

CS162
Operating Systems and
Systems Programming
Lecture 6

Programming Techniques and Teams

February 6, 2012
Anthony D. Joseph and Ion Stoica
<http://inst.eecs.berkeley.edu/~cs162>

Goals for Today

- Tips for Programming in a Project Team
- The Software Process

Note: Some slides and/or pictures in the following are adapted from slides ©2005 Silberschatz, Galvin, and Gagne. Slides courtesy of Anthony D. Joseph, John Kubiawicz, AJ Shankar, George Necula, Alex Aiken, Eric Brewer, Ras Bodik, Ion Stoica, Doug Tygar, and David Wagner.

2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 6.2

The Role of Software Engineering

- Developing software efficiently
 - » Minimize time
 - » Minimize dollars
 - » Minimize ...
- First, we'll go through some tips for working in a team
- Then, we'll talk about more formal processes

2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 6.3

Tips for Programming in a Project Team



- Big projects require more than one person (or long, long time)
 - Big OS: thousands of person-years!
- It's very hard to make software project teams work correctly
 - Doesn't seem to be as true of big construction projects
 - » Empire state building finished in **one** year: staging iron production thousands of miles away
 - » Or the Hoover dam: built towns to hold workers

"You just have to get your synchronization right!"

2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 6.4

Big Projects

- What is a big project?
 - Time/work estimation is hard
 - Programmers are eternal optimists (it will only take two days!)
 - » This is why we bug you about starting the project early
- Can a project be efficiently partitioned?
 - Partitionable task decreases in time as you add people
 - But, if you require communication:
 - » Time reaches a minimum bound
 - » With complex interactions, time increases!
 - Mythical person-month problem:
 - » You estimate how long a project will take
 - » Starts to fall behind, so you add more people
 - » Project takes even more time!



2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 6.5

Techniques for Partitioning Tasks

- Functional
 - Person A implements threads, Person B implements semaphores, Person C implements locks...
 - Problem: Lots of communication across APIs
 - » If B changes the API, A may need to make changes
 - » Story: Large airline company spent \$200 million on a new scheduling and booking system. Two teams “working together.” After two years, went to merge software. Failed! Interfaces had changed (documented, but no one noticed). Result: would cost another \$200 million to fix.
- Task
 - Person A designs, Person B writes code, Person C tests
 - May be difficult to find right balance, but can focus on each person’s strengths (Theory vs systems hacker)
 - Since Debugging is hard, Microsoft has *two* testers for *each* programmer
- Most CS162 project teams are functional, but people have had success with task-based divisions [poll]

2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 6.6

Communication

- More people mean more communication
 - Changes have to be propagated to more people
 - Think about person writing code for most fundamental component of system: everyone depends on them!
- Miscommunication is common
 - “Index starts at 0? I thought you said 1!”
- Who makes decisions? [poll]
 - Individual decisions are fast but trouble
 - Group decisions take time
 - Centralized decisions require a big picture view (someone who can be the “system architect”)
- Often designating someone as the system architect can be a good thing
 - Better not be clueless
 - Better have good people skills
 - Better let other people do work



2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 6.7

Coordination

- More people ⇒ no one can make all meetings!
 - They miss decisions and associated discussion
 - Example from earlier class: one person missed meetings and did something group had rejected
 - Why do we limit groups to 5 people?
 - » You would never be able to schedule meetings otherwise
 - Why do we require 4 people minimum?
 - » You need to experience groups to get ready for real world
- People have different work styles
 - Some people work in the morning, some at night
 - How do you decide when to meet or work together?
- What about project slippage?
 - It will happen, guaranteed!
 - Ex: everyone busy but not talking. One person way behind. No one knew until very end – too late!
- Hard to add people to existing group
 - Members have already figured out how to work together



2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 6.8

How to Make it Work?

- People are human. Get over it.
 - People will make mistakes, miss meetings, miss deadlines, etc. You need to live with it and adapt
 - It is better to anticipate problems than clean up afterwards.
- Document, document, document
 - Why Document?
 - » Expose decisions and communicate to others
 - » Easier to spot mistakes early
 - » Easier to estimate progress
 - What to document?
 - » Everything (but don't overwhelm people or no one will read)
 - Standardize!
 - » One programming format: variable naming conventions, tab indents, etc.
 - » Comments (Requires, effects, modifies)—javadoc?



2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 6.9

Suggested Documents for You to Maintain

- Project objectives: goals, constraints, and priorities
- Specifications: the manual plus performance specs
 - This should be the first document generated and the last one finished
- Meeting notes
 - Document all decisions
 - You can often cut & paste for the design documents
- Schedule: What is your anticipated timing?
 - This document is critical!
- Organizational Chart
 - Who is responsible for what task?



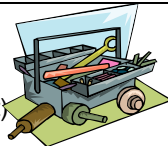
2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 6.10

Use Software Tools

- Source revision control software (CVS, SVN, git)
 - Easy to go back and see history
 - Figure out where and why a bug got introduced
 - Communicates changes to everyone (use RCS's features)
- Use an Integrated Development Environment
 - Structured development model
- Use automated testing tools
 - Write scripts for non-interactive software
 - Use “expect” for interactive software
 - Microsoft rebuilt Vista, W7 kernels every night with the day's changes. Everyone ran/tested the latest software
- Use E-mail and instant messaging consistently to leave a history trail



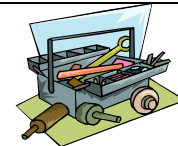
2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 6.11

Integrated Development Environments

- Structured Compile-Edit-Debug environment
 - Organizes top-level projects, folders, and files in a hierarchical structure
 - Makes it easy to find uses of variables, procedures, ...
 - Formats code for easier interaction
 - Interacts with version control infrastructure
- Projects consist of:
 - Files, interdependencies, configurations, version control information, etc.
 - May also manage non-project information:
 - » Global preferences, windows layout, search and navigation history, local change history (like version control, but local changes only)
- Different IDEs support different languages
 - MS Visual Studio (C/C++/C#.NET), IBM Eclipse (Java)



2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012

Lec 6.12

Test Continuously

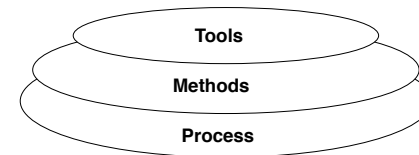
- Integration tests all the time, not at 11pm on due date!
 - Write dummy stubs with simple functionality
 - » Let's people test continuously, but more work
 - Schedule periodic integration tests
 - » Get everyone in the same room, check out code, build, and test.
 - » Don't wait until it is too late!
- Testing types:
 - Unit tests: white-/black-box check each module in isolation (use JUnit?)
 - Daemons: subject code to exceptional cases
 - Random testing: Subject code to random timing changes
- Test early, test later, test again
 - Tendency is to test once and forget; what if something changes in some other part of the code?



2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 6.13

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, etc.
- Tools: automate processes and methods

2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 6.14

The 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
- Process = *how* things are done
 - In contrast to what is done
- Ideal Project (to me)
 - Core functionality is reasonably attainable
 - But extra features are cool and can be implemented as time permits

2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 6.15

Administrivia

- Piazza
 - Use Piazza instead of emailing TA's
 - Avoid asking private questions (we get the same questions over and over)
 - We will work on faster response time for answers
- Project One
 - Mac OS X Lion temporary workarounds: Use instructional machines or supported OS in a VM to cross-compile
 - Testing: Create a plan for *automated* white-/black-box unit testing and integration testing
 - Individual part and group design due Thu 2/9 11:59P
 - Most questions should be directed to the TA's

2/6/12

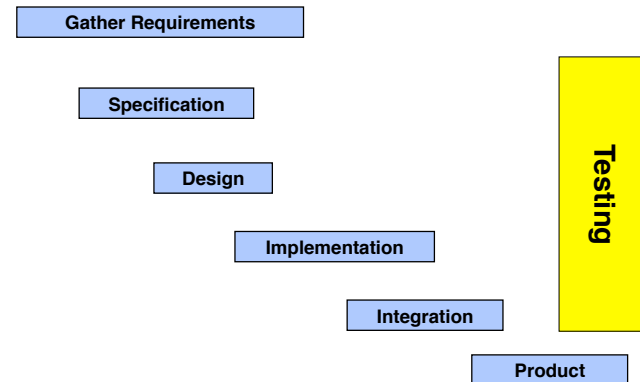
Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 6.16

5min Break

2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 6.17

Waterfall Process Phases

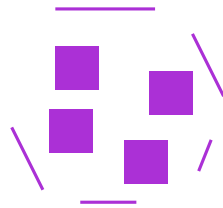


2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 6.18

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

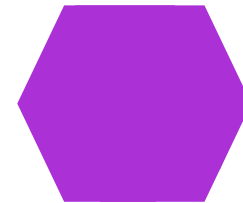


2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 6.19

2. Specification

- A written description of *what* the system does
 - In all circumstances
 - » For all inputs
 - » In each possible state
 - Don't assume correct inputs or states!
- Because it covers all situations, much more comprehensive than requirements

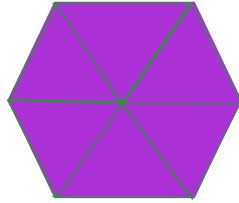


2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 6.20

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

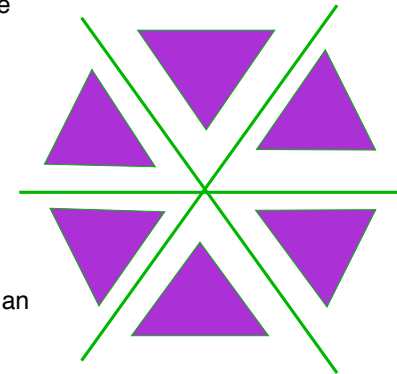


2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 6.21

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

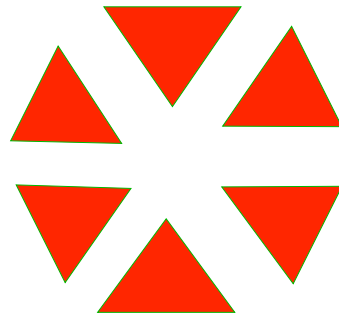


2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 6.22

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

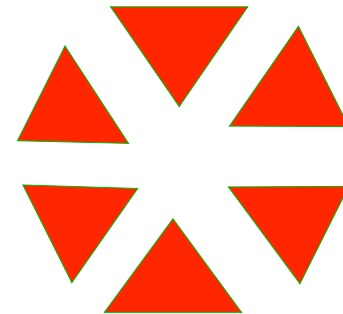


2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 6.23

5. Integration

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

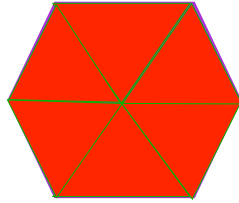


2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 6.24

5. Integration

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

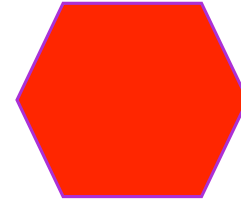


2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 6.25

6. Product

- Ship/Deploy and be happy!
- Actually, start maintenance...



2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 6.26

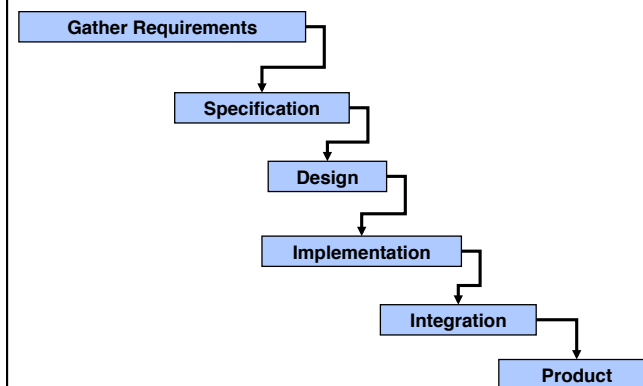
A Software Process

- 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

2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 6.27

The Waterfall Model



2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 6.28

The Waterfall Model (Cont'd)

- 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

2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 6.29

The Waterfall Model (Discussion)

- What are the risks with the waterfall model?
- The major risks are (my opinions):
 - 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

2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 6.30

My Opinions

- The waterfall model seems to be adopted from other fields of engineering
 - This is how to build bridges
- Not much 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

2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 6.31

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

2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 6.32

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

2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 6.33

Case Study #1

- California DMV software (1987-1993)
- 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 and expected delivery slipped to 1998 (or 11 years)!

2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 6.34

Case Study #2

- FBI Virtual Case File system (2000-2003)
 - Trilogy project: thought to take 3 years and \$380 million (including PC and networking upgrades)
- Replace FBI's Automated Case Support (ACS) software
 - Developed in-house by the bureau, considered obsolete when deployed in 1995...
- In 2002, Congress granted Trilogy another \$123 million
 - In 2004, contractor requests another \$50m, FBI pays contractor \$16m to salvage system and another \$2m to perform external review
- In 2005, FBI scraps project
 - Continues to use "obsolete" ACS...

2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 6.35

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...

2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 6.36

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.

2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 6.37

Fundamental Assumption

- We do not know much about the final product
 - No matter what we think
 - Environment will change
 - Requirements will change
 - Tools will change
 - Design will change
 - Better to roll with the punches than go for the KO
 - » (A terrible analogy)

2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 6.38

Comments on Rapid Prototyping

- Hard to throw away the prototype
 - Slogan “the prototype is the product”
 - Happens more often than you might think!
 - Best way to avoid: write prototype in another language
- But prototyping is so useful
 - Much more realistic to show users a system rather than specification documents

2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 6.39

More Comments

- A prototype exposes design mistakes
 - “Man, this is a pain in the butt to code up, even in the prototype”
- Easy to do with web technologies
 - Scripting languages are flexible
 - Browsers are forgiving
 - Much of the glue (CGI, etc.) is already there
 - Ruby on Rails makes everything easy
 - » Even using a DBMS

2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 6.40

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 were incorrect!
- Waterfall model with “feedback loops”
 - Iterative model

2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 6.41

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

2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 6.42

Iterative Models: Plan for Change

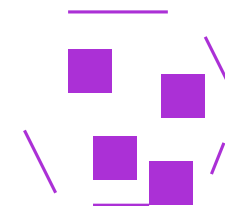
- Use the same stages as the waterfall model
- But plan to iterate the whole cycle several times
 - Each cycle is a “build”
 - Smaller, lighter-weight than entire product
- Break the project into a series of builds which lead from a skeletal prototype to a finished product
- This is the model we use in Berkeley research projects!
 - Also used by Microsoft (internally), Google, Facebook, Twitter, and many others

2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 6.43

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

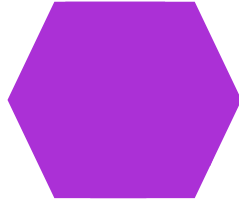


2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 6.44

Specification

- A written description of *what* the system does
 - In all circumstances
 - » For all inputs
 - » In each possible state
 - Don't assume correct inputs or states!
- Still need this
 - Worth significant time
- Recognize it will evolve
 - Be aware of what aspects are under-specified

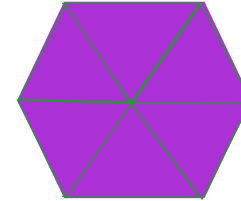


2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 6.45

Design

- Decompose system into modules and specify interfaces
- Design for change
- Which parts are most likely to change?
 - Put abstraction there

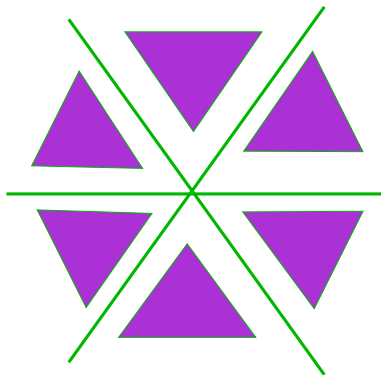


2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 6.46

Design

- Decompose system into modules and specify interfaces
- Design for change
- Which parts are most likely to change?
 - Put abstraction there

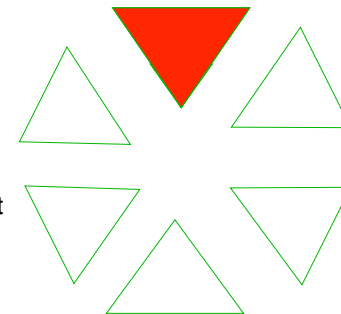


2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 6.47

Design

- Plan incremental development of each module
- From skeletal component to full functionality
- From most critical to least critical features
 - Example: Two engineers at Facebook implemented photo sharing with just one feature – tagging

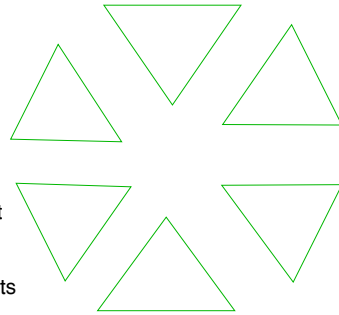


2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 6.48

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

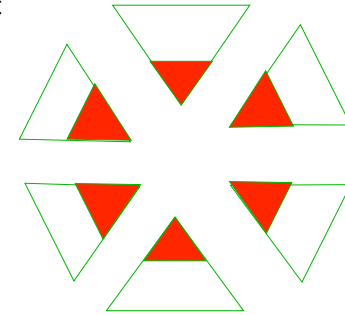


2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 6.49

Implementation: Subsequent Builds

- After build 1, always have a demo to show (or product to deploy)
 - To customers
 - To the team
 - Communication!
- Each build adds more functionality

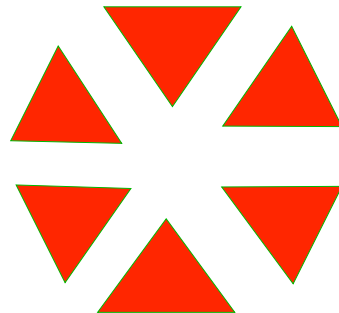


2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 6.50

Integration

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



2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 6.51

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?

2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 6.52

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

2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 6.53

In Practice

- Most consumer software development uses the iterative model
 - Daily builds
 - System is *always* working
 - Always getting feedback
 - Microsoft, Google, Facebook, Twitter are well-known examples
- Many systems that are hard to test use something more like a waterfall model
 - E.g., unmanned space probes

2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 6.54

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 (and ship/deploy it), then evolve it over time
 - Postpone some of the thinking

2/6/12

Anthony D. Joseph and Ion Stoica CS162 ©UCB Spring 2012 Lec 6.55