CS3505/5020
Software Practice II

Lobby Guidance
Testing
C# Unit Tests
Building the lobby

• A nearly complete example is available:
  – “Getting started with XNA Game Studio” ->
    “Going Beyond: XNA Game Studio in 3D”
    “Tutorial #5: Adding Multiplayer and Networking
    Support to the game”

• You may liberally borrow pieces of the example code.
  – Put an appropriate comment in your code:
  – “// Borrowed from tutorial”
Building the lobby

● Chat is a significant complication
  – A helper class to convert events to a typed string would be useful.

● Interoperability capabilities in XNA
  – Not directly supported – your executable can only see other players with the same executable.
  – Change your project’s Assembly GUID to match another team’s Assembly GUID – your executables should be able to see each other when your GUIDs match.
  – Look in AssemblyInfo.cs, use GUID tool to generate new GUIDs.
Completing the project

● Perhaps Tuesday after spring break is a bit ambitious:
  – I could delay this until Friday, April 2
  – Pros – more time for you and your team of 2
  – Cons – less time for you and your team of 4

  – If I extend it, there will be an overlapping assignment due the following week.

  – Students may choose, 2/3 majority vote required to extend.
An Overview of Testing

● Testing is a process, and there are many:
  » Black box, white box, unit, incremental, integration, regression, acceptance, in-line, load, stress, usability, security, alpha, beta, etc., etc., etc.
  » Why so many?

● Testing serves at least one purpose:
  – Revealing software faults
  – Design through testing
  – Proving correctness / completeness?
    » Yes, in limited circumstances.
An Overview of Testing

- Testing relies on a definition of correctness:
  - Functional correctness
    » Does the software produce the correct result? Are any side effects correct?
    » Can be tested via return values and side effects.
  - Formal correctness
    » Does the software use/implement the correct algorithm? Is it secure? Is it robust? Is it thread safe?
    » Can be tested via source code inspection and formal methods.
  - Suitability
    » Does the software achieve the desired goal? Is it fast enough?
Quality testing – Tests should be...

- Tests should be comprehensive
  - Impossible for most situations
  - Strive for good coverage (boundary cases, etc.)

- Tests should be repeatable
  - Ad hoc testing is fine for a prototype, but not for production code.
  - Documented test cases, test suites, and other testing software address this.

- Running tests should be automatic
  - This reduces the overhead in testing
  - This simplifies regression testing
Purposeful testing

Before writing tests, consider:
- What is the cost of not finding bugs (severity)
- What is the likelihood of a bug occurring (frequency)
- What is the relevance of the bug (importance)
- How hard is it to find the bug (investment)
Important types of testing

Black box testing
- Component specification is given to the tester
- The tester guesses about situations that might not work
- Pros:
  » The tester is not biased by reading the code
  » (The tester makes no assumptions about functionality)
- Cons:
  » Some code logic may never be tested
Important types of testing

• White box testing
  – Source code is given to the tester
  – The tester examines the code and creates tests to guarantee statement coverage
  – Pros:
    » All statements will be tested
  – Cons:
    » The tester may assume something about required functionality based on what is seen in code
Important types of testing

- **Unit testing**
  - Units of software are the smallest testable part of a program.
  - The goal is to test units in isolation so that only one part is being tested.
    - This is important – if some class is dependent on other code in other classes, what exactly will you be testing?
    - This is challenging – how do you test a web service without having clients send you data?
      - There is a testing tool for every purpose...
Important types of testing

- **Integration testing**
  - Tests combinations of software modules – client/server systems, combinations of individual applications, concurrent systems, etc.
  - Unit testing does not address errors that arise during integration
  - Distributed software or concurrent software may rely on interactions that are not testable with unit tests
Important types of testing

- Regression testing
  - Reappplication of tests to ensure functionality does not change after software is modified
  - Ensures that bug fixes do not cause other errors
  - Critical in large systems
  - Many software packages support automatic regression testing, slicing, version comparisons, and other operations to help locate faults injected during revisions
Important types of testing

- **Stress / load testing**
  - Does the software survive under normal work loads? How about extreme work loads?

- **Security testing**
  - Is the software safe from both internal and external attacks?
Important types of testing

- **Alpha testing**
  - In-house simulation of end user testing

- **Beta testing**
  - Live (distributed) testing using actual end users
Testing

- There is one universal purpose in testing:
  - Break software!

  » Tests designed to pass are poor tests – they prove nothing (except in formal methods).
  » Tests should be written with ‘hostility’ in mind – try to break the code being tested.
  » Tests that fail reveal weakness in code – this is useful information.
  » Some tests can provide fault localization information – see fault localization, program comparison, and program slicing literature / tools
Testing

- Agile methods often use testing for an additional purpose:
  - Design!
    » Test cases drive the development process.
    » Write a test, and only then write software to pass it.
    » Software is considered acceptable when it passes the tests.
  - Unit tests are commonly used for test-driven design (TDD).
  - Remember Windows v. Linux holy wars? Perhaps Emacs v. Vi or PC v. Mac?
    » These are nothing compared to TDD v. traditional design debates.
Test-Driven Design: Pros

- Writing tests defines the interface a component should have with the outside world.
  - This is usually a tough design decision, and almost always wrong the first time. Test first helps clarify this.

- Tests define minimum functionality, meaning you can strive to write minimal code.
  - Smaller code is usually better code – it avoids unneeded features.
Test-Driven Design: Pros

- Newly authored code is tested by default.
  - Testing implemented features is harder than implementing tested features.

- Regression testing is simple.
  - Additional authors can work on your code so long as the tests pass. The tests serve as sample code.
Test-Driven Design: Cons

- (All of these are avoidable by skilled programmers)

- Tests as design documents focuses on process, not usability.
  - The only products designed are those that the tester can test.

- Code is not written with quality or planned enhancement in mind.
  - It might be important to add member functions to a class even though they are not currently used.
  - Instead, code is written to ‘just pass the tests’
Test-Driven Design: Cons

- Test-driven design devalues exploration (exploration is a key part of software engineering).
  - Some problems are hard enough to require iterative problem solving in software
  - Prototyping is also important – determining which solutions might be best prior to determining overall functionality

- Existing tests discourage refactoring
  - Refactoring requires modifying code and tests (double the workload) – it is easier to not change anything
  - Refactoring is important to agile methods
Testing

● Another purpose in testing:
  – Proof of correctness

  » This is extremely important for concurrent systems, embedded systems, and software implemented in hardware (microcode)
  » “Formal methods” uses specifications of desired software behavior along with “proof” engines that reason about the possible outcomes of a piece of code.
  » If any possible outcome violates the requirements of the specification, the validation fails.
Testing

- A few final thoughts:
  - Writing tests is no easier than writing code – it is often harder
    » This does not mean it is not important!!!
  - Errors in tests often directly translate into errors in code.
  - Testing of non-deterministic processes with deterministic tests will lead to failure.
    » Don’t use unit tests on multithreaded code
    » Don’t assume compositions or convolutions of code will work if the parts work.
A brief unit test example in C#

- Square root method