy.
  - Research subjects are more or less software related
    - Bugs in distributed systems.
    - Scientific programs running too slow.
    - Software vulnerabilities.
  - Research ideas in your field can directly come from 510.
    - Data lineage
    - Software vulnerabilities
Algorithmic software engineering (hard)
- Machine tractable and technical
  - Program analysis, model checking, software verification, testing.
- The first 2/3 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 (relaxed)
- Machine intractable
  - Process, architecture, design patterns.
- The last 1/3.
- Relaxed and work on the project
CS510 Content

This course will not cover:

- Project management
- Requirements Analysis
- Software metrics
- UML
Grading Policy

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

http://www-cgi.cs.purdue.edu/cgi-bin/xyzhang/doku.php?id=home
More Details about Course Conduction

Tiny Projects: select 3 out of the following 4 small projects on using tools. (15%)
- Using Valgrind memcheck to debug a given faulty program without knowing the fault. (5%)
- Using FireEye to perform combinatorial testing on a given program. (5%)
- Using Klee to perform dynamic test generation. (5%)
- Using Chess to test a given concurrent program. (5%)

Term project (40%)
- (One person) a dynamic data slicing tool.
- (Two persons) a full slicing tool that handles both data and control dependences.
- (Three persons) full slicing + GDB support
- (Four persons) full slicing + GDB support + Checkpointing support.
- Other project options are possible, but require instructor's approval.
More Details about Course Conduction

The three tiny projects and most homework assignments will occur in the first two months of the semester.

The last month or more will be focused on the large project.

Each one (??) will have one short paper presentation (15 minutes). One presentation scheduled for each week. More will be scheduled towards the end.

- The short one happens every Thursday. Papers are selected from ICSE09 and FSE09.

I will occasionally post challenges, mostly about devising (not implementing) solutions for realistic challenges.

- Chances to earn extra credits.

Attendance will not be taken (??)

Will try to post course readings a week ahead, slides one night ahead.
Books Used

- Aditya P. Mathur, Foundation of Software Testing.
- Alfred V. Aho, Monical S. Lam, Ravi Sethi, Jeffrey D. Ullman, Compilers Principles, Techniques, & Tools.
Suggested Readings

- Hanne Riis-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?