Jessan - Google

- test driven development
  - user profiles
- skeleton up quickly, then flesh out
  - scalable in many dimensions
  - code reviews
- work on the hardest pieces first
Mid-quarter Assessment

Class feedback
SDS and ZFR

- stabilization time before release
- daily build in a clean environment
  - measuring quality
    - automated tools
    - compiler warnings
    - Jslint
    - number of modules reused in the system
    - number of designs patterns used
    - Readability
    - Time to fix most bugs is < value
Beta Release

• basic skeleton with tracer-bullet functionality
  • binary release
  • source release
• updated docs with change tracking on
Quality Assurance

Thinking about Your Work

Turn off the autopilot and take control. Constantly critique and appraise your work.

John Lambert, Microsoft Test Lead Coming to Monday’s class!

CSE 403, Spring 2008, Alverson
With material from Marty Stepp 403 lectures.
Readings

- Pragmatic Programmer: p 237-247 - Ruthless Testing
- Code Complete, McConnell: Chapter 22 Developer Testing

Summary due Monday on PP p237-247 and Code Complete p499-504 only.

Summaries will always be due Mondays going forward
Outline

- Quality from the get-go
- Test
  - What makes a good tester
  - Types of testing
- Bugs
Quality Assurance

Things we can do to ensure we produce a high quality (low defect) product

Two main approaches to QA:
- Process: Build in quality from the start
- Test: Add quality through removing bugs
How can you build in quality from the start?

Spring08 says:

• Code reviews
• Build modules so they’re testable and test driven development
• Coding standard
• Building good code (good interfaces) … good design in hand
• Refactor as you go
From Apple’s C++ Style Guide:

- **Type names** – begin with a capital letter. ie. Book

- **Member names** - begin with an `f`, for "field." Member function names need only begin with a capital letter. ie.: fVisible, Draw.

- **Global names** *(including static members of classes)* - begin with a `g`. ie: gApplication, TGame::gPlayingField.

- **Local and parameter names** - begin with a word whose initial letter is lower case. ie.: i, thePort, aRegion.

- **Constant Names** - begin with a `k`. ie: kSaveDialogResID.

- **Multiple-word names** - the first word should follow the convention for the type of the name, and subsequent words should immediately follow, with the first letter of each word capitalized. ie. TSortedList
Up to 4x the normal number of defects are reported for released products that were developed under excessive schedule pressure (Jones, 94)
Test – the most common QA practice

Two components of test:

- **Verify**: “Did we build the system right?”
- **Validate**: “Did we build the right system?”
What makes a good tester?

- **Analytical**
  - Think of all the odd cases that could happen
  - Think of all the details, describe them
  - Be able to **understand** the system

- **Methodical**
  - Repeatable process, identifying all steps/all variables
  - Code coverage

- **Brutally honest**
  - Report the true data
How do test engineers fail?

- Desire to “make it work”
  - Impartial judge, not “handyman”
- Trust in opinion or expertise
  - Trust no one – the truth (data) is in there
- Failure to follow defined test procedure
  - How did we get here?
- Failure to document the data
- Failure to believe the data
Some testing jargon

Black box testing
Treats the system as atomic
Best simulates the customer experience

White box testing
Examines the system internals
Trace data flow directly (ie, in the debugger)
Bug report contains more detail on source of defect
May obscure timing problems (race conditions)
In black box, the tests are usually intended to cover the space of behavior. Often developer driven.

In white box, the tests are usually intended to cover the space of parts of the program. Often tester driven.
General types of testing

- Functional (include boundaries)
- Coverage/path
- Performance
  - Stress
- Usability
- Compatibility
- Localization
- Security
- Penetration
- Interoperability
- Integration
- UI
Boundary testing

- **boundary testing**: test inputs starting from known good values and progressing through reasonable but invalid to known extreme and invalid

  - Is this black or white box?
Boundary testing

- Imagine we are testing a `Date` class with a `getDaysInMonth(int month, int year)` method.

- What are some important conditions and good boundary tests for this method?
  - reasonable: 3, 2008  2, 2002  2, 2000
  - unreasonable: -1  MaxInt  MinInt  0
  - leapyear  Feb  2,2100
Unit test tools can capture tests

- Capture your boundary tests in a test harness (jig) for easy, automated, running

- JUnit (java) and NUnit (c#), popular harnesses, plugin to eclipse, Ruby Test:::Unit

http://www.nunit.org/
http://www.junit.org/
http://www.jsunit.net/
Attributes of good unit tests

• Well-defined inputs and outputs
  o Consider environment as inputs
  o Consider ‘side effects’ as outputs

• Clearly defined initial conditions
  Clearly described expected behavior

• Specific – small granularity provides greater precision in analysis

• Test must be at least as verifiable as feature
Coverage testing

- **coverage testing**: an attempt to use test input that will pass once over each path in the code

  - Is this black or white box?

  - Can you describe some tests for `getDaysInMonth(month, year)`

  - `int foo(int bar) {`
    - `if (bar <= 0)`
      - `return computationA;`
    - `else`
      - `return computationB;`
  `}`
Coverage analysis

- Shows where you have holes in your test suite
- Guides productive test development: try to get the most coverage with the least effort
- Should you aim for 100% coverage?
Coverage tools

NCollection Code Coverage Report

Modules summary

nUnitConsoleApplication 36.36%

Module nUnitConsoleApplication
Account 100%
AccountTest 51.8%
nUnitConsoleApplication.Program 100%

Plugins for eclipse

Google: nunit [junit] code coverage
Integration/system testing

Shows that the major subsystems that make up the project work well together.

- **big bang**: no stubs; do unit testing, then throw all parts together

- **bottom-up**: integrate upward into double, triple, quadruple module test

- **top-down**: test top layer (UI) first, then add layers to replace underlying stubs

- **big bang**: + faster (if everything works) - can be costly, error-prone
- **bottom-up**: + fewer stubs needed - tests UI last; UI is important!
- **top-down**: + focuses on user experience - needs many stubs
GUI testing

Testing a product that uses a GUI, to ensure it meets its written specifications

- Difficulties:
  - Many operations to test (ie. MS Wordpad has 325)
  - Order of testing matters
  - Regression testing is hard given GUI evolution
  - Need to test on an array of browsers (web apps)
Approaches to GUI testing

- **Automated UI testing**
  - scripts that use your app and look for failures
  - a black-box system test
  - Selenium: http://www.openqa.org/selenium/

- **Manual tests**
  - human beings click through predetermined paths
  - need to write down the specific tests each time
  - http://members.tripod.com/~bazman/checklist.html

- **Ad-hoc tests**
  - human beings are "turned loose" on the app to see if they can break it
Web app compatibility testing

- Motivation
  - ensure that your web app is compatible with various browsers, platforms, etc.
  - ensure that your app's HTML code complies with web standards
  - ensure that you have no broken links or other HTML errors

- W3C HTML validator:
  http://validator.w3.org/

- W3C link checker:
  http://validator.w3.org/checklink

(let's try it out on the 403 home page)
Load testing

Creating demand on a system or device and measuring its response

- How many hits should the system be able to handle?
- What should be its performance under these circumstances?
- Will the system withstand abnormal load (stress testing)?

- tools
  - JMeter: http://jakarta.apache.org/jmeter/
  - curl-loader: http://sourceforge.net/projects/curl-loader
  - Database Open-source Test Suite (DOTS):
    http://ltp.sourceforge.net/
  - others: http://www.opensourcetesting.org/performance.php
Pragmatic Programmer Tips

Test early, test often, test automatically

Coding ain’t done ‘til all the tests run

Find bugs once