Coding and Testing

Unit - 4

Coding

- The goal of the coding or programming activity is to implement the design in the best possible manner.
- Coding affects both testing and maintenance profoundly.
- The goal should of coding is not to simplify the job of programmer; but to simplify the job of tester and maintainer.
- Programs should not be constructed so that they are easy to write, but so that they are easy to read and understand.

Programming Principles

- The main task before a programmer is to write quality code with few bugs in it.
- The additional constrain is to write code quickly.
- Writing a good quality code is a skill that can only be acquired by practice.
- Good programming is a practice, independent of target programming language.
- Well structured programming languages make the programmer's job simpler.

Common Coding Errors

- Memory Leaks: Memory is allocated to a programme which is not freed subsequently.
- Freeing an Already Freed Resource
- NULL dereferencing: attempt to access the contents of a location that points to NULL.
- Lack of Unique Addresses: aliasing creates violation of unique addresses when we expect different address. [strcat(src,dstn) ; if src is aliased to dstn, then its runtime error.]

Common Coding Errors

- Synchronization Errors:
  - Deadlocks
  - Race conditions
  - Inconsistent Synchronization
- Array Index Out of Bounds:
- Arithmetic Exceptions:
- Off by One:

Common Coding Errors

- Enumerated Data Types: while assigning values to enum data types, overflow and underflow of data is possible.
- Illegal use of & instead of &&
- String Handling Errors
- Buffer Overflow

Structured Programming

- Structured programming practice helps develop programs that easier to understand.
- A program has a static and dynamic structure.
- The static structure is the structure of the text of the program, which is usually just a linear organization of statements of the program.
- The dynamic structure of the program is the sequence of statements executed during the execution of the program.

Structured Programming

- Same static and dynamic structure will help to understand the dynamic behaviour of the programme.
- The goal of the structured programming is to ensure that the static structure and the dynamic structure are the same.
- The structured programming is often regarded as “goto-less” programming.
- No meaningful programme can be written as a sequence of simple statements without any branching and repetition.
- The key property of a structured statement is the it has a single-entry and a single-exit.

Structured Programming

- The most common single-entry and single-exit statements are
  - Selection (if-else type)
  - Iteration (loops)
  - Sequencing (s1;s2;.....)
- Any piece of code with a single-entry and sing-exit cannot be considered a structured construct.
- The basic objective of using structured constructs is to linearized the control flow so that the execution behaviour is easier to understand.
- Any unstructured construct should be used only if the structured alternative is harder to understand.

Information Hiding

- The software solution to a problem always contains data structures that are meant to represent information in the problem domain.
- A piece of information in the problem domain is used only in a limited number of ways in the problem domain.
- *"When the information is represented as data structures, only some defined operations should be performed on the data structures."* is the principle of information hiding.

Information Hiding

- The information captured in the data structures should be hidden from the rest of the system.
- Only the access function on the data structures that represent the operations performed on the information should be visible.
- Rest of the modules in the software should only use these access function to access and manipulate the data structures.

Information Hiding

- Information hiding can reduce the coupling between modules and make the system more maintainable.
- Many of the older languages like Pascal, C, FORTRAN, do not provide mechanisms to support data abstraction.
- With such languages, information hiding can be supported only by a disciplined use of the language, and by programmer's efforts.
- Most modern OO languages provide linguistic mechanisms to implement information hiding.

Coding Standards

- Coding standards provide guidelines for programmers regarding:
  - Naming (Naming Conventions)
  - File organization(how files should be named, what file should contain, etc.)
  - Statements and declarations (declaration and executable statements in the source code)
  - Layout and comments (to make programme more readable)

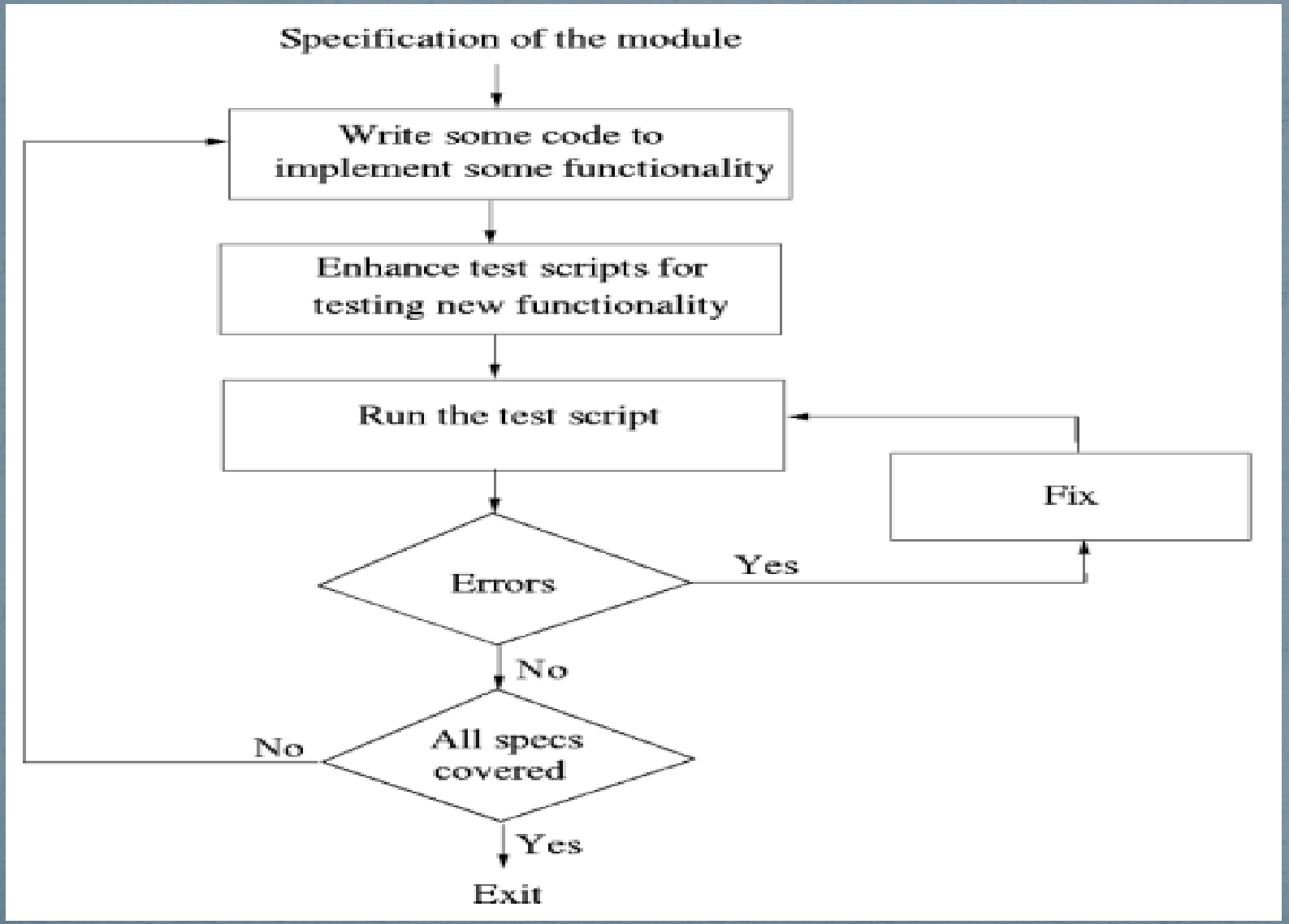
Coding Process

- The coding activity starts when
  - some form of design has been done and
  - the specifications of the modules to be developed are available.
- In a top-down implementation, we start by assigning modules at the top of the hierarchy and proceed to the lower levels.
- In a bottom-up implementation, the development starts with first implementing the modules at the bottom of the hierarchy and proceeds up.
- Coding process directly impacts on integration and testing.

Incremental Coding Process

- Code is developed incrementally.
- Write code for implementing only part of the functionality of the module.
- This code is compiled and tested with some quick tests to check the code that has been written so far.
- When the code passes these tests, the developer proceeds to add further functionality to the code, which then tested again.

Incremental Coding Process



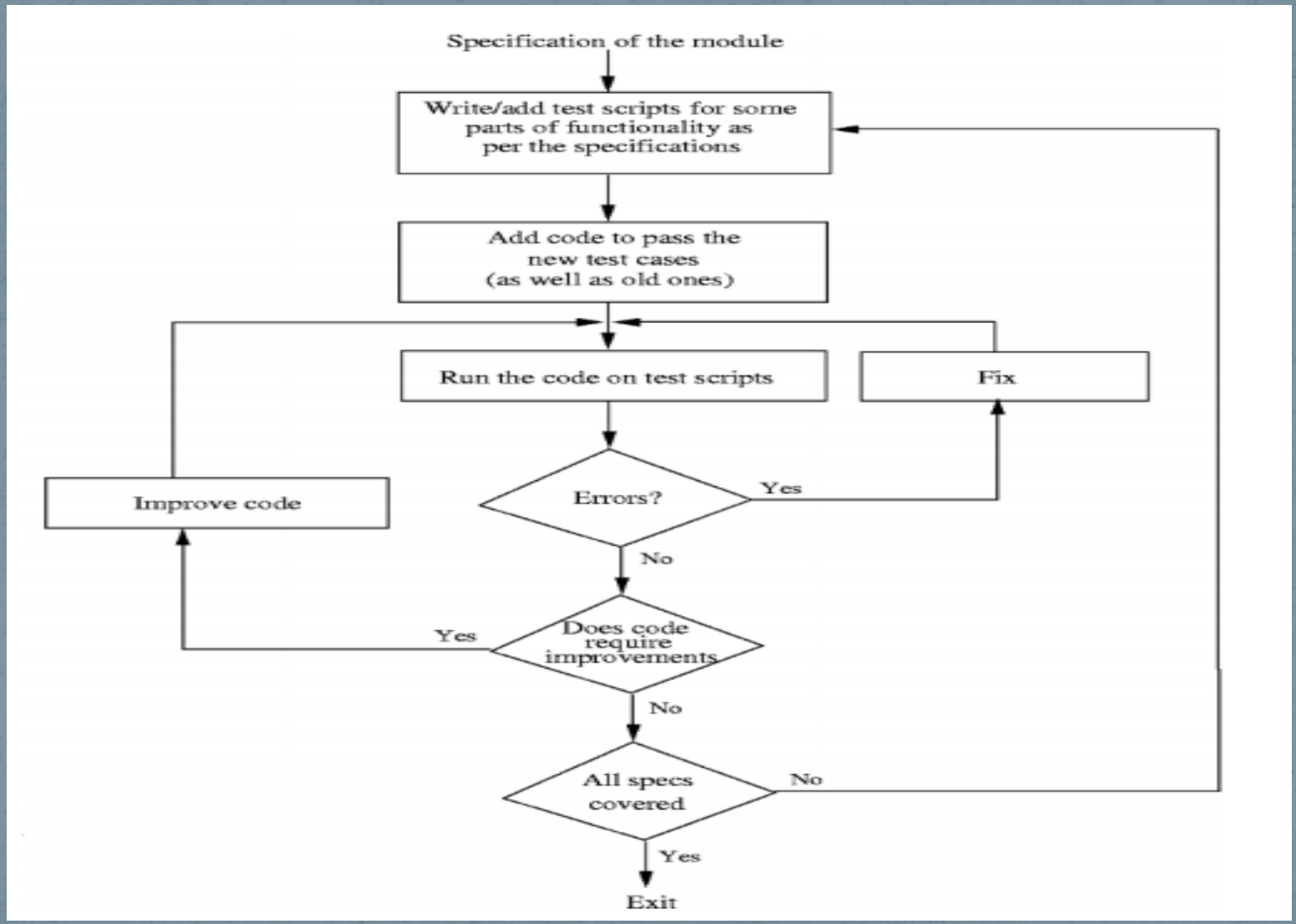
Advantages of incremental coding process

- The code is incrementally tested.
- It facilitate debugging- and error found in some testing can be safely attributed to code that was added since last successful testing.
- For automatic test scripts, testing can be done as frequently as desired, and new test cases can be added easily.
- These test scripts can also be used with some enhancements for the final unit testing that is often done before checking in the module.

Test Driven Development (TDD)

- In TDD, a programmer first writes the test scripts, and then writes the code to pass the tests.
- The whole process is done incrementally.
- Test scripts are written based on the specification and code being written to pass the tests.
- This new approach has been adopted in the extreme programming (XP) methodology.

TDD


Specification of the module
Write/add test scripts for some parts of functionality as per the specifications
Add code to pass the new test cases (as well as old ones)
Run the code on test scripts
Fix
Errors?
Yes
Improve code
No
Does code require improvements
Yes
No
All specs covered
No
Yes
Exit

TDD

- ✓ The approach says that you write just enough code to pass the tests.
- ✓ Writing test cases before the code is written makes the development usage driven.
- ✓ Prioritization for code development- first few tests are likely to focus on using the main functionality.
- ✗ Special cases may not get handled because of less priority.

Pair Programming

- The code is not written by individual programmers but by a pair of programmers.
- The process is that, one person will type the programme while the other will actively participate and constantly review what is being typed.
- The roles are rotated frequently making both equal partners and having similar roles.

Pair Programming

- ✓ The code is getting reviewed as it its being typed.
- ✓ Better decisions being taken about data structures, algorithms, interfaces, logic, etc.
- ✓ Special conditions, which frequently result in errors, are also handled in a better manner.
- ✗ it may result in loss of productivity by assigning two people for a programming task.
- ✗ issues of accountability and code ownership when the pairs are not fixed and rotate.

Refactoring

- Code also changes when requirements change or when new functionality is added.
- Due to the changes being done, we often ended up with code whose design is not as good as it should be.
- Once the design embodied in the code becomes complex, then enhancing the code to accommodate required changes becomes more complex, time consuming, and error prone.
- The productivity and quality starts decreasing.

Refactoring

- Refactoring is the technique to improve existing code and prevent this design decay with time.
- Refactoring is part of coding in that it is performed during the coding activity, but is not regular coding.
- Refactoring generally results in one or more of the following:
  - Reduced coupling
  - Increased cohesion
- To minimized the side effects of refactoring:
  - Refactor in small steps
  - Have test scripts available to test existing functionality

Verification

- Code Inspections
- Static Analysis
- Unit Testing

Code Inspections

- Code inspections are generally held after code has been successfully compiled and other forms of static tools have been applied.
- The documentation to be distributed to the inspection team includes the code to be reviewed and the design document.
- The team for inspection should include the programmer, the designer, and the tester.

Static Analysis

- Analysis of programs by methodically analyzing the program text is called static analysis.
- Static analysis is usually performed mechanically by the aid of software tools.
- During static analysis the program itself is not executed, but the program text is the input to the tools.
- The aim of static analysis is to detect errors or potential errors in the code and to generate information that can be useful in debugging.

Static Analysis v/s Compilers

- Compilers perform some limited static analysis.
- However, the analysis performed by compilers focuses around code generation and not defect detection.
- Static analysis tools explicitly focus on detecting errors.

Unit Testing

- The unit testing is like regular testing where programs are executed with some test cases except tha***t the focus is on testing smaller programs or modules called units.***
- A unit may be a function, a small collection of functions, a class, or a small collection of classes.
- During unit testing the tester (programmer) will execute the unit for a variety of test cases and study the actual behaviour of the unit being tested for these test cases.

Software Testing

- In a software development project, errors can be introduced at any stage during the development.
- Though errors are detected after each phase by techniques like inspections, some errors remain undetected.
- *"TESTING IS THE ACTIVITY WHERE THE ERRORS REMAINING FROM ALL THE PREVIOUS PHASES MUST BE DETECTED."*
- Hence testing performs a very critical role for ensuring quality.

Software Testing

- During testing, the software to be tested is executed with a set of test cases and the behaviour of the system for the test cases is evaluated to determine if the system is performing as expected.
- Clearly, the success of testing in revealing errors depends critically on the test cases.
- Testing a large system is a complex activity, and like any complex activity it has to be broken into smaller activities. (use of incremental testing).

Error, Fault, and Failure

- Error
  - The term error is used in two different ways.
  - Error refers to the discrepancy between a computed, observed or measured value and the true, specified or theoretically value.
  - Error refers to the difference between the actual output of a software and the correct output.
  - In this interpretation, error essentially is a measure of the difference between the actual and ideal.
  - Error is also used to refer to human action that results in software containing a defect or fault.

Error, Fault, and Failure

- Fault
  - Fault is a condition that causes the software to fail to perform its required function.
  - A fault is the basic reason for software malfunction and is synonyms with the commonly used phrase “bug” .

Error, Fault, and Failure

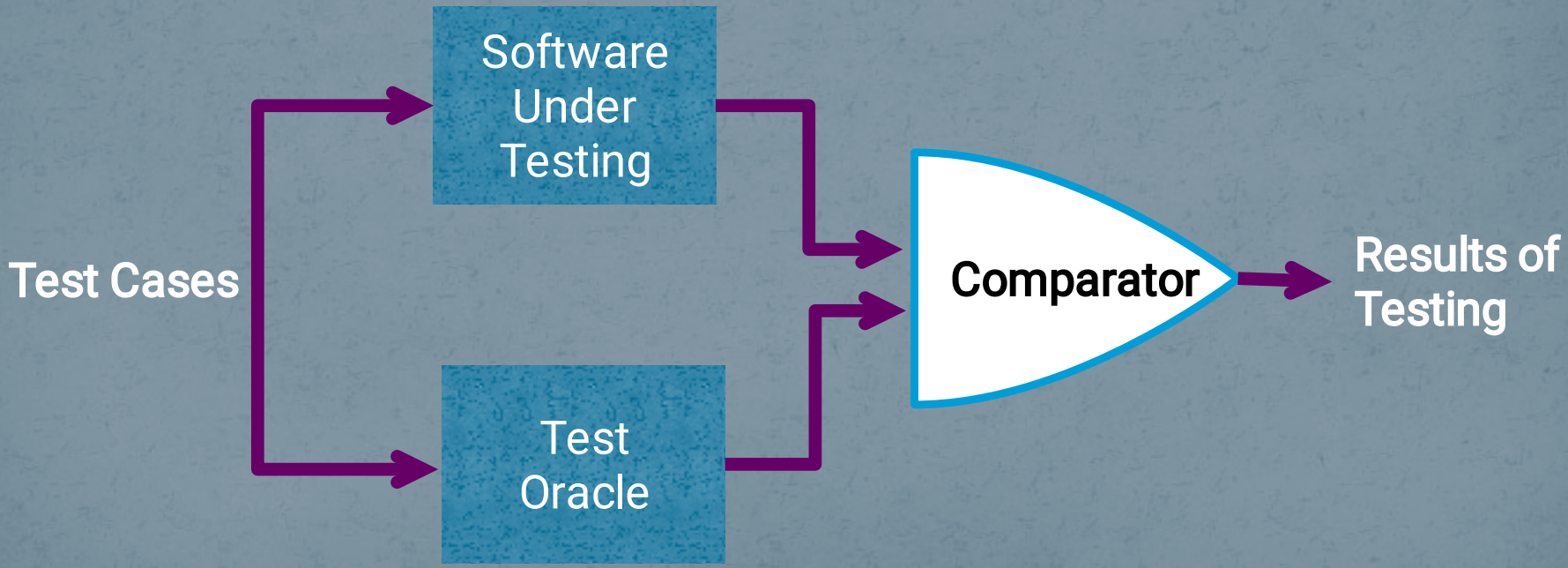
- Failure
  - Failure is the inability of a system or a component to perform a required function according to its specifications.
  - A software failure occurs if the behaviour of the software is different from the specified behaviour.
  - Failures may be caused due to functional as well as performance reasons.
  - A failure is produced only when there is a fault in the system.
  - However, the presence of the fault does not guarantee a failure.
  - In other words, faults have the potential to cause failures and their presence is a necessary but not sufficient condition for failure to occur.

Test Oracle

- For testing any program we need to have a description of the expected behaviour of the program and a method of determining whether the observed behaviour conforms to the expected behaviour.
- For this we need a Test Oracle..
- A test oracle is a mechanism, different from the program itself, that can be used to check the correctness of the output of the program for the test cases.
- Conceptually, we can consider testing as a process in which the test cases are given to test oracle and the program under testing.
- The output of the two is then compared to determine if the program behaved correctly for the test cases.

Test Oracle

Ideally, we would like to automated oracle, which always gives a correct answer. However, often the oracles are human beings, who mostly compute by hand what the output of the program should be.



Test Oracle

- As it is often extremely difficult to determine whether the behaviour conforms to the expected behaviour, our “human oracle” itself may make mistakes.
- As a result, when there is a discrepancy in the result of the program and the oracle, we have to verify the result produced by the oracle, before declaring that there is a fault in the program.
- This is one of the reasons why testing is so cumbersome and expensive.

Test Cases

- Having proper test cases is central to successful testing.
- The reason for this is that if there is a fault in a program, the program can still provide expected behaviour for many inputs.
- Only for the set of inputs that exercise the fault in the program will the output of the program deviate from the expected behaviour.
- Hence, it is fair to say that a testing process is as good as its test cases.
- Ideally we would like to determine a set of test cases such that successful execution of them implies that there are no errors in the program.
- This ideal goal cannot be achieved due to practical and theoretical constraints.

Test Cases

- Each test cases cost money, as effort is needed to generate the test cases, machine time is needed to execute the program for that test case, and more effort is needed to evaluate the results.
- Therefore we would also like to minimize the number of test cases needed to detect errors due to the reliability and economic constraints associated with selecting a proper set of test cases.
- The goal during selecting test cases is to ensure that if there is an error/fault in the program, then it is exercised by one of the test cases.
- An ideal test case set is one that succeeds (meaning that its execution reveals no errors) only if there are no errors in the program.

Test Cases

- One possible ideal set of test cases is one that includes all the possible inputs to the program. This is often called “Exhaustive Testing”
- However, exhaustive testing is impractical and unfeasible, as even for small programs the number of elements in the input domain can be extremely large.
- Hence, a realistic goal for testing is to select a set of test cases that is close to ideal.
- How should we select our test cases? On what basis should we include some element of program domain in the set of test cases and not include others?
- For this we need some testing criteria. That is, we define a criterion that we believe should be satisfied during testing.

Test Criteria

- The criterion then becomes the basis for test case selection and a set of test cases is selected that satisfy that criterion.
- There are two aspects of test case selection –

[1] Specifying a criterion for evaluating a set of test cases and

[2] a procedure for generating a set of test cases that satisfy a given criterion.

Test Criteria

- There are two desirable properties for a testing criterion:

[1] Reliability and [2] Validity

[1] Reliability

A criterion is reliable if all the sets (of test cases) that satisfy the criterion detect the same errors. That is, it is insignificant to which of the sets satisfying the criterion is chosen; every set will detect exactly the same errors.

[2] Validity

- A criterion is valid if for any error in the program, there is some set satisfying the criterion that will reveal the error.

Test Criteria

- A fundamental theorem of testing says that *"a testing criterion is valid and reliable, if a set satisfying that criterion succeeds (revealing no faults) then the program contains no errors ."*
- Getting a criterion that is reliable, valid and can be satisfied by a small number of test cases, is usually not possible.
- Even devising a criterion itself is very difficult.

Psychology of Testing

- Devising a set of test cases that will guarantee that all errors will be detected is not feasible.
- Moreover, there are no formal or precise methods for selecting test cases.
- Even though there are a number of heuristics and rules of thumb for deciding the test cases, selecting test cases is still largely a creative activity, which relies a lot on the ingenuity of the tester.
- Due to this reason, the psychology of the person performing the testing becomes important.

Psychology of Testing

- The aim of testing is often considered to be demonstrating that a program works by showing that it has no errors.
- This is the opposite of what testing should be viewed as.
- The basic purpose of the testing phase is to ensure some reliability of the software, or detect the errors that may be present in the program.
- Hence one should not start testing with the intent of showing that a program works; but the intent should be to show that a program does not work.
- With this mind we define testing as ‘Testing is the process of executing a program with the intent of finding errors’.

Black Box Testing

- Black box testing is done to validate all the functional requirements of the system by checking for incorrect or missing functions, interface errors etc.
- In black box testing, the tester only knows the inputs that can be given to the system and what output the system should give.
- The internal structure of the code is not considered while designing functional tests and hence are called “black box”.
- For each test case, input values, expected outputs and actual outputs are recorded.

White Box Testing

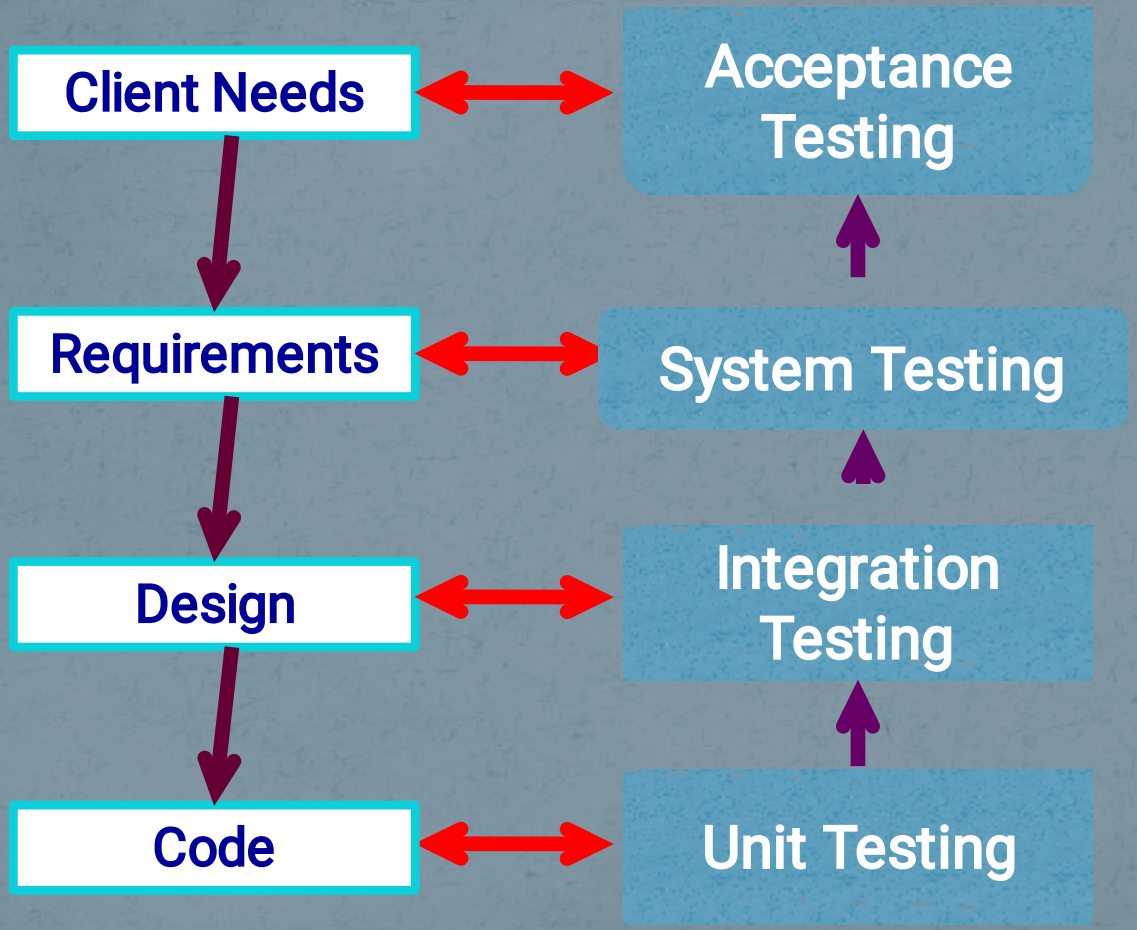
- White box testing is concerned with testing the implementation of the program.
- The intent of this testing is to exercise the different programming structures and data structures used in program.
- To test the structures of a programme, structural testing aims to achieve test cases that will force the desired coverage of different structures.
- The criteria for structural testing are generally quite precise as they are based on program structures, which are formal and precise.

Approaches for Testing

Black-Box	White-Box
Functional/Behaviour Testing (Focuses on functional requirements of the software).	Glass-box/Structural Testing (Focuses on the internal structure of the program).
Structure of the program is not considered.	Structure of the program is considered.
Test cases are decided solely on the basis of the requirements or specifications.	Test cases are decided solely on the basis of internals of the module or the program i. e. from an examination of the program's logic.
Types of error : incorrect or missing functions, interface errors, errors in external database access, behavior or performance errors, initialization or termination errors, etc.	Types of error : logical error, run-time error, data-flow error, looping error , etc.
Examples : Graph-based testing, Equivalence Partitioning, Boundary Value analysis, Cause-effect Graphing, Orthogonal Array Testing, pair-wise testing, State-based testing, etc.	Examples: Statement coverage testing, condition testing, data-flow testing, mutation testing, Basic Path Testing, Control Structure Testing, etc.

Testing Process

Levels of Testing



Unit Testing

- First level of testing.
- Different modules are tested against the specifications produced during design for the modules.
- Essentially for verification of the code produced during the coding phase and hence the goal is to test the internal logic of the modules.
- Done by the programmer of the module.
- Considered for integration and use by others.

Integration Testing

- Next level of testing.
- Tested modules are combined into subsystems, which are then tested.
- Goal : To see if the modules can be integrated properly.
- Emphasis is on testing interfaces between modules.
- This testing activity can be considered as testing the design.

System Testing

- Entire software system is tested.
- The reference document for this process is the requirements document.
- Goal : To see if the software meets its requirements.
- This essentially validation exercise and in many situations it is the only validation activity.

Acceptance Testing

- It is performed with realistic data of the client to demonstrate that the software is working satisfactorily.
- Focuses on the external behavior of the system.
- Mostly functional testing is performed at this level.

Regression Testing

- Regression testing is performed when some changes are made to an existing system.
- Changes are fundamental to software; any software must undergo changes.
- Frequently a change is made to “upgrade” the software by adding new features and functionality.
- The modified software needs to be tested to make sure that the new features to be added do indeed work.
- In such circumstances, testing has to be done to make sure that the modification has not had any undesired side effect of making some of the earlier services faulty.

Regression Testing

- That is besides ensuring the desired behaviour of the new services, testing has to ensure that the desired behaviour of the old services is maintained.
- For regression testing, some test cases that have been executed on the old system are maintained, along with the output produced by the old system.
- These test cases are executed again in the modified system and its output compared with the earlier output to make sure that the system is working as before on these test cases.
- Test cases for system should be properly documented for future use in regression testing.

Test Plan

- In general testing commences with a test plan and terminates with acceptance testing.
- A Test Plan is a general document for the entire project that
  - defines the scope, approach to be taken and the schedule of testing
  - identifies the test items for the entire testing process and the personnel responsible for the different activities of testing.
- The test planning can be done well before the actual testing commences and can be done in parallel with the coding and design activities.

Test Plan

- The inputs for forming the test plan are:
  1. Project Plan
  2. Requirements Document and
  3. System Design Document
- A test plan should contain the following:
  1. Test Unit Specification
  2. Features to be tested
  3. Approach for testing
  4. Test deliverables
  5. Schedule and task allocation