Lecture 2 - Introduction - Outline

• What is software engineering

• Difficulties of software engineering

• Why Software Engineering is different from programming.

• \textit{Processes, Models and Tools}.

• Course Outline.
Impact

• On a Company’s business process: billing, accounting, inventory, personnel, manufacturing, forecast, email, documentation, WWW... (typically 4-5% of revenue not profit)

• On Life (!): Medical (imaging, instruments, etc) airplanes (B747 has 100 processors), automotive (fuel, ignition, braking)

• On Industrial Processes: Mechanical CAD, VLSI CAD

• On the Bottom Line: $10^{12}$ per year (peaks from y2k & euro conversion).

• others....Privacy, Civil Liberties, Democratic processes,
Examples of Software Systems

• Application Examples:

<table>
<thead>
<tr>
<th>System</th>
<th>Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barcode Scanner</td>
<td>10-50Kloc</td>
</tr>
<tr>
<td>Cellphone</td>
<td>2 Mloc</td>
</tr>
<tr>
<td>B2 Bomber</td>
<td>3.5 Mloc</td>
</tr>
<tr>
<td>M$ Office</td>
<td>$10^7$ Loc</td>
</tr>
<tr>
<td>Car Shifter</td>
<td>20KLoc</td>
</tr>
<tr>
<td>ATM Network</td>
<td>600 Kloc</td>
</tr>
<tr>
<td>Shuttle</td>
<td>$10^7$-$10^8$ Loc</td>
</tr>
<tr>
<td>Telephone Switch</td>
<td>$10^7$-$10^8$ Loc</td>
</tr>
</tbody>
</table>

Boeing 777:

2.5 Mlocs of custom code, 1.5 Million lines of COTS.

79 different supplies, each supplying part of the system.

Specifications of the software, done through SCD (software control diagrams). 100 pages for "simplest systems" to 10,000 pages for most complex.

Interface descriptions for each system placed in database. 40,000 data items, and 3,000 analog systems. Tool built to automatically find discrepancies. 50,000 found and removed!

System was written in ADA.
Attributes of Software

- **Cost**: Varies, $20-$40 a line is a reasonable guess (approximately).
- **Quality**: Depending on application domain: word processors vs. telephone switches.
- **Lifetime**: days (e.g., applets) to decades (e.g., Boeing 747, 5ESS)
- **Complexity**: Not the same as size. O/S Kernel, vs. a text editor.
- **Usability**: Software controlling a VCR vs. Software controlling a Cruise Missile
- **Portability**: Mobile code Vs. Embedded Application (e.g. software controlling a thermostat).
- **Companies/Cultures**: flyb ynigh.com, EA, Oracle, Boeing, Lucent.
Failure Models

- **Requirements Failures:**
  - End-user naivete
  - Engineers aren’t human (like the users are).
  - Engineers don’t have a life (like the users have a life).
  - Changes in business, law, time, etc.

- **Project management failures:**
  - Difficulties of estimating.
  - . . . communication between developers.
  - . . . co-ordination
  - . . . personnel turnover
  - . . . finding good people

- **Inherent Difficulties (Fred Brooks)**
  - *Complexity:* for it’s size/cost, more than other endeavours.
  - *Invisibility:* few helpful visualizations (e.g., Blue prints)
  - *Changeability/Conformity:* everyone wants to change software first when reality changes.
Why Software Engineering is different from programming

- Those darn customers—understanding ill-defined problems.
- External constraints
  - legalities, time/budget restrictions, physics, etc etc.
- Team work—interaction, co-ordination, training, etc.
- Long running—maintainers not the same as designers.
- Everything changes, all the time. Stability = Death.
- Programmers define computations—software engineers make real life systems (cars, nuclear reactors) work.
- Scale—cost & effort can increase more than linearly.
Software Engineering: 

Processes, Models, and Tools

- **Processes:** Systematic ways of organizing teams and tasks so that there is a clear, traceable path from customer requirements to the final product. e.g.,:
  - Waterfall, Prototyping, Spiral etc.

  Processes help organize and co-ordinate teams, prepare documentation, reduce bugs, manage risk, increase productivity, etc.

- **Models:** Well-defined formal or informal languages and techniques for organizing and communicating arguments and decisions about software. e.g:
  - specification languages (Z, etc),
  - design models (Petri Nets, UML, etc)
  - models of code (CFGs, SeeSoft, etc)).

  Models help stake-holders communicate: customers with developers, designers and developers, developers and testers etc. If they are formal, they also can help support automation.

- **Tools:** Programs which automate or otherwise support software development tasks: e.g.,
  - Make, CVS, Purify, Eclipse, etc.

  Tools increase productivity, quality and can reduce costs
Course Outline

• Software Design
  – Designing like a Pro—the use of MVC and other design patterns.
  – Software Architectures: Client/Server, 3 Tier, Blackboard, etc.
  – OO Design Methodology: OMT.

• Requirements/HCI

• Software Processes

• Implementation and Testing
  – Formal approaches to verification: Z, Petri Nets, Correctness proofs.
  – Testing
  – Coding defensively—tricks from the battlefront.
Assignment 1 Goal: to illustrate requirements elicitation

- Put yourself in the users’ shoes.
- Imagine yourself using the system. Are all the expected responses clear from the specification?
- What can go wrong? (ordinary things). What’s a graceful response?
- Can different features interact?
- If I ask the customer this question, is s/he going to think I’m a fool?