Software Processes

CS169

What is Software Engineering For?

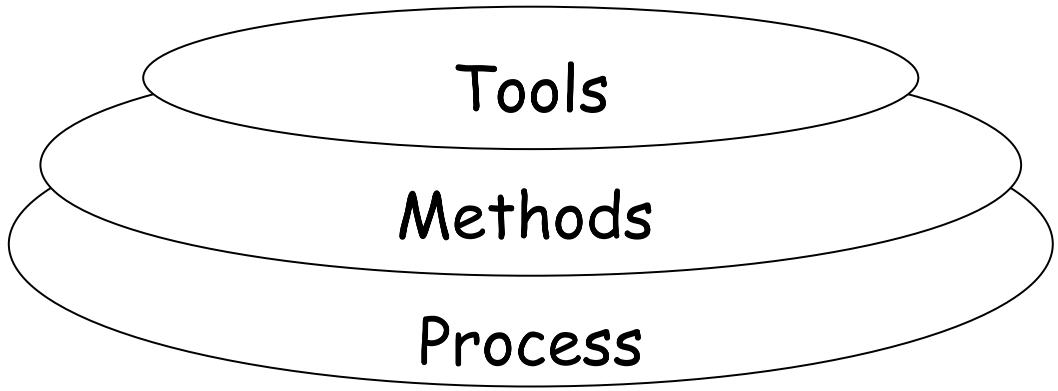
- Solve two problems:
  - How do we know what code to build?
    - How do we know the code works?
  - How do we develop software efficiently?
    - Minimize time
    - Minimize dollars
    - Minimize …

- How do we organize these activities?

Software Process

- Most projects follow recognized stages
  - From inception to completion
- These steps are a "software process"
  - Arrived at by trial and (lots of) error
  - Represent a good deal of accumulated wisdom
- **Process** = **<u>how</u>** things are done
  - In contrast to <u>what</u> is done

Software Engineering Layers

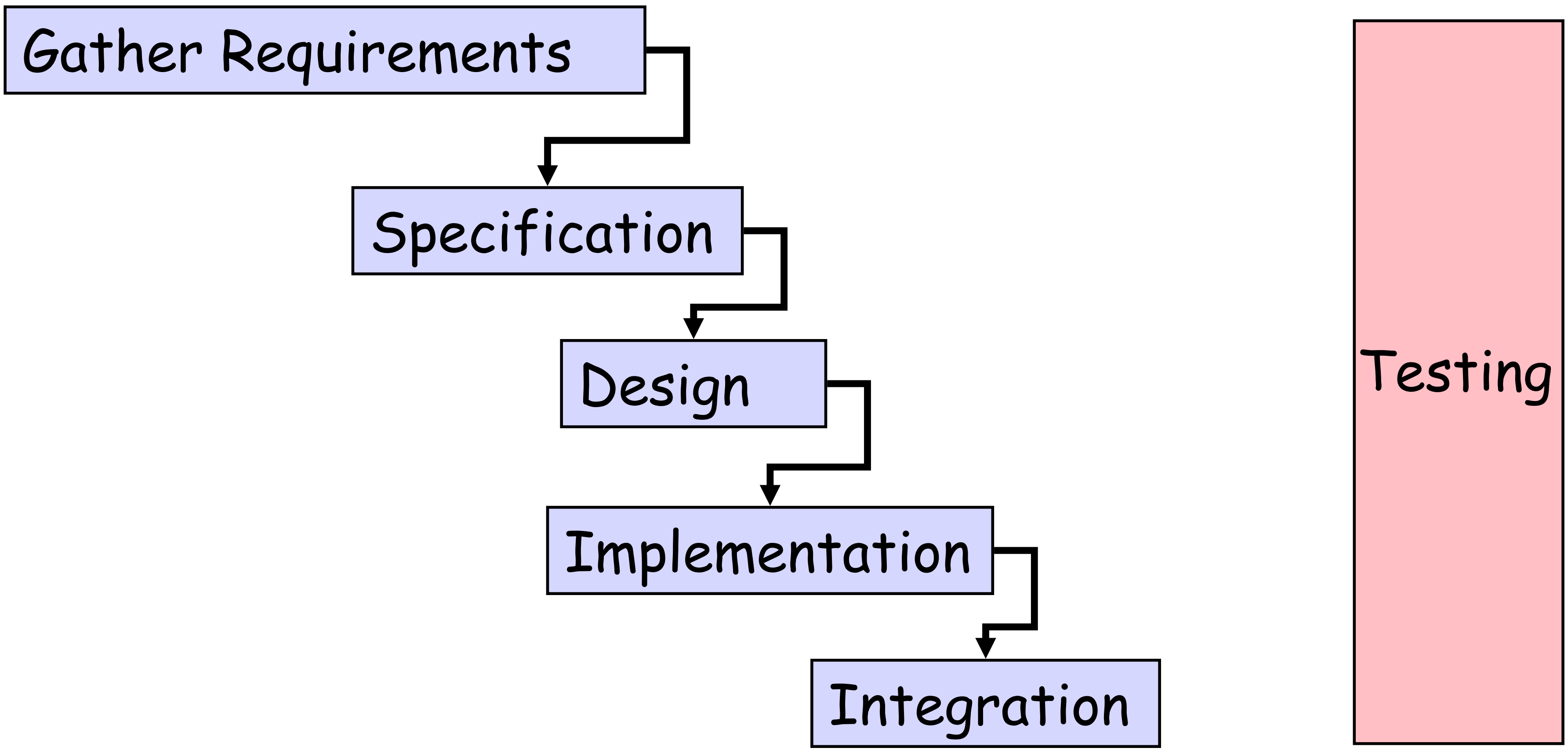


- Process: framework of the required tasks
  - e.g., waterfall, extreme programming
- Methods: technical "how to"
  - e.g., design review, code review, testing,
- Tools: automate processes and methods

Why do we Need a Software Process?

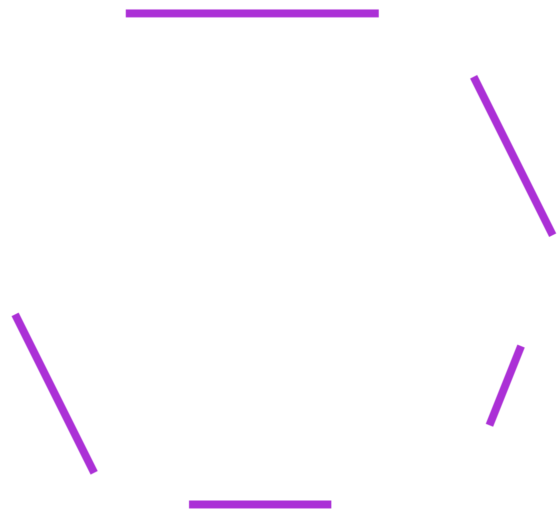
- Consider the ad-hoc process (no-process):
  - Alternate in ad-hoc manner between:
    - Some thinking about what we need to build
    - Some coding
    - Some talking to customers
    - Some testing
- This may work for very small prototypes
- For complex software we learned from past mistakes that it is worth to have a systematic approach (software process)

The Waterfall Model



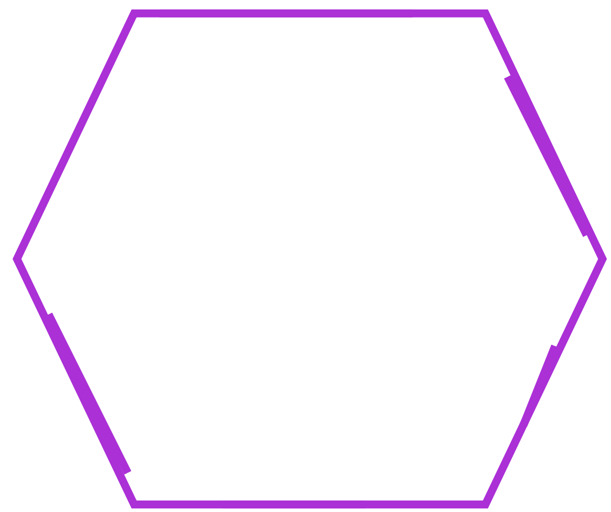
1. Gather Requirements

- Figure out what this thing is supposed to do
  - A raw list of features
  - Written down . . .
- Purpose:
  - Try to ensure we don't build the wrong thing
  - Gather information for planning
- Talk to users, clients, or customers (stakeholders) !
  - But note, they don't always know what they want
  - Sometimes customer != user



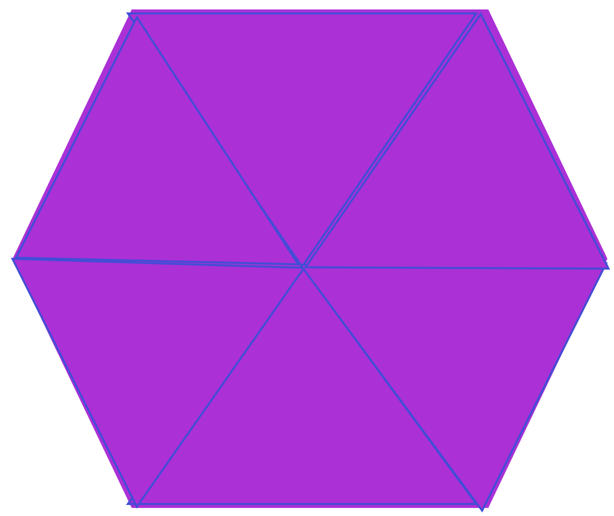
2. Specification

- *A written description of* ***what*** *the system does*
  - In all circumstances
    - For all inputs
    - In each possible state
- A written document
- It covers all situations, much more comprehensive than requirements



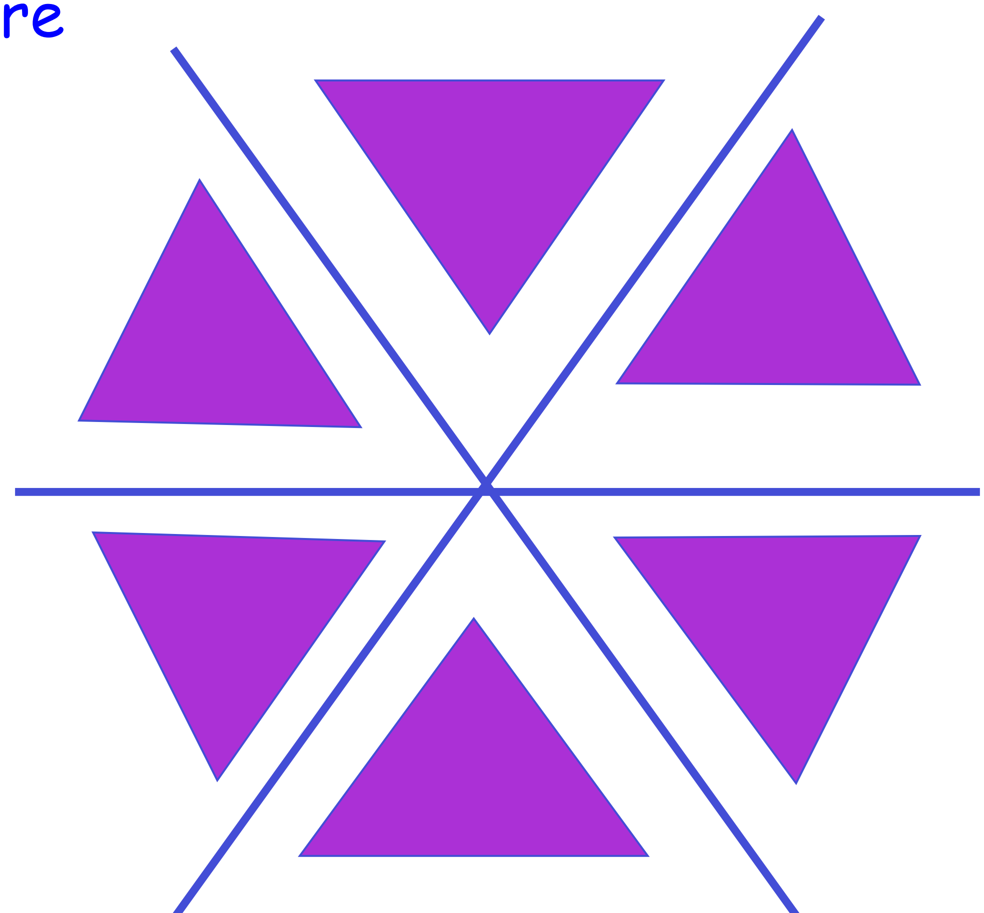
3. Design

- The system architecture
- Decompose system into modules



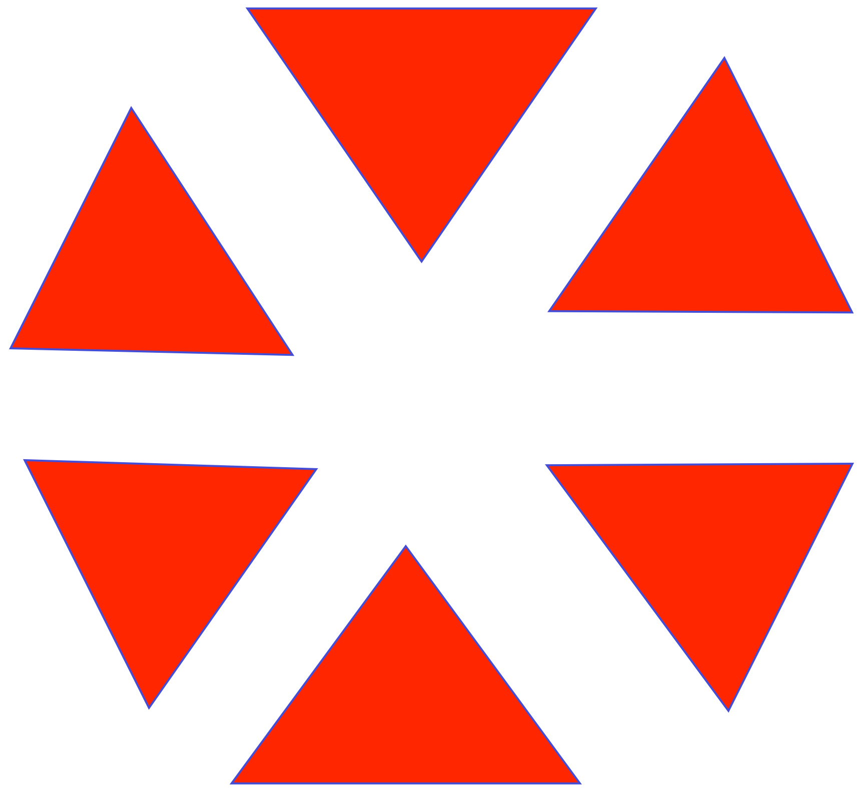
3. Design

- The system architecture
- Decompose system in modules
- Specify interfaces between modules
- Much more of *how* the system works, rather than *what* it does



4. Implementation

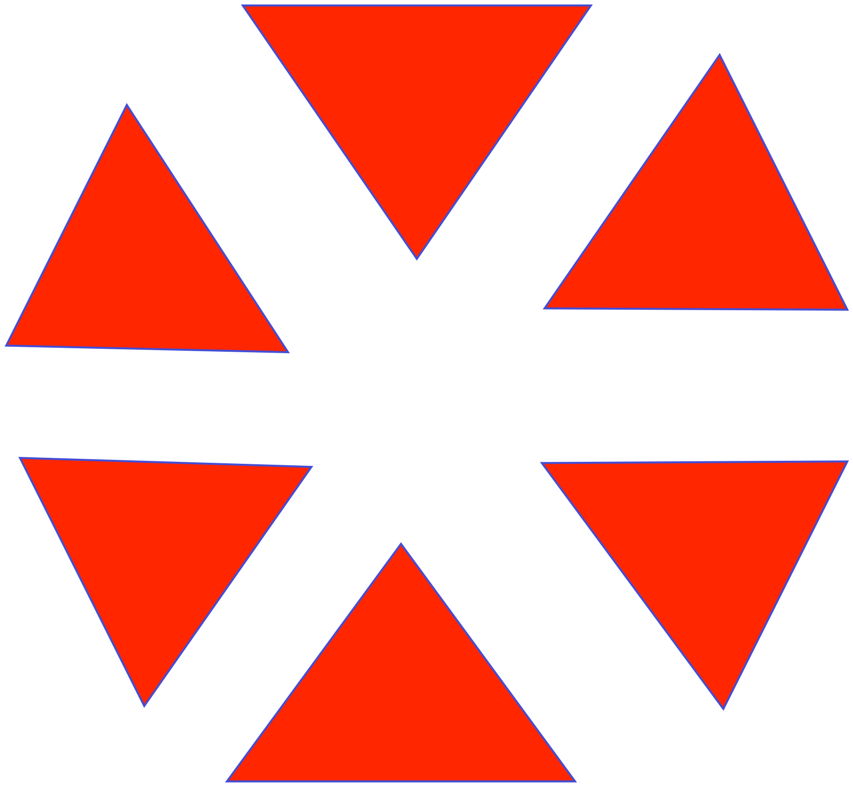
- Code up the design
- First, make a plan
  - The order in which things will be done
  - Usually by priority
  - Also for testability
- Test each module



5. Integration

- Put the pieces together

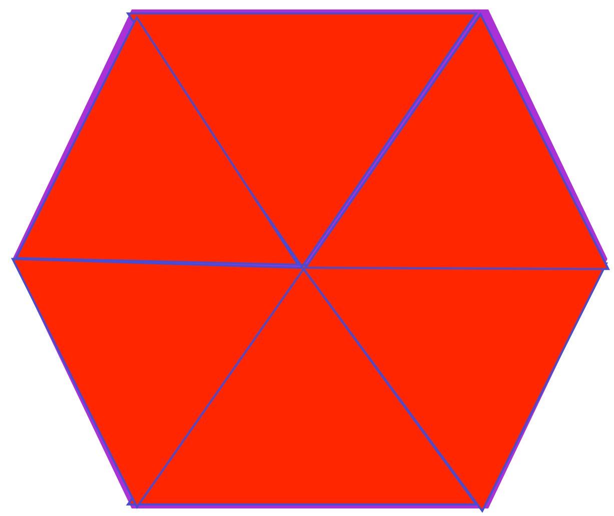
- A major QA effort at this point to test the entire system



5. Integration

- Put the pieces together

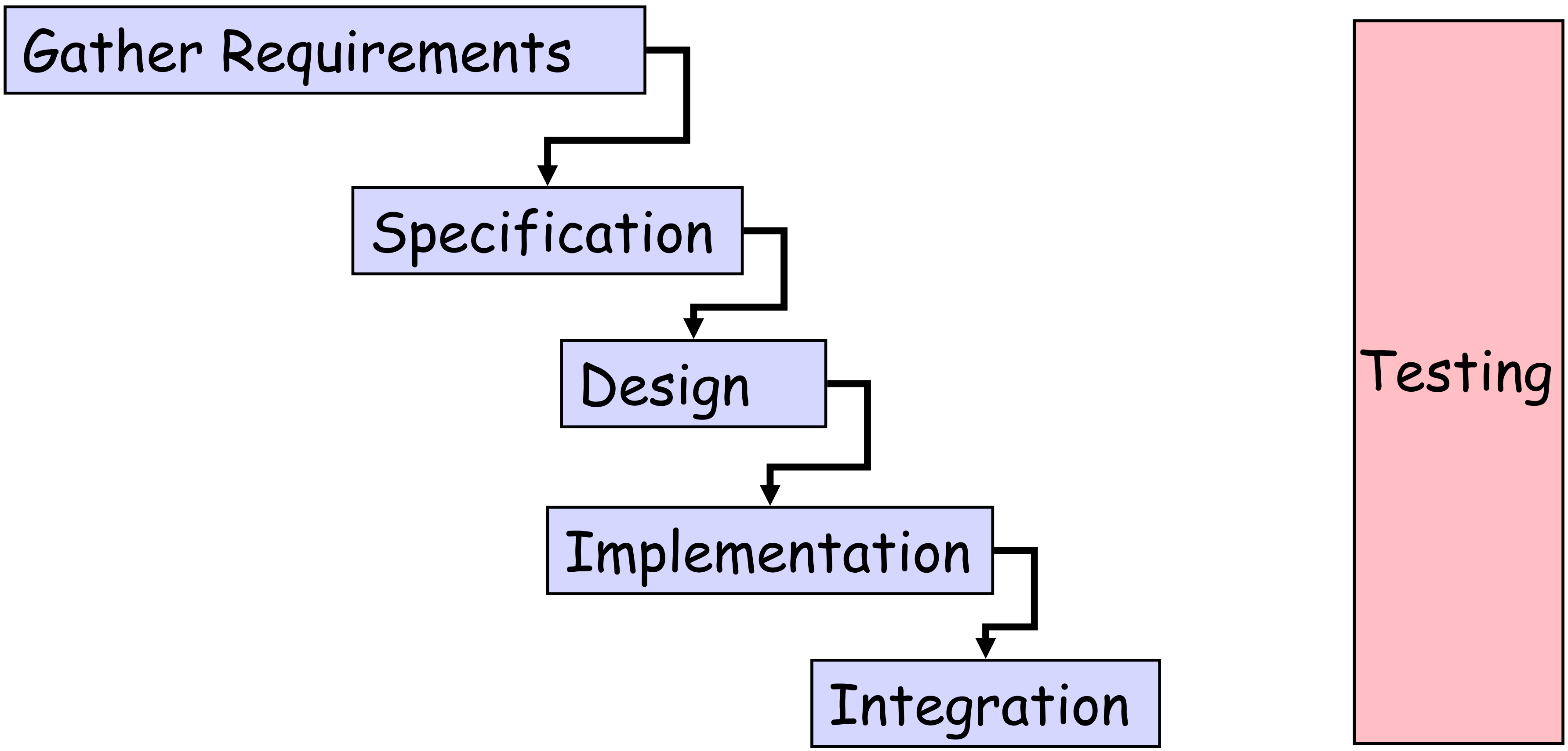
- A major QA effort at this point to test the entire system



A Software Process

- This is called the *waterfall model*
  - one of the standard models for developing software
- Each stage leads on to the next
  - Original model (1970) allowed for feedback between stages

The Waterfall Model



The Waterfall Model (Cont.)

- There is testing after each phase
  - Verify the requirements, the spec, the design
  - Not just the coding and the integration

- Note the outside-in design
  - Requirements, spec, design

- Inside-out implementation
  - Implement, integrate subparts, integrate product

The Waterfall Model (Discussion)

- What are the risks with the waterfall model?

Opinions

- The major risks are:
  - Relies heavily on being able to accurately assess requirements at the start
  - Whole process can take a long time before the first working version is seen
  - Little feedback from users until very late
    - Unless they read and understand specification documents
    - And they know what they want
  - Problems in the specification may be found very late
    - Coding or integration

Opinions

- The waterfall model seems to be adopted from other fields of engineering
  - This is how bridges are built
- But many good aspects
  - Emphasis on spec, design, testing
  - Emphasis on communication through documents
- (I believe) Very little software is truly built using the waterfall process
  - Where is it most, least applicable?

Waterfall Example at NASA

- Space shuttle control software
- Each execution controls $4B equipment, lives, "dreams of a nation"
  - No beta testing
  - 420k lines program, had 17 errors in 11 versions
  - Commercial equivalent would have 5000 bugs
- Secret sauce is the process

Waterfall Example at NASA

- A third of the effort before coding starts
- Specifications are written down and negotiated at length
  - Change to add GPS support (1% of code = 7k lines)
    - Spec for the change is 2500 pages !
  - Total spec is 40,000 pages
- Spec is almost pseudo-code
  - Very little flexibility once the spec is set

Waterfall Example at NASA

- When you find a mistake, don't just fix the mistake, fix what allowed the mistake in the first place
  - Unclear API
  - Insufficient tests
  - Improper use of tools
- Validation and review at all levels
  - 85% of bugs found before formal testing begins
- Process relies heavily on two databases:
  - Revision history
  - Bug database

Waterfall Example at NASA

- Flip-side:
  - 420,000 lines program maintained by 260 people at $32 million cost a year
    - That is $8/line of code/year
- Such a process is too expensive for many software products
  - Perhaps overkill too
- But how to reach right compromise …

An Opinion on Time

- Time is the enemy of all software projects
- Taking a long time is inherently risky

"It is hard to make predictions,
especially about the future"

- Why is time so important ?

Why Time is Important?

- The world changes, sometimes quickly
- Other people produce competitive software
- Technologies become obsolete
  - Some products are obsolete before they first ship!
- Software usually depends on many 3rd-party pieces
  - Compilers, networking libraries, operating systems, etc.
  - All of these are in constant motion
  - Moving slowly means spending lots of energy keeping up with these changes

The Flip Side: Advantages to Being Fast

- In the short-term, we can assume the world will not change
  - At least not much
- Being fast greatly simplifies planning
  - Near-term predictions are much more reliable
- Unfortunately, the waterfall model does not lend itself to speed . . .

Iterative Models: Plan for Change

- Use the same stages as the waterfall model
- But plan to iterate the whole cycle several times
  - Each cycle is a “build”
  - Smaller, lighter-weight than entire product
- Break the project into a series of builds which lead from a skeletal prototype to a finished product

Iterative Process

- Gather requirements
  - As before, but don't spend too much time
  - Realize that there are diminishing returns
  - Without something to show probably can't get full requirements
- Specification:
  - Still important
  - Recognize it will evolve
  - Think about what areas are more likely to change?

Iterative Process (cont.)

- Design:
  - Design for expected change
  - Put abstraction in places where you expect change
- Implementation:
  - Critical pieces first
  - Can leave some parts unimplemented
- Iterate:
  - Show to customer the prototype
  - Update the requirements

Advantages

- Find problems sooner
  - Get early feedback from users
- A prototype is useful in refining requirements
  - Much more realistic to show users a system rather than specification documents
- A prototype exposes design mistakes
- Experience building a prototype will improve greatly the accuracy of plans
  - When build 3 of 4 is done, product is 75% complete

Disadvantages

- Main risk is making a major mistake in requirements, spec, or design
  - Because we don't invest as much time before build 1
  - Begin coding before problem is fully understood

- Trade this off against the risks of being slow
  - Often better to get something working and get feedback on that, rather than study problem in the abstract for too long

In Practice

- Most consumer software development uses the iterative model
  - Daily builds
  - System is *always* working
  - Weekly deployments

- Many systems that are hard to test use something more like a waterfall model
  - E.g., unmanned space probes

Conclusions

- Important to follow a good process
- Waterfall
  - top-down design, bottom-up implementation
  - Lots of upfront thinking, but slow, hard to iterate
- Iterative, or evolutionary processes
  - Build a prototype quickly, then evolve it
  - Postpone some of the thinking
- Extreme programming, next …