Project Management and Control

Slides by:
John Guttag
Michael Ernst
MIT EECS

Documentation

- Goal: reduce opportunity for misunderstandings
- Do not rely on oral communication
  - Record decisions in writing
- Not a waste to document things that may change
  - Provide target for criticism
  - Good way to find mistakes early
- When a teammate finds a flaw in your design
  - Triumph for both of you
  - He/she found flaw
  - You explained things well

Documentation, cont.

- Use version control system
  - For documentation as well as code
- Keep change log
  - Date
  - Person
  - Reason
  - Summary
- Verilog/circuit diagrams are most important documentation
  - Property of project
    - Not property of designer
  - Shared coding standard and design patterns help
    - Must be enforced by management

Team Organization

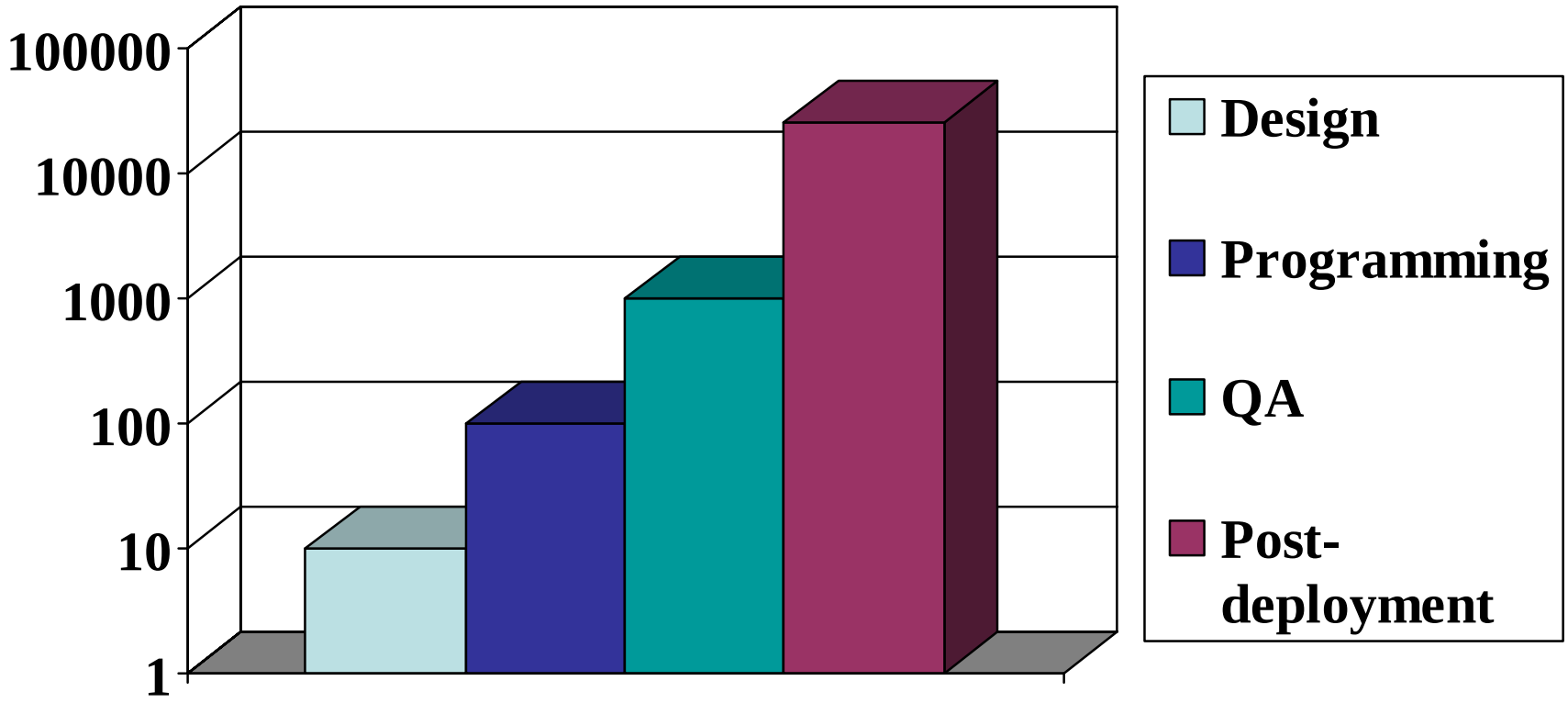
- Most importantly, you need one
- Two distinct considerations
  - How decisions get made (management structure)
  - How information flows (communication structure)
    - Have a plan for this
- Two poles
  - Centralized
    - Needed for large projects
  - Decentralized
    - What I recommend for your project

Decentralized Organization

- Key decisions made jointly
  - Requirements
  - High level design
  - Schedule
  - Who will work on what
  - Response to slippage
- Lower level design exchanged for examination
  - Everyone responsible for everything
  - Design 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

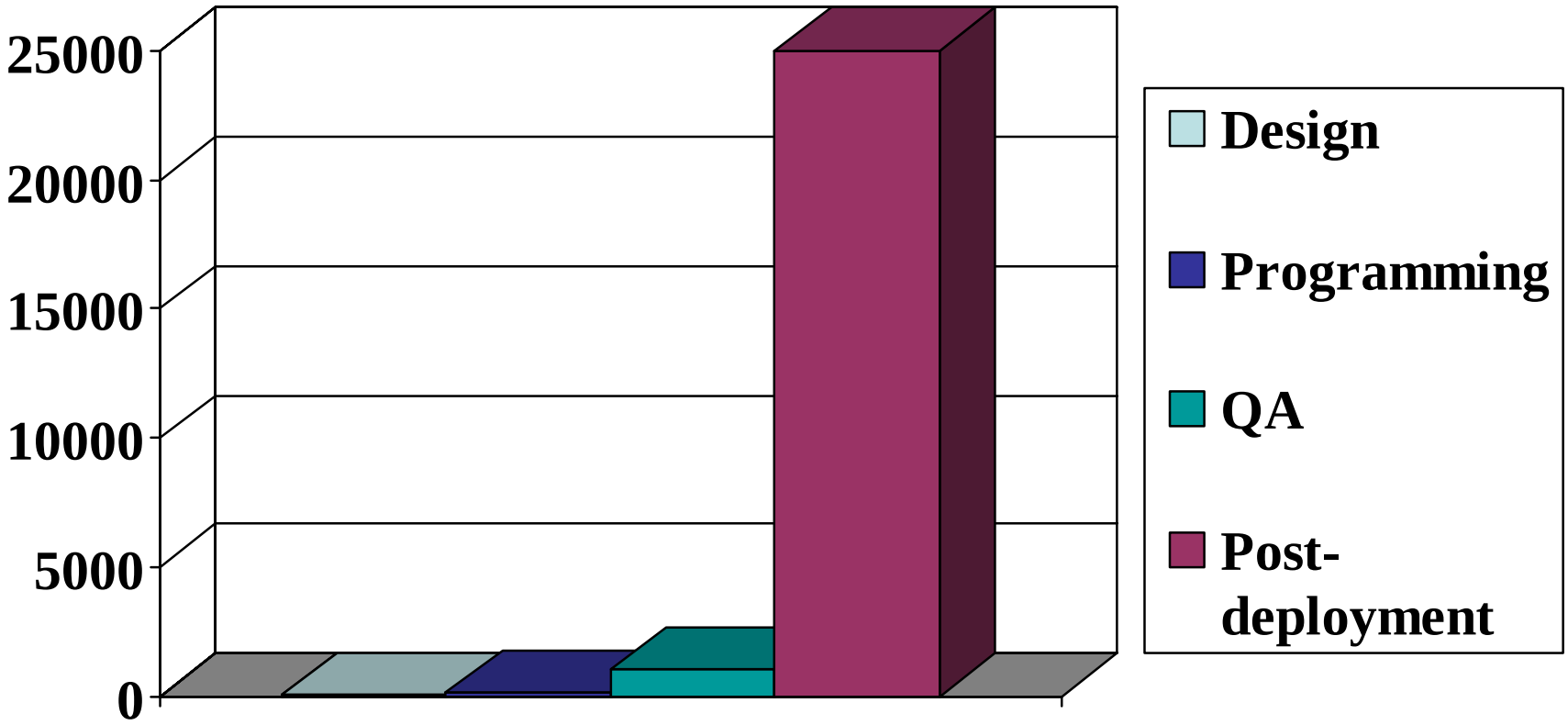
Some Process Models, But First

- Keep in mind cost to fix defects

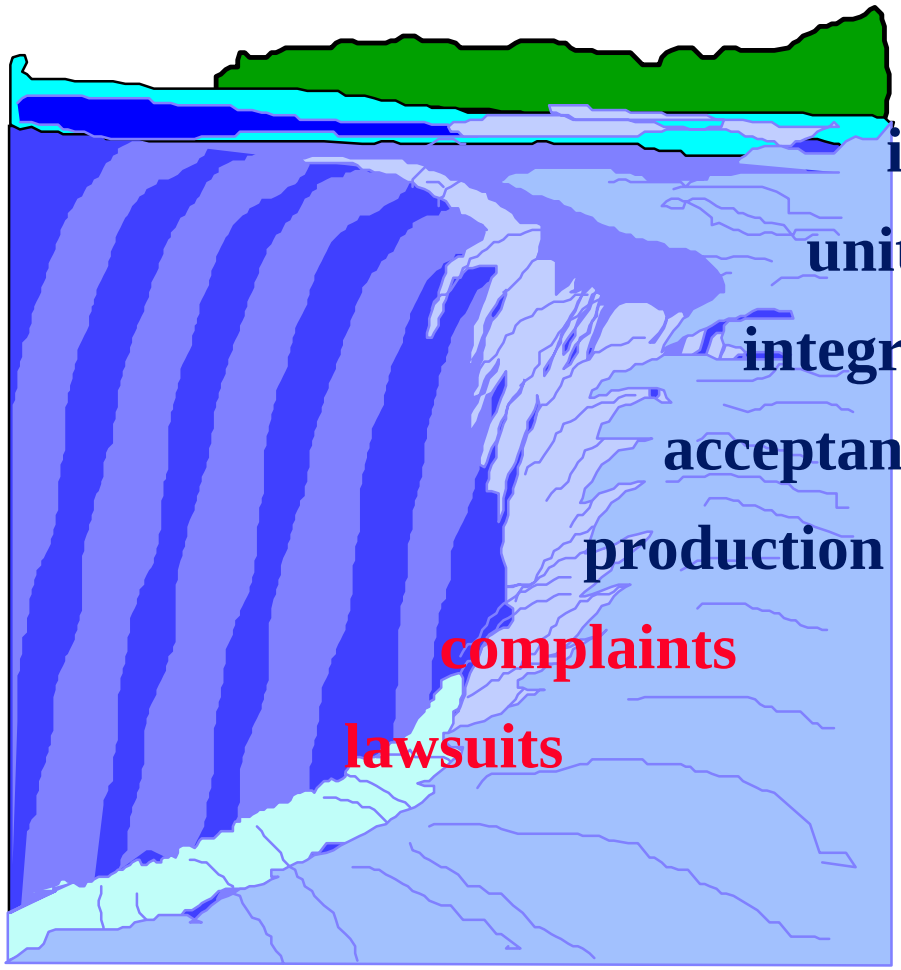


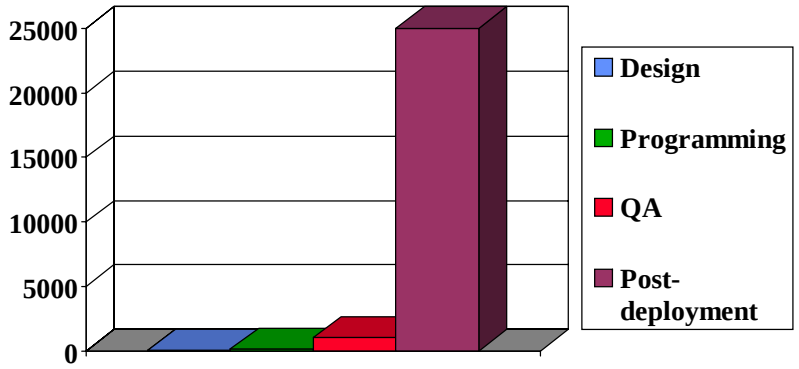
Some Process Models, But First

- Keep in mind cost to fix defects



The Waterfall Model

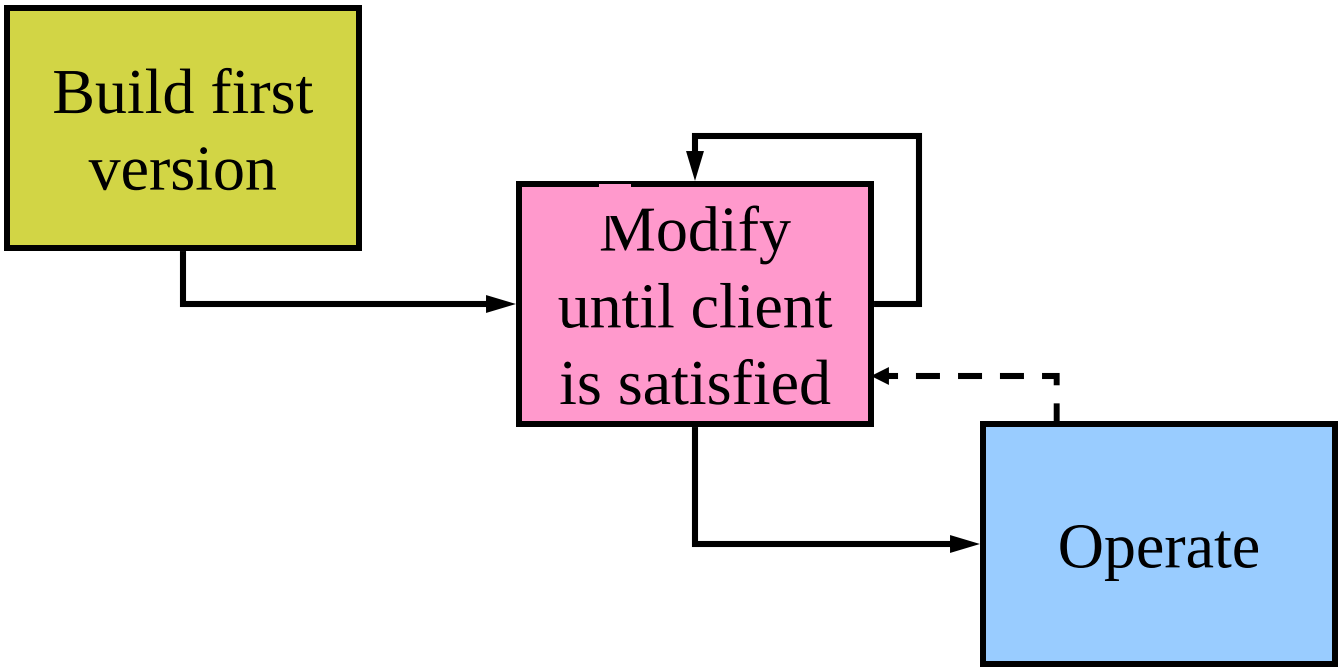




Waterfall Model

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

Hacking Model



Hacking Model Has Problems

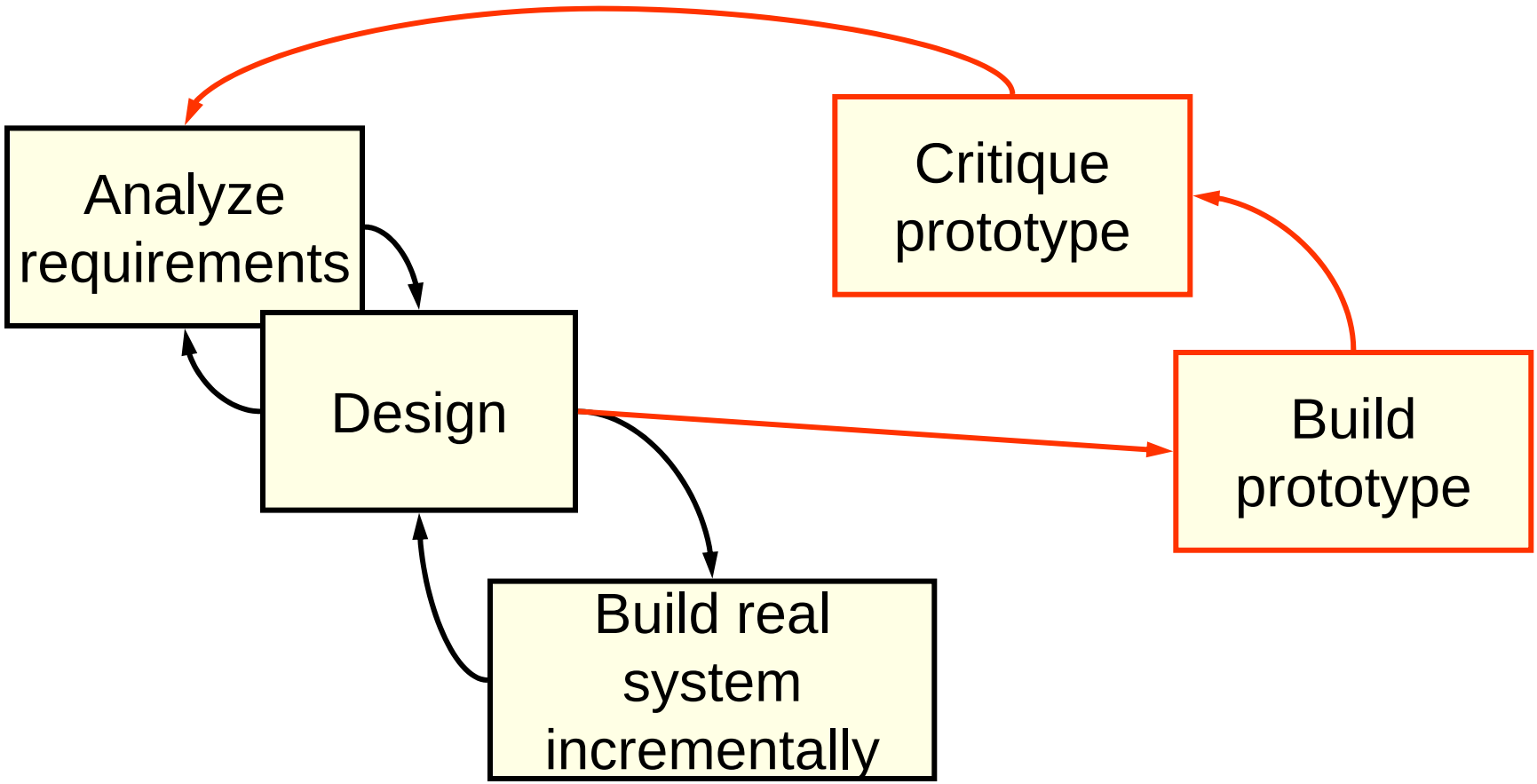
- No specification
  - Figure out specification as you design
- Useful for
  - Small designs
  - Easily accessible client
- If applied to something too complex
  - Lots of rework
  - High costs at a late stage

Characteristics of a Better 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

Prototyping/Incremental Model



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

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 I/O pins
- Understand design
  - Find "gotchas" early on
- Understand building blocks
  - Hardware
  - Programming environment
  - Tool kits and libraries

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
  - Analog checkoff in Lab 3

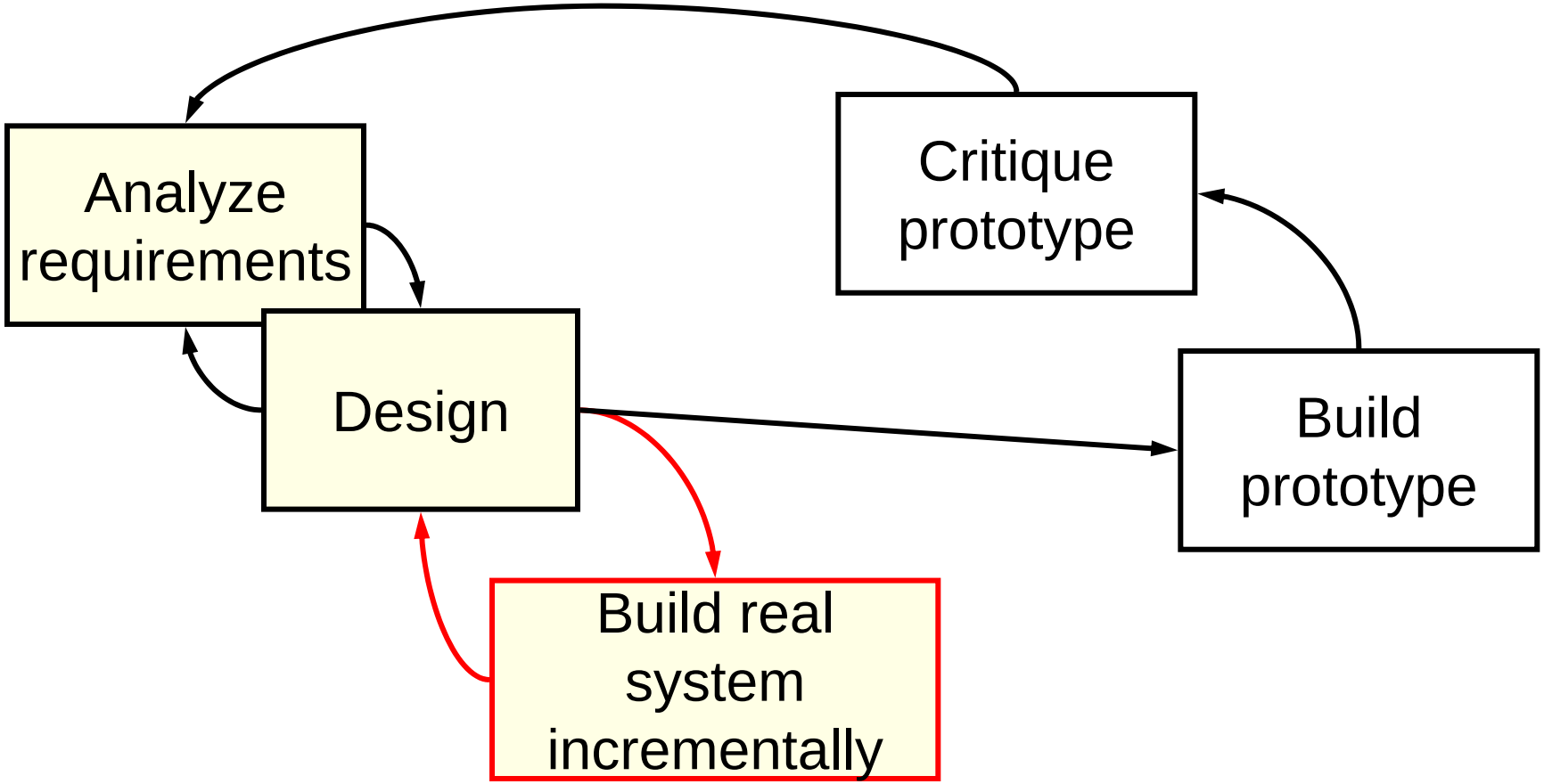
Effective Prototyping

- 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

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**

Prototyping/Incremental Model



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

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

"Bad clients make for bad architecture… Clients get the building they deserve."
-- Frank O. Gehry

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 design 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

Scheduling Is Crucial

- Usually gets far less attention than appropriate
  - Made to fit other constraints
- Needed to make slippage visible
  - Like an quarterly 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

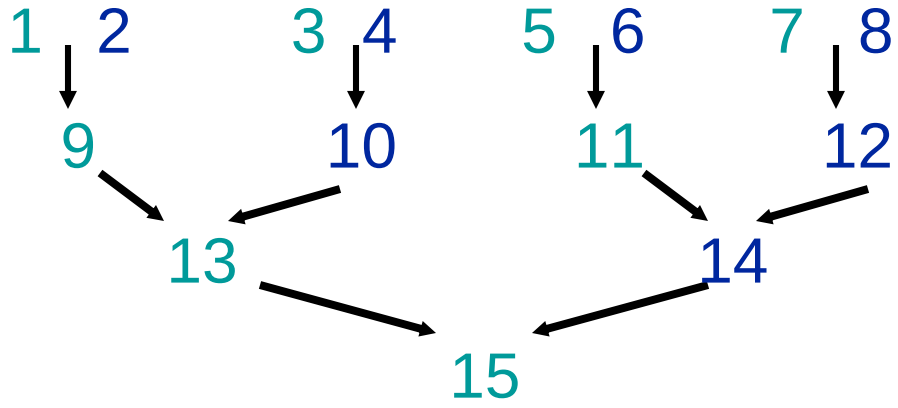
Even if you know that it will take twice as long as you think

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

Optimism is the Root of Problem

- People assume that all will go well
  - Every task will take as long as it ought to take
- Consider following schedule



- 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

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

Effort Is Not the Same as Progress

- Cost is product of workers and time
  - Easy to track
- Progress is more complicated
  - Hard to track
- People don't like to admit lack of progress
  - Think they can catch up before anyone notices
  - Not usually possible
- Design process and architecture to facilitate tracking

How Does a Project Get to Be One Year Late?

- One day at a time
- It's not the hurricanes that get you
- It's the termites
  - Tom missed a meeting
  - Mary's keyboard broke
  - A new release of the CAD tools came out
  - …

How Does a Project Get to Be One Year Late?

- One day at a time
- It's not the hurricanes that get you
- It's the termites
  - Tom missed a meeting
  - Mary's keyboard broke
  - A new release of the CAD tools came out
  - …
- Remember, "It ain't over 'till it's over."
  - If you find yourself ahead of schedule
    - Don't relax
    - Don't add features

Controlling Schedule

- First, you must have one
- Need verifiable milestones
- Some non-verifiable milestones
  - 90% of coding done
  - 90% of debugging done
  - Design complete
- Need 100% events
  - Module 100% coded
  - Unit testing successfully complete
- Need critical path chart
  - Know effects of slippage
  - Know what to work on when

Getting to the End

- Rule of thumb for complex projects
  - 1/3 planning (not all up front)
  - 1/6 coding
  - 1/4 component test and early system test
  - 1/4 system test

When is the project over?

Never?

The project is done when

It is in users' hands

Significant fraction of resources freed up