SE2831 Introduction to Software Verification

Dr. Walter Schilling

Fall, 2011

You may use one (1) 8.5 x 11 sheet of paper with notes on it for the exam.

1. Week #1
   (a) Lecture #1
      i. No Class due to Labor day.
   (b) Lecture #2 Introduction to Software Testing
      i. Explain the relationship between the cost of fixing a defect and the phase in which the defect is discovered.
      ii. Justify the importance of software testing from an economic standpoint.
      iii. Explain through case studies the root cause of one or more software failures.
2. Week #2
   (a) Lecture #1 An Overview of Software Development Lifecycles
      i. List and explain Polyas four principles to problem solving.
      ii. Draw the Waterfall model for software development
      iii. Explain the purpose for each step within the waterfall process
      iv. Draw a representation of an incremental model of software development
      v. List the advantages of an incremental model over a waterfall model
      vi. Compare and contrast the incremental model and the waterfall model
      vii. Explain and draw the spiral model for software development
      viii. Draw the V Model for software development
      ix. Explain the roles of testing within the V Model
   (b) Lecture #2 Your First Unit Tests
      i. Define testing.
      ii. Define the relationship between errors, defects, and failures
      iii. Compare and Contrast the four main levels of testing.
      iv. Explain why it is impossible to test every case of program execution.
      v. Construct rudimentary test case for a simple software method.
3. Week #3
   (a) Lecture #1 Equivalence Class Testing
      i. Define the terms black box test and white box test.
      ii. Explain the concept of a test oracle.
      iii. Define equivalence class.
      iv. Explain the concept of design by contract. (Reading only)
      v. Compare and contrast defensive testing and testing by contract. (Reading only)
      vi. Given a software description, define the equivalence classes for a given problem.
      vii. Given a software description, construct test cases using the equivalence partitioning method.
      viii. Based on Equivalence class testing, determine the minimum number of test cases necessary to test a given software system.
   (b) Lecture #2 Boundary Value Testing
      i. Define a software boundary condition.
      ii. Explain why boundary conditions represent commonly occurring mistakes.

iii. Given a software description, construct test cases using the boundary value testing method.
iv. Compare and contrast boundary value testing with equivalence class testing.
v. Based on boundary value testing, determine the minimum number of tests required to test a given software system.
vi. Explain how one can combine boundary value testing with equivalence class testing to solve multi-dimensional data sets.
vii. Construct test cases which would allow for the testing of multi-dimensional data sets.

4. Week #4
   (a) Lecture #1 Automating your unit tests with JUnit.
      i. Explain the purpose for the JUnit framework.
      ii. Draw the architecture for the JUnit framework.
      iii. Define the JUnit terms test fixture, unit test, test case, test suite, and test runner.
      iv. Draw the initialization flow for JUnit when executing tests.
      v. Critique the limitations of JUnit.
      vi. Construct a rudimentary unit test using JUNit.
   (b) Lecture #2 State Transition Testing
      i. Define the terms state, transition, event, and action.
      ii. Explain the concept of a state transition table.
      iii. Given a state diagram, construct a state transition table for the problem.
      iv. Explain how state transition test cases can be constructed from a state machine definition.
      v. Construct a set of test cases from a state diagram.

5. Week #5
   (a) Lecture #1
      i. Catch up on anything not yet covered.
      ii. Review materials for the exam.
   (b) Lecture #2 Midterm Exam
      i. Successfully master the exam material presented thusfar.

6. Week #6
   (a) Lecture #1 Mock Objects
      i. Explain the concept of testing stubs.
      ii. Explain the concept of the Mock Object testing pattern.
      iii. List the three steps necessary for testing a system with mock objects.
      iv. Construct a mock object for a rudimentary system and use this object for the purpose of constructing unit tests.
      v. Construct a mock object using EasyMock.
   (b) Lecture #2 White Box Testing and Control Flow Testing
      i. Define white box testing.
      ii. List advantages and disadvantages of white box testing versus black box testing.
      iii. Define control flow testing.
      iv. Define the terms process block, decision point, and junction point in the context of a control flow graph.
      v. Construct a control flow graph from source code.

7. Week #7
   (a) Lecture #1 Cyclomatic Complexity
      i. Calculate the McCabe Cyclomatic complexity for a source code module based on a control flow graph.
      ii. Define the term basis path.
      iii. Construct test cases from the control flow graph which fully exercise the software module.
      iv. Critique Control flow testing, listing its advantages and disadvantages versus other testing techniques.
   (b) Lecture #2 Scripted Testing

i. Explain the code and fix programming paradigm.
ii. Define the terms repeatability, objectivity, and auditability.
iii. List the documents present within the IEEE 829 test document set.
iv. Explain the purpose for a software test plan.
v. Explain the purpose for a test design specification.
vi. Explain the purpose for a test case specification.
vii. Explain the purpose for a test procedure specification.

8. Week #8

(a) Lecture #1 Exploratory Testing

i. Define Exploratory testing.
ii. Define a testing charter.
iii. Define timebox.
iv. Critique the advantages and disadvantages of exploratory testing.
v. Compare and Contrast Scripted testing with exploratory testing, highlighting the differences of the approaches.

(b) Lecture #2 Assessing the Quality of Tests: Mutation Testing

i. Define the term mutant.
ii. Explain the process of mutation testing.
iii. Define the terms Killed Mutant, Alive/Live Mutant, and Stubborn Mutant.
iv. Quantify the time complexity of mutation testing.
v. Understand the limitations of mutation testing.
vi. Perform mutation testing using a mutation testing software tool.

9. Week #9

(a) Lecture #1 What makes for good tests?

i. Compare and contrast the advantages of whitebox and blackbox testing to system testing.
ii. Explain the concept of regression testing.
iii. Explain the advantages and disadvantages of automated testing.
iv. Explain the concept of bug clustering.
v. Describe the concept of smoke testing.

(b) Lecture #2 Design issues and testability.

i. Define testability.
ii. Define the term fault density.
iii. Visualize the relationship between cyclomatic complexity and the probability of fault manifestation.
iv. Visualize the defect density curve and relation to lines of code and justify module size decisions.
v. Interpret the defect density curve as it applies to software testing.
vi. Critique the usage of cyclomatic complexity as a test measure.
vii. Define Fan-In and Fan-Out.
viii. Define the Depth of the inheritance tree.
ix. Compare and contrast DIT_A and DIT_T, noting the advantages and disadvantages of one versus the other.

10. Week #10

(a) Lecture #1 When to write tests and who should write tests

i. Critique the advantages of writing tests before development has commenced.
ii. Critique the advantages of having a second person write tests independent from the developer.

(b) Lecture #2 Review and conclusion.

i. Review for the final exam.
ii. Analyze the effectiveness of the course.

1 Lab Outcomes

1. Lab 1: The Triangle Problem
   (a) To review and refresh Java programming skills after the summer break
   (b) Construct a simple Java program using File IO and mathematical calculations.
   (c) Construct Java programs which properly handle exceptions.
   (d) Develop rudimentary test cases to verify that a program is functioning properly
   (e) Describe a set of test cases for a given domain problem
   (f) Use those developed test cases to test a developed program
2. Lab 2: The Bank of MSOE - Part 1
   (a) Analyze and interpret use case scenarios to develop test cases.
   (b) Organize testing in a meaningful fashion in order to ensure maximum defect detection.
   (c) Document and describe a set of test cases for a given problem.
   (d) Construct a set of test scripts to aid in testing.
3. Lab 3: The Bank of MSOE - Part 2
   (a) Perform software testing using a series of developed test cases.
   (b) Write accurate and complete defect reports.
   (c) Enter defect reports into a defect tracking tool.
4. Lab 4: Introduction to JUnit: The Tax Code
   (a) Construct a series of JUnit tests using a combination of equivalence classes and boundary value analysis.
   (b) Execute JUnit tests to uncover simple programming mistakes.
   (c) Correct programming mistakes from within a source code module.
   (d) Regression test software to ensure that the fixing of defects has not broken other portions of the software.
   (e) Write accurate and complete defect reports
   (f) Enter defect reports into a defect tracking tool
   (g) Track uncovered defects to closure using a defect tracking system.
5. Lab 6: Mock Objects and Watching the Stock Market
   (a) Use Mock Objects to verify the behavior of a class under test.
   (b) Obtain familiarity with EasyMock, a tool for developing Mock Objects
   (c) Apply JUnit to the testing of an existing Java application
   (d) Apply boundary conditions to determine proper test results for the system
   (e) Debug existing Java code and remove faults which have been injected by an external entity
6. Lab 7: State transition testing
   (a) Analyze and construct test cases from a state diagram.
   (b) Perform state testing on an existing class, noting any found defects.
7. Lab 8: Mutation Testing
   (a) Investigate the quality of existing test suites using Mutation testing
   (b) Analyze existing JUnit test case effectiveness using a mutation testing tool
8. Lab 9: Code Coverage Analysis with EMMA
   (a) Understand the importance of code coverage in assessing testing of software deliverables.
   (b) Use a code coverage tool to uncover both tested and untested segments of source code within a software module.
9. Lab 10: Complexity Analysis with CyVis
   (a) Use source code analysis to determine the complexity of software modules
   (b) Using source code metrics, critique the quality of a software product
   (c) Determine which modules may require extra testing or are otherwise overly complex and may pose problems during development