

What is Software Engineering and why does it matter?

Original slides by Chris Wilcox,
Colorado State University

Computer Science: Disciplines

- Computer Graphics
- Computer Networking and Security
- Parallel Computing
- Database Systems
- Artificial Intelligence
- **Software Engineering**

All kinds of interesting stuff is going on at
Colorado State University!

CS270 - Spring Semester 2014

2

Software Engineering

IEEE Computer Society Definition:

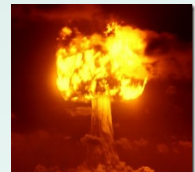
“**Software engineering** is the application of a **systematic, disciplined, quantifiable** approach to the development, operation, and maintenance of software, and the study of these approaches; that is, the application of engineering to software.”

CS270 - Spring Semester 2014

3

Software Disasters

- Mariner Bugs Out (1962)
- Almost World War III (1983)
- Medical Machine Kills (1985)
- Wall Street Crash (1987)
- AT&T Lines Dead (1990)



CS270 - Spring Semester 2014

4

Software Engineering

Doing the right thing






- Software that users want and need
- Software that benefits society

Doing the thing right

- Following a good software process
- Developing your programming skills

Software Engineering

No Silver Bullet, but lots of progress

- Assembly Programming 
- High Level Languages (Fortran, C) 
- Object Oriented Languages (C++, Java)
- Card Reader 
- Computer Terminal 
- Bitmapped Display
- Command line 
- Graphical tools (Eclipse, Visual Studio)

What is software?

Non-physical manifestation of information

- Media, genetic code, operating system, application

Intellectual Property

Executable software

Non-executable software

- Problem statement, requirements document, software design, test plan, source code

Software media is not software

Nature of Software

- Demand for software is high and rising, we hear about the perpetual 'software crisis'.
- Untrained people can hack something together, thus software is often of poor quality.
- Software creation is labor intensive, must use engineering (not manufacturing) skills.
- Software does not wear out, but its requirements and the environment change.
- Software development cannot be automated, and it's easy to modify but hard to fix.

Quality Issues

Information systems:

- Data integrity, security, availability, transaction performance, usability

Distributed systems:

- System reliability, adaptability to network partitioning, fault tolerance

Embedded systems:

- Response time, reliability, safety, usability

Commercial Software (COTS):

- Reusability versus generality, cost

Stakeholders

1. Users

- Those who use the software
- Needs: efficiency, reliability, usability, functionality

2. Customers

- Those who pay for the software
- Needs: low cost, reliability, increased productivity, flexibility

3. Software developers

- Those who write the software
- Needs: high-quality documentation, tools, design

4. Development Managers

- Those who manage the project
- Needs: minimal development time, cost, few defects

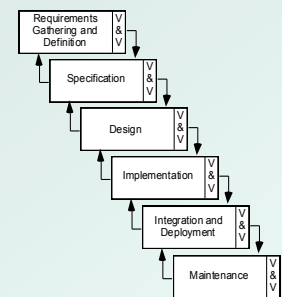
The “Problem”

- Programs are written by programmers, not users, how to understand requirements?
- Large gaps exist between the problem and solution, user and computer.
- Human domain is informal, computer domain is formal, translation is difficult.
- Key requirements can easily be expressed informally, formal specification is hard.
- Programs are formal (and must be in order to compile into machine instructions).

Waterfall Model

The classic way of looking at software development:

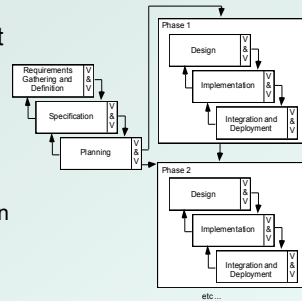
- Series of carefully planned stages
- Verify and validate output at each stage
- Allows stepping back, in a limited way
- Hard to handle changing requirements



Phased Release

Introduces the idea of *incremental* development of software.

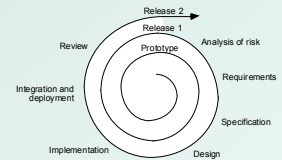
- Project is broken into separate *phases*.
- Each phase released to customers when ready.
- System available earlier than waterfall approach.
- Requirements still must be final before development.



Spiral Model

Explicitly embraces prototyping and an *iterative* approach to software development.

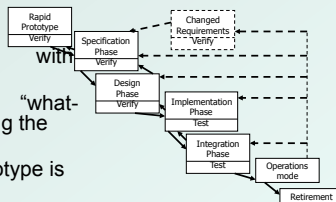
- Start by developing a small prototype
- Followed by a series of waterfall processes
- Review software at end of each phase
- Repeat until software meets requirements (and beyond)!



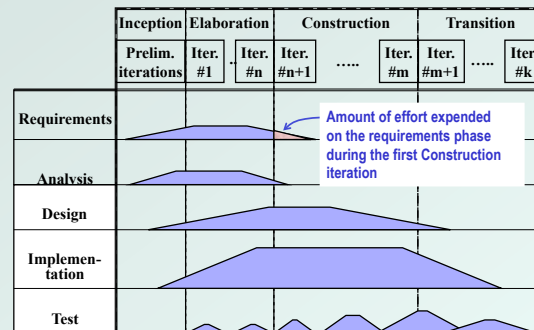
Incremental Development

A rapid prototype is a working model that is functionally equivalent to part of the product

- Specification is correct when client is satisfied prototype
- Lessons about "what-not-to-do" are learned during the prototyping
- Speed is of essence, prototype is eventually thrown away



Unified Model



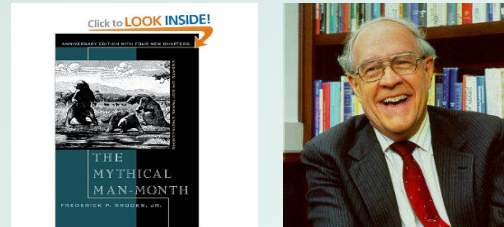
The agile manifesto

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements to harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, shorter is better.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them.
- Working software is the primary measure of progress, simplicity is essential.

<http://agilemanifesto.org/principles.html>

The Mythical Man-Month

Brooks's Law: Adding manpower to a late software project makes it later.



Joys of the Craft (Brooks)

- First is the sheer joy of making things..., especially things of his own design.
- Second is the pleasure of making things that are useful to other people.
- Third is the fascination of fashioning complex puzzle-like objects of interlocking moving parts and watching them work in subtle cycles...
- Fourth is the joy of always learning.
- Finally, there is the delight of working in such a tractable medium (as we shall see later, this has its own problems).

Woes of the Craft (Brooks)

- First one must perform perfectly. If one character, one pause of the incantation is not strictly in the proper form, the magic doesn't work.
- Next, other people set one's objectives, provide one's resources, and furnish one's information. One rarely controls the circumstances...
- The next woe is that designing grand concepts is fun; finding nitty little bugs is just work.
- The last woe is that the product over which one had labored so long appears to be obsolete upon (or before) completion.

No Silver Bullet — Essence and Accidents of Software Engineering

- Brooks says “there is no single development, in either technology or management technique, which by itself promises even one order of magnitude improvement within a decade in productivity, in reliability, in simplicity.”
- Brooks makes a distinction between **accidental complexity** and **essential complexity**, and asserts that most of what software engineers now do is devoted to the essential.

Conclusions

Software is indispensable to our modern lifestyle.
Engineering discipline is needed for good software:

- Be good at what you do.
- And get ready for change.

Why does it matter?

- So that you will prosper.
- For the benefit of society at large.

