CS 370
The Importance of Prerequisites

DR. MICHAEL J. REALE
FALL 2015
Introduction

- When building a house ➔ need to prepare
  - Blueprints
  - Permits
  - Survey the site

- Important to prepare prerequisites before software construction starts
  - Called “upstream work”

- Prerequisites include:
  - Software Requirements
  - Software Architecture/Design

- Carpenter’s expression: “Measure twice; cut once.”
Why Do Programmers Skip This Step?

- Do not know how to do “upstream work”
- Want to get started coding as soon as possible
- Unsympathetic managers
  - WISCA or WIMP syndrome
    - “Why Isn’t Sam Coding Anything?”
    - “Why Isn’t Mary Programming?”
  - Possible solutions:
    - Agree to code → bad idea
    - Pretend to code → ethically questionable
    - Educate your boss → best idea where possible
    - Find another job
Why are Prerequisites Important?

One can make the case for prerequisites in three ways:
- Appeal to Logic
- Appeal to Analogy
- Appeal to Data
Appeal to Logic

- Appeal to Logic
  - Need to know cost
    - Time, people, resources
  - Know what you’re building!
    - Build the wrong thing $\rightarrow$ have to throw work away and start over $\rightarrow$ EXPENSIVE
  - Risk reduction
    - Try to get major risks out of the way BEFORE you start building
Appeal to Analogy

- Building a house $\rightarrow$ need blueprints, permits, etc.

- Food chain
  - Requirements $\rightarrow$ Architecture $\rightarrow$ Design $\rightarrow$ Code
  - “Pollutants” will propagate down food chain

- Building housing development (iterative project)
  - Don’t have houses fully planned BUT know where electrical/sewer lines are, etc.
## Appeal to Data

- **Appeal to Data**
  - Studies over last 25 years → pays to do things right the first time
    - Hewlett-Packard, IBM, Hughes Aircraft, TRW, and others
  - Early a defect is in the “food chain” → more expensive to repair
    - Error in requirements → MOST expensive to repair later
  - True whether project is:
    - Highly sequential → 100% of requirements/design up front
    - Highly iterative → 5% of requirements/design up front

### Time Detected

<table>
<thead>
<tr>
<th>Time Introduced</th>
<th>Requirements</th>
<th>Architecture</th>
<th>Construction</th>
<th>System Test</th>
<th>Post-Release</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>1</td>
<td>3</td>
<td>5-10</td>
<td>10</td>
<td>10-100</td>
</tr>
<tr>
<td>Architecture</td>
<td>--</td>
<td>1</td>
<td>10</td>
<td>15</td>
<td>25-100</td>
</tr>
<tr>
<td>Construction</td>
<td>--</td>
<td>--</td>
<td>1</td>
<td>10</td>
<td>10-25</td>
</tr>
</tbody>
</table>

Table from “Code Complete”, pg. 29
Different Approaches

- Two development approaches:
  - Sequential ("Waterfall model")
  - Iterative

- Usually project is some combination of the two

- Plan-driven $\rightarrow$ traditionally more sequential
- Agile $\rightarrow$ traditionally more iterative
Do We Need Prereqs with Iterative Projects?

- Projects without prerequisites → iterative cost < sequential cost
  - Errors detected closer to time of insertion
  - HOWEVER:
    - Discovering errors at end of iteration → still need to redesign, recode, retest → unnecessary cost
    - Costs absorbed piecemeal → average cost will be about the same
- (Iterative + prerequisites) cost < (Iterative only) cost
Which to Choose...

- **Sequential, up-front**
  - Requirements → fairly stable
  - Design → straightforward and fairly well understood
  - Dev team → familiar with applications area
  - Project → contains little risk
  - Long-term predictability → important
  - Cost of changing requirements/design/code downstream → likely to be high

- **Iterative**
  - Requirements → not well understood or unstable
  - Design → complex and/or challenging
  - Dev team → unfamiliar with applications area
  - Project → contains lot of risk
  - Long-term predictability → NOT important
  - Cost of changing requirements/design/code downstream → likely to be low