Software Development Lifecycle

thinking about the process

Next week

• Monday:
  – interactive section on extreme programming
  – Play SimSE (think SimCity but with software engineers)
  – Bring a laptop!

Software Development Lifecycle

thinking about the process

How complex is software?

• Measures of complexity:
  – lines of code
  – number of classes
  – number of modules
  – module interconnections and dependencies
  – time to understand
  – # of authors
  – ... many more

Windows Server 2003: 50 MSLoC
Debian 5.0: 324 MSLoC

How complex is software?

• Google keeps all their code in a single repository, all at HEAD
  • Sept 16, 2015 WIRED article reported that code is 2 billion lines of code
  • [Link](http://www.wired.com/2015/09/google-2-billion-lines-code-and-one-place/)
<table>
<thead>
<tr>
<th>Managing software development</th>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Requirements</td>
<td>• Why do we need a lifecycle process?</td>
</tr>
<tr>
<td>• Design</td>
<td>• Lifecycle models and their tradeoffs</td>
</tr>
<tr>
<td>• Implementation</td>
<td>– code-and-fix</td>
</tr>
<tr>
<td>• Testing</td>
<td>– waterfall</td>
</tr>
<tr>
<td>• Maintenance</td>
<td>– spiral</td>
</tr>
<tr>
<td></td>
<td>– staged delivery</td>
</tr>
<tr>
<td></td>
<td>– agile (scrum)</td>
</tr>
<tr>
<td></td>
<td>– ... there are many others</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ad-hoc development</th>
<th>Ad-hoc development disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Creating software without any formal guidelines or process</td>
<td>• Some important actions (testing, design) may go ignored</td>
</tr>
<tr>
<td>• Advantage: easy to learn and use!</td>
<td>• Unclear when to start or stop each task</td>
</tr>
<tr>
<td>• Disadvantages?</td>
<td>• Scales poorly to multiple people</td>
</tr>
<tr>
<td></td>
<td>• Hard to review or evaluate one’s work</td>
</tr>
</tbody>
</table>

> The later a problem is found in software, the more costly it is to fix.

<table>
<thead>
<tr>
<th>What makes a lifecycle?</th>
<th>Benefits of using a lifecycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Requirements</td>
<td>• provides a work structure</td>
</tr>
<tr>
<td>• Design</td>
<td>• forces thinking about the “big picture”</td>
</tr>
<tr>
<td>• Implementation</td>
<td>• helps prevent decisions that are individually on target but collectively misdirected</td>
</tr>
<tr>
<td>• Testing</td>
<td>• assists management and progress control</td>
</tr>
<tr>
<td>• Maintenance</td>
<td>How do we combine them?</td>
</tr>
</tbody>
</table>
What are some drawbacks?

Are there analogies outside of SE? Consider the process of building the Prudential Project with little attention to process

Survival Guide: McConnell p24

Project with little attention to process


Project with early attention to process

Let’s talk about some lifecycle models

Code-and-fix model
**Code-and-fix model**

- **Advantages**
  - Low overhead
  - Applicable to small, short-lived projects

- **Dangers**
  - No way to assess progress and manage risks
  - Hard to accommodate changes
  - Unclear what and when will be delivered
  - Hard to assess quality

**Waterfall model**

- **System requirements validation**
- **Software requirements validation**
- **Preliminary design validation**
- **Detailed design validation**
- **Operations & maintenance revalidation**
- **Test**

**Waterfall model advantages**

- Works well for well-understood projects
  - Tackles all planning upfront
  - No midstream changes leads to efficient software development process

- Supports experienced teams
  - Orderly, easy-to-follow sequential model
  - Reviews help determine readiness to advance

**Waterfall model limitations**

- Difficult to do all planning upfront
- No sense of progress until the end
- Integration occurs at the very end
  - Defies the “integrate early and often” rule
  - Without feedback, solutions are inflexible
  - Final product may not match customer’s needs
- Phase reviews are massive affairs
  - It takes a lot of inertia and $ to make changes

**Spiral model**

- Oriented towards phased reduction of risk
- Take on the big risks early
  - Are we building the right product?
  - Do we have customers for this product?
  - Is it possible to use existing technology?
  - Tomorrow’s technology?
- Progresses carefully toward a result
Spiral model advantages

- Especially appropriate at the beginning of the project, allowing requirement fluidity
- Provides early indication of unforeseen problems
- Allows for change
- As costs increase, risks decrease!

Addresses the biggest risk first

Spiral model disadvantages

- A lot of planning and management
- Requires customer and contract flexibility
- Developers must be able to assess risk

Staged delivery model

- Can ship at the end of any release cycle
- Intermediate deliveries show progress, satisfy customers, and lead to feedback
- Problems are visible early (e.g., integration)
- Facilitates shorter, more predictable release cycles

Very practical, widely used and successful

Staged delivery model disadvantages

- Requires tight coordination with documentation, management, marketing
- Product must be decomposable
- Extra releases cause overhead

What’s the best model?

Consider
- The task at hand
- Risk management
- Quality / cost control
- Predictability
- Visibility of progress
- Customer involvement and feedback

Aim for good, fast, and cheap. But you can’t have all three at the same time.
Next week

- Monday:
  - interactive section on extreme programming
  - Play SimSE (think SimCity
    but with software engineers)
  - Bring a laptop!