

6.170 – laboratory in software engineering

lecture 19 – march 30, 1999

Prototyping

John Chapin

Lab for Computer Science

MIT

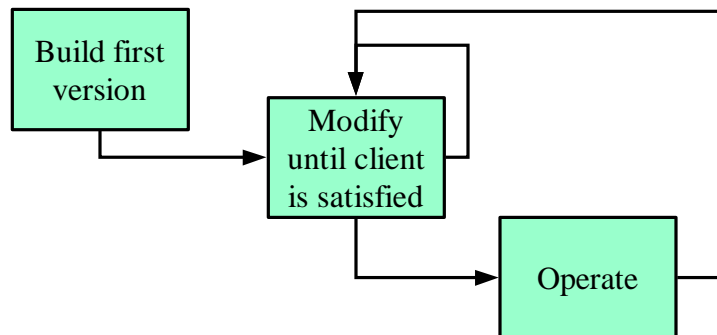
jchapin • 30-Mar-99 • 1

Outline

- Development models
- Prototyping
 - things to learn
 - dangers
- Tomorrow: sit with teammates
- For Thursday section: read entire PS5 handout carefully,
think about where the challenges are

jchapin • 30-Mar-99 • 2

Hacking model

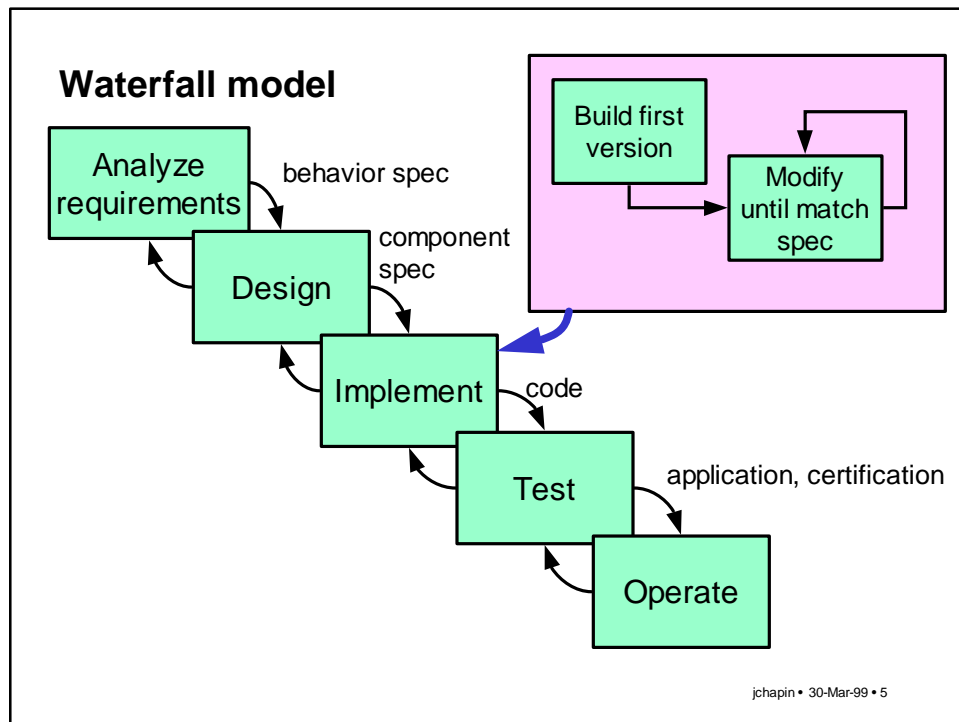


jchapin • 30-Mar-99 • 3

Hacking model discussion

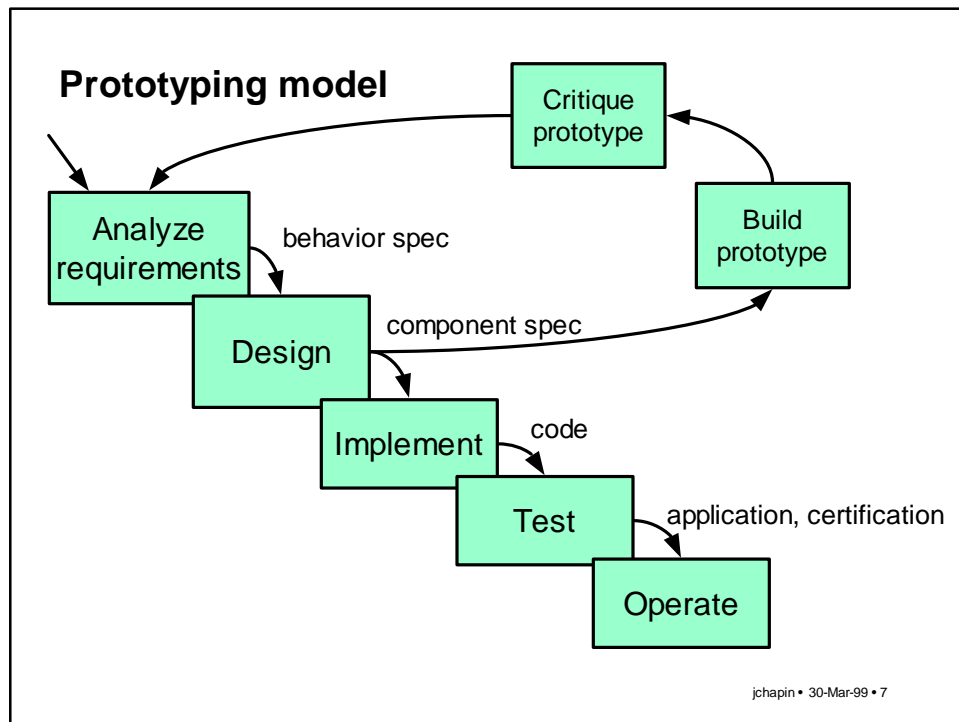
- No spec: figure out spec as you build the system
- Used for: perl script that parsed your midcourse evaluations
- If applied to something too complex:
 - Lots of rework
 - High costs at a late stage

jchapin • 30-Mar-99 • 4



Discussion of waterfall model

- Used for: implementing WtDigraph and Table in ps2
- Waterfall model works well when:
 - A set of high quality, stable user requirements exist
 - The developers have previously built similar systems
 - The project is not very complex» [faulk97]
- Otherwise: lots of rework, high late-stage costs
- Why? The developers get it wrong the first time.



Discussion of prototyping model

- Used for: our implementation of PS5
- Often multiple prototypes
- 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*
 - also, much cheaper if mistakes discovered early
- Note: there are more sophisticated models, notably the "spiral model" which adds risk analysis

jchapin • 30-Mar-99 • 8

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
 - for example, simulate the GUI in Macromind Director
 - to do well, needs usability lab or simulation environment
- Prototype as part of design:
 - implement part of the design, critique it
 - find [gotchas](#) early on
 - discover cleaner [structure](#)
 - make better feature [tradeoffs](#)
 - one developer can do it on their own
- Today we only consider design prototypes

jchapin • 30-Mar-99 • 9

outline of rest of talk

- what you can learn from a prototype about:
 - specs
 - gotchas
 - dynamic effects
- how to prototype effectively

jchapin • 30-Mar-99 • 10

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
- want to reuse their code anyway
- have to "try out" the desired operations
- examples
 - planning courses for a Course VI major
 - activating & using COM objects under Windows
 - controlling a UNIX SVR4 STREAMS module

jchapin • 30-Mar-99 • 11

Learn how to implement a spec

- specs you are asked to implement may be
 - ambiguous
 - incomplete or incorrect
 - too complex
- must implement correctly anyway
- "correct" in this context: system behaves as desired when your implementation is composed with existing clients
- examples:
 - going on a first date
 - redrawing efficiently when animating under Swing
 - implement an internet router

jchapin • 30-Mar-99 • 12

Learn subtleties of a spec

- hard problems may lurk inside innocuous specs
- build prototype to understand specs better
- 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?

jchapin • 30-Mar-99 • 13

Learn cases missed in a design

- easy to miss cases when doing design
- build prototype to learn whether design handles all cases
 - what should you take in your backpack on a long hiking trip? Easy to forget something if you've never hiked before.
 - Air traffic control:
 - airplane is removed from the system after it has landed
 - no LANDED message
 - system must guess: over runway, no radar data
 - what about aborted landings? oops

jchapin • 30-Mar-99 • 14

Dynamic effects: performance

- Can't analyze performance details without a working system
- Fixing 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?

jchapin • 30-Mar-99 • 15

Start with questions

- A prototype is code 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

jchapin • 30-Mar-99 • 16

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

jchapin • 30-Mar-99 • 17

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

jchapin • 30-Mar-99 • 18

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

jchapin • 30-Mar-99 • 19

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 required to get it 100% right vs 90% 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

jchapin • 30-Mar-99 • 20