50MLOC = 50 million lines of code

- 50 lines/page-side \(\Rightarrow\) 1M page-sides
- 1K page-sides/ream \(\Rightarrow\) 1K reams
- 2 inches/ream \(\Rightarrow\) 2K inches
- 2K inches = 167 feet \(\approx\) twice the height of the Allen Center

- 5 words/LOC \(\times\) 50 wpm \(\Rightarrow\) 50MLOC/5M min
- 5M min = 83,333 hr = 3,472 days \(\approx\) 10 years

Addressing software complexity

What are/is the ...?
- Requirements
- Design
- Implementation
- Testing plan
- ...

Who does the ...?
- Requirements
- Design
- Implementation
- Testing
- ...

- Two sides of the same coin
- Different approaches, representations, etc. are needed for the artifact-oriented components
- Different skill-sets, knowledge, etc. are needed for the human-oriented components

Outline

- What is a software development lifecycle?
- Why do we need a lifecycle process?
- Lifecycle models and their tradeoffs
  - “Code-and-fix”
  - Waterfall
  - Spiral
  - Evolutionary prototyping
  - Staged delivery
  ...there are many others (XP, scrum, ...!)
- Main recurring themes (Agile processes)

Ad-hoc development

- **ad-hoc development**: creating software without any formal guidelines or process
- Advantage: easy to learn and use!
- Disadvantages?

- Some disadvantages of ad-hoc development:
  - some important actions (testing, design) may go ignored
  - not clear when to start or stop doing each task
  - does not scale well to multiple people
  - not easy to review or evaluate one’s work

- A common observation: The later a problem is found in software, the more costly it is to fix.
**Lifecycle stages**

- Virtually all lifecycles share
  - Requirements
  - Design
  - Implementation
  - Testing
  - Maintenance
- Key question: how do you combine them, and in what order?

**The software lifecycle**

- Software lifecycle: series of steps / phases, through which software is produced
  - from conception to end-of-life
  - can take months or years to complete
- Goals of each phase:
  - mark out a clear set of steps to perform
  - produce a tangible item
  - allow for review of work
  - specify actions to perform in the next phase

**Benefits of using a lifecycle**

- It provides us with a structure in which to work
- It forces us to think of the "big picture" and follow steps so that we reach it without glaring deficiencies
- Without it you may make decisions that are individually on target but collectively misdirected
- It is a management tool

**Benefits of using a lifecycle**

- It provides us with a structure in which to work
- It forces us to think of the "big picture" and follow steps so that we reach it without glaring deficiencies
- Without it you may make decisions that are individually on target but collectively misdirected
- It is a management tool

**Benefits of using a lifecycle**

- It provides us with a structure in which to work
- It forces us to think of the "big picture" and follow steps so that we reach it without glaring deficiencies
- Without it you may make decisions that are individually on target but collectively misdirected
- It is a management tool

**Are there analogies outside of SE?**

Consider the process of building the Paul Allen Center

**Drawbacks?**
Project with little attention on process

Survival Guide: McConnell p24

Project with early attention on process


“Code-and-fix” model

Advantages
• Little or no overhead - just dive in and develop, and see progress quickly
• Applicable sometimes for very small projects and short-lived prototypes

But DANGEROUS for most projects
• No way to assess progress, quality or risks
• Unlikely to accommodate changes without a major design overhaul
• Unclear delivery features (scope), timing, and support

Classic waterfall model

Classic waterfall advantages
• Can work well for projects that are: very well understood but complex
  – Tackles all planning upfront
  – The ideal of no midstream changes equates to an efficient software development process
• Can provide support for an inexperienced team
  – Orderly sequential model that is easy to follow
  – Reviews at each stage determine if the product is ready to advance
Classic waterfall limitations

- Difficult to specify all reqs of a stage completely and correctly upfront
- No sense of progress until the very end
- Integration occurs at the very end
  - Defies integrate early and often rule
  - Solutions are inflexible, no feedback until end
  - What is delivered may not match customer real needs
- Phase reviews are massive affairs
  - It takes a lot of inertia ($$) to make any change

Spiral model – risk oriented

- Oriented towards phased reduction of risk
- Take on the big risks early and make some decisions
  - are we building the right product?
  - do we have any customers for this product?
  - is it possible to implement the product with the technology that exists today? tomorrow?
- Walks carefully to a result -- tasks can be more clear each spiral

Spiral model

Advantages
- Especially appropriate at the beginning of the project when the requirements are still fluid
- Provides early indication of unforeseen problems and allows for change
- As costs increase, risks decrease!
  - Always addresses the biggest risk first

Advantages
- Especially appropriate at the beginning of the project when the requirements are still fluid
- Provides early indication of unforeseen problems and allows for change
- As costs increase, risks decrease!
  - Always addresses the biggest risk first

Limitations?
Spiral model

Advantages

- Especially appropriate at the beginning of the project when the requirements are still fluid
- Provides early indication of unforeseen problems and allows for change
- As costs increase, risks decrease!
  - Always addresses the biggest risk first

Limitations?

- Lots of planning and management
- Requires flexibility of the customer & contract
- Relies on developers to have risk-assessment expertise

Staged delivery model

Advantages

- Can ship at the end of any release-cycle
- While not feature complete, intermediate deliveries show progress, satisfy customers, and provide opportunity for feedback
- Problems are visible early (ie. integration)
- Facilitates shorter, more predictable release cycles

Limitations?

- Requires tight coordination with documentation, mgmt, mktg
- Must be decomposable
- Extra “release” overhead

Evolutionary prototyping model

Develop a skeleton system and evolve it for delivery
Evolutionary prototyping model

Another popular and successful model, especially for custom products

**Advantages**
- Addresses risks early
- Produces steady signs of progress
- Useful when requirements are changing rapidly or customer is non-committal

**Limitations?**

Why are there so many models?

**Choices are good!**

- The choice of a model depends on the project circumstances and requirements
- A good choice of a model can result in a vastly more productive environment than a bad choice
- A cocktail of models is frequently used in practice to get the best of all worlds. Models are often combined or tailored to environment

How do you evaluate models?

- Consider
  - The task at hand
  - Risk management
  - Quality / cost control
  - Predictability
  - Visibility of progress
  - Customer involvement and feedback
- Theme: Aim for good, fast, and cheap. But you can't have all three at the same time.

Model category matrix

<table>
<thead>
<tr>
<th>Model type</th>
<th>Risk mgmt.</th>
<th>Quality/cost ctrl</th>
<th>Predictability</th>
<th>Visibility of progress</th>
<th>Customer involvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>code-and-fix</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>waterfall</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>spiral</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>evolutionary prototyping</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>staged delivery</td>
<td>3</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>design-to-schedule</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>
What's the best SW dev model?

- A system to control anti-lock braking in a car
- A hospital accounting system that replaces an existing system
- An interactive system that allows airline passengers to quickly find replacement flight times (for missed or bumped reservations) from terminals installed at airports