MODULE 4

CODING AND TESTING

- The goal of coding is to implement the design in best possible way.
- While coding, program should not be constructed so that it easy to write instead it should be understandable and readable.
- There are many different criteria for judging the program like readability, size of the program, execution time, and required memory.
- Readability and understandability are the main objectives that help in producing software that is more maintainable.

Programming Principles and Guidelines

- The main task before a programmer is to write quality code with few bugs in it.
- Good programming is a practice independent of target programming language.

Common Coding Errors

- Software errors are reality that all programmers have to deal with.
- Though errors can occur in wide variety of ways, some types of errors are found more commonly like:

(i) Memory leaks

- It is a situation where the memory is allocated to the program which is not freed subsequently.
- This error is common failures which occur in languages that do not have automatic garbage collection.
- They have little impact on small programs but drastic for long programs.
- A software program with memory leaks keeps consuming memory, till at some point of time the program may come to an exceptional halt because of lack of free memory.

(ii) Freeing an Already Freed Resource

- In programs, resources are first allocated and then freed.
- This error occurs when the programmer tries to free the already freed resource.
- The impact of this error is more severe if we have some malloc statement between the two free statements, there is a chance that the first freed location is now allocated to the new variable and the subsequent free will deallocate it.

(iii) NULL Dereferencing

- It occurs when we try to access the contents of location that points to NULL.
- It is a common occurring error which can bring software system down.
- It is also difficult to detect NULL dereferencing as it may occur only in some paths and under certain situations.

(iv) Lack of Unique Addresses

- Aliasing creates problems and among them is violation of unique addresses when we expect different addresses.
- For example, in string concatenation function, we expect source and destination addresses to be different.
- If this is not the case, it can lead to runtime errors.\

(v) Synchronization errors

- These errors are hard to find as they don’t occur so often but when occurs it causes serious damage to the system.
- There are different categories of synchronization errors and some of them are as follows:
  1. Deadlocks
  2. Race conditions
  3. Inconsistent synchronization

- Deadlock is a situation in which one or more threads mutually lock each other.
- Race condition occurs when two threads try to access the same resource and result of the execution depends on order of execution of errors.
- Inconsistent synchronization is also common error representing situation where there is a mix of locked and unlocked accesses to some shared variables.

(vi) Array Index Out of Bounds

- Array index goes out of bounds, leading to exceptions.
- Array index values cannot be negative or should not exceed their bounds.

(vii) Arithmetic Exceptions

- These include errors like divide by zero and floating point exceptions.
- The result of these may vary from getting unexpected results to termination of the program.

4.1 Some Programming Practices

(i) Control Constructs

It is desirable to use a few standard control constructs rather than wide variety of constructs, just because they are available in language.

(ii) Gotos

Goto should be used sparingly and in disciplined manner. Only when the alternative is using gotos is more complex should the gotos be used.

(iii) Information Hiding

The access functions for the data structures should be made visible while hiding the data structure behind these functions.

(iv) Nesting

If nesting of if-then-else constructs becomes too deep, then the logic become harder to understand. It is often difficult to which a particular else cause is associated.

For example, in the below case:

```
if C1 then S1
    else if C2 then S2
        else if C3 then S3;
```

If these are disjoint, the structure can be converted as:

```
if C1 then S1
if C2 then S2
if C3 then S3
```

This sequence of statements will produce same result but is much easier to understand.

(v) Module size

Programmer should carefully examine any function with too many statements and large modules will not be functionally cohesive. The guideline for modularity should be cohesion and coupling.

(vi) Module Interface

A module with complex interface should be carefully examined. If the interface is complex with more than 5 parameters should be examined and broken into modules with simpler interface.

(vii) Side Effects

When module is invoked, it creates side effects of modifying program state beyond the modification of parameters in the interface.

(viii) Robustness

A program might face exceptional conditions like overflow, and in such situations programs should not crash or halt instead should produce some meaningful message and exit successfully.

(ix) Switch case with default

If there is no default case in switch statement, the behavior can be unpredictable at development stage as it can result in bug like NULL dereferencing, memory leak etc.

(x) Empty Catch Block

There are chances that if an exception is caught, there is no action defined and some of the operations may not be performed. It is always good to use catch block even if it is just an error message.

(xi) Trusted Data Sources

Checks should be made before accessing the input data, particularly if it is being provided by the user or is being obtained over the network. Some checks should be done like parity checks, hashes, etc. to ensure the validity of incoming data.

(xii) Give Importance to Exception

Most programmers give less importance to the possible exceptional cases and tend to work with main flow. Though main work is done in main path, it is the exceptional paths that often cause software systems to fail.

4.2 Coding Standards

- Programmers spend more time reading the code than writing code.
- Prime importance is to write code in a manner that it is easy to read and understand.
- Coding standards provide rules and regulations for some aspects of programming in order to make code easier to read.
- Most organizations that develop software regularly develop coding standards.
- The major coding standards include the following:

(a) Naming Conventions

- Package names should be in lower case.
- Variable names should be nouns starting with lower case.
- Constant names should all be uppercase.
- Method names should be verbs starting with lowercase.
- Variables with a large scope should have long names and short names with small scope.
- Private class variables should have the _ suffix.

(b) Files

There are conventions on how files should be named, and what files should contain, such that reader can get some idea about file contents.

For Eg: Java source files should have extension .java

Each file should contain class name same as the file name.

(c) Statements

- These guidelines are for the declaration and executable statements in the source code.
- Not everyone organization will agree to this and develop their own guidelines without restricting the flexibility of programmers.
- Some of the common statement guidelines includes the following:

(i) Variables should be initialized where declared.

(ii) Declare related variables together in a common statement.

(iii) Class variables should never be declared public.

(iv) Loop variables should be initialized immediately before the loop.

(v) Avoid use of break and continue in a loop.

(d) Commenting and Layout

- Comments are textual statements that are meant for the program reader to understand the code.
- Comments should explain what the code is doing or why the code is there.
- Providing comments for modules is most useful, as it forms unit of testing, compiling, verification and modification.
- Comments for a module are known as *prologue* which describes the functionality and purpose of the module.
- If the module is modified, then the prologue should also be modified.
- Some guidelines of this are as follows:
  (i) Single line comments for a block of code should be aligned with the code.
  (ii) There should be comments for all major variables

TESTING

- Testing is intended to show that a program does what it is intended to do and to discover program defects before it is put into use.
- The main testing objectives includes:
  a) Testing is a process of executing a program with intent of finding an error.

b) A good test case is one that has a high probability of ending an undiscovered error.

c) A successful test is one that uncovers all errors.

- The testing principle includes the following:

  a) All tests should be traceable to customer requirement.

  b) Test should be planned long before testing begins.

  c) Exhaustive testing is not possible

  d) To be most effective, testing should be conducted by an independent third party.

4.3 Black Box Testing

- It is also known as Behavioral Testing
- It is a software testing method in which the internal structure/design/implementation of the item being tested is not known to the tester.
- These tests can be functional or non-functional, though usually functional.
- This method is named so because the software program, in the eyes of the tester, is like a black box; inside which one cannot see.
- This method attempts to find errors in the following categories:

  a) Incorrect or missing functions
  b) Interface errors
  c) Errors in data structures or external database access
  d) Behavior or performance errors
  e) Initialization and termination errors

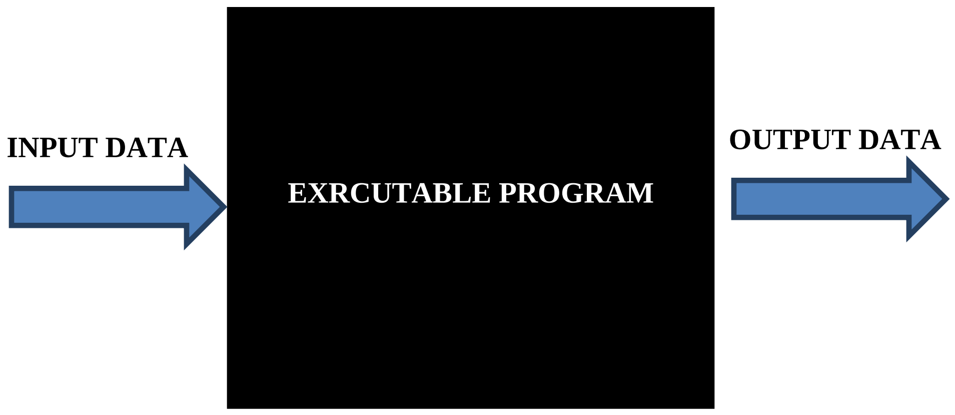

Fig: Black Box Testing

4.3.1 Techniques

- There are different techniques involved in black-box testing and some are as follows:

  (a) Equivalence class portioning

  - The natural approach is to divide the input domain into a set of equivalence classes, so that if the program works correctly for a value, then it will work correctly for all the other values in that class.
  - The equivalence class partitioning method tries to approximate this ideal. An equivalence class is formed of the inputs for which the behavior of the system is specified or expected to be similar.
  - Each group of inputs for which the behavior is expected to be different from others is considered a separate equivalence class.
  - The rationale of forming equivalence classes like this is the assumption that if the specifications require the same behavior for each element in a class of values, then the program is likely to be

constructed so that it either succeeds or fails for each of the values in that class.

- One common approach for determining equivalence classes is that ifthere is reason to believe that the entire range of an input will not be treated inthe same manner, then the range should be split into two or more equivalenceclasses, each consisting of values for which the behavior is expected to be similar.
- Another approach for forming equivalence classes is to consider any specialvalue for which the behavior could be different as an equivalence class.
- Once equivalence classes are selected for each of the inputs, then the issueis to select test cases suitably.
- One strategy is to select each test case covering as many valid equivalenceclasses as it can, and one separate test case for each invalid equivalence class.
- A somewhat better strategy which requires more test cases is to have a test casecover at most one valid equivalence class for each input, and have one separatetest case for each invalid equivalence class.
- In the latter case, the number of testcases for valid equivalence classes is equal to the largest number of equivalenceclasses for any input, plus the total number of invalid equivalence classes.

(b)Boundary Value Analysis

- It has been observed that programs that work correctly for a set of values inan equivalence class fail on some special values.
- These values often lie on theboundary of the equivalence class. Test cases that have values on the boundariesof equivalence classes are

therefore likely to be “high-yield” test cases, andselecting such test cases is the aim of boundary value analysis.

- In boundary value analysis, we choose an input for a test case from an equivalenceclass, such that the input lies at the edge of the equivalence classes.
- Boundaryvalues for each equivalence class, including the equivalence classes of the output,should be covered.
- Boundary value test cases are also called “extreme cases.”
- In case of ranges, for boundary value analysis it is useful to select theboundary elements of the range and an invalid value just beyond the two ends(for the two invalid equivalence classes).
- So, if the range is $0.0 < x < 1.0$, thenthe test cases are 0.0, 1.0 (valid inputs), and −0.1, and 1.1 (for invalid inputs).

(c) Pair ways testing

- Many of the defects in software generally involve one condition, that is, some special value of one of the parameters. Such a defect is called a single-mode fault.
- Single-mode faults can be detected by testing for different values of differentparameters.
- However, all faults are not single-mode and there are combinations of inputs that reveal the presence of faults.
- These multimodefaults can be revealed during testing by trying different combinations ofthe parameter values—an approach called *combinatorial testing*.
- Some research has suggested that most software faults are revealed on somespecial single values or by interaction of a pair of values.

- Mostfaults tend to be either single-mode or double-mode.
- For testing for double-modefaults, we need not test the system with all the combinations of parametervalues, but need to test such that all combinations of values for each pair ofparameters are exercised. This is called *pairwise testing*.

(d) State based Testing

- There are some systems that are essentially stateless in that for the same inputs they always give the same outputs or exhibit the same behavior.
- There are, however, manysystems whose behavior is state-based in that for identical inputs they behavedifferently at different times and may produce different outputs.
- The reasonfor different behavior is that the state of the system may be different, so the behavior and outputs of the system depend not only on the inputsprovided, but also on the state of the system.
- The state of the system dependson the past inputs the system has received, so the state representsthe cumulative impact of all the past inputs on the system.
- If the set of states of a system is manageable, a state model of the system can be built.
- The state model shows what state transitions occur and what actions areperformed in a system in response to events.
- When a state model is built fromthe requirements of a system, we can only include the states, transitions, andactions that are stated in the requirements or can be inferred from them.

- Ifmore information is available from the design specifications, then a richer statemodel can be built.
- A state model for a system hasfour components:
    a) **States**: Represent the impact of the past inputs to the system.
    b) **Transitions**: Represent how the state of the system changes from one stateto another in response to some events.
    c) **Events:** Inputs to the system.
    d) **Actions:** The outputs for the events.

4.3.2 Advantages

- Tests are done from a user's point of view and will help in exposing discrepancies in the specifications.
- Tester need not know programming languages or how the software has been implemented.
- Tests can be conducted by a body independent from the developers, allowing for an objective perspective and the avoidance of developer-bias.
- Test cases can be designed as soon as the specifications are complete.

4.3.3 Disadvantages

- Only a small number of possible inputs can be tested and many program paths will be left untested.
- Without clear specifications, which are the situation in many projects, test cases will be difficult to design.
- Tests can be redundant if the software designer/developer has already run a test case.

4.4 White Box Testing

- White box testing is concerned with testing the implementation of the program.
- The intent of this testing is not to exercise all the different input or output conditions but to exercise the different programmingstructures and data structures used in the program.
- White-box testing is also called structural testing.
- To test the structure of a program, structural testing aims to achieve testcases that will force the desired coverage of different structures.
- One approach to structural testing: control flow-basedtesting, which is most commonly used in practice.

4.4.1 Control Flow based Testing

- Most common structure-based criteria are based on the control flow of theprogram.
- In these criteria, the control flow graph of a program is consideredand coverage of various aspects of the graph is specified as criteria.
- Let the control flow graph (or simply flow graph) of a program P be G. Anode in this graph represents a block of statements that is always executedtogether, i.e., whenever the first statement is executed, all other statementsare also executed.
- An edge (i, j) (from node i to node j) represents a possibletransfer of control after executing the last statement of the block representedby node i to the first statement of the block represented by node j.

- A nodecorresponding to a block whose first statement is the start statement of P iscalled the start node of G, and a node corresponding to a block whose last statement is an exit statement is called an exit node.
- A path is a finitesequence of nodes (n1, n2, ..., nk), k > 1, such that there is an edge (ni, ni+1)for all nodes ni in the sequence (except the last node nk).
- A complete path is a path whose first node is the start node and the last node is an exit node.
- The simplest coveragecriterion is statement coverage, which requires that each statement of theprogram be executed at least once during testing.
- In other words, it requiresthat the paths executed during testing include all the nodes in the graph. Thisis also called the all-nodes criterion.
- This coverage criterion is not very strong, and can leave errors undetected.
- A more general coverage criterion is branch coverage, which requires thateach edge in the control flow graph be traversed at least once during testing.
- In other words, branch coverage requires that each decision in the program beevaluated to true and false values at least once during testing.
- Testing basedon branch coverage is often called branch testing.
- The trouble with branch coverage comes if a decision has many conditionsin it.
- A more general coverage criterion is onethat requires all possible paths in the control flow graph be executed duringtesting.
- This is called the path coverage criterion or the all-paths criterion, andthe testing based on this criterion is often called path testing.
- The difficultywith this criterion is that programs that contain loops can have an infinitenumber of possible paths.

4.5 Testing Strategic Issue

- Even the best strategy will fail if a series of overriding issues are not addressed.
- A software testing strategy will succeed only whensoftware testers:

  (1) Specify product requirements in a quantifiable manner long before testing commences

  (2) State testing objectives explicitly

  (3) Understandthe users of the software and develop a profile for each user category

  (4) Developa testing plan that emphasizes “rapid cycle testing,”

  (5) Build “robust” softwarethat is designed to test itself

  (6) Use effective technical reviews as a filter prior to testing

  (7) Conducttechnical reviews to assess the test strategy and test cases themselves

  (8) Develop a continuous improvement approach for the testingprocess.

4.6 Unit Testing

- Once a programmer has written the code for a module, it has to be verifiedbefore it is used by others.
- Testing remains the most common method of this verification and at the programmer level the testing done for checking the codethe programmer has developed is called unit testing.
- Unit testing is like regular testing where programs are executed with sometest cases except that the focus is on testing smaller programs or

moduleswhich are typically assigned to one programmer (or a pair) for coding.

- A unit may be a function or a small collection of functions for procedurallanguages, or a class or a small collection of classes for object-orientedlanguages.
- It suffices that during unit testing the tester, who is generallythe programmer, will execute the unit with a variety of test cases and studythe actual behavior of the units being tested for these test cases.
- Based on thebehavior, the tester decides whether the unit is working correctly or not.
- Ifthe behavior is not as expected for some test case, then the programmer findsthe defect in the program (an activity called debugging), and fixes it.
- Afterremoving the defect, the programmer will generally execute the test case thatcaused the unit to fail again to ensure that the fixing has indeed made the unitbehave correctly.
- An issue with unit testing is that as the unit being tested is not a completesystem but just a part, it is not executable by itself.
- Furthermore, in itsexecution it may use other modules that have not been developed yet.
- Due tothis, unit testing often requires drivers or stubs to be written. Drivers play the role of the “calling” module and are often responsible for getting the test data,executing the unit with the test data, and then reporting the result.
- Stubs areessentially “dummy” modules that are used in place of the actual module tofacilitate unit testing.

4.7 Integration Testing

- Integration testing is a systematic technique for constructing the software architecture while at the same time conducting tests to uncover errors associated with interfacing.
- The objective is to take unit-tested components and build a program structure that has been dictated by design.
- There is often a tendency to attempt non-incremental integration where all components are combined in advance and the entire program is tested as a whole.
- In Incremental integration the programis constructed and tested in small increments, where errors are easier toisolate and correct; interfaces are more likely to be tested completely; and a systematictest approach may be applied.
- A number ofdifferent incremental integration strategies are developed like:

(a) Top-Down Integration

- Modules are integrated by moving downward through the control hierarchy, beginning with the main control module (main program).
- Modules subordinate (and ultimately subordinate)to the main control module are incorporated into the structure in either a depth-first or breadth-first manner.
- Depth-first integration integrates all components on amajor control path of the program structure.

- Breadth-first integration incorporates all components directly subordinate at each level, moving acrossthe structure horizontally.

(b)Bottom-Up Integration

- Bottom-up integration testing, as its name implies, begins construction and testing with atomic modules (i.e., components at the lowest levels in the program structure).
- Because components are integrated from the bottom up, the functionality provided by components subordinate to a given level is always available and the need for stubs is eliminated.

(c) Regression Testing

- Regression testing is the re-execution of some subset of tests that have already been conducted to ensure that changeshave not propagated unintended side effects.
- Regression testing helps to ensurethat changes (due to testing or for other reasons) do not introduce unintendedbehavior or additional errors.
- Regression testing may be conducted manually, by re-executing a subset of all test cases or using automated capture/playback tools.

(d)Smoke Testing

- Smoke testing is an integration testing approach that is commonly used when product software is developed.
- It is designed as a pacing mechanism fortime-critical projects, allowing the software team to assess the project on a frequentbasis.

4.8 Validation Testing

- Validation testing begins at the culmination of integration testing, when individual components have been exercised, the software is completely assembled as a package, and interfacing errors have been uncovered and corrected.
- Testing focuses on user-visible actions and user-recognizable output from the system.

4.8.1 Validation Test Criteria

- Software validation is achieved through a series of tests that demonstrate conformity with requirements.
- A test plan outlines the classes of tests to be conducted, and a test procedure defines specific test cases that are designed to ensure that:
  - (a) Functional requirements are satisfied
  - (b) All behavioral characteristics are achieved,
  - (c) All content is accurate and properly presented,
  - (d) All performance requirements are attained,
  - (e) Documentation is correct,
  - (f) Usability and other requirements are met

4.8.2 Configuration Review

- An important element of the validation process is a configuration review.

- The intent of the review is to ensure that all elements of the software configurationhave been properly developed, are cataloged, and have the necessary detail tosupport activities

4.8.3 Alpha and Beta Testing

- It is virtually impossible for a software developer to foresee how the customer will really use a program.
- Instructions for use may be misinterpreted; strange combinations of data may be used; output that seemed clear to the tester may be unintelligible to a user in the field.
- Most softwareproduct builders use a process called alpha and beta testing to uncover errorsthat only the end user seems able to find.
- The alpha test is conducted at the developer's site by a representative groupof end users. The software is used in a natural setting with the developer "lookingover the shoulder" of the users and recording errors and usage problems.
- The beta test is conducted at one or more end-user sites. Unlike alpha testing,the developer generally is not present.
- Therefore, the beta test is a "live" applicationof the software in an environment that cannot be controlled by the developer.
- The customer records all problems (real or imagined) that are encounteredduring beta testing and reports these to the developer at regular intervals.
- A variation on beta testing, called customer acceptance testing, is sometimesperformed when custom software is delivered to a customer under contract.

- The customer performs a series of specific tests in an attempt to uncover errorsbefore accepting the software from the developer.

4.9 System Testing

- Software is incorporated with other system elements (e.g., hardware, people, information), and a series of system integration and validation tests are conducted.
- These tests fall outside the scope of the software process and are not conducted solely by software engineers.
- However, steps taken during software design and testing can greatly improve the probability of successful software integration in the larger system.

4.9.1 Recovery Testing

- Recovery testing is a system test that forces the software to fail in a variety of ways and verifies that recovery is properly performed.
- If recovery is automatic (performed by the system itself), re-initialization, check-pointing mechanisms, data recovery, and restart are evaluated for correctness.
- If recovery requires human intervention, the mean-time-to-repair (MTTR) is evaluated to determine whether it is within acceptable limits.

4.9.2Security Testing

- Security testing attempts to verify that protection mechanisms built into a system will, in fact, protect it from improper penetration.
- Penetration spans a broad range of activities: hackers who attempt topenetrate systems for sport, disgruntled employees who attempt to penetrate forrevenge, dishonest individuals who attempt to penetrate for illicit personal gain.

4.9.3Stress Testing

- Stress testing executes a system in a manner that demands resources in abnormal quantity, frequency, or volume.
- A variation of stress testing is a technique called sensitivity testing.
- In somesituations (the most common occur in mathematical algorithms), a very smallrange of data contained within the bounds of valid data for a program may causeextreme and even erroneous processing or profound performance degradation.

4.9.4 Performance Testing

- Performance testing occurs throughout all steps in the testing process.
- Even at the unit level, the performance of an individual module may be assessed as tests are conducted.
- However, it is not until all system elements are fully integrated that the true performance of a system can be ascertained.
- Performance tests are often coupled with stress testing and usually require both hardware and software instrumentation.

4.9.5 Deployment Testing

- Deployment testing, sometimes called configuration testing, exercises the software in each environment in which it is to operate.
- In addition, deployment testing examines all installation procedures and specialized installation software (e.g., “installers”) that will be used by customers, and all documentation that will be used to introduce the software to end users.