Software Engineering: Introduction

CSE 403 - Spring 1997

Outline

- Why software engineering?
- Challenges of software development and maintenance.
- What is software engineering?
  - vs. programming
- What are we going to do this quarter.

Engineering?

- **en·gi·neer·ing**
  - *n*. 1. The application of scientific **principles** to practical ends as the **design**, **construction**, and operation of **efficient** and **economical** structures, equipment, and **systems**. 2. The profession of or the work performed by an engineer.

Software & Society

- Economics
  - At least several hundred billion dollars/year are spent worldwide on software---and it's growing fast.
- Essential part of almost all enterprises
  - Accounting, budgeting, market analysis, forecasting & projections, planning, manufacturing control, e-mail, WWW, etc. etc.
- Key part of many engineering products
  - Medical systems (imaging, treatments, etc.), airplanes (B747-400 has 125 processors), avionics (GPS, FMS, etc.), automobiles (fuel injection, ABS.), etc. etc.
- Essential for designing many engineering products.
  - Stress simulations, VLSI designs, Virtual building (B777 walkthrough.)

Sample Code Sizes

[Jon Jacky]

- Hand-held bar code scanner: 10-50KLOC [Symbol Technologies Inc.]
- Electronic four-speed transmissions: 19KLOC [Chrysler]
- Cellular telephones: 30KLOC [Motorola]
- Automated teller network: 600KLOC [IBM]
- Telephone call-router: 2.1MLOC [DSC Communications Corp.]
- B-2 Stealth bomber: 3.5MLOC [Leonard Lee]
- Seawolf submarine combat computers: 3.6MLOC [Leonard Lee]

Challenge

- Delivery of software system satisfying customer requirements on time and on budget.
- Why is software hard to engineer?
  - Inherent complexity.
    - Software systems are important and complex artifacts, just like bridges, buildings and airplanes.
    - For its size, is more complex than other human artifacts---they have more varied parts than physical artifacts (otherwise we could collapse them into a single abstraction) [Brooks]
  - "Software crisis" (a 25+-year-old phrase)
    - We cannot, as a rule, produce or maintain high-quality software at reasonable price and on schedule.
    - "Software systems are like cathedrals: first we build them and then we pray."[Redwine]

“Wrong” Products

- In many cases, the developer constructs a “wrong” system.
  - That is, the system is not what the customer wants, e.g. “That’s not what we wanted!”
- This arises from a mixture of problems.
  - Communication problem between the client and the developer, i.e., not understanding each other.
  - Changes in external environments, e.g., emergence of a new user-interaction mode.
  - Perturbation.
    - On a delivery of software, the customer has a concrete point from which to verbalize their requirements.

Poor Quality

- Software systems are often unreliable.
  - "Sorry, sir. We can't take your order because our computer system is down."
  - A bank system that cleared the same check twice. [Personal experience]
  - Infamous back-door on fingerd.
- Software systems are often unsafe.
  - The Therac-25 radiation dosing machine killed people due to a combination of software errors and inappropriate system design. [Leveson]
- Software systems are often inflexible.
- Other qualities?

Constant Maintenance

- Maintenance is generally considered to include correction, retargeting, and enhancement.
  - Only "useful" software goes through maintenance.
- Maintenance is different for software than for tangible products.
  - In contrast to cars, software maintenance is not done because parts wear out.
  - Different expected life-span.
    - Year 200 problem.
- What causes the need for software maintenance, and how does that compare to other engineered artifacts?

Poor Estimates

- Delivery time and development costs are generally underestimated.
  - "Take your best 'worst' estimate and multiply it by two."
  - "How does a project get to be a year late? ... One day at a time."
- For example, on the Metro transit case on which David Notkin served as an expert witness.
  - Incurred costs were about $14M over the initial budget (which was several million dollars)
  - The developer took about three years over the initial agreement of about one year.
  - The system was not finished or delivered.
- Many projects are canceled before completion.

Software Engineering vs. Programming

- Handling under- and ill-defined problems.
- External constraints.
  - Customer's needs, time/budget restrictions, changes in technology, etc.
- Groups required to build and maintain the software.
  - Requires additional communication, which add costs and complexity.
  - Human relations.
- Projects run for a long time.
  - Implies personnel turnover, etc.
  - Maintainers are usually not original designers/implementors. There are systems for which the system is older than the people maintaining it.

Software Engineering vs. Programming

- Change is critical and unavoidable in software engineering.
  - "Software changes or becomes useless." [Belady & Lehman]
    - The social and technological context in which the software system is used changes.
  - Since it's intangible, it seems easy to change. But in practice, it's not!
- Defining computations vs. real world modeling.
  - Programming tends to define computations, while software engineering usually models the real world.
  - In real world models, software is usually only one part of a system. (a nuclear power plant, a telephone system, a business enterprise, etc.)

Software Engineering vs. Programming

- Scale
  - Many programming, long-term use of systems, long-term development efforts, multiple versions, high costs, ...
  - An increase in scale in software systems is usually not linear, because the number of different kinds of components increase [Brooks]
    - Building a larger system is not simply a matter of building bigger (or more) hash tables, for example.
- How do other engineering disciplines deal with the problem of scaling up?