9. Software Testing Strategies

Abdus Sattar
Assistant Professor
Department of Computer Science and Engineering
Daffodil International University
Email: abdus.cse@diu.edu.bd

Discussion Topics

- Program Testing
- Aim of Testing
- Verification vs Validation
- Design of Test Cases
- Functional Testing Vs. Structural Testing
- Black Box Testing
- White Box Testing

Program Testing

- Testing a program consists of providing the program with a set of test inputs (or test cases) and observing if the program behaves as expected. If the program fails to behave as expected, then the conditions under which failure occurs are noted for later debugging and correction.
- Some commonly used terms associated with testing are:
  - **Failure:** This is a manifestation of an error (or defect or bug). But, the mere presence of an error may not necessarily lead to a failure.
  - **Test case:** This is the triplet [I,S,O], where I is the data input to the system, S is the state of the system at which the data is input, and O is the expected output of the system.
  - **Test suite:** This is the set of all test cases with which a given software product is to be tested.

Aim of Testing

- The aim of the testing process is to identify all defects existing in a software product.
- However for most practical systems, even after satisfactorily carrying out the testing phase, it is not possible to guarantee that the software is error free. This is because of the fact that the input data domain of most software products is very large. It is not practical to test the software exhaustively with respect to each value that the input data may assume.
- Even with this practical limitation of the testing process, the importance of testing should not be underestimated.
- It must be remembered that testing does expose many defects existing in a software product. Thus testing provides a practical way of reducing defects in a system and increasing the users' confidence in a developed system.

Verification vs Validation

- **Verification** is the process of determining whether the output of one phase of software development conforms to that of its previous phase.
  - Verification is concerned with phase containment of errors.
- **Validation** is the process of determining whether a fully developed system conforms to its requirements specification.
  - Aim of validation is that the final product be error free.

Design of Test Cases

- Exhaustive testing of almost any non-trivial system is impractical due to the fact that the domain of input data values to most practical software systems is either extremely large or infinite.
- Therefore, we must design an optional test suite that is of reasonable size and can uncover as many errors existing in the system as possible.
- But larger test suite does not always detects more error. For example lets consider the following code:
  - ```
    if (x>y)
        max = x;
    else
        max = x;
    ```
    ```

Design of Test Cases(Cont..)

- For the above code segment, consider the following test suites
  - Test suite 1: **{ (x=3,y=2); (x=2,y=3) }**
  - Test suite 2: **{ (x=3,y=2); (x=4,y=3); (x=5,y=1) }**
- Test suite 1 can detect the error.
- Test suite 2 can not detect the error despite of being large.
- Therefore, systematic approaches should be followed to design an optimal test suite. In an optimal test suite, each test case is designed to detect different errors.

Functional Testing Vs. Structural Testing

- In the black-box testing approach, test cases are designed using only the functional specification of the software, i.e. without any knowledge of the internal structure of the software. For this reason, black-box testing is known as functional testing.
- On the other hand, in the white-box testing approach, designing test cases requires thorough knowledge about the internal structure of software, and therefore the white-box testing is called structural testing.

Unit testing

- Unit testing is undertaken after a module has been coded and successfully reviewed. Unit testing (or module testing) is the testing of different units (or modules) of a system in isolation.
- In order to test a single module, a complete environment is needed to provide all that is necessary for execution of the module. That is, besides the module under test itself, the following steps are needed in order to be able to test the module:
  - The procedures belonging to other modules that the module under test calls.
  - Nonlocal data structures that the module accesses.
  - A procedure to call the functions of the module under test with appropriate parameters.

Unit Testing

module
to be
tested

interface

local data structures

boundary conditions

independent paths

error handling paths

test cases

Black Box Testing

- In the black-box testing
  - test cases are designed from an examination of the input/output values only
  - no knowledge of design or code is required.
- The following are the two main approaches to designing black box test cases.
  - Equivalence class portioning
  - Boundary value analysis

Black Box Testing

- **In Black Box Testing we just focus on inputs and output of the software system** without bothering about internal knowledge of the software program.

Equivalence Class Partitioning

- In this approach, the domain of input values to a program is partitioned into a set of equivalence classes.
- The following are some general guidelines for designing the equivalence classes:
  - If the input data values to a system can be specified by a range of values, then one valid and two invalid equivalence classes should be defined.
  - If the input data assumes values from a set of discrete members of some domain, then one equivalence class for valid input values and two equivalence classes for invalid input values should be defined.

Equivalence Class Partitioning

- **Example 1:** A software can compute the square root of an input integer which can assume values in the range of 0 to 5000. Design 3 Equivalence Class Partitioning test cases.
- **Answer:** There are three equivalence classes:
  - The set of negative integers.
  - the set of integers in the range of 0 and 5000 and
  - the integers larger than 5000.
  - Therefore, the test cases must include representatives for each of the three equivalence classes and possible 3 test sets can be:
  - Test suite 1: {-5,500,6000}, Test suite 2: {-15,900,6020} and Test suite 3: {-3,4999,5100}.

Equivalence Class Partitioning

- **Example 2:** Design the black-box test suite for the following program. The program computes the intersection point of two straight lines and displays the result. It reads two integer pairs (m1, c1) and (m2, c2) defining the two straight lines of the form $y=mx + c$.
- **Answer :** The equivalence classes are the following:
  - Parallel lines (m1=m2, c1≠c2)
  - Intersecting lines (m1≠m2)
  - Coincident lines (m1=m2, c1=c2)
  - Now, selecting one representative value from each equivalence class, the test suit {(2, 2) (2, 5)}, {(5, 5) (7, 7)}, {(10, 10) (10, 10)} are obtained.

Equivalence Class partitioning

Example 3:

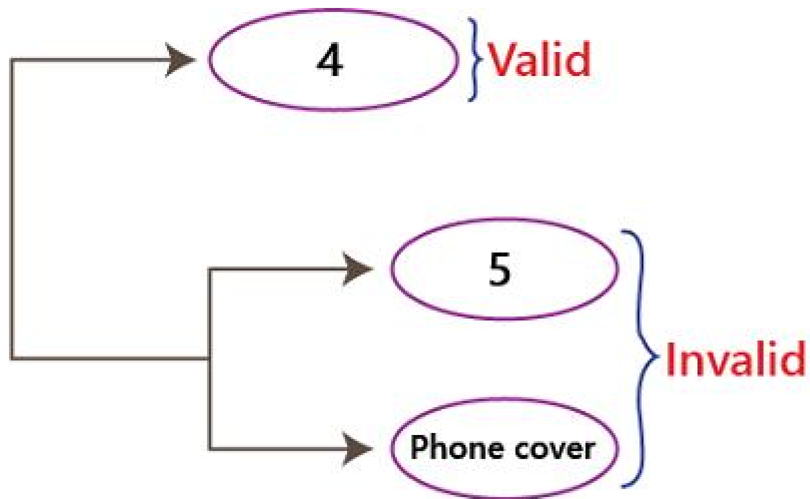
if we are doing online shopping, mobile phone product, and the different **Product ID -1,4,7,9**

Here, **1 → phone covers 4 → earphones 7 → charger 9 → Screen guard**

And if we give the product id as **4**, it will be accepted, and it is one valid value,

and if we provide the product id as **5 and phone cover**,

it will not be accepted as per the requirement, and these are the two invalid values.



Boundary Value Analysis

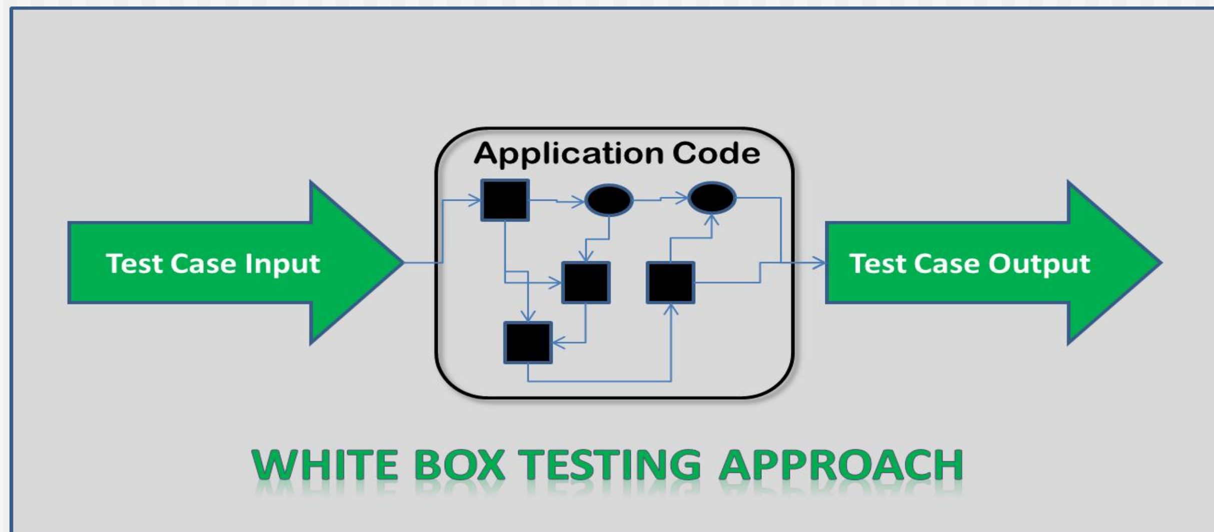
- A type of programming error frequently occurs at the boundaries of different equivalence classes of inputs. The reason behind such errors might purely be due to psychological factors. Programmers often fail to see the special processing required by the input values that lie at the boundary of the different equivalence classes. For example, programmers may improperly use < instead of <=, or conversely <= for <. Boundary value analysis leads to selection of test cases at the boundaries of the different equivalence classes.
- **Example:** For a function that computes the square root of integer values in the range of 0 and 5000, the test cases must include the following values: {0, -1,5000,5001}.

WHITE-BOX TESTING

- WHITE BOX TESTING (also known as Clear Box Testing, Open Box Testing, Glass Box Testing, Transparent Box Testing, Code-Based Testing or Structural Testing) is a software testing method in which the internal structure/design/implementation of the item being tested is known to the tester.
- One white-box testing strategy is said to be *stronger than* another strategy, if all types of errors detected by the first testing strategy is also detected by the second testing strategy, and the second testing strategy additionally detects some more types of errors. When two testing strategies detect errors that are different at least with respect to some types of errors, then they are called *complementary*.

White Box Testing

- White-box testing (also known as clear box testing, glass box testing, transparent box testing, and structural testing) is a method of testing software that tests internal structures or workings of an application, as opposed to its functionality



WHITE-BOX TESTING

- Example: Consider the Euclid's GCD computation algorithm:

```
int compute_gcd(x, y) {
        int x, y;
        while (x! = y) {
                if (x>y)
                    x= x – y;
                else
                    y= y – x;
                }
        return x;
}
```

- By choosing the test set {(x=3, y=3), (x=4, y=3), (x=3, y=4)}, we can exercise the program such that all statements are executed at least once.

References

Software Maintenance Models

https://www.professionalqa.com/software-maintenance-models

Chapter 9 software maintenance

https://www.slideshare.net/abhinavtheneo/chapter-9-software-maintenance