Announcements

• Lab 3 is due tonight (by 11:59 PM)

• Midterm is on Wednesday October 7th
  – Attendance is REQUIRED for the Midterm
    • If you skip that class, I will not allow you to make it up later

Today’s Topics

• Midterm

• Software Engineering

• Group Activity
Midterm – Wednesday Oct 7th

• Approximately one third of the exam is based on Software Engineering Slides

• Allowed one 8.5x11in cheat sheet with HAND WRITTEN notes

• The rest of the exam is on Objective C and iOS SDK

• Question Formats:
  – Multiple choice
  – True/False
  – Short Answer

Midterm Software Engineering

• Be familiar with (able to list and/or define) the following:
  – Software engineering myths/realities
  – Software Design Models
    • Waterfall
    • Rapid Application
    • Evolutionary
      – Spiral
      – Prototyping
    • Agile Development
      – Extreme Programming
    • Copyright
    • Engineering Ethics
Midterm – Objective C

– Understand the following concepts:
  • Objects
  • Memory Management
  • Model/View/Control
  • View Controllers

– Be able to explain the following commands
  • `synthesize`
  • `- (id) myMethod` vs `+(id) myMethod`

– Syntax covered in class (slides)

Questions on Midterm?
What is Software Engineering?

Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, and the study of these approaches; that is, the application of engineering to software.

“Guide to Software Engineering Body of Knowledge”
– Bourque

Software Engineering ≠ Software Programming

- **Software programming**
  - Single developer
  - “Toy” applications
  - Short lifespan
  - Single or few stakeholders
    - Architect = Developer = Manager = Tester = Customer = User
  - One-of-a-kind systems
  - Built from scratch
  - Minimal maintenance
Software Engineering ≠ Software Programming

• **Software engineering**
  – Teams of developers with multiple roles
  – Complex systems
  – Indefinite lifespan
  – Numerous stakeholders
    • Architect ≠ Developer ≠ Manager ≠ Tester ≠ Customer ≠ User
  – System families
  – Reuse to amortize costs
  – Maintenance accounts for over 60% of overall development costs

Software Engineers’ Myths

• **Once the program is written, I’m done**
  – Between 60-80% of effort expended after delivery
• **Until the program is written, quality is uncertain**
  – Formal design reviews
  – Formal code reviews
  – Test-first approaches
  – Prototyping to verify design and structure
  – Prototyping to validate requirements
• **The only deliverable is the program itself**
  – Lots of documentation: installation guides, usage guides, maintenance guides, API definitions and examples
Software Engineers’ Myths

• **Documentation is Software-Engineering busy work**
  – Focus is on quality, not quantity
  – Documentation can be hard for engineers to write, just as C++ may be difficult for poets.
  – Conserve energy: documented code can serve as a basis for useful documentation
    • JavaDoc
    • Doxygen

• **Literate Programming**
  – You write code for other people to read

Software Engineering “Axioms”

• **Adding developers to a project will likely result in further delays and accumulated costs**

• **Basic tension of software engineering**
  – better, cheaper, faster — pick any two!
  – functionality, scalability, performance — pick any two!

• **The longer a fault exists in software**
  – the more costly it is to detect and correct
  – the less likely it is to be properly corrected

• **Up to 70% of all faults detected in large-scale software projects are introduced in requirements and design**
  – detecting the causes of those faults early may reduce their resulting costs by a factor of 100 or more
Software Design Models

Software Production Process

- **Phases and Actions to Build, Deliver, Evolve Product**
- **Objectives**
  - Construct Programs from Idea to Completion
  - Produce Reliable, Predictable, and Efficient SW
- **Difficult to Automate**
  - Software Production is a Highly Intellectual Activity
  - Interactions of Individuals from Various Backgrounds
  - Interface to OS, Hardware, Databases, etc.
- **Production Models Focus on the Software Lifecycle**
  Emphasizing the Process from Start to Finish
### Motivation

- **Increase in Application Complexity and Requirements**
  - Led to Separation Between Developers and Users
- **Software Now Targets Users without “Computer Expertise”**
  - Higher Level of Quality and Reliability Needed
  - Software Development as Group Activity
- **Software Development Needs to:**
  - Manage Complexity in Modern Applications
  - Provide Detailed Careful Analysis of User Requirements
- **Goals of a Model are:**
  - Determine Appropriate Stages
  - Establish Transition Criteria for Progressing from One Stage to Another

### Different Models

- **Waterfall Model**
- **Rapid Application Model**
- **Evolutionary Models**
  - Spiral
  - Prototyping
- **Agile Model**
### Waterfall Model

- Also called classic life cycle, proposed by Winston Royce in 1970
- Original proposal allowed for feedback and loops
- In practice, strictly linear
- Called a “prescriptive” process model

<table>
<thead>
<tr>
<th>Stage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication</td>
<td>Initiation, requirements gathering</td>
</tr>
<tr>
<td>Planning</td>
<td>Estimating, scheduling, tracking mechanisms</td>
</tr>
<tr>
<td>Modeling</td>
<td>Analysis and design</td>
</tr>
<tr>
<td>Construction</td>
<td>Code and test</td>
</tr>
<tr>
<td>Deployment</td>
<td>Delivery, support, feedback</td>
</tr>
</tbody>
</table>
Waterfall Model – Classic Approach

- Multiple Phases for Development
- Complete One Phase before Next Begins

- Completion of All Phases (thru Delivery) Implies a Valid, Verified Product

Waterfall Model - Evaluation

- Contributions to Understanding Software Processes
  - Software Development Must be Disciplined, Planned, and Managed
  - Implementation Delayed Until Objectives Clearly Understood

- Characteristics of Waterfall Model
  - Linear: From Beginning to End w/o Backtracking
  - Rigid
  - Monolithic: All Planning is Oriented to Single Delivery Date

- What are the Problems with this Process?
### Waterfall Model – Problems

**Problems with Waterfall Model**
- Forces Cost Estimation and Project Planning to Occur After Limited Analysis Performed
- Unrealistic to Assume all Requirements Frozen before Development Starts
  - User’s Often Don’t Know Exact Requirements
  - Particularly Early in the Process
- Does not Stress Anticipating Changes
- Enforces Standards Based on Producing Particular Documents at Specific Times

### RAD–Rapid Application Deployment

- Breaks problem into pieces
- Utilizes concurrent design and construction
- Huge integration exercise at the end
- Alleged 60-90 day time span

---

<table>
<thead>
<tr>
<th>Communication</th>
<th>Planning</th>
<th>Design</th>
<th>Construction</th>
<th>Deployment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Problems with RAD?

- Requires sufficient human resources
- Must commit to rapid development process
  - Vision of design must remain consistent among teams
  - Tends to fade or become chaotic over time
- Requires a project that can be componentized
- Levels of abstraction and insulation between teams can cause performance issues
- Use of cutting-edge technology in one team can sink the whole project if it fails

## Evolutionary Model

- Fred Brooks Advocates Producing a Product Twice
  - First Version is Throwaway to Provide Means for Users to Offer Feedback on Exact Requirements
  - Second Version Developed using Waterfall
- Evolutionary or Incremental Approach
  - Emphasizes Stepwise Development
  - Flexible and Non-Monolithic
  - Postpones Parts of Some Stages
Evolutionary Models

- Examples
  - Spiral
  - Prototyping
- Iterative approaches
- Increasingly more complete versions of the product are generated
- Articulated deliveries can help planning
  - Revising design delivers a more on-target product
  - Revisiting implementation can remedy a bad initial approach

Spiral Model

- Purpose:
  - Provide a Framework for Design Production Process Guided by Risk Levels
- Guiding Principles:
  - Level of Risk (Potential Adverse Circumstance)
- Risk Management:
  - Barry Boehm (created Spiral Model) states: “Discipline whose objectives are to identify, address, and eliminate software risk items before they become either threats to successful software operation or a major source of expensive software rework.”
- Focus on Identifying and Eliminating High Risk Problems by Careful Process Design/Assessment
Spiral Model

- **Cyclical Model is Four Stages:**
  1. Identify Objectives and Design Alternatives
  2. Evaluate Alternatives and Identify/Deal with Potential Risks
  3. Develop and Verify Next Level Product
  4. Review Results and Plan for Next Iteration
- **Allows Unstated Requirements to Become Part of Specification of Next Cycle**
  - Robustness Approximates Correctness More Closely
The Spiral Model

- Determine objectives, alternatives, constraints
- Evaluate alternatives, identify, resolve risks
- Plan next phase
- Communication
- Quick plan
- Quick design
- Construction of prototype
- Deployment, delivery, feedback

Prototyping Model

- Requirements Plan
- Life-cycle plan
- Concept of Operation
- S/W requirements
- Hardware, models, benchmarks
- Develop, verify next-level product
Prototyping Model

- **Useful when**
  - Insufficient requirements exist at start
  - Behavior of some components unknown
    - New or strange OS
    - Hardware “in progress”
    - HCI (Human-Computer Interface) factors not yet firm
    - Algorithmic uncertainties: speed, space

- **Potential Problems or Issues?**
  - Testing may be minimal
  - Not intended for ultimate delivery of longevity
  - Little or no documentation is produced
- **Customer and team must agree on this approach up-front**
- **Expectations should not be overly high on either side**

Problems in Evolutionary Models

- **Potential Problems:**
  - Large Time Gap Between Requirements Specification and Delivery
  - Emphasis on User Interface and Not Product
    - May Miss Functional Requirement
    - May Underestimate DB Complexity/Interactions
Advantages in Evolutionary Models

- Product May Closely Follow User Requirements
- Supports Anticipation of Change
- More Flexible Than Just Waterfall Approach

Agile Programming

- Iterative and evolutionary development
- Timeboxing
  - Set amount of time for iteration
  - Adapt future iteration based on the realities
- Adaptive planning
- Incremental delivery
- More focused on success than sticking with a plan
- Working software is valued and considered measure of progress
Agile Programming

• Learn enough to start
• Work together in the same space
• Express solution in features
• Maintain working system, fully tested, ready to ship
• Deliver features consistently
• Evolve the design as you learn

Extreme Programming?

• 2am, marathon, Cheetos,
• Hacking
• undocumented, brilliant,
• tests?, mysterious,
• Jolt Cola, protective,
• ad-hoc, brittle, cubicle,
• fear change, cowboy,
• overly elegant, code ego,
• real soon now, stress
• difficult to talk to
Extreme Programming!

- 2am, marathon, Cheetos,
- hacking, greasy,
- undocumented, brilliant,
- tests?, mysterious,
- Jolt Cola, protective,
- ad-hoc, brittle, cubicle,
- fear change, cowboy,
- overly elegant, code ego,
- real soon now, stress
- difficult to talk to

8-5, teamwork,
- code reviews,
- communication,
- coding standards,
- snacks, communal code
tests, tests, tests,
maintainable,
documented
predictable,
copes with change

Extreme Programming

- **Code is written in response to a user story**
  - 4x6 card describing requirements

- **Start with the smallest set of features**
  - Release early and often

- **Simple Design**
  - Use simplest possible design that gets job done
Extreme Programming

- Continuous Testing
  - Tests are written before programming
  - When tests are passed, job is done

- Continuous Integration
  - New code is added daily
    - All test must be passed though

- Pair Programming
  - Two programmers at one machine

Pair Programming

- Two programmers share one computer
  - One is the driver
    - Does all the writing of code
  - Other is observer
    - Watches and guides
    - Focuses on strategic issues
      - How this module fits with others
    - Typically the more experienced programmer

- Claim:
  - Pair programming is more productive than having two separate programmers

- NYtimes article:
Group Activity