CSE 403, Winter 2010
PHASE 4 (40 points): Testing Procedure Specification (TPS)

due Fri Feb 26, 11:59pm

In this phase you will submit two major deliverables:

1. A set of **unit test cases** that test part of the functionality of your upcoming "beta" code, and
2. A "testing procedure specification" report (**TPS report**, IEEE 829) that describes three other major types of testing that your group intends to perform on the product by the time it reaches "v1" status.

1. Unit Test Cases

Submit at least **five (5)** major unit test case program/class files that each test a major class or subsystem of your project.

Framework: Write these tests using the appropriate *"-unit"* framework for your language. For example, if you are using Java, use JUnit, TestNG, JUnitEE, or similar. In PHP, use PHPUnit or SimpleTest. In Ruby, consider Test::Unit, RSpec, or ZenTest. In Python, use PyUnit. For C# .NET, use NUnit or csUnit. And so on.

Some web frameworks (such as CakePHP) incorporate their own unit testing primitives; you may want to use these in your project. If your project incorporates multiple languages or technologies, you may wish to investigate unit testing for those tools such as JSUnit (JavaScript), DbUnit (databases), XMLUnit, utMySQL, etc.

Breadth: Each test file should test only one class/subsystem; its area of testing should be labeled (commented) clearly. Each test class should contain several smaller test cases (usually implemented as test methods) that each test a small part of the functionality of the class(es) under test. Your code does not need to be thoroughly covered by testing and quality-assurance tools at this point, but we will look to see that you have chosen a broad set of non-trivial functionality to test.

We generally suggest that your unit tests focus their coverage on the "model" and/or data-driven aspects of the project, since these are often conducive to the unit testing process; but you can write tests for any part you like.

Depth: For full credit, each test class should be fairly comprehensive over the surface of the class(es) that it is testing. That is, there should be at least one testing method/case that covers each major method, constructor, etc. of the class(es) under test. It is likely that some of these methods are more complex than others; more complex methods (such as those that accept several parameters, or those that perform complex operations on the object's state) may need to be covered by multiple test methods in order to be considered well covered. For example, if you were testing a method that adds an element to a list in sorted order, you would want one test case that adds to the front of the list, another that adds to the end, another to the middle, another to an empty list, another to a one-element list, another to a large list, and so on.

It is often wise to concoct various test "helper" methods that allow you to describe tests at a high level as calls to another helping method. This way your test program does not become unwieldy or redundant, and it is easier to add more tests.

Bug-tracking: Since test cases reveal the presence of bugs, you should update your project's bug-tracking system as you work on this phase and complete and run your test cases. As we are grading we will examine your bug-tracker to look for an adequate number of reported bugs. The bugs should be thoroughly filled out with information such as: description, steps to reproduce the bug, severity, who the bug is assigned to, its current status (open, resolved, won't fix, etc.) Many bugs can still be unresolved at this point in time; the main point is that some have been found, not that all have been fixed.

Black/white box: Each test case should be labeled (commented) as to whether it is a black-box test (where the tester is not aware of all of the internal details of the code under test) or a white-box test (where the tester looks at the code and uses this information to guide the creation of test cases). At least one of your unit tests must be black-box and at least one other must be white-box. The tests should fit their given category. That is, if the test is black-box, it should focus on externally discoverable behavior such as boundary cases, parameter values, unexpected/edge cases, and combinations of calls. If the test is white-box, it should focus on covering each path and statement of the code in the class under test.

Test-driven development: At least one of your tests must be written in the "test-driven development" style. This means that the test must be written to cover a class or subsystem that *has not been written or finished yet*. Generally the class under test is created with empty or trivial "stub" methods that do not yet produce the proper expected behavior. It is expected that this test case will fail when it is run. The idea is that now the developer must write the class such that it will pass all of the test cases. The test case should be comprehensive enough that once the code under test is written and passes it, the developers have reasonable assurance that the class under test has been well implemented.

2. Testing Procedure Specification ("TPS") Report

Your TPS report is a document describing your plans for how you will further test your product to assure its quality. Submit a document, approximately **2-3 pages in length**, addressing the following pieces of information:

a. Further Unit Testing

List which additional classes of your project you intend to unit test, and if appropriate, which of their methods and behavior. For any classes you do not plan to unit test, briefly justify your decision not to do so.

Also *briefly* describe the **process** of your unit test development, both so far and looking forward. Who has written the tests so far, and who will write the remaining tests? What tests will be black-box tests (written by someone who has not seen the source code being tested) versus white box (by someone who has seen the complete source code)? What are some of the more significant cases you plan to test, such as boundary conditions and expected error cases?

In a later phase, your project's eventual unit testing submission must demonstrate significant unit test **coverage**, covering approximately a given percentage or more of the overall project code. In your test plan document you should describe how you plan to prove to the customer that you have achieved this coverage. You will eventually need to use a tool that displays unit test coverage percentages, such as NoUnit, EMMA, Cobertura, or Hansel (though not on this phase).

b. System Testing

Describe the ways in which you plan to test your product as a whole. For full credit, your eventual product must undergo non-trivial testing in at least **three (3) areas of system testing**, such as the following:

- Integration testing
- Automated UI "functional" testing
- Performance testing / profiling
- Load / stress / reliability testing
- Security testing / auditing
- Usability testing
- Internationalization / localization testing

For example, you could perform documented usability using clearly defined test scenarios with users from outside your group. You could create automated UI tests using a framework such as Selenium. Part of your grade will depend on your choosing reasonable functionality to test and effective means to ensure that your tests will produce a meaningful outcome.

It may be acceptable to perform only two (2) of the above areas, rather than 3, if you perform two major tasks and deliverables for a particular area. For example, you could do an extensive set of load tests as well as a large set of reliability tests. If you have a non-standard set of testing you want to perform, please ask the customer for approval.

Other kinds of testing not listed above may be acceptable; ask your customer for approval if you are interested in performing a kind of testing not listed. We will not accept ad-hoc testing or customer acceptance testing as one of your three categories, though, since these are not as conducive to the goals of the project or to grading.

In your test plan document, describe which kinds of system testing you plan to perform, and also how you intend to demonstrate to the grader that you have performed them adequately. Testing that is done anecdotally without any resulting evidence (e.g. "We looked over all our code to make sure it was secure. And it was!") may not be given credit.

Submission and Grading:

Submit your testing code online through the course web site. The code should be gathered into a single archive such as a .ZIP file. Each file in the archive should be clearly labeled. The TPS report should be submitted as a .DOC, .ODT, or PDF file. It can be separate or part of the ZIP archive.

Your bug-tracking system should actually exist when we go to examine it, and we should be able to log in and access it successfully on the first attempt. Sufficient bugs must be listed per developer with suitable details marked on each bug.

Your TPS phase does not need to reflect "customer" interaction, but you may still want to have an optional meeting with your customer about it, and/or ask your customer about various aspects of this phase before you officially submit it. If you make your request a reasonable amount of time before the due date, the customer will do his/her best to try to accommodate such requests and give feedback in a timely manner.