

CS194

March 31, 2011

CS194 -- Getting Started

Form a group and choose a topic

Use **Piazza** to find partners/topics

Get topic approval: cs194mail@gmail.com

Proposal: due midnight, Sunday, April 10

In-class presentation, April 12/14
(there will be a handout on this next week)

CS194 -- Getting Started

ME310: SUDS today!

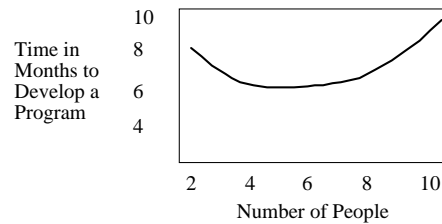
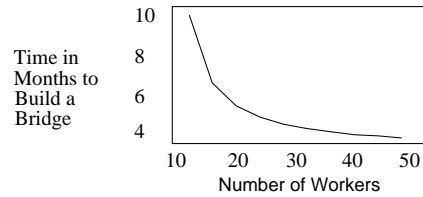
Today: 6:15

Peterson Building room 204

Software engineering deals with the problems that arise when programs are

- large
- involve many programmers
- exist over a long period of time.

The Mythical Man-month



The Traditional Development Process

Requirements

The development team meets with the client to determine what the client needs.

Answers the question: What do you need?

Specifications

The functionality of the program is defined.

Answers the question: What does the program do?

The Traditional Development Process

Design

The structure of the program is determined.

Answers the question: How does the program work?

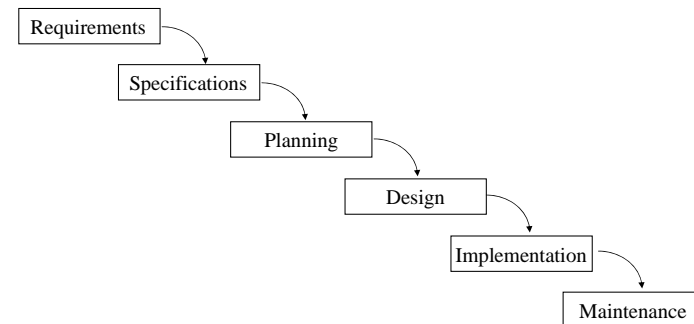
Implementation

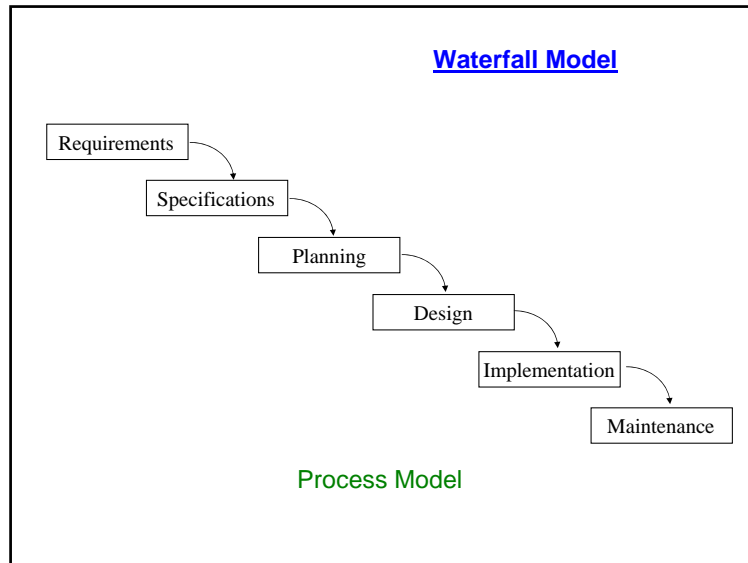
Coding and testing.

Maintenance

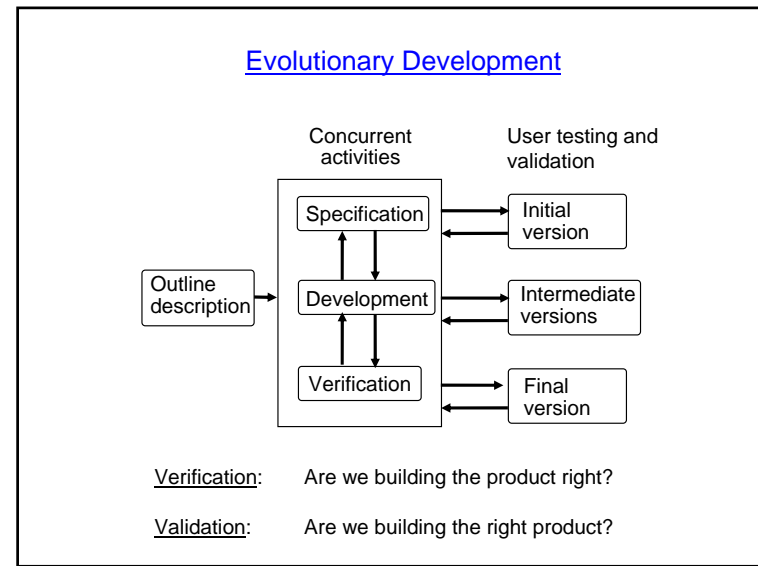
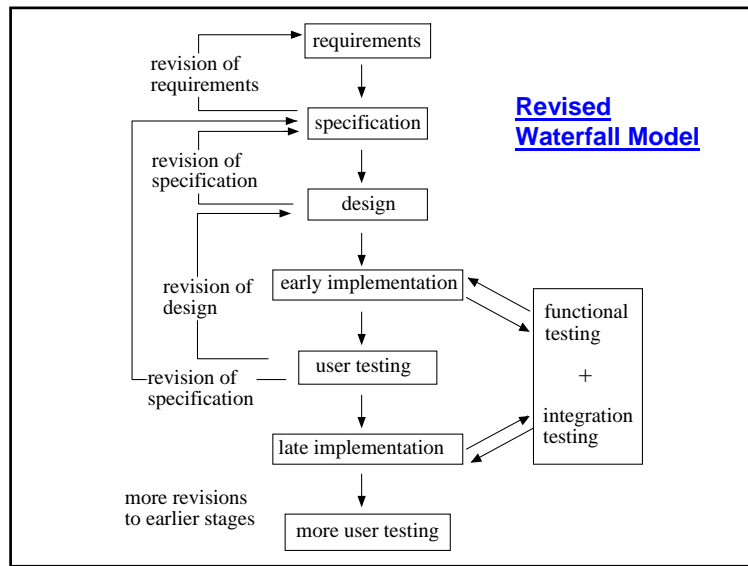
After the initial release, the program is corrected or upgraded.

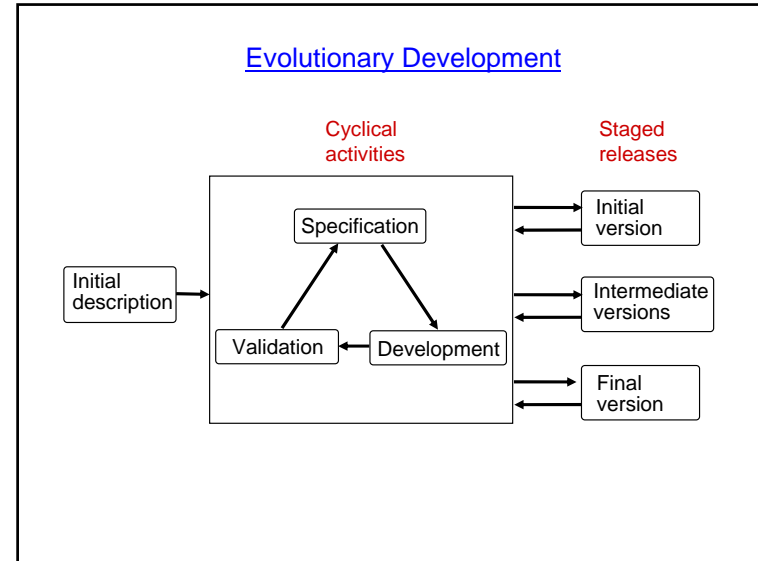
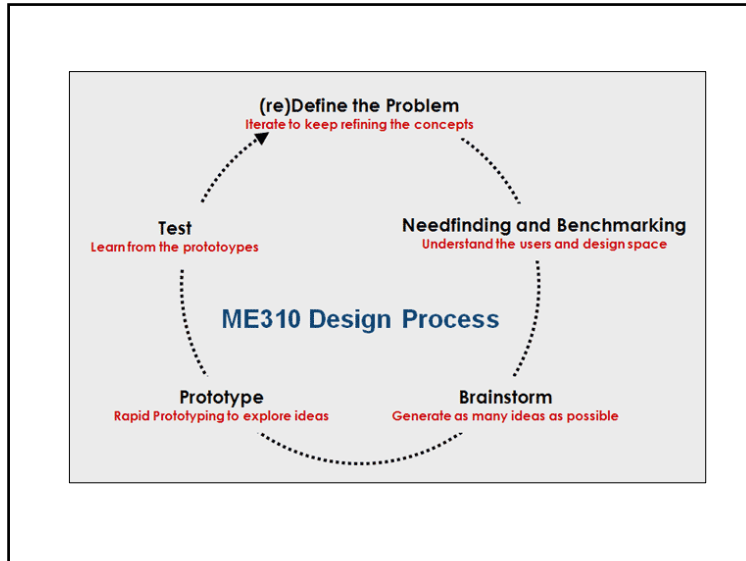
Waterfall Model





Phase	Relative Time to Fix Bugs
Requirements	1
Specification	2
Design	4
Implementation	10
Integration	30
Maintenance	200





Agile Software Development

Agility is the ability to create and respond to change in order to profit in a turbulent business environment.

Agility is the ability to create and respond to change in order to profit in a turbulent business environment.

Companies that can

- innovate better and faster
- respond quickly to
 - competitive initiatives
 - new technology
 - customer's requirements

Agility is the ability to create and respond to change in order to profit in a turbulent business environment.

Companies that can

- innovate better and faster
- respond quickly to
 - competitive initiatives
 - new technology
 - customer's requirements

will win.

The following are **flawed assumptions**:

- It is possible to plan the project well enough so that adhering to the plan produces success.
- It is possible to protect against late changes to a large systems project.

Alan MacCormack (Harvard Business School) asked executives at a software company to provide examples of "good" and "bad" software projects that had led to shipping products.

MacCormack and colleagues rated the products in terms of market acceptance, expert quality rating, productivity, etc.

Alan MacCormack (Harvard Business School) asked executives at a software company to provide examples of "good" and "bad" software projects that had led to shipping products.

MacCormack and colleagues rated the products in terms of market acceptance, expert quality rating, productivity, etc.

The "good" products were market failures.

The "bad" products were a marketplace success.

The executives considered a good project to be one where

- the specification was completed up front
- the design had been frozen
- the project was executed efficiently
- the team built what they set out to build

The executives felt that the bad projects were ones where the final results were very different from the original goal.

The executives considered a good project to be one where

- the specification was completed up front
- the design had been frozen
- the project was executed efficiently
- the team built what they set out to build

The executives felt that the bad projects were ones where the final results were very different from the original goal.

MacCormack:

"The people who were overseeing projects assumed that the good projects were the ones that delivered to the spec.

In fact, good projects are ones that deliver to the market."

Another study:

MacCormack, A. D. "Product-Development Practices That Work: How Internet Companies Build Software." *Sloan Management Review* 42, no. 2 (winter 2001): 75-84.

20 projects from 17 companies were evaluated.

MacCormack identified four practices that lead to success:

1. An **early release** of the evolving product to the customer.
2. Getting **rapid feedback from the customer** and incorporating that feedback into new design experiments.
3. A **team structure** that will allow the right decisions to be made on the fly.
4. Choosing a product architecture that **allows for change** rather than attempting to get optimal performance.

The Agile Software Development movement was born in Feb., 2001 at the Lodge at Snowbird, Utah.

17 people got together to talk about newly emerging alternatives to document-driven, rigorous software development.

They produced the **Agile Manifesto**.

The Agile Manifesto

We value:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

Principles behind the Agile Manifesto

We follow these principles:

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must work together daily throughout the project.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Working software is the primary measure of progress.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design enhances agility.

Simplicity--the art of maximizing the amount of work not done--is essential.

The best architectures, requirements, and designs emerge from self-organizing teams.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Do these techniques really work?

Do these techniques really work?

Robert Austin (Harvard) surveyed IT managers and found the following characteristics of successful projects:

- They were all iterative.
- They all relied on fast cycles and early delivery.
- They all got functioning software into the hands of business users early in the process.
- They were preceded by little or no traditional ROI-style analysis of the project as a whole.

The basic problem: Risk

- Schedule slips
- Project canceled
- System goes sour
- Defect rate
- Business misunderstood
- Business changes
- False feature rich

The basic problem: Risk

- | | |
|------------------------|---|
| Schedule slips | Short release cycles |
| Project canceled | Smallest release that makes sense |
| System goes sour | Maintain a suite of tests |
| Defect rate | Testing by programmers and customers |
| Business misunderstood | Make the customer part of the team |
| Business changes | Short release cycles |
| False feature rich | Address only the highest priority tasks |

Agile Methodologies

- Scrum
- Dynamic Systems Development Method (DSDM)
- Crystal Methods
- Feature-Driven Development (FDD)
- Lean Development (LD)
- Extreme Programming (XP)
- Adaptive Software Development (ASD)

Agile Methodologies

Scrum

Dynamic Systems Development Method (DSDM)

Crystal Methods

Feature-Driven Development (FDD)

Lean Development (LD)

Extreme Programming (XP)

Adaptive Software Development (ASD)

