
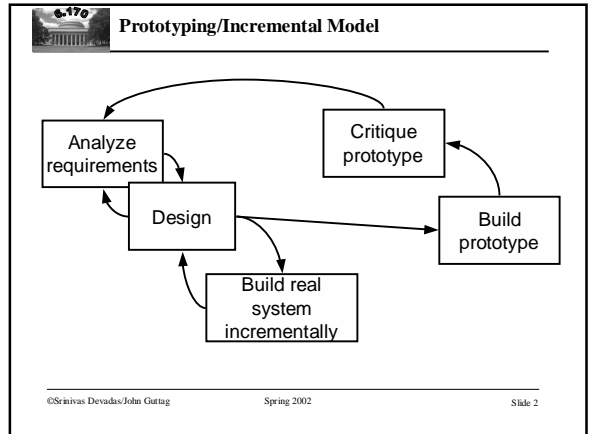


6.170 Lecture 20 Project Management and Control



John Guttag
MIT EECS



Prototype Phase

Not hacking
Carefully design prototypes
Discard prototypes rather than change-until-done

Part of requirements analysis
Learn the customer's needs more precisely
Build a mock-up, demo it, get feedback

Quick and dirty?
Dirty is easy

Lots of wasted work?
Plan to throw one away, you will anyway
Much cheaper if mistakes discovered early

©Srinivas Devadas/John Guttag Spring 2002 Slide 3

Some Uses of Prototypes

Sell project to management

Understand requirements
Build a mock up, get feedback from users
Learn the customers needs

Understand design
Find "gotchas" early on

©Srinivas Devadas/John Guttag Spring 2002 Slide 4

Be Mindful of Cost of Correcting a Defect

Measured by IBM

Design: \$10	QA: \$1000
Programming: \$100	Post-deployment: \$25,000

Phase	Cost (\$)
Design	10
Programming	100
QA	1000
Post-deployment	25000

©Srinivas Devadas/John Guttag Spring 2002 Slide 5

Some Uses of Prototypes

Understand building blocks
Hardware
Programming environment
Tool kits and libraries

©Srinivas Devadas/John Guttag Spring 2002 Slide 6

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

Example
Physics package

©Srinivas Devadas/John Guttag Spring 2002 Slide 7

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 log
Don't revise or throw out

©Srinivas Devadas/John Guttag Spring 2002 Slide 8

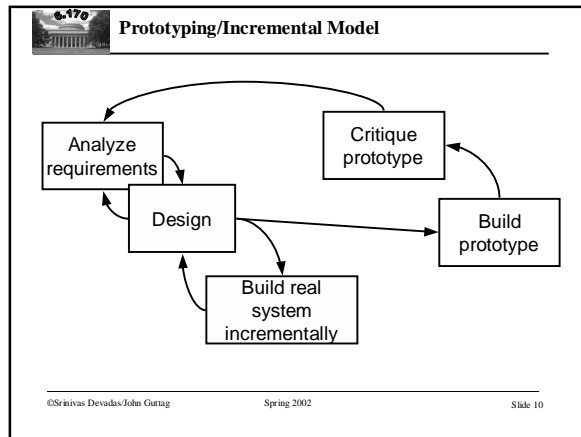
6.170 Prototyping Pitfalls

Worthless prototype
Doesn't answer right (or any) questions

Failure to discard prototype
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 goes to last 10% of getting it right
Features get added or schedule compressed

©Srinivas Devadas/John Guttag Spring 2002 Slide 9



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
Not throw away code

©Srinivas Devadas/John Guttag Spring 2002 Slide 11

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

©Srinivas Devadas/John Guttag Spring 2002 Slide 12

 **Scheduling**

“More software projects have gone awry for lack of calendar time than for all other causes combined.”
-- Fred Brooks

“More students have ...” -- John Guttag


Three central questions of software business

- 3) When will it be done?
- 2) How much will it cost?
- 1) When will it be done?

Facts

1. Estimates almost always too optimistic
2. Estimates reflect what one wishes to be true
3. We confuse effort with progress
4. Progress is poorly monitored
5. Slippage is not aggressively treated

©Srinivas Devadas/John Guttag Spring 2002 Slide 13

 **Scheduling Is Crucial**


Usually gets far less attention than appropriate
Made to fit other constraints

Needed to make slippage visible
Like an annual business plan
Must be objectively checkable by outsiders

Unrealistically optimistic schedules a disaster
Decisions get made at wrong time
Decisions get made by wrong people
Decisions get made for wrong reasons

The great paradox
Everything will take twice as long as you think
Even if you know that it will take twice as long as you think


©Srinivas Devadas/John Guttag Spring 2002 Slide 14

 **In Large Projects**

Estimates don't change as activity approaches
No matter how wrong they end up being

Once activity has started
Overestimates of cost come steadily down
Underestimates do not change until near scheduled end

©Srinivas Devadas/John Guttag Spring 2002 Slide 15

 **Optimism Root of Problem**

People assume that all will go well
Every task will take as long as it ought to take


Consider following schedule

```
graph TD
    1 --> 9
    2 --> 9
    3 --> 10
    4 --> 10
    5 --> 11
    6 --> 11
    7 --> 12
    8 --> 12
    9 --> 13
    10 --> 13
    11 --> 14
    12 --> 14
    13 --> 15
    14 --> 15
```

Suppose that on average each task takes as long as planned
Odd tasks over-run by 10 days, even under-run by 10

How close to schedule does project finish?
-40 days

©Srinivas Devadas/John Guttag Spring 2002 Slide 16

 **Estimating With One's Heart**

Desires of client
Can dictate scheduled completion date
Cannot dictate the actual completion date

Don't let client push you into an unrealistic plan
Have courage to trust pessimistic estimates
Not an easy thing
“Madame No” sometimes gets fired

Evaluate your team honestly
Remember productivity differs greatly

©Srinivas Devadas/John Guttag Spring 2002 Slide 17