CSE 403
Lecture 1

Introduction

Reading:
*Rapid Development*  Ch. 3.3

slides created by Marty Stepp
http://www.cs.washington.edu/403/
Software engineering

- **software engineering**: Creating and maintaining software applications by applying technologies and practices from computer science, project management, and other fields.

  - Software engineering is about people working in teams under stress to create value for their customers.

  - Software engineering has accepted as its charter, "How to program if you cannot." -- E. Dijkstra

  - The first step toward the management of disease was replacement of demon theories and humours theories by the germ theory. That very step, the beginning of hope, in itself dashed all hopes of magical solutions. It told workers that progress would be made stepwise, at great effort, and that a persistent, unremitting care would have to be paid to a discipline of cleanliness. So it is with software engineering today. -- F. Brooks
Aspects of software engr.

1. **Processes** necessary to turn a concept into a robust deliverable that can evolve over time

2. Working with **limited time and resources**

3. Satisfying a **customer**

4. Managing **risk**

5. Teamwork and **communication**
Ties to many fields

- **computer science** (algorithms, data structures, languages, tools)
- **business/management** (project mgmt, scheduling)
- **economics/marketing** (selling, niche markets, monopolies)
- **communication** (managing relations with stakeholders: customers, management, developers, testers, sales)
- **law** (patents, licenses, copyrights, reverse engineering)
- **sociology** (modern trends in societies, localization, ethics)
- **political science** (negotiations; topics at the intersection of law, economics, and global societal trends; public safety)
- **psychology** (personalities, styles, usability, what is fun)
- **art** (GUI design, what is appealing to users)

• necessarily "softer"; fewer clearly right/wrong answers
Roles of people in software

- **customer / client**: wants software built
  - often doesn't know what he/she wants

- **managers / designers**: plan software
  - difficult to foresee all problems and issues in advance

- **developers**: write code to implement software
  - it is hard to write complex code for large systems

- **testers**: perform quality assurance (QA)
  - it is impossible to test every combination of actions

- **users**: purchase and use software product
  - users can be fickle and can misunderstand the product
Projects

- you make proposals (then vote on which projects to develop)
  - start thinking about ideas today
  - ideas from previous quarters are linked from the web sites

- project development in stages
  - reflects modern methodologies for effective development
  - you get feedback from us after each stage, but also regularly during the development at each stage

- project teams need to have at least 5-6 members
  - otherwise it'd be toy development, and you'd miss on some of the most important experiences
What's in it for you

• what you'll learn
  – get exposure to **software development practices** in use today
  – learn how to **collaborate** with others toward a common goal
  – see **how software is produced**, from idea to ship to maintenance
  – get **experience working in a large team** toward a common goal
  – be able to **articulate and understand ideas** in a conversation
  – understand **issues and tradeoffs** in decisions as a manager

• tools you'll use
  – design diagram (UML) software
  – integrated development environments (IDEs)
  – test suites (JUnit) and benchmarking / profiling software
  – content management systems (CVS, Subversion)
  – performance testing (profiling) software
Unique aspects of course

• cross-disciplinary nature of the subject
• larger-size teams
• opportunity to propose and work on your own ideas
• instructors and TAs in the "coach" role
• (some) mistakes along the way are encouraged, not penalized
• few clearly right/wrong answers
• plans always change
• content: software design, testing, project management, etc.
• "**Work together** (in the same place) as much as possible."

• "Well-run and consistently scheduled **meetings** help a lot."

• "We often underestimated tasks. If we had spent more time analyzing each task and **breaking it down** into smaller chunks, our estimated times would have been more accurate."

• "Don't underestimate the difficulty of **learning** new programming languages, frameworks and tools."

• "Make small, **frequent updates** and commits to your source repository. Failing to do this results in merges that can be a nightmare."
What is a software project?

- "Good, fast, cheap ... Choose 2"
Making software is hard

• Historically, ~ 85% of software projects "fail." Why?
  – management sets unrealistic expectations; devs don't correct them
  – overestimating the positive impact of shiny new tools and hardware
  – hired developers based on availability despite warning signs
  – personality conflicts between developers
  – changes in rate structure requirements in middle of work
  – one delay causes another (dev delay leads to test delay, etc.)
  – hacks and shortcuts
  – developers end up working "death marches" (6-day, 10-hour weeks)
  – overestimating how nearly done you are ("I'm 90% there!")
  – software written doesn't match the spec
  – developer time taken away by other tasks
  – tons of bugs come out in testing
  – developers don't listen to testers; ignore severity of bugs reported
  – management breaking promises (bonuses, time off, etc.)
## Kinds of mistakes made

<table>
<thead>
<tr>
<th>People</th>
<th>Process</th>
<th>Product</th>
<th>Technology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Undermined motivation</td>
<td>Overly optimistic schedules</td>
<td>Requirements gold-plating</td>
<td>Silver-bullet syndrome</td>
</tr>
<tr>
<td>Weak personnel</td>
<td>Insufficient risk management</td>
<td>Feature creep</td>
<td>Overestimated savings from new tools or methods</td>
</tr>
<tr>
<td>Uncontrolled problem employees</td>
<td>Contractor failure</td>
<td>Developer gold-plating</td>
<td>Switching tools in the middle of a project</td>
</tr>
<tr>
<td>Heroics</td>
<td>Insufficient planning</td>
<td>Push-me, pull-me negotiation</td>
<td>Lack of automated source-code control</td>
</tr>
<tr>
<td>Adding people to a late software project</td>
<td>Abandonment of planning under pressure</td>
<td>Research-oriented development</td>
<td></td>
</tr>
<tr>
<td>Noisy, crowded offices</td>
<td>Wasted time during the &quot;fuzzy front end&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Friction between developers and customers</td>
<td>Shortchanged upstream activities</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unrealistic expectations</td>
<td>Inadequate design</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of effective project sponsorship</td>
<td>Shortchanged quality assurance</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of stakeholder buy-in</td>
<td>Insufficient management controls</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of user input</td>
<td>Premature or overly frequent convergence</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Politics placed over substance</td>
<td>Omitting necessary tasks from estimates</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wishful thinking</td>
<td>Planning to catch up later</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Code-like-hell programming</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Is SWE different?

Are the problems faced in software any different than those faced in other engineering fields? Arguments in favor:

– testing software quality is hard (example: Halting Problem)
– lower barrier to entry
– immaturity of the discipline
– customer expectations: quality, delivery timeline, etc.
– fast pace of technological change
– software is easier to copy

Arguments against:

– software isn't always "soft"
  • change is not easy, yet requirements do change
  • change often forces a rewriting of major parts of the software
– developers still need to plan, execute, test, and sell
– the discipline is still in its infancy