Testing I

Week 14
Agenda (Lecture)

- Concepts and principles of software testing
- Verification and validation
- Non-execution based testing
- Execution based testing
- Feasibility of testing to specification
- Feasibility of testing to code
- Black box testing
Agenda (Lab)

- Implementation
- Submit a weekly project progress report at the end of this week lab session
Software Test

• Software process and a testing phase
  – A separate testing phase?
• Testing
  – Non-execution based and execution based
• Mindset
  – Test-oriented process models
Verification vs. Validation

• **Verification**: "Are we building the product right?"
  – The software should conform to process that is chosen.

• **Validation**: "Are we building the right product?"
  – The software should do what the user really requires.

• Testing (V&V) is a whole life-cycle process
  – V & V must be applied at each stage in the software process
The Relative Cost of Finding a Fault at Each Phase
Non-execution Based Testing

- Testing software without running test cases
- Non-execution based testing includes reviewing software and analyzing software
- Applied to the early phases or workflows such as requirement, specification and design, and even implementation
- Process models and organizations provide guidelines for non-execution based testing
  - IEEE standard for software reviews [IEEE 1028]
Walk-through and Inspection

• Walk-through is less formal and inspection is more formal
Inspections

Software Inspection

- Requirement and specification
- Formal or semi-formal specification
- High-level design
- Detailed design
- Program

Program Testing
Inspection Success

• Many different defects may be discovered in a single inspection.
• Using domain and programming knowledge reviewers are likely to have seen the types of error that commonly arise.
The Inspection Process

- Planning
- Overview
- Individual preparation
- Inspection meeting
- Rework
- Follow-up
Walk-Through

• Less formal approach to review
• Uncover faults and record them for later correction
Case Studies

• 67 percent of all the faults were located by inspections before unit testing was started
• 82 percent of all detected faults were discovered during design and code inspections
• 93 percent of all detected faults were found during inspections
• At the JPL, on average, each 2-hour inspection exposed 4 major faults and 14 minor faults
  – Translated into dollar terms, this meant a savings of $25,000 per inspection
Execution-based Testing

- “Program testing can be a very effective way to show the presence of bugs, but it is hopelessly inadequate for showing their absence” – Dijkstra, 1972
- “Execution-based testing is a process of inferring certain behavioral properties of a product based on the results of executing the product in a known environment with selected inputs”
- Incremental approaches to the execution-based testing
  - Unit-testing
  - Integration testing
  - Product testing
  - Acceptance testing / alpha or beta testing
Feasibility of Testing to Specification

• Two inputs
  – One has five values
  – The other has seven values
  – How many test cases are needed
  – 5 X 7 = 35
• 30 inputs
  – Each input has four different values
  – How many test cases are required?
  – If a program has 1.1 X 10^{18} possible inputs and one test can be run every microsecond, how long would it take to execute all of the possible inputs?
Feasibility of Testing to Code

Read (kmax) // kmax is an integer between 1 and 18
for (k = 0; k < kmax; k++) do
{
    read (myChar) // myChar is the character A, B, or C
    switch (myChar)
    {
        case 'A':
            block A;
            if (cond1) blockC;
            break;
        case 'B':
            block B;
            if (cond2) blockC;
            break;
        case 'C':
            block C;
            break;
    }
}
Feasibility of Testing to Code


diagram

Feasibility of Testing to Code

A

k < 18

[yes]

[no]

blockA

myChar

blockB

cond1

true

false

true

false

cond2

blockC

blockD

cond1

true

false

cond2

true

false
1. INTRODUCTION
   1.1 PURPOSE
   1.2 BACKGROUND
   1.3 SCOPE
   1.4 PROJECT IDENTIFICATION
2. SCENARIOS FOR TEST
3. TEST STRATEGY
   3.1 TESTING TYPES
      3.1.1 FUNCTION TESTING
      3.1.2 NON-FUNCTION PROFILE
      3.1.3 RELIABILITY TESTING
      3.1.4 STRESS TESTING
      3.1.5 VOLUME TESTING
      3.1.6 ROBUSTNESS TESTING
      3.1.7 SECURITY TESTING
      3.1.8 INSTALLATION TESTING
   3.2 TOOLS
4. RESOURCES
   4.1 WORKERS
   4.2 SYSTEMS
5. PROJECT MILESTONES
6. DELIVERABLES
   6.1 TEST MODEL
   6.2 TEST LOGS
   6.3 DEFECT REPORTS
Black Box Testing

- Behavioral
- Functional
- Data-driven
- Input/output-driven
Black Box Testing (cont’d)

• Exhaustive black-box testing generally requires billions and billions of test cases

• The art of testing is to devise small, manageable set of test cases to maximize the chances of detecting a fault, while minimizing the chances of wasting a test case due to having the same fault detected by more than one test case

• Every test case must be chosen to detect a previously undetected fault
Equivalence Testing

• Equivalent partitioning
• A black-box testing method
• Divides input domain of a product into classes of data
• Equivalent classes are used to define test cases that uncover classes of error and reduce the total number of test cases that must be developed
  – With boundary value analysis
• An equivalence class represents a set of valid or invalid state for input conditions
Equivalence Testing - Example

- The possible blood sugar level (including safe, unsafe, and undesirable) is between 1 and 35.
- Equivalence classes for this example
  - Equivalence class1:
  - Equivalence class2:
  - Equivalence class3:
Boundary Value Analysis

• Maximize the chances of finding a fault
• Experience has shown that, when a test case on or just one side of the boundary of an equivalence class is selected, the probability of detecting a fault increases
Type of Equivalence Class

• A range of values
• A set of values
  – The input must be letter
• A specific value
  – The response must be followed by a # sign
How to Perform Equivalence Testing

• For each range (L, U)
  – Select five test cases: less than L, equal to L, greater than L but less than U, equal to U, and greater than U

• For each set S
  – Select two test cases: a member of S and a non-member of S

• For each precise value P
  – Select two test cases: P and anything else
Exercises

• How many minimum number of test cases should be prepared for a range (R1, R2) listed in either the input or output specifications?

• How many minimum of number test cases should be prepared when it is specified that an item must be a precise value?