



Lecture 18

Software Project Management



MIT EECS

MIT 6.170 Slide 1




Plan for Project Management Lectures

Achieving technical success
 It works, i.e., meets customer's needs
 You know it works
 Basis for future projects
 New people can work on it

Focus on management and process issues
 Big picture matters
 Difference between success and failure on large projects
 Also of relevance to your final project

Outline
 Team Organization Issues
 Development Process Models
 The Big Picture: Anatomy of a Product Design Cycle

MIT 6.170 Slide 2




Team Organization

Most importantly, you need one

Two distinct considerations
 How decisions get made (management structure)
 How information flows (communication structure)
 Have a plan for this

Two poles
 Decentralized
 What I recommend for your project
 Centralized
 Needed for large projects like Windows

MIT 6.170 Slide 3




Decentralized Organization

Key decisions made jointly (a plus and a minus)
 Requirements
 High level design
 Schedule
 Who will work on what
 Response to slippage

Lower level design and code exchanged for examination
 Everyone responsible for everything
 Code reviews tremendously helpful
 Try it, you'll like it

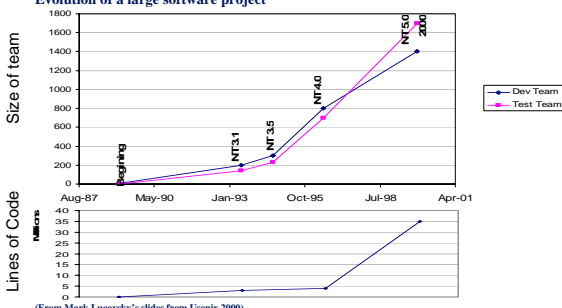
Do need a leader at all times
 Chair meetings, provide agenda, make decisions
 Leadership changes based on relevant expertise
 Someone needs to exercise oversight to notice problems

MIT 6.170 Slide 4



Windows NT Development Project

Evolution of a large software project




The top graph shows the 'Size of team' (Y-axis, 0 to 1800) over time (X-axis, Aug-87 to Apr-01). It tracks two teams: 'Dev Team' (blue line with diamonds) and 'Test Team' (magenta line with squares). Key milestones are marked: 'Beginning' (Aug-87), 'NT 3.1' (Jan-93), 'NT 3.5' (Oct-95), 'NT 4.0' (Jul-98), and 'NT 5.0 2000' (Apr-01). Both teams show exponential growth, with the Dev Team reaching approximately 1400 and the Test Team reaching approximately 1600 by the end of the project.

The bottom graph shows 'Lines of Code' in 'Millions' (Y-axis, 0 to 40) over the same time period. A single blue line shows a steady increase from near zero in 1987 to approximately 35 million lines of code by 2001.

(From Mark Lucovsky's slides from Usenix 2000)

MIT 6.170 Slide 5



Team Culture


Need to establish a culture to Scale a development team
 Common way of evaluating designs, making tradeoffs etc.
 Common way of developing code and reacting to problems
 Common way of establishing ownership of problems

Creating and maintaining the culture
 Goal setting can be the foundation for the culture
 Keeping a culture alive as a team grows is a challenge
 Some parts of the culture may not scale

At the beginning (NT 3.1)
 Everyone owns all the code
 Sloppiness is not tolerated
 Accept that mistakes will happen; react and correct them fast

At the end (NT 5.0)
 Small independent teams where everyone owns all the code
 Well define process when checking into the main source base

MIT 6.170 Slide 6




Establish a set of Goals

Puts everyone on the same mindset
 The few visionaries cannot check everything
 Goals will make the team keep to the vision, build a team culture

Goals set at the beginning of Windows
 Portability – Ability to target more than one processor architecture
 Reliability – Nothing should be able to crash the OS (right!)
 Extensibility – Ability to extend the OS over time
 Compatibility – With DOS, OS/2, POSIX
 Performance – is less important than all of the above


MIT 6.170 Slide 7



Building a Functioning Team

The five important factors in Windows
 Development Teams
 Source Code Control System
 Process Management
 Serialized Development
 Defects

MIT 6.170 Slide 8




Development teams

At the beginning (NT 3.1)
 Start very small (5), grows very slowly to 200
 NT culture was commonly understood by all

At the end (NT 5.0)
 More than 1400 developers!
 Mass assimilation of other teams into the NT team
 Original culture kept by the oldtimers,
 But keeping it alive was hard
 Diluted culture led to many conflicts and delays
 Less accountability
 Reliability vs. new features

MIT 6.170 Slide 9




Source Code Control System

At the beginning (NT 3.1)
 Very simple tool
 10-12 well isolated source “projects” (3M LOC)
 Informal project separation worked well
 Developer can easily stay in sync with all changes made

At the end (NT 5.0)
 Multiple branches, parallel isolated development
 180 source “projects” (35M LOC)
 Keeping in-sync become very hard (1 week to setup)

MIT 6.170 Slide 10




Process Management

At the beginning (NT 3.1)
 Save sync period in effect for 4 hours each morning
 All other times, check-in when ready
 Build lab synchs during the safe sync period and do a complete build (5 hours to build)
 1-2 breaks was normal
 Do minimal boot test and start stress test at 4pm (on 100 machines)

At the end (NT 5.0)
 Developers are not allowed to check-in without explicit written permission.
 100 changes approved per day.
 A break is taken very seriously (it affect over 5000 people)
 Complete build time is 8 hours
 Start stress test at 4pm (on 1000 machines)

MIT 6.170 Slide 11



Defect Rates Serialized Development

Impact of buggy code grows with the team.

| Version | Defects per year per developer | Defects per day |
|---------------|--------------------------------|-----------------|
| NT 3.1 | ~2 | ~1 |
| NT 3.5 | ~2 | ~2 |
| NT 4.0 | ~3 | ~6 |
| NT 5.0 (2000) | ~4 | ~15 |

| Version | Time to fix a defect | Total defect fix time |
|---------------|----------------------|-----------------------|
| NT 3.1 | ~10 | ~10 |
| NT 3.5 | ~10 | ~20 |
| NT 4.0 | ~10 | ~200 |
| NT 5.0 (2000) | ~10 | ~600 |

MIT 6.170 Slide 12

The Team Environment

At the beginning (NT 3.1)
 Fast and loose development, a lot of fun and energy
 Few barriers to getting work done
 Defects serialized parts of the process, but didn't stop it
 Minimal down time

At the end (NT 5.0)
 Source code control system was bursting at the seams
 Excessive process management required
 Serialized the entire process
 1 defect stops 1400 developers and 5000 team members

Making large teams work like a set of small teams is hard!

MIT 6.170 Slide 13

Goals of a Development Process Model

Reduces cost of late-stage problems
 Adequacy of specification
 Performance limitations

Rapidly builds necessary competencies
 Use new knowledge to improve design

Focuses on reducing risk
 Make (and discover) mistakes early

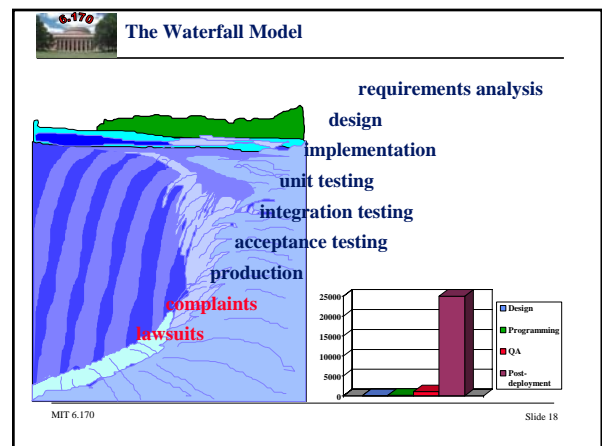
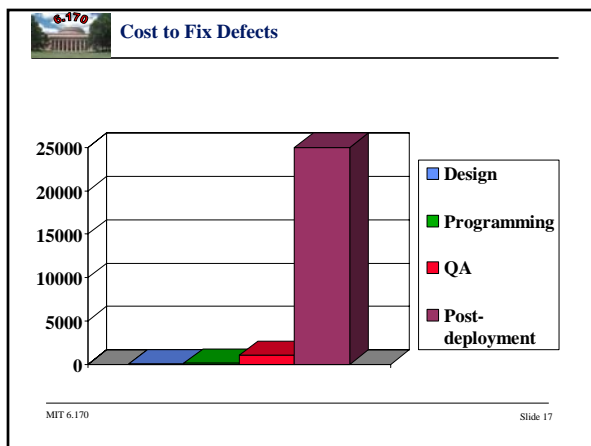
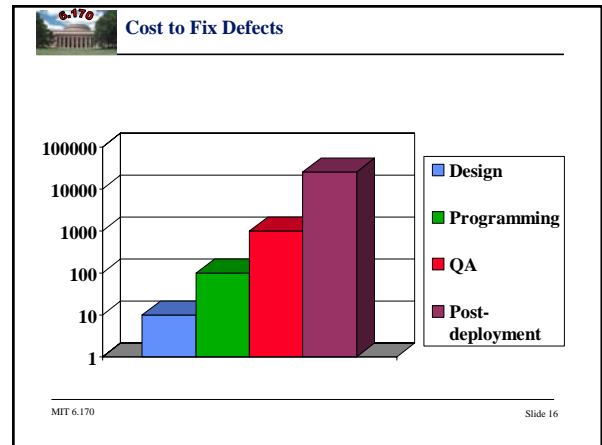
Look at one better model
 Not the only reasonable model
 Different situations call for different models

MIT 6.170 Slide 14

Development Process Models

Waterfall Model
 Hacking or Organic Development Model
 Prototyping or Incremental Development Model

MIT 6.170 Slide 15



Waterfall Model: Pros and Cons

Works well when

- The requirements are high quality and stable
- The developers have previously built similar systems
- The project is not very complex

When used in other situations

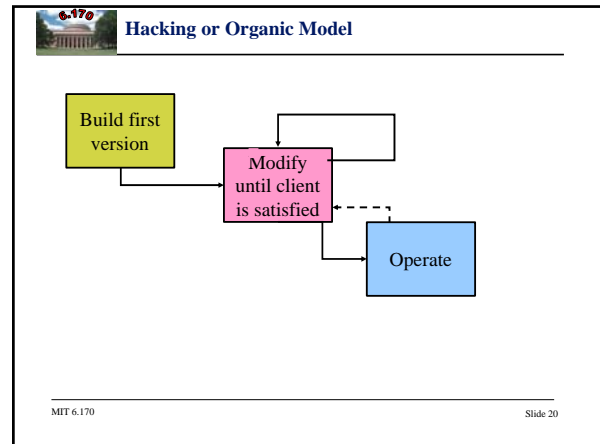
- Lots of rework
- High late stage costs
- Because everyone gets it wrong the first time

Rarely right for most companies

Might be right for late stage development

- E.g., customization

MIT 6.170 Slide 19



Hacking or Organic Model: Pros and Cons

No specification

- Figure out specification as you code

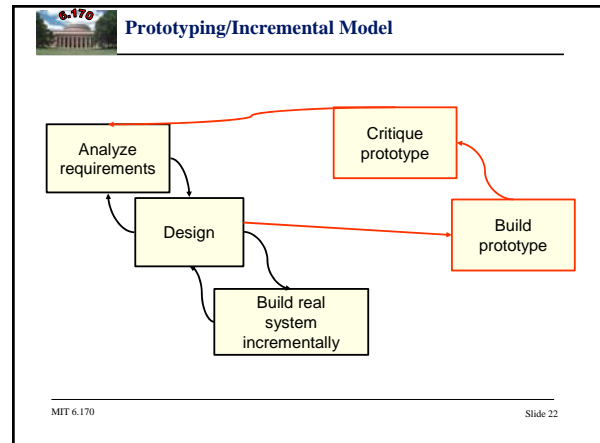
Useful for

- Short programs
- Easily accessible client

If applied to something too complex

- Lots of rework
- High costs at a late stage

MIT 6.170 Slide 21



Prototype Phase

Not hacking

- Carefully design prototypes
- Discard prototypes rather than change-until-done

Quick and dirty?

- Dirty is easy

Lots of wasted work?

- Plan to throw one away, you will anyway
- Much cheaper if mistakes discovered early

MIT 6.170 Slide 23

Some Uses of Prototypes

Sell project to management

- Be careful what you wish for
- “Managing up” is important

Understand requirements

- Build a mock up, get feedback from users
- Learn the customers needs
- More than just the UI

Understand design

- Find “gotchas” early on

Understand building blocks

- Hardware
- Programming environment
- Tool kits and libraries

MIT 6.170 Slide 24

6.170 Understand Building Blocks

May have specifications that are

- Ambiguous
- Incomplete or inaccurate
- Unclear about preconditions
- Unclear about performance

Trying building blocks out early in appropriate context

- Informs design
- Modularizes debugging process

Examples

- Gizmoball: physics package
- RSS reader: XML, RSS, HTML, database

MIT 6.170 Slide 25

6.170 Effective Prototyping

A prototype is built to answer questions

- Know what questions you wish to answer
- Write them down at the start

Use list to decide

- What functionality to implement
- What tests to run
- When discard prototype

Keep a lab notebook

- Record decisions and rationale
- Treat as a log
- Don't revise or throw out

MIT 6.170 Slide 26

6.170 Prototyping Pitfalls

Worthless prototype

- Doesn't answer right (or any) questions

Failure to discard prototype

- The longer you wait, the harder it gets
- Must have drop dead date for prototyping phase

Second system effect

- Prototype makes problem seem easier than it is
- Much of the work is the "last 10%" of getting it right
- Features get added or schedule compressed

MIT 6.170 Slide 27

6.170 Prototyping/Incremental Model

```

    graph TD
      A[Analyze requirements] --> B[Design]
      B --> C[Build prototype]
      C --> D[Critique prototype]
      D --> A
      B --> E[Build real system incrementally]
      E --> B
  
```

Example: "Extreme programming" (XP)

MIT 6.170 Slide 28

6.170 Incremental Development Phase

Short cycles: weeks, not years

| | |
|------------------------|---------------|
| Design | Redesign |
| Implement | Reimplement |
| Validate | Revalidate |
| Assess risk | Reassess risk |
| Consolidate & optimize | |

Smallest steps representing visible progress

- New behavior
- Better performance
- Reduced amount of code
- Better platform for future development

Not same as prototyping phase

- Don't throw away code

MIT 6.170 Slide 29

6.170 Advantages of Incremental Model

Feedback

- Easier to measure progress
- Better documentation
- Reality check

Leads to more modular designs

- Piecewise validation easier
- Changes easier

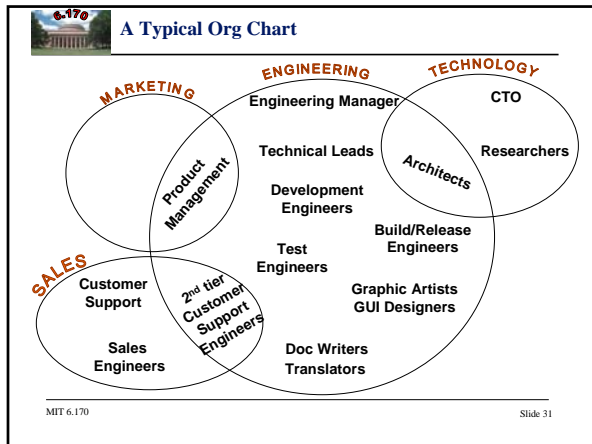
Better customer-vendor relationship

- Less adversarial
- Shared problem solving

Difficulty

- Executives/customers must pay attention
- A serious problem in real world

MIT 6.170 Slide 30



Anatomy of a New Product Development Cycle

Product Management

- Analysis of the company's strengths
- Analysis of the current product line and trends
- Customer Survey
- Market Survey
- Technology ideas from the CTO office

Product Idea

Marketing Requirement Document (MRD)

MIT 6.170 Slide 32

Anatomy of a New Product Development Cycle

Product Management

- Analysis of the company's strengths
- Analysis of the current product line and trends
- Customer Survey
- Market Survey
- Technology ideas from the CTO office

Product Idea

Marketing Requirement Document (MRD)

MIT 6.170 Slide 33

Anatomy of a New Product Development Cycle

Architect

- Take the MRD
- Figure-out how to divide the work into components
- Identify possible hard/showstopper problems
 - Get a researcher/engineer to do a Proof-of-Concept (PoC)
- Identify the high-level tasks

Development Leads for each Component

- Define the tasks and project the work required
- Define the interfaces to the rest of the components

Test Lead

- Create a test plan

Engineering Manager

- Assign Resources / People
- Identify a timeline and a release plan

→ Leads to an Engineering Requirements Document (ERD)

MIT 6.170 Slide 34