

Introduction

- **Software in our lives**
- **What is *software engineering*?**
- **Software engineering - precis of a short history by Barry Boehm, ICSE'06 Keynote speaker**
- **Software disasters**
 - Ariane 5 rocket
- **Software quality**
- **Software myths**

Software is ubiquitous

- **Systems software**
 - OS, compiler, loader
- **Business software**
 - Payroll, accounting
- **Scientific and engineering software**
 - Computer-aided design, simulation, weather prediction, ...
- **PC software**
 - Spreadsheets, word processing, games, ...
- **Embedded software**
 - Cars, microwave ovens, cable boxes, light switches, "smart dust", ...

Software is ubiquitous

- Internet software
 - B2C: business-to-customer (e.g. amazon.com)
 - B2B: business-to-business
 - Main target for web services technology
 - B2E: business-to-employee
 - Intra-company access
 - For mobile and remote employees

Software

Software is:


- Executable programs
- Data associated with these programs
- Documents: user requirements, design documents, user/programmer guides, etc.
- Software plays a key role
 - Produces, manages, and presents information
 - Information society
 - Next step after industrial society

Software Crisis?

- People that design and build software have to deal with many problems
- **Software crisis** for the last 30 years?
 - In reality, things are not that bad
 - Many more successes than failures
 - But problems are persistent
- The SE field is still immature
 - e.g., compare with civil engineering, etc.

Definition of Software Engineering

IEEE Standard 610.12

 The application of a **systematic, disciplined, quantifiable** approach to the development, operation, and maintenance of software; that is, the application of engineering to software, and

 The study of approaches as in (1).

That's a mouthful, isn't it?

Definition of Software Engineering

Based on Webster's definition of "engineering"

- The application of science and mathematics by which the properties of software are made useful to people
- Includes computer science and the sciences of making things useful to people
 - Behavioral sciences, economics, management sciences

Definition of Software Engineering

Pressman's book:

A discipline that encompasses

- process of software development
- methods for software analysis, design, construction, testing, and maintenance
- tools that support the process and the methods

Process, Methods, Tools

- **Various tasks required to build software**
 - e.g. design, testing, etc. (more later)
- **SE process: the organization and management of these tasks**
- **SE methods: ways to perform the tasks**
 - e.g. methods for software testing, for designing classes
- **SE tools: assist in performing the tasks**
 - e.g. design tools, IDEs like Eclipse
 - UML tools: Rational Rose,
 - Testing tools like JUnit, JCover

Importance of Historical Perspective

- **Santayana half-truth:**
 - "Those who cannot remember the past are condemned to repeat it"
- **Don't remember failures?**
 - Likely to repeat them
- **Don't remember successes?**
 - Not likely to repeat them

History of SW Development

- **1950's: engineer software like hardware**
 - **Applications: airplanes, bridges, circuits**
 - **Economics: computer time more valuable than people time**
 - **Boehm supervisor, 1955: "We're paying \$600/hour for that computer, and \$2/hour for you, and I want you to act accordingly."**

CS431F06, BG Ryder/A Routev

Cf Barry Boehm, ICSE06 Keynote

11

History of SW Development

- **1960's: software is NOT LIKE hardware**
 - **Properties:**
 - **Invisible, complex, had to be executed by computers, hard to change, doesn't wear out (or does it?), unconstrained by physical laws of nature**
 - **Demand for programmers exceeded supply**
 - **Cowboy programmers as heros; hacker culture;**
 - **Code and fix process**
 - **Better tools: compilers, operating systems, utilities**
 - **Departments of Computer Science formed**
 - **Successes: Apollo, BofA check processing, ESS**
 - **Problems: failure of most large systems, unmaintainable code; undiagnosible systems**

CS431F06, BG Ryder/A Routev

Cf Barry Boehm, ICSE06 Keynote

12

History of SW Development

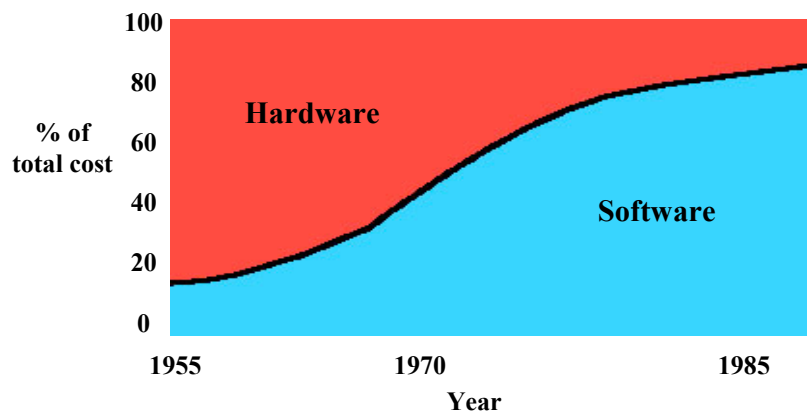
- 1970's Formal approaches developed
 - Structured programming, elimination of goto's
 - Waterfall model of development
 - Developed notions of requirements and design phases of SW creation
 - Problems with formal methods
 - Successful for small, critical programs
 - Proofs show presence of defects, not correctness
 - Lack of expert programmer community

CS431F06, BG Ryder/A Routev

Cf Barry Boehm, ICSE06 Keynote

13

Large-Organization HW/SW Cost Trends (1973)



CS431F06, BG Ryder/A Routev

Cf Barry Boehm, ICSE06 Keynote

14

History of SW Development

- 1980's Synthesis, productivity, reuse, objects
 - Major SW productivity enhancers
 - Working faster: tools and environments
 - Working smarter: processes and methods
 - Work avoidance: reuse, simplicity; objects
 - Technology silver bullets: AI, Do what I mean, programming by example
 - Develop processes by which to produce software, standards and conformance
 - Reuse: OO Libraries (Smalltalk, Eiffel, C++)

CS431F06, BG Ryder/A Routev

Cf Barry Boehm, ICSE06 Keynote

15

"No Silver Bullet", Fred Brooks

IEEE Computer, 1987

- Automated solutions are good for "accidental" software problems
 - Simple inconsistencies, noncompliance, inferences
- They do not do well on "essential" software problems
 - **Changeability**: adapting themselves to unanticipated changes
 - **Conformity**: working out everything the computer needs to "know"
 - Devoid of intuition, common-sense reasoning
 - **Complexity**: integrating multiple already-complex programs
 - **Invisibility**: communicating their likely behavior to humans
- Closest thing to silver bullet: **great designers**

CS431F06, BG Ryder/A Routev

Cf Barry Boehm, ICSE06 Keynote

16

History of SW Development

- 1990's predictability through modeling (heavyweight, but scalable)
- Rapidity of change - agile development (lightweight and not scalable)
- Other trends, '90s-early 00's: Y2K, reverse engineering, COTS, Open Source SW, Legacy codes
 - Software as the primary competitive discriminator
 - 80% of aircraft functionality

CS431F06, BG Ryder/A Routev

Cf Barry Boehm, ICSE06 Keynote

17

Existing SW Problems

- Software is too expensive
- Software takes too long to build
- Software quality is low
- Software is too complex to support and maintain
- Software does not age gracefully
- Not enough highly-qualified people to design and build software

CS431F06, BG Ryder/A Routev

18

Software Cost Overruns - Consequences

- **Many projects are cancelled**
 - People may get fired or may quit
- **Product features are not implemented**
 - Bad quality: not enough money to get it right, but more expensive in the long run
- **Loss of revenue and market share**
 - Both for the vendor and for the client
 - E.g., baggage system at Denver airport
 - Cost: \$1 million per day
- **Projects may become obsolete**
 - Technology changes rapidly
 - Competing products already on the market

CS431F06, BG Ryder/A Routev

19

An Overall Survey of Experiences

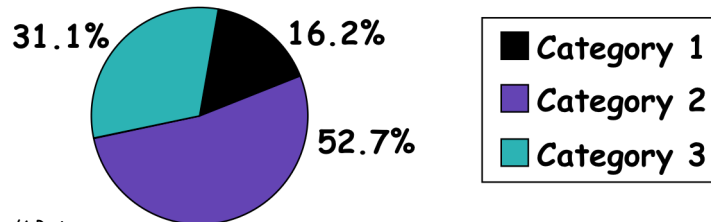
- **Studies of IT projects by the Standish Group (1995 and 1998)**
 - 350+ IT managers, 8000+ applications
- **What percentage of projects were**
 - cancelled before being completed?
 - over budget and/or late?
 - completed on time and on budget?
- **What was the cost/time overrun?**

CS431F06, BG Ryder/A Routev

20

Success Rate

- Category 1: on time and on budget, with all initially specified features
- Category 2: over budget or over time, with fewer features than specified
- Category 3: cancelled



CS431F06, BG Ryder/A Routev

21

Overruns and Deficiencies

- Cost and time overruns
 - Averages for category 2 and category 3
- Cost overruns: **189%** of original estimate
- Time overruns: **222%** of original estimate
- Feature deficiencies: only 61% of the originally specified features were implemented
 - Average for category 2

CS431F06, BG Ryder/A Routev

22

Some Reasons for Failure

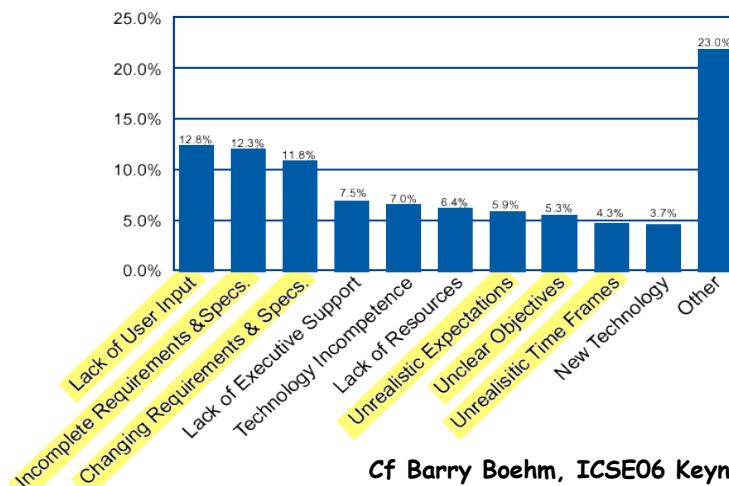
- Lack of user involvement
- Incomplete requirements and specs
- Changing requirements and specs
- Lack of executive support (politics)
- Lack of planning and management
- Inadequate resources and time
 - Death-march projects
- Technological incompetence

CS431F06, BG Ryder/A Routev

23

Why Software Projects Fail

- Average overrun: 89.9% on cost, 121% on schedule, with 61% of content



Cf Barry Boehm, ICSE06 Keynote

CS431F06, BG Ryder/A Routev 352 companies - 8,000 software projects. Source: The Standish Group, 1995

24

Standish Survey Summary

- **It is common for projects to fail**
 - A third of the projects from this survey
 - Estimated cost for 1995: \$81 billion
- **It is common for projects to go over budget/time, and to have fewer features**
 - Half of the projects in this survey
 - Time/cost is double of initial estimates
 - Estimated cost for 1995: \$56 billion
- **Similar results from the 1998 survey**

CS431F06, BG Ryder/A Routev

25

Software Quality

- **Software defects result in failures**
 - Example 1: Windows crashes while you play a game at home
 - Example 2: The software that controls a nuclear reactor crashes
- **Direct loss of life and money**
 - Millions of dollars
- **Indirect loss: missed opportunities**
 - e.g. online purchases are down for a day
- **Loss of credibility, bad publicity**

CS431F06, BG Ryder/A Routev

26

Ariane 5



Example: Ariane 5

- Ariane 5 rocket
- Built by the European Space Agency
- First launch: June 1996
- Crashed 40 seconds after launch
- Cost: \$500 million for rocket and contents; \$7 billion for development
- No people on board
- Problem: software failure

<http://www.ima.umn.edu/~arnold/disasters/ariane.html>
<http://www.around.com/ariane.html>

What Happened?

- **Overflow when velocity was converted from 64-bit integer to 16-bit integer**
- **The exception was not caught**
 - **Inertial Reference System failed**
 - Backup system failed for the same reason
 - **Rocket went off course**
 - **Self-destruct module (correctly) activated**
- **The code was OK for Ariane 4**
 - **Same software, different environment**
 - Code was for earlier phase in flight but was left active

CS431F06, BG Ryder/A Routev

29

Software Myths

- **"If we get behind schedule, we can just add more people and catch up"**
- **Fact: Adding people to a late project makes it even later**
 - **Brooks: Adding more people may make the project even later**
 - The people working now have also to teach the new people

CS431F06, BG Ryder/A Routev

30

Software Myths

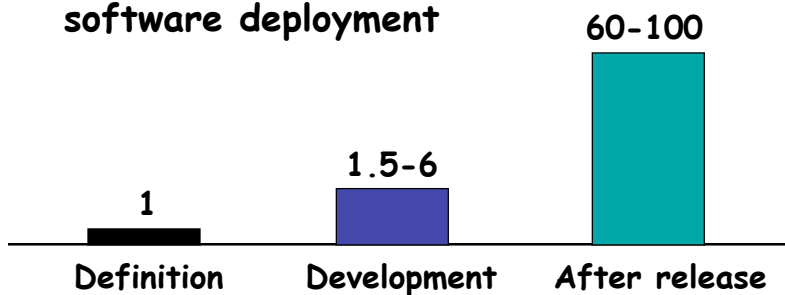
- 'Mythical man-month': "men and months are interchangeable only when a task can be partitioned among many workers with no communication among them" [Brooks, '75]
- Cost of pairwise communication that may be needed between n workers is $n(n-1)/2$ for coordinated tasks.
- "A general statement of objectives is enough to start programming"
- Fact: Incomplete requirements are a major cause for project failures

CS431F06, BG Ryder/A Routev

31

Software Myths

- "Changes in requirements are easy to deal with because software is flexible"
- Fact: Changes are hard and expensive
 - Especially during coding and after software deployment



CS431F06, BG Ryder/A Routev

32

Software Myths

- “Once we get the program running, we are done”
- **Fact: Most effort (60-80%) comes after the software is delivered for the first time**
 - Bug fixes, feature enhancements, etc.
- “Until the program is running, I cannot assess quality”
- **Fact: software reviews can be applied as soon as code is written and are very effective; pair programming techniques as well.**

Software Myths

- “The only product is the running program”
- **Fact: Need the entire configuration**
 - Documentation of system requirements, design, programming, and usage
- “SE will slow us down by requiring unnecessary documentation”
- **Fact: SE is about quality; Brooks recommends time division of: 1/3 planning; 1/6 coding; 1/4 component test and early system test; 1/4 system test**

Software Engineering

- **Software is complex, expensive, late, low-quality, hard to maintain**
- **Goal: approach these problems using software engineering**
 - **Engineering disciplines: civil engineering, etc.**
 - **Body of knowledge, established practices, professional education, certification, etc.**
- **Key problem: the field is very young**
 - **The term "SE" was introduced in 1968**