How Do We Make Software?

Monica Lam
Stanford University
Software

- Encapsulation of knowledge
  - that can be applied automatically, over and over again
Why is Software Engineering Unique?

- Abstract
- Complex
- Malleable
- Correctness vs. function / time tradeoff
Complexity

- Kolmogorov complexity
  - Complexity of an object can be measured by the size of the Turing machine that generates the object

- Millions and millions of lines of code

- Can model engineered artifacts
  ... and simulate them in their environments
Software Development Process

- Requirements
- Software architecture design + Algorithms
- Coding
- Testing
- Software maintenance
Requirements Evolve

- Full specification not possible
- Working prototype needed to understand the requirements
- Requirements change across time
- Compatibility with legacy system
Architecture Design

- Need to understand domain & automate manual process
- “Plan to throw one away”, Fred Brooks
- Architectural integrity: small, excellent design team
Architectural Concepts

Abstraction
Modularity
-- Recursively applied --
Components for Common System Tasks

- Graphical tool kits
- Databases
- Secure communication protocols
- Network protocols
- Operating systems
Programming Tools

- High-level programming languages and compilers e.g. Java
- Higher level languages
  - Matlab, spreadsheets
- Domain-specific abstractions
  - Object-oriented programming
Coding is Hard

- Productivity varies by orders of magnitude
- Mythical man-months
- Brooks’s Law: “Adding manpower to a late software project makes it later”
But Deceptively Simple

- Majority of a code handles exceptions
  - 90% correct → 10% of the code
    → < 1% debugging time

- Hard to estimate software development time

- Exponential # paths through a program
  - Paths not tested are wrong
Correctness vs. Speed/Features

- Less correct
  - faster time to market
  - more features
  - faster code

- more money
Hardware Design

- Errors are very expensive!
- Introduce features carefully
- Invest in CAD/verification tools
- Test extensively
- Relies on software to take up the slack or mask its error
Software Quality

- Quality assurance: testing
  - a large number of “testers”
  - not tied to development

- Very fragile & brittle

- Fix the high-priority errors
Software is Full of Errors

- Error rate: 1-4.5 errors per 1000 lines
- Windows 2000
  - 35M LOC,
  - 63000 known bugs at the time of release
  - 2 per 1000 lines
- Large consumer software
  - 100% correct is infeasible
The Bar is Too Low

- Many simple and deadly errors
- Security: buffer overruns
- Exploit known since 1989
- Program accesses a buffer out of bounds
- Can be eliminated if written in Java
- Can be caught dynamically as program runs
Problems

- False economy
  - Slammer worm costs $1 billion
  - MS Blaster costs Stanford $800,000

- Monopoly

- Patch available before the attack
  - 6 months → 1 week → ... → 0 day
On the Research Front

- Software integrity checkers
  - thousands of errors in windows / linux
  - memory leaks
  - detects anomalies as program runs
- Higher level debugging tools
- Error recovery

Economic incentives can greatly speed up progress
Conclusions

- Software engineering is complex and hard, but is deceptively simple

- Current economic model
  - Passes cost of unreliability to unwitting consumers
  - Lack of software tool investment