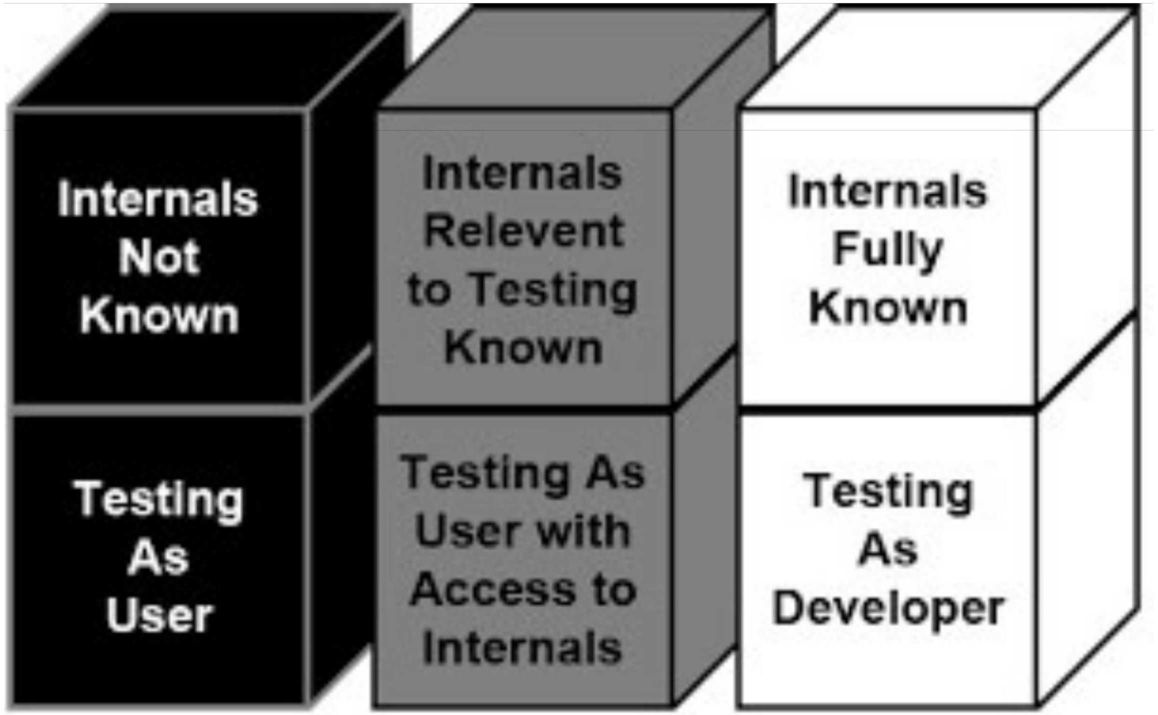
Unit 5.4
Software Engineering

Software Testing

Views of Test Objects

Black Box Testing	White Box Testing	Grey Box Testing
Close Box Testing Testing based only on specification	Open Box Testing Testing based on actual source code	Partial knowledge of source code



Internals Not Known

Testing As User

Internals Relevent to Testing Known

Testing As User with Access to Internals

Internals Fully Known

Testing As Developer

Black Box Testing

- □ Also known as specification-based testing
- □ Tester has access only to running code and the specification it is supposed to satisfy.
- □ Test cases are written with no knowledge of internal workings of the code
- □ No access to source code
- □ So test cases don’t worry about structure
- □ Emphasis is only on ensuring that the contract is met

Black Box Testing Cont.

□ Advantages

- Scalable; not dependent on size of code
- Testing needs no knowledge of implementation
- Tester and developer can be truly independent of each other
- Tests are done with requirements in mind
- Does not excuse inconsistencies in the specifications
- Test cases can be developed in parallel with code

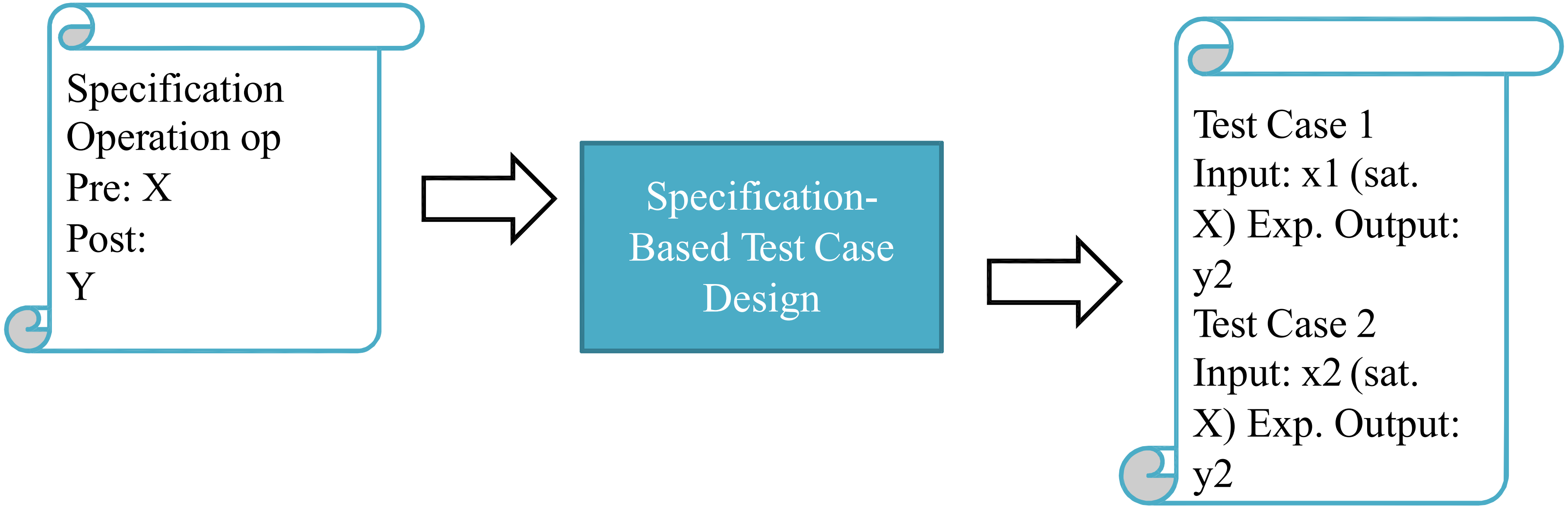
□ Disadvantages

- Test size will have to be small
- Specifications must be clear, concise, and correct
- May leave many program paths untested
- Weighting of program paths is not possible

Black Box Testing Cont.

Test Case Design

- □ Examine pre-condition, and identify equivalence classes
- □ All possible inputs such that all classes are covered
- □ Apply the specification to input to write down expected output



Specification
Operation op
Pre: X
Post:
Y

Specification-Based Test Case Design

Test Case 1
Input: x1 (sat. X) Exp. Output: y2
Test Case 2
Input: x2 (sat. X) Exp. Output: y2

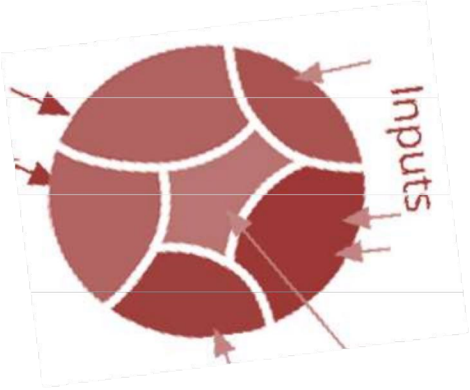
Black Box Testing Cont.

- □ Exhausting testing is not always possible when there is a large set of input combinations, because of budget and time constraint.
- □ The special techniques are needed which select test-cases smartly from the all combination of test-cases in such a way that all scenarios are covered.

Two techniques are used

Equivalence Partitioning

Boundary Value Analysis (BVA)



Black Box Testing Cont.

Equivalence Partitioning

- □ Input data for a program unit usually falls into a number of partitions, e.g. all negative integers, zero, all positive numbers
- □ Each partition of input data makes the program behave in a similar way
- □ Two test cases based on members from the same partition is likely to reveal the same bugs

By identifying and testing one member of each partition we gain 'good' coverage with 'small' number of test cases

Testing one member of a partition should be as good as testing any member of the partition

Black Box Testing Cont.

Example - Equivalence Partitioning

□ Example: for binary search the following partitions exist

- Inputs that conform to pre-conditions
- Inputs where the precondition is false
- Inputs where the key element is a member of the array
- Inputs where the key element is not a member of the array

□ Pick specific conditions of the array

- The array has a single value
- Array length is even
- Array length is odd

Black Box Testing Cont.

Example - Equivalence Partitioning

□ Example: Assume that we have to test field which accepts SPI (Semester Performance Index) as input (SPI range is 0 to 10)

SPI

* Accepts value 0 to 10

Equivalence Partitioning		
Invalid	Valid	Invalid
<=-1	0 to 10	>=11

□ Valid Class: 0 – 10, pick any one input test data from 0 to 10

□ Invalid Class 1: <=-1, pick any one input test data less than or equal to -1

□ Invalid Class 2: >=11, pick any one input test data greater than or equal to 11

Black Box Testing Cont.

Boundary Value Analysis (BVA)

□ It arises from the fact that most program fail at input boundaries

□ Boundary testing is the process of testing between extreme ends or boundaries between partitions of the input values.

□ In Boundary Testing, Equivalence Class Partitioning plays a good role

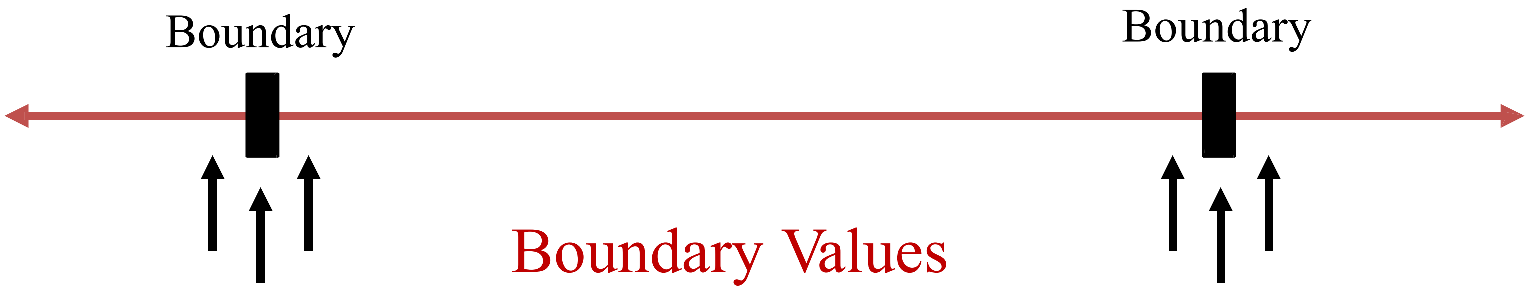
□ Boundary Testing comes after the Equivalence Class Partitioning

□ The basic idea in boundary value testing is to select input variable values at their:

Just below the minimum	Minimum	Just above the minimum
Just below the maximum	Maximum	Just above the maximum

Black Box Testing Cont.

Boundary Value Analysis (BVA)



□ Suppose system asks for “a number between 100 and 999 inclusive”

□ The boundaries are 100 and 999

□ We therefore test for values

99 100 101

Lower boundary

999 999 1000

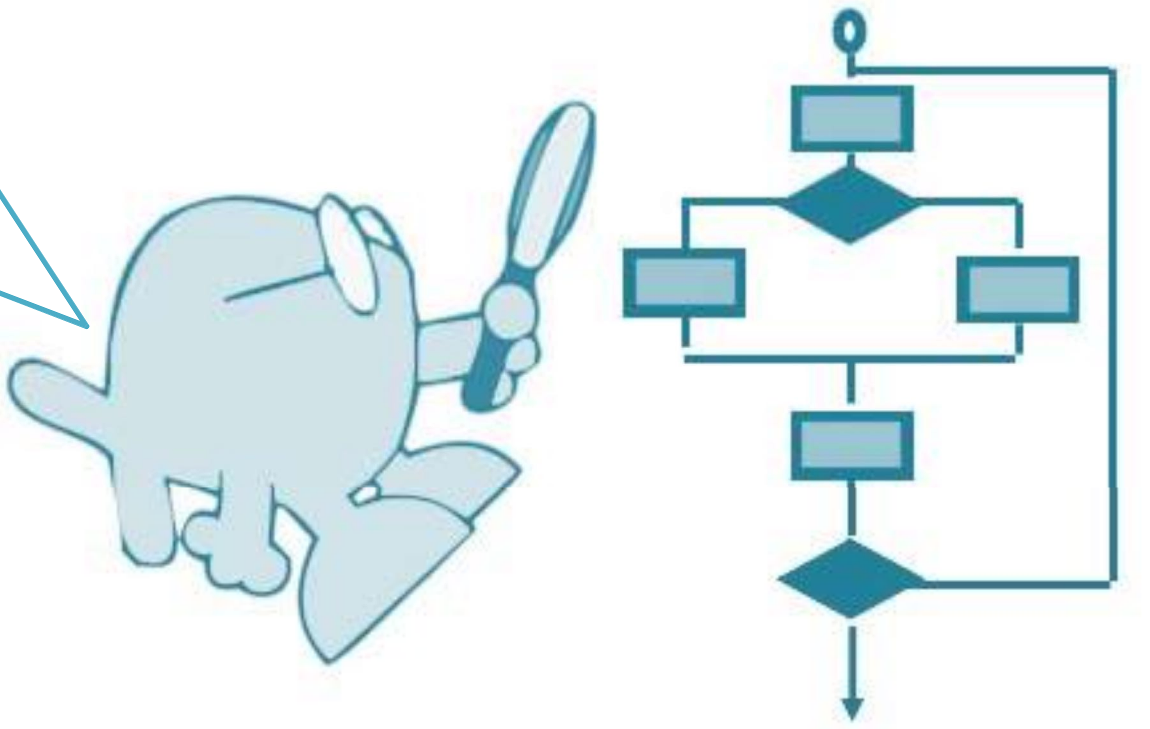
Upper boundary

White Box Testing

- □ Also known as structural testing
- □ White Box Testing is a software testing method in which the internal structure/design/implementation of the module being tested is known to the tester
- □ Focus is on ensuring that even abnormal invocations are handled gracefully
- □ Using white-box testing methods, you can derive test cases that
  - Guarantee that all independent paths within a module have been exercised at least once
  - Exercise all logical decisions on their true and false sides
  - Execute all loops at their boundaries
  - Exercise internal data structures to ensure their validity

White Box Testing

...our goal is to ensure that all statements and conditions have been executed at least once ...



It is applicable to the following levels of software testing

- Unit Testing: For testing paths within a unit
- Integration Testing: For testing paths between units
- System Testing: For testing paths between subsystems

White Box Testing Cont.

□ Advantages

- Testing can be commenced at an earlier stage as one need not wait for the GUI to be available.
- Testing is more thorough, with the possibility of covering most paths

□ Disadvantages

- Since tests can be very complex, highly skilled resources are required, with thorough knowledge of programming and implementation
- Test script maintenance can be a burden, if the implementation changes too frequently
- Since this method of testing is closely tied with the application being testing, tools to cater to every kind of implementation/platform may not be readily available

White-box testing strategies

- □ 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.
- □ White-box testing strategies
  - Statement coverage
  - Branch coverage
  - Path coverage

Statement Coverage

- □ It aims to design test cases so that every statement in a program is executed at least once
- □ Principal idea is unless a statement is executed, it is very hard to determine if an error exists in that statement
- □ Unless a statement is executed, it is very difficult to observe whether it causes failure due to some illegal memory access, wrong result computation, etc.

Statement Coverage Cont.

Consider the Euclid's GCD computation algorithm

```
int compute_gcd(x, y)
int x, y;
{
        while (x! = y){
        if (x>y) then
                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.

Branch coverage

- □ In the branch coverage based testing strategy, test cases are designed to make each branch condition to assume true and false values in turn
- □ It is also known as edge Testing as in this testing scheme, each edge of a program’s control flow graph is traversed at least once
- □ Branch coverage guarantees statement coverage, so it is stronger strategy compared to Statement Coverage.

Path Coverage

- □ In this strategy test cases are executed in such a way that every path is executed at least once
- □ All possible control paths taken, including
  - All loop paths taken zero, once and multiple items in technique
  - The test case are prepared based on the logical complexity measure of the procedure design
- □ Flow graph, Cyclomatic Complexity and Graph Metrices are used to arrive at basis path.

Grey Box Testing

- Combination of white box and black box testing
- Tester has access to source code, but uses it in a restricted manner
- Test cases are still written using specifications based on expected outputs for given input
- These test cases are informed by program code structure

Configuration Testing

- □ Configuration variability and instability are important factors that make WebApp testing a challenge.
- □ Hardware, operating system(s), browsers, storage capacity, network communication speeds, and a variety of other client-side factors are difficult to predict for each user.
- □ One user's impression of the WebApp and the manner in which he/she interacts with it can differ significantly.
- □ Configuration testing is to test a set of probable client-side and server-side configurations
  - to ensure that the user experience will be the same on all of them and,
  - to isolate errors that may be specific to a particular configuration

Test Plan

- What: a document describing the scope, approach, resources and schedule of intended testing activities; identifies test items, the features to be tested, the testing tasks, who will do each task and any risks requiring contingency planning;
- Who: QA;
- When: (planning)/design/coding/testing stage(s);

Test documentation

- **Test Plan (cont'd)**
  - – Why:
    - Divide responsibilities between teams involved; if more than one QA team is involved (ie, manual / automation, or English / Localization) – responsibilities between QA teams ;
    - Plan for test resources / timelines ;
    - Plan for test coverage;
    - Plan for OS / DB / software deployment and configuration models coverage.
  - - QA role:
    - Create and maintain the document;
    - Analyze for completeness;
    - Have it reviewed and signed by Project Team leads/managers.

Test documentation

- **Test Case**
  - What: a set of inputs, execution preconditions and expected outcomes developed for a particular objective, such as exercising a particular program path or verifying compliance with a specific requirement;
  - Who: QA;
  - When: (planning)/(design)/coding/testing stage(s);
  - Why:
    - Plan test effort / resources / timelines;
    - Plan / review test coverage;
    - Track test execution progress;
    - Track defects;
    - Track software quality criteria / quality metrics;
    - Unify Pass/Fail criteria across all testers;
    - Planned/systematic testing vs Ad-Hoc.

Test documentation

- **Test Case (cont'd)**
  - Five required elements of a Test Case:
    - ID – unique identifier of a test case;
    - Features to be tested / steps / input values – what you need to do;
    - Expected result / output values – what you are supposed to get from application;
    - Actual result – what you really get from application;
    - Pass / Fail.

Test documentation

- **Test Case (cont’d)**
  - – Optional elements of a Test Case:
    - Title – verbal description indicative of testcase objective;
    - Goal / objective – primary verification point of the test case;
    - Project / application ID / title – for TC classification / better tracking;
    - Functional area – for better TC tracking;
    - Bug numbers for Failed test cases – for better error / failure tracking (ISO 9000);
    - Positive / Negative class – for test execution planning;
    - Manual / Automatable / Automated parameter etc – for planning purposes;
    - Test Environment.

Test documentation

Test Case (cont’d)

- Inputs:
  - Through the UI;
  - From interfacing systems or devices;
  - Files;
  - Databases;
  - State;
  - Environment.
- Outputs:
  - To UI;
  - To interfacing systems or devices;
  - Files;
  - Databases;
  - State;
  - Response time.

Test documentation

- **Test Case (cont'd)**
  - Format – follow company standards; if no standards – choose the one that works best for you:
    - MS Word document;
    - MS Excel document;
    - Memo-like paragraphs (MS Word, Notepad, Wordpad).
  - Classes:
    - Positive and Negative;
    - Functional, Non-Functional and UI;
    - Implicit verifications and explicit verifications;
    - Systematic testing and ad-hoc;

Summary

□ Coding Standard and Guidelines

□ Code Review, Walk Throughand Inspection

□ Software Documentation

□ Test Strategies for Conventional Software

- Unit Testing
- Integration Testing
- Validation Testing
- Alpha and Beta Test
- System Testing
- Acceptance Testing

□ White Box and Black Box Testing

□ Testing Object Oriented Applications

- Testing Web Applications
- Dimensions of Quality
- Content Testing
- User Interface Testing
- Component-Level Testing
- Navigation Testing
- Configuration Testing
- Security Testing
- Performance Testing

□ Verification and Validation