Software Engineering Process

Kevin Cathey
Where are we going?
<table>
<thead>
<tr>
<th>Last Week</th>
<th>iPhone Application Technologies Workshop</th>
</tr>
</thead>
<tbody>
<tr>
<td>This Week</td>
<td>Software Engineering Process</td>
</tr>
<tr>
<td>Thanksgiving Break</td>
<td>Write some code, yo</td>
</tr>
<tr>
<td>2 Dec</td>
<td>Options: Introduction to MyCampus application architecture or iPhone Graphics</td>
</tr>
<tr>
<td>2 Dec ++</td>
<td>Project work</td>
</tr>
</tbody>
</table>
Software Engineering Process
“Programming today is a race between software engineers striving to build bigger and better idiot-proof programs, and the Universe trying to produce bigger and better idiots. So far, the Universe is winning.”

Rick Cook, The Wizardry Compiled
“Introduce the core concepts of Software Engineering: what it means to work on a team to make a product by an immovable deadline.”

Me on the purpose of today’s talk
Software Engineering
From a High Level
Software Engineering Timeline

Start (what we have)

19 NOV

Goals (the end product)
Milestones

• Milestones denote sub-goals and deadlines when certain kinds or priorities of work need to be completed.

• Example:
  ▪ Milestone 1: Have some cool demos
  ▪ Milestone 2: Be feature complete
  ▪ Milestone 3: Only allow priority one bugs
  ▪ Milestone 4: Release candidate (RC)
  ▪ Milestone 5: Final release (GM — “Golden Master”)

• During the process, you seed beta versions of software.
Software Engineering Timeline

Start (what we have)

Goals (the end product)

NOV 19
Software Engineering Timeline

Now (what we have)
- Milestone 1 (Sake)
  - Maps done
- Milestone 2 (Maguro)
  - Feature complete
- Milestone 3 (Fugu)
  - GM

MyCampus (the end product)
Main Components of Software Engineering
Software Engineering Components

What you will encounter when making software

• Software Configuration Management (SCM)
  ▪ Version Control
  ▪ Automated Building
  ▪ Bug Tracking

• Refactoring

• Testing

• Debugging

• Reverse Engineering
Software Configuration Management

Version Control

• Why?
  ▪ Keep track of versions of software as it changes.
    ▪ What’s broken since revision 11?
  ▪ Keep track of old versions in order to recreate customer issues.
  ▪ Have independent versions that get merged (branches).
  ▪ Backup — Avoid catastrophes.

• Common version control systems
  ▪ Subversion (SVN) *
  ▪ CVS
  ▪ Git
  ▪ Perforce
## Software Configuration Management

### Version Control Terminology

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Repository</strong></td>
<td>Central location of all software configuration items.</td>
</tr>
<tr>
<td><strong>Software Config. Item</strong></td>
<td>Any item that is versionable: source file, folder, image, etc.</td>
</tr>
<tr>
<td><strong>Revision</strong></td>
<td>Numbered snapshot of the repository.</td>
</tr>
<tr>
<td><strong>Working copy</strong></td>
<td>Version of repository running locally on your machine, where you do your work.</td>
</tr>
<tr>
<td><strong>Commit</strong></td>
<td>Putting changes back into the repository from a working copy.</td>
</tr>
<tr>
<td><strong>Trunk</strong></td>
<td>Main line of development.</td>
</tr>
<tr>
<td><strong>Branch</strong></td>
<td>Independent version of some part of trunk (or trunk in its entirety).</td>
</tr>
<tr>
<td><strong>Tag (Submission)</strong></td>
<td>Version of software known to work and released to others for testing (has a “build” number).</td>
</tr>
<tr>
<td><strong>Top of tree</strong></td>
<td>Latest revision (last revision in trunk... top of tree... get it?)</td>
</tr>
<tr>
<td><strong>Merge</strong></td>
<td>Taking changes from one place and welding them into another; a forward change.</td>
</tr>
<tr>
<td><strong>Back-out</strong></td>
<td>A reverse merge; a backward change.</td>
</tr>
</tbody>
</table>
Software Configuration Management

Automated Building

• Periodically (every night, every three days, etc) you build the entire project or set of projects.

• This way, you can catch commits that break the build, and back them out quickly.

• Every morning, receive an email on the building status of the software. If there are problems, you can respond quickly.

• Automated build systems
  ▪ Simple cron job
  ▪ buildbot
Software Configuration Management

Bug Tracking

• Everything is a bug. If the software doesn’t do it now, it's a bug.
• One objective (mostly) way to track progress.
• **Typical bug pieces**: title, description, diagnosis (running dialog of what might be wrong), state, assignee, milestone, priority, classification, originator (duh), modification dates, other things (source changes, dependencies, fix order, personal label, etc).

• Change control
  - **Commit-level**: code reviews. Ideally you wouldn’t commit non-reviewed code to trunk.
  - **High-level**: At certain milestones, only allow certain kinds or priorities of bugs to be committed.

• Change tracking system we’re going to use: Trac.
Trac

• Project management wiki.

• Capabilities
  ▪ View and edit project milestones and details.
  ▪ File and track bugs.
  ▪ Integrate with SVN repository to browse source and see change logs.

• Our Trac wiki: http://acm.uiuc.edu/projects/MacWarriors
Demo
SVN and Trac
Refactoring

A refactoring is a change made to the architecture, design, or general implementation of a piece of software that does not change the visible functionality of an application.

Why you refactor:
- Make things clearer.
- Prepare things for expansion.

When not to refactor: refactoring is not fixing a bug. If it doesn’t work before, refactoring won’t fix it.

This is where design and architecture evolve.

Great books and articles:
- Refactoring by Martin Fowler
- OO Design Quality Metrics by Robert Martin
“If it isn’t tested, it doesn’t work.”

Ralph Johnson, SE professor
Testing

• One of the most critical parts of Software Engineering.
• Tests…
  ▪ find bugs, among many things.
  ▪ prove the existence of bugs, not their absence.
  ▪ make changing things easier.
  ▪ provide a concrete quality metric.
  ▪ help you see and understand what software should and should not do — they provide examples.
Testing

- **Kinds of tests**: unit tests, system tests, integration tests, fuzz tests, stress tests, regression tests, …

- Test delivery options:
  - **Automated**: run over and over. Run for every build.
  - **Manual**: run occasionally (when you have the time).

- Test development methods:
  - **Black-box**: Written with no knowledge of internal workings of the software — Does the software do what it should?
  - **White-box**: Developer writes them with the understanding of the internal workings of the program — Does the software not do what it shouldn’t?

- Testing framework we will be using: SenTestingKit.
Other Items

• Debugging
  • You have to know how to use GDB efficiently.

• Reverse Engineering
  • Critical when working on your first huge pre-existing project.
  • Check out *A Pattern Language for Reverse Engineering* by Demeyer, Ducasse, and Nierstrasz.
Software Development Cycle

Review

Start with a project
Make core classes
Design an interface

Code  Build  Debug  Write tests  Implement

Profile
Software Development Process

Submission is started

Integrate bugs to Build

Build and run tests

Create tag

Build bugs to Verify

Verify bug to Close

Submitter

Bug is filed

Get bug in

Analyze

Software Dev. Cycle

Branch trunk

Merge with trunk

Get bug in

Analyze

Run all unit tests

Commit

Create tag

Build bugs to Verify

Integrate bugs to Build

Run all unit tests

Commit

Bug to Integrate

You

Verify bug to Close
So... exactly how?
Division of Labor

• In MyCampus, there are a number of modules:
  ▪ Maps
  ▪ Food
  ▪ Events
  ▪ Directory
  ▪ Parking
  ▪ Courses (?)

• Per module, team up in pairs.

• All modules will descend from same base class. Each pair will fill in the implementation for module.

• Don’t have to be involved in architecture, unless you want to.
macwarriors
macwarriors