

# Software Testing Strategies

---

SCS3207/ IS3103

# Software Testing

---

Software testing was the first software quality assurance tool applied to control the software product's quality before its shipment or installation at the customer's premises.

At first, testing was confined to the final stage of development, after the entire package had been completed.

Later, SQA professionals were encouraged to extend testing to the partial in-process products of coding,

- which led to software module (unit) testing and integration testing.

# Software Testing cont.

---

❖ Testing is certainly not the only type of SQA tool applied to software code.

Additional tools are code inspections and code walkthroughs, methods implemented on code printout without actually running the program.

These procedures, which are similar to those applied in design inspection and walkthroughs, yield good results in identifying code defects.

Nevertheless, these tools, because they are based solely on the review of documents, can never replace testing, which examines the software product's functionality in the form actually used by the customer.

# Code Inspections

---

The main purpose of code inspection is to find defects and it can also spot any process improvement if any.

An inspection report lists the findings, which include metrics that can be used to aid improvements to the process as well as correcting defects in the document under review.

Preparation before the meeting is essential, which includes reading of any source documents to ensure consistency.

Inspections are often led by a trained moderator, who is not the author of the code.

# Code Inspections cont..

---

The inspection process is the most formal type of review based on rules and checklists and makes use of entry and exit criteria.

It usually involves peer examination of the code and each one has a defined set of roles.

After the meeting, a formal follow-up process is used to ensure that corrective action is completed in a timely manner.



# Code Walkthrough

---

Code Walkthrough is a form of peer review in which a programmer leads the review process and the other team members ask questions and spot possible errors against development standards and other issues.



# Software testing – strategies

---

## White box testing

- data processing and calculation correctness tests
- path coverage vs. line coverage
- McCabe's cyclomatic complexity metrics
- software qualification and reusability testing
- advantages/disadvantages of white box testing

## Black box testing

- Equivalence classes for output correctness tests
- Operation / Revision / Transition factor testing classes
- advantages/disadvantages of white box testing

# Definition

---

## Frame 9.1 Software tests – definition

Software testing is a **formal** process carried out by a **specialized testing team** in which a software unit, several integrated software units or an entire software package are examined by **running the programs on a computer**. All the associated tests are performed according to **approved test procedures** on **approved test cases**.



# Objectives

---

## **Frame 9.2**   **Software testing objectives**

### ***Direct objectives***

- To identify and reveal as many errors as possible in the tested software.
- To bring the tested software, after correction of the identified errors and retesting, to an acceptable level of quality.
- To perform the required tests efficiently and effectively, within budgetary and scheduling limitations.

### ***Indirect objective***

- To compile a record of software errors for use in error prevention (by corrective and preventive actions).

# Software Testing Strategies

---

There are basically two testing strategies:

1. “Big bang testing”: tests the software as a whole, once the completed package is available.
2. “Incremental testing”: tests the software piecemeal—
  - software modules are tested as they are completed (unit tests)
  - followed by groups of modules composed of tested modules integrated with newly completed modules (integration tests)
  - Once the entire package is completed, it is tested as a whole (system test)

# Big Bang vs. Incremental

---

Unless the program is very small and simple, applying the “big bang” testing strategy presents severe disadvantages.

- Identification of error in the entire software package when perceived as one “unit” is very difficult and, despite the vast resources invested, is not very effective.
- Moreover, performing perfect correction of an error in this context is frequently laborious.
- Obviously, estimates of the required testing resources and testing schedule tend to be rather fuzzy.

## Big Bang vs. Incremental cont..

---

In contrast, the advantages of incremental testing, because it is performed on relatively small software units, yields higher percentages of identified errors and facilitates their correction.

As a result, it is generally accepted that incremental testing should be preferred.

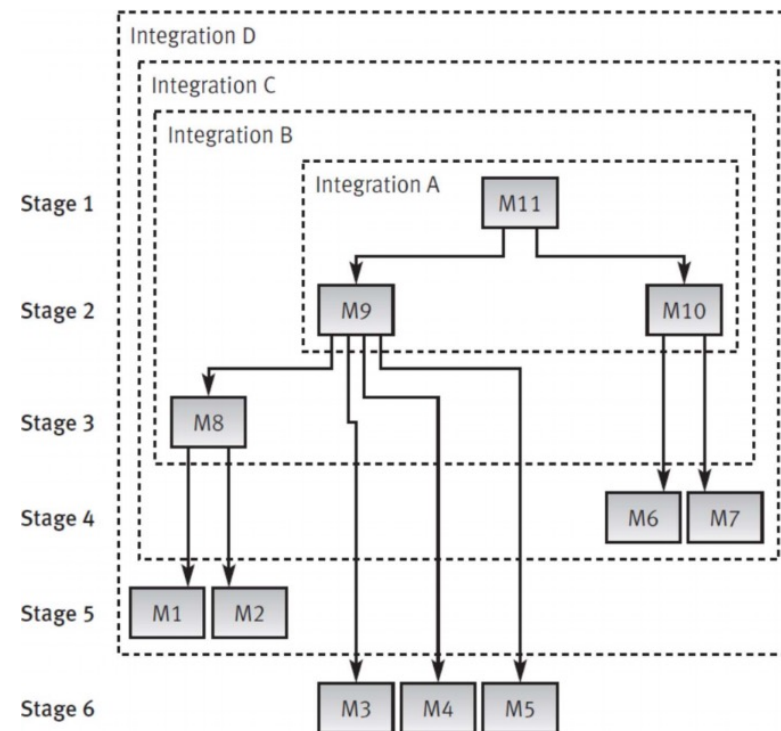
# Incremental Testing

---

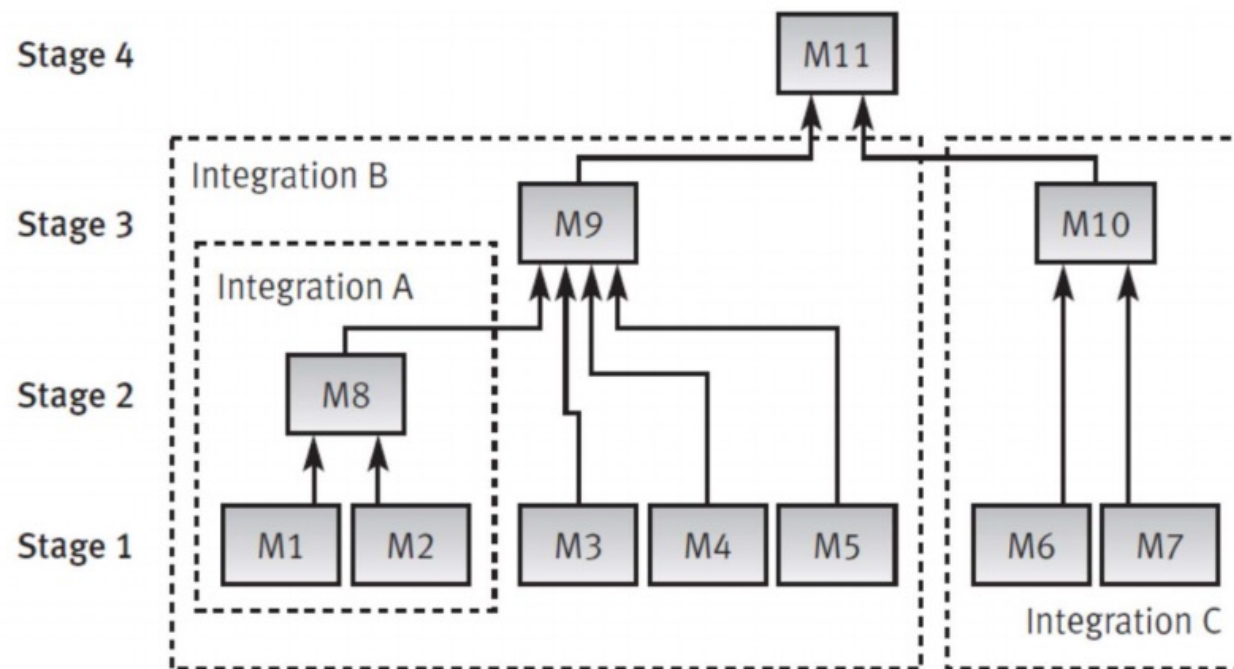
There are two basic incremental testing strategies: bottom-up and top-down.

1. In top down testing, the first module tested is the main module, the highest level module in the software structure; the last modules to be tested are the lowest level modules.
2. In bottom-up testing, the order of testing is reversed: the lowest level modules are tested first, with the main module tested last.

# Top-Down Testing



# Bottom-Up Testing



# Stubs and Drivers for Incremental Testing

---

Stubs and drivers are software replacement simulators required for modules not available when performing a unit test or an integration test.

In top-down testing of incomplete Systems: **Stubs** (also called “Dummy Module”) replaces an unavailable lower level module, subordinate to the module tested.

In bottom-up testing until the upper level modules are developed : A **driver** is a substitute module but of the upper level module that activates the module tested. The driver is passing the test data on to the tested module and accepting the results calculated by it.



# Software Test Classification

---

Classification according to testing concept:

1. Black box (functionality) testing : identifies bugs only according to malfunctioning of the software as revealed from its outputs, while disregarding the internal paths of calculations performed by the software.
2. White box (structural) testing : examines the internal paths of calculations in order to identify bugs.

# Software Test Classification cont..

---

Classification according to requirements:

An extension of McCall's model to the classification of the tests carried out to ensure full coverage of the respective requirements

This model classifies all software requirements into 11 software quality factors. The 11 factors are grouped into three categories

1. **Product operation factors** – Correctness, Reliability, Efficiency, Integrity, Usability.
2. **Product revision factors** – Maintainability, Flexibility, Testability.
3. **Product transition factors** – Portability, Reusability, Interoperability.

Factor category	Quality requirement factor	Quality requirement sub-factor	Test classification according to requirements
Operation	1. Correctness	1.1 Accuracy and completeness of outputs, accuracy and completeness of data	1.1 Output correctness tests
		1.2 Accuracy and completeness of documentation	1.2 Documentation tests
		1.3 Availability (reaction time)	1.3 Availability (reaction time) tests
		1.4 Data processing and calculations correctness	1.4 Data processing and calculations correctness tests
		1.5 Coding and documentation standards	1.5 Software qualification tests
	2. Reliability		2. Reliability tests
	3. Efficiency		3. Stress tests (load tests, durability tests)
	4. Integrity		4. Software system security tests
	5. Usability	5.1 Training usability	5.1 Training usability tests
		5.2 Operational usability	5.2 Operational usability tests
Revision	6. Maintainability		6. Maintainability tests
	7. Flexibility		7. Flexibility tests
	8. Testability		8. Testability tests
Transition	9. Portability		9. Portability tests
	10. Reusability		10. Reusability tests
	11. Interoperability	11.1 Interoperability with other software	11.1 Software interoperability tests
		11.2 Interoperability with other equipment	11.2 Equipment interoperability tests

# Test Scenarios and Test Cases

---

Test scenarios provides high-level description of the functionalities we need to test for.

Test cases are the conditions which we test for any particular scenario and test cases provides more details of functionalities.

# When Bugs are Found

---

Bug/Defect:

This is the output of testing effort .

Any behavior of the system which is not according to the requirement and any behavior which negatively impacts the system is a bug/defect.

Bug can be related to functional behavior, UI behavior, Usability behavior, Performance behavior and Security behavior of the application.



# Bug/Defect Life Cycle

---

It is a cyclic process which a defect follows through during its lifetime.

It begins when a tester logs the bug and ends when he decides to close it after thorough verification.

Therefore, the software bug life cycle is related to the defect found during testing.

# White Box Testing

---

Realization of the white box testing concept requires verification of every program statement and comment.

White box testing enables performance of:

- data processing and calculations correctness tests (white box correctness test),
- software qualification tests,
- maintainability tests and
- reusability tests.

# Data processing and calculation correctness tests

---

Applying the concept of white box testing, which is based on checking the data processing for each test case, immediately raises the question of coverage of a vast number of possible processing paths and the multitudes of lines of code.

Two alternative approaches have emerged:

- Path coverage: Every computational operation in the sequence of operations created by each test case (“path”) must be examined.
- Line coverage: This type of verification allows us to decide whether the processing operations and their sequences were programmed correctly for the path in question, but not for other paths.



# Path Coverage

---

“Path coverage” is defined as the percentage of possible paths of software processing activated by the test cases.

- Different paths in a software module are created by the choice in conditional statements, such as IF–THEN–ELSE or DO WHILE or DO UNTIL.
- Ideally one test case for each possible path → Impractical
- Directed mainly to high risk software modules

# Example: Imperial Taxi Service

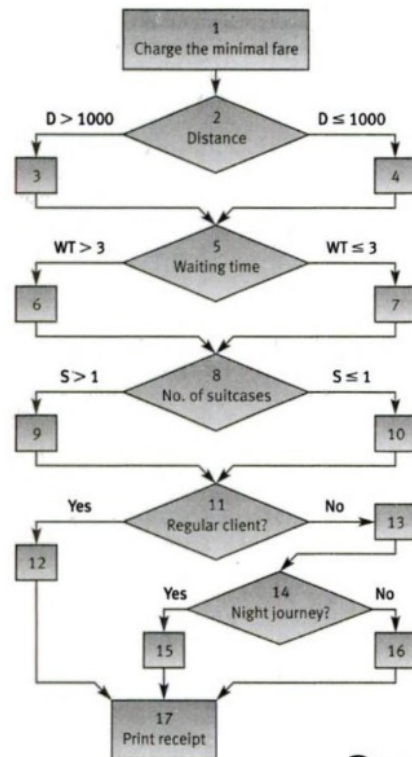


Table 9.3: The Imperial Taxi example – the full list of paths

No.	The path
1	1-2-3-5-6-8-9-11-12-17
2	1-2-3-5-6-8-9-11-13-14-15-17
3	1-2-3-5-6-8-9-11-13-14-16-17
4	1-2-3-5-6-8-10-11-17
5	1-2-3-5-6-8-10-11-13-14-15-17
6	1-2-3-5-6-8-10-11-13-14-16-17
7	1-2-3-5-7-8-9-11-12-17
8	1-2-3-5-7-8-9-11-13-14-15-17
9	1-2-3-5-7-8-9-11-13-14-16-17
10	1-2-3-5-7-8-10-11-12-17
11	1-2-3-5-7-8-10-11-13-14-15-17
12	1-2-3-5-7-8-10-11-13-14-16-17
13	1-2-4-5-6-8-9-11-12-17
14	1-2-4-5-6-8-9-11-13-14-15-17
15	1-2-4-5-6-8-9-11-13-14-16-17
16	1-2-4-5-6-8-10-11-12-17
17	1-2-4-5-6-8-10-11-13-14-15-17
18	1-2-4-5-6-8-10-11-13-14-16-17
19	1-2-4-5-7-8-9-11-12-17
20	1-2-4-5-7-8-9-11-13-14-15-17
21	1-2-4-5-7-8-9-11-13-14-16-17
22	1-2-4-5-7-8-10-11-12-17
23	1-2-4-5-7-8-10-11-13-14-15-17
24	1-2-4-5-7-8-10-11-13-14-16-17

# Line Coverage

---

The line coverage concept requires that for full line coverage, every line of code to be executed at least once during the process of testing.

Basis path testing involves execution of all possible blocks in a program and achieves maximum path coverage with the least number of test cases.

This concept requires far fewer cases than path coverage, but leaves most of the possible paths untested.

**Table 9.4: The Imperial Taxi example – the minimum number of paths**

No.	The path
1	1-2-3-5-6-8-9-11-12-17
23	1-2-4-5-7-8-10-11-13-14-15-17
24	1-2-4-5-7-8-10-11-13-14-16-17

# Line Coverage cont..

---

It is used to calculate and measure the number of statements in the source code which can be executed given the requirements.

Line Coverage = (Number of statements exercised / Total number of statements) \* 100


Line Coverage allows the tester to identify

- Unused Statements
- Dead Code
- Unused Branches

# Line Coverage Example

---

```
Prints (int a, int b) {  
    int result = a+ b;  
    If (result> 0)  
        Print ("Positive", result)  
    Else  
        Print ("Negative", result)  
}
```



If A=3 and B=9

Number of executed statements = 5

Total Number of Statement = 7

Line coverage =  $5/7 = 71\%$

If A= -3 and B= -9

Line coverage = ?%

# McCabe's cyclomatic complexity metrics

---

The cyclomatic complexity metrics developed by McCabe (1976) measures the complexity of a program or module at the same time as it determines the maximum number of independent paths needed to achieve full line coverage of the program.

An Independent path is any path on the program flow graph that includes at least one edge that is not included in any former independent paths.

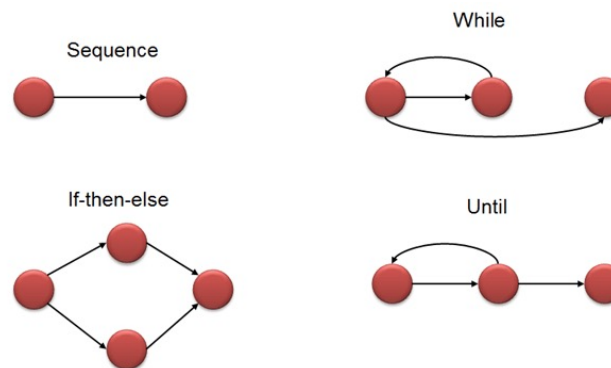
# McCabe's cyclomatic complexity metrics cont..

---

Flow Graph notation for a program depict several nodes connected through the edges.

These Flow diagrams for statements like if-else, While, until and normal sequence of flow.

Nodes represent processing tasks while edges represent control flow between the nodes.



# McCabe's cyclomatic complexity metrics cont (2)..

---

The cyclomatic complexity metric  $V(G)$

$$V(G) = R$$

$$V(G) = E - N + 2$$

$$V(G) = P + 1$$

Where,

$R$  = the number of regions

$E$  = number of edges

$N$  = number of nodes

$P$  = number of decisions (nodes having more than one leaving edge)



# McCabe's cyclomatic complexity metrics cont..

The cyclomatic complexity metric  $V(G)$  for the Imperial Taxi Service example:

$$V(G) = R = 6$$

$$V(G) = E - N + 2 = 21 - 17 + 2 = 6$$

$$V(G) = P + 1 = 5 + 1 = 6$$

Table 9.6: The ITS example – the maximum set of independent paths

Path no.	The path	Edges added by the path	Number of edges added by the path
1	1-2-3-5-6-8-9-11-12-17	1-2, 2-3, 3-5, 5-6, 5-8, 8-9, 9-11, 11-12, 12-17	9
2	1-2-4-5-6-8-9-11-12-17	2-4, 4-5	2
3	1-2-3-5-7-8-9-11-12-17	5-7, 7-8	2
4	1-2-3-5-6-8-10-11-12-17	8-10, 10-11	2
5	1-2-3-5-6-8-9-11-13-14-15-17	11-13, 13-14, 14-15, 15-17	4
6	1-2-3-5-6-8-9-11-13-14-16-17	14-16, 16-17	2

Programs with  $V(G) < 5$  are considered simple.

# Software Qualification Testing

---

Qualification testing is of crucial importance for coding in the development and maintenance stages.

Software that qualifies is coded and documented according to standards, procedures and work instructions.

Software qualification testing determines whether software development responded positively to questions reflecting a specific set of criteria.

- Does the code fulfil the code structure instructions and procedures, such as module size, application of reused code, etc.?
- Does the coding style fulfil coding style procedures?
- Do the internal program documentation and “help” sections fulfil coding style procedures?

# Maintainability Tests

---

Maintainability tests : refer to special features, such as those installed for detection of causes of failure, module structures that support software adaptations and software improvements, etc.

It basically defines that how easy it is to maintain the system.

Maintenance Testing is done on the already deployed software. The deployed software needs to be enhanced, changed or migrated to other hardware. The Testing done during this enhancement, change and migration cycle is known as maintenance testing.

# Types of Maintenance Strategy

---

Corrective maintenance – implemented right after a defect has been detected. The maintainability of a system can be measured in terms of the time taken to diagnose and fix problems identified within that system.

Perfective maintenance – Enhancements. The maintainability of a system can also be measured in terms of the effort taken to make required enhancements to that system.

- This can be tested by recording the time taken to achieve a new piece of identifiable functionality such as a change to the database, etc. A number of similar tests should be run and an average time calculated. The outcome will be that it is possible to give an average effort required to implement specified functionality.

## Types of Maintenance Strategy cont..

---

Adaptive maintenance – Adapting to changes in environment. The maintainability of a system can also be measured in terms on the effort required to make required adaptations to that system. This can be measured in the way described above for perfective maintainability testing.

Preventive maintenance –Maintenance carried out at predetermined intervals or according to prescribed criteria, aimed at reducing the failure risk or performance degradation of the equipment. This refers to actions to reduce future maintenance costs.

# Reusability Tests

---

Reusability tests examine the extent that reused software is incorporated in the package and the adaptations performed in order to make parts of the current software reusable for future software packages



# Black box testing

---

Black box (functionality) testing : identifies bugs only according to malfunctioning of the software as revealed from its outputs, while disregarding the internal paths of calculations performed by the software.

Some testing classes are unique to black box testing.

Still, due to the special characteristics of each testing strategy and the test classes unique to white box testing, black box testing cannot automatically substitute for white box testing.

# Equivalence classes for output correctness tests

---

Output correctness tests are, in most cases, among the tests that consume the greater part of testing resources.

Equivalence partitioning or equivalence class partitioning (ECP) is a software testing technique that divides the input data of a software unit into partitions of equivalent data from which test cases can be derived. In principle, test cases are designed to cover each partition at least once.

The output correctness tests apply the concept of test cases.



# Equivalence classes for output correctness tests cont..

---

Improved choice of test cases can be achieved by the efficient use of equivalence class partitioning.

- An equivalence class (EC) is a set of input variable values that produce the same output results or that are processed identically.
- Test cases are defined separately for the valid and invalid ECs.
- Importantly, as the equivalence class method is a black box method, equivalence class partitioning is based on software specification documentation, not on the code.

# Test cases and boundary values

---

According to the definition of equivalence classes, one test case should be sufficient for each class.

However, when equivalence classes cover a range of values (e.g. monthly income, apartment area), the tester has a special interest in testing border values when these are considered to be error prone.

In these cases, the preparation of three test cases – for mid range, lower boundary and upper boundary values – is recommended.



# Equivalence Class Partitioning Example

---

AGE

Enter Age

\* Accepts value from 18 to 60

Equivalence Class Partitioning		
Invalid	Valid	Invalid
$\leq 17$	18-60	$\geq 61$

# Revision Factor Testing

---

Easy revision of software is a fundamental factor assuring a software package's successful, long service and its successful sales to larger user populations.

- Maintainability tests
- Flexibility tests
- Testability tests

# Transition factor testing classes

---

The software characteristics required to be operative, with minor adaptations, in different environments, and those needed to incorporate reused modules or to permit interfacing with other software packages are all among the transition features required from a software system, especially for commercial software packages aimed at a wide range of customers.

- (1) Portability tests
- (2) Reusability tests
- (3) Tests for interoperability requirements:
  - Software interfacing tests
  - Equipment interfacing tests.

# Exercise

---



1. What are the benefits and drawbacks of the top down approach?
2. What are the benefits and drawbacks of the bottom up approach?
3. What are the advantages of path testing?
4. What are the advantages and disadvantages of white box testing?
5. What are the advantages and disadvantages of black box testing?