Software Process

CS169
Lecture 2

(again based on G. Necula slides)

Administrivia

• Added 20 more so far...
  - Will limit enrollment to ~65 students
• Only one TA so far...
• Start thinking about project proposal
  - Bonus points for proposals that will be popular!

Review: What is Software Engineering For?

• Solve two problems:
  - How do we know the code works?
  - How do we develop software efficiently?
    • Minimize time
    • Minimize dollars
    • Minimize ...
• To do either we need specifications

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 = how things are done
  - In contrast to what 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

Waterfall Process Phases

Gather Requirements
  Specification
  Design
  Implementation
  Integration
  Product
1. Gather Requirements

- Figure out what this thing is supposed to do
  - A raw list of features
  - Written down...

- Usually a good idea to talk to users, clients, or customers!
  - But note, they don’t always know what they want!

- Purpose: Make sure we don’t build the wrong thing

2. Specification

- A written description of what the system does
  - In all circumstances
    - For all inputs
    - In each possible state

- A written document

- Because it covers all situations, much more comprehensive than requirements

3. Design

- The system architecture

- Decompose system into 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

6. Product
- Ship and be happy
- Actually, start maintenance

A Software Process
- This is called the waterfall model
  - one of the standard models for developing software
- Each stage leads on to the next
  - No iteration or feedback between stages

The Waterfall Model
- Gather Requirements
- Specification
- Design
- Implementation
- Integration
- Product

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 top down design
  - Requirements, spec, design
- Bottom up implementation
  - Implement, integrate, product

The Waterfall Model (Discussion)
- What are the risks with the waterfall model?
My Opinions

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

My Opinions

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

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”
- Yogi Berra

Why?

- The world changes, sometimes quickly
- Technologies become obsolete
  - Many products obsolete before they first ship!
- Other people produce competitive software
- 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

A Case Study

- California DMV software (’87 ’93)
- Attempt to merge driver & vehicle registration systems
  - Thought to take 6 years and $8 million
- Spent 7 years and $50 million before pulling the plug
  - Costs 6.5x initial estimate & expected delivery slipped to 1998 (or 11 years)

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 . . .
Something Faster: Rapid Prototyping

- Write a quick prototype
- Show it to users
  - Use to refine requirements
- Then proceed as in waterfall model
  - Throw away the prototype
  - Do spec, design, coding, integration, etc.

Comments on Rapid Prototyping

- Hard to throw away the prototype
  - Slogan "the prototype is the product"
  - Happens more often than you might think!
- 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

Opinions on Reality

- Neither of these models is true to life
- In reality, feedback between all stages
  - Specifications will demand refined requirements
  - Design can affect the specification
  - Coding problems can affect the design
  - Final product may lead to changes in requirements
    - I.e., the initial requirements weren’t right!
- Waterfall model with "feedback loops"

What to Do?

- Accept that later stages may force changes in earlier decisions
- And plan for it
- The key: Minimize the risk
  - Recognize which decisions may need to be revised
  - Plan to get confirmation/refutation as soon as possible

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

Gather Requirements

- Same idea as before
- Talk to users, find out what is needed
- But recognize diminishing returns
- Without something to show, probably can’t get full picture of requirements on the first iteration
**Specification**

- A written description of what the system does
  - In all circumstances
    - For all inputs
    - In each possible state
- Still need this
  - Worth significant time
- Recognize it will evolve
  - Be aware of what aspects are under-specified

**Design**

- Decompose system into modules and specify interfaces
  - Design for change
    - Which parts are most likely to change?
      - Put abstraction there
- Plan incremental development of each module
  - From skeletal component to full functionality
- From most critical to least critical features

**Implementation: Build 1**

- Get a skeletal system working
- All the pieces are there, but none of them do very much
- But the interfaces are implemented
- This allows
  - A complete system to be built
  - Development of individual components to rely on all interfaces of other components

**Implementation: Subsequent Builds**

- After build 1, always have a demo to show
  - To customers
  - To the team
  - Communication
- Each build adds more functionality
Integration

- Integration and major test for each build
- Stabilization point
- Continues until last build
  - But may begin shipping earlier builds

Advantages

- Find problems sooner
  - Get early feedback from users
  - Get early feedback on whether spec/design are feasible
- More quantifiable than waterfall
  - When build 3 of 4 is done, product is 75% complete
  - What percentage have we completed at the implementation stage of the waterfall model?

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

In Practice

- Most consumer software development uses the iterative model
  - Daily builds
  - System is always working
  - Microsoft is a well-known example
- 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