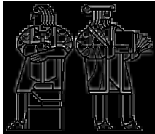


# 6.170 Lecture 19 Prototyping



John Chapin  
MIT EECS  
3-15-2000



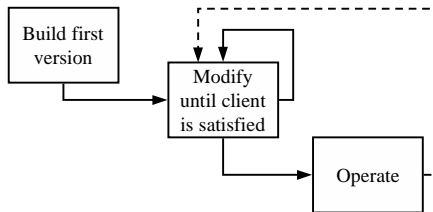
## Outline

### Development models

**Prototyping**  
things to learn  
dangers



## Hacking model



## Hacking model discussion

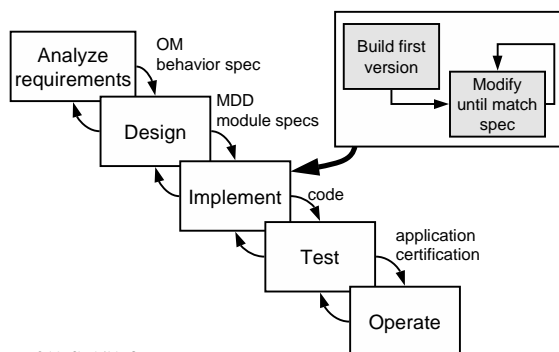
**No specification**  
figure out specification as you code

**Used for:**  
scripts  
short programs

**If applied to something too complex**  
Lots of rework  
High costs at a late stage



## Waterfall model



## Waterfall model discussion

**Used for:**  
Implementing WtDigraph and Table in ps5

**Waterfall model works well when:**  
The requirements are high quality and stable  
The developers have previously built similar systems  
The project is not very complex  
[faulk97]

**When used in other situations:**  
Lots of rework  
High late-stage costs

**Because the developers get it wrong the first time.**



### Characteristics of a better model

**Reduce cost of late-stage problems**

adequacy of specification  
performance limitations

**Rapidly build necessary competencies**

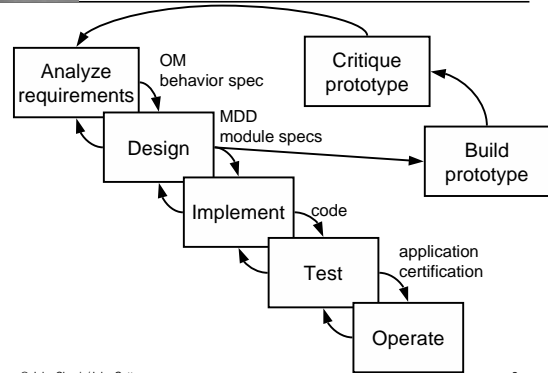
learn OS and language tool details  
use new knowledge to improve design

**Focus on reducing risk**

make (and discover) mistakes early



### Prototyping model



### Discussion of prototyping model

**Not hacking**

carefully design prototypes  
discard prototypes rather than change-until-done

**Lots of wasted work?**

Fred Brooks: Plan to throw one away, you will anyway  
Much cheaper if mistakes discovered early

**There are more sophisticated models**

the "spiral model" adds formal risk analysis



### Two kinds of prototypes

**Prototype as part of requirements analysis:**

learn the customer's needs more precisely  
build a mock-up, demo it, get feedback

**Prototype as part of design:**

implement part of the design, critique it  
find gotchas early on  
discover cleaner structure  
make better feature tradeoffs

**Today we only consider design prototypes**



### Remainder of the lecture

**What you can learn from a prototype about:**

Specifications  
Gotchas  
Dynamic effects

**How to prototype effectively**



### Specs: Learn how to use a module

**Code provided by others may have specs that are:**

ambiguous  
incomplete or incorrect  
so complex you can't be sure you have satisfied the preconditions

**Have to "try out" the desired operations**

**Examples**

activating & using COM objects under Windows  
controlling a UNIX SVR4 STREAMS module



### Specs: Learn how to implement a spec

**Specifications you are asked to implement may be**  
ambiguous  
incomplete or incorrect  
too complex

**must implement correctly anyway**

**"correct"**

system behaves as desired when your  
implementation is composed with existing clients

**examples:**

redrawing efficiently when animating under Swing  
implement an internet router



### Gotchas: Learn subtleties of a spec

**Hard problems may lurk inside innocuous specs**

**Examples:**

Friend says "meet me in the infinite corridor at 11 AM"

In Java try-catch-finally, finally clause always runs  
what about when a thread exits due to an  
uncaught exception?



### Gotchas: Learn cases missed in a design

**Easy to miss cases when doing design**

what should you take in your backpack on a long  
hiking trip?

Air traffic control:

airplane must be deleted after it has landed  
no LANDED message from radar  
system must guess  
location over runway, no radar data -> landed  
what about aborted landings? oops



### Dynamic effects: performance

**Can't analyze performance without a working system**

**Perf. problems may require major design changes**

**Build prototype to study:**

Which paths are the common case?  
Which data structures will have many entries?  
How fast are other system components (eg. Swing)

**Example:**

Can you redraw everything on screen each time the  
ball moves in your gizmoball game?



### How to prototype effectively

**A prototype is built to answer questions**

So, be clear what questions you are trying to answer!

**Write down the list of questions in advance**

**Use the list to decide:**

what functionality to implement  
what to test  
when to discard and restart



### Keep a lab notebook

**Prototype answers other questions too**

**Every decision you make gives valuable information**

what design options did you consider?  
why did you choose the options you did?  
what did you try then change because it didn't work?

**Keep a thorough lab notebook**

never erase/throw out notes (use a bound book)  
keep sequential

**These records improve your ability to:**

critique prototype  
get the final design right  
justify decisions in the final design



## Pitfalls

### How to go wrong when prototyping

- worthless prototype
- pressure to reuse code
- second system effect



## Pitfall #1: worthless prototype

### Waste of time scenario:

- Day 1: receive assignment, read quickly
- Day 2-7: have fun hacking prototype
- Day 8-10: think carefully about spec and design
  - realize that prototype doesn't fulfill spec
- Day 11: start over

### To make prototype useful:

- Do best possible design effort first
- Keep list of questions that arise, use to drive prototype



## Pitfall #2: pressure to reuse code

### Disaster scenario:

- one week to final deadline
- prototype works "sort of"
- no time to discard and restart,
  - let's keep working on this version

### Avoiding disaster:

- keep prototype work focused on the list of questions
- set a milestone for finishing prototype phase
- leave enough time to redesign and implement from scratch



## Pitfall #3: second system effect

### Disaster scenario

- prototype done carefully, finished early
- that was easy!
- let's add a few neat features in the redesign
- oops, now too complex, hard to get even basic features working right

### Huge difference in effort to get it 90% right vs 100% right

### Avoiding disaster:

- identify opportunities for extensions when doing proto.
- leave hooks in the redesign to add the extensions
- but get the basic functionality completely done first



## Summary

### Prototyping

- a development model
- focuses on reducing risk

### Effective prototyping

- write down a list of questions
- set a milestone date to finish prototyping phase