White Box Testing

Introduction

- IEEE definition of software lists four components which are needed in order to assure the quality of software applications:
- computer programs or the code which is the brain behind any application;
- procedures that define the flow of the program, its methods and the way of functioning;
- documentation needed for developers and users;
- the data that includes parameters.
- The computer programs or the source code of the software is an important artifact that needs to be tested for ensuring quality of the software product. The testing that encompasses the verification of the computer programs and the logic of the application is known as White Box testing.

Introduction

- IEEE defines White box testing as “The testing that takes into account the internal mechanism of a system or component”. White box testing takes care of the intricacies of the product and evaluates it for accuracy and precision, to meet the requirement specifications.

Introduction

- White box testing verifies the designs and codes involved in the development of a software product. It involves validating whether the code has been implemented as per design specifications and also validating the security concerns of the software’s functionality. Thus skilled testers with knowledge of programming are required to conduct white box testing.

Introduction

- White box testing helps the software tester to find out the correct type of input data used to test the application effectively, i.e. the tester is aware of the internal coding and hence is able to optimize the code.
- When the tester removes the extra line of code, it enables the testers to find the hidden defects.

Introduction

- white box testing helps a software tester to perform the following functions:

1. Test independent paths within a unit or a module.
2. Test the logical correctness (test both the true and false conditions).
3. Test loops, specifically at their boundaries and check the operational boundary correctness.
4. Test internal data structures to ensure their validity.

White box testing provides greater stability and reusability of test cases. The software application is tested in a thorough way and thus raises the customer satisfaction and confidence.

White box testing

- White box testing is also referred to as glass box testing or structural testing or open
- box testing or clear box testing, due to its nature of examining the internal workings.

Static White Box Testing

- Static white box testing methodology involves testing the internal logic and structure of the code without compiling and running the program.
- Advantage of performing static white box testing is that bugs are found early and those that cannot be uncovered by dynamic white box are also identified.
- A software tester needs to have the knowledge of software coding and the internal logic. This enables a software tester to differentiate between the statement or the path which works from those which do not work.

Static White Box Testing

- Static white box testing involves a procedure of analyzing data flow, control flow, information flow, and also testing the intended and unintended software behaviors.
- specs that are examined which include codes, branches, paths, and internal logic.

Examining the Design and Code

- Examining and reviewing the codes without execution
- Three methods through which bugs are identified and captured
  - reviews,
  - Inspections,
  - walkthroughs.
- Three methods use procedures and error detecting techniques for analyzing the bugs found and also subsequently correct them. There are planned review meetings that are held and the developers and the testers discuss the nature of the application and the probable areas of defects.
- the areas are identified, are either corrected or marked for dynamic white box testing

Note

- The responsibility of performing static white box testing varies from one development team to another development team. In some organizations, the programmers organize and run reviews by inviting the testers as observers, while in some organizations the testers perform the task by asking the programmers, who wrote the code along with other peers, to help them in their reviews.

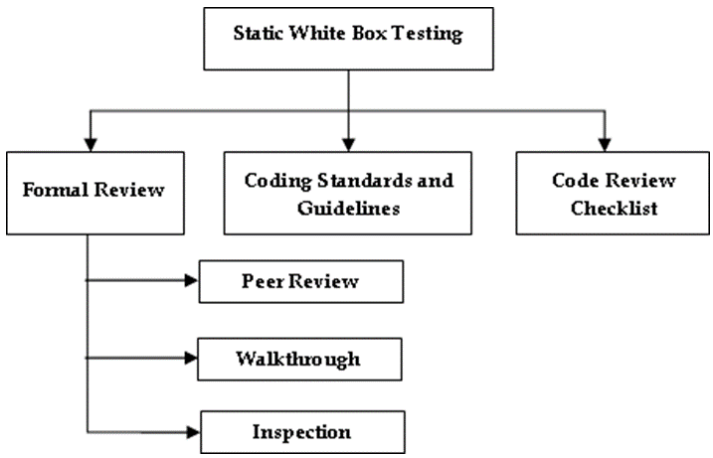
Static white box testing

- Static white box testing is found to be a cost-effective method of testing.
- The advantage of performing a static white box testing is that it provides the testers with better ideas about the test cases while implementing them during software development.
- Programmers write codes and a few peers conduct reviews on them. The development team conducts the static white box testing and reviews the results. In addition, the development team also invites testers to observe the process of testing.

Static white box testing

- Static white box testing is seldom carried out in the software testing process, as there is a misconception that it is time consuming, expensive and not productive when compared to the other alternative testing methodologies.

Static White Box Testing



Formal Review

- involves formal meetings between the programmers and testers (or between programmers)
- there will be discussions pertaining to inspection of the software's design and code.
- formal reviews are considered to be the first nets that capture bugs, since, prospective defect areas are discussed

Formal Review

- For a successful formal review, four essential elements are required, and they are:
    1. ***Identify Problems:*** A formal review must identify the problems related to design and code of the software. Participants are required to avoid criticism and carry a positive attitude and must be diplomatic and not emotional about reviews.
    2. ***Follow Rules:*** Formal reviews follow a fixed set of rules. The number of lines of code to be reviewed per day along with the time spent on the job is to be strictly adhered to. This is done so that reviews can be carried out more smoothly.
    3. ***Prepare:*** Formal reviews expect each participant to prepare and contribute towards the meeting. Based on the type of review, each participant's role differs. The problems discovered in the review process are generally found during the preparation process and not actually during the review.
    4. ***Write a Report:*** The formal review group produces a written report summarizing all the results of the review and the report is made available to the rest of the development team. Thereby, the problems encountered are shared with the team.

Steps involved in formal review

- peer reviews, walkthroughs, and inspections
- **Peer Reviews**
- Peer reviews are the informal reviews where team members conduct reviews amongst themselves. They are also known as buddy reviews.
- conducted with a programmer who has been involved in designing the architecture or code along with other programmers or testers, who act as reviewers.
- participants involved in the review are required to adhere to the four key elements of formal review (identify problems, follow rules, prepare and write a report).

Steps involved in formal review

- **Walkthroughs**
- the programmer who developed the code presents the code to a group consisting of five or six member team of programmers and testers.
- A walkthrough is conducted to provide an overview about the structure of the code in the presence of a senior programmer and other reviewers.
- presenter reads through the code line by line, or function by function and explains what each function and line of code means. Relevant comments and queries are addressed during the walkthrough session.
- participants in a walkthrough is more than those in the peer review session, it is important to follow rules and have periodic follow-up meetings.
- On completion of the review, the presenter makes a report of the meeting and also the way the bugs were addressed

Steps involved in formal review

- Inspections
- follows a structured format
- Different from walkthrough and peer review.
- person who presents a code is not the real programmer. Participants are called inspectors.
- inspectors are provided the task of reviewing the code from the user's and tester's perspective. This helps in bringing out the views about the product from various perspectives, thus helping in identifying the different bugs in the product.
- On completion of the inspection, another meeting is conducted by the inspectors alone to discuss the defects that were found and they work with the moderators who are competent programmers to identify the areas of rework. The programmer then rectifies the defects and the moderators verify the same to ensure that it is done properly. Re-inspections are conducted based on the criticality of a software bug that is found.

Coding Standards and Guidelines

- In the formal review method, inspectors look only for the problems and omissions in the code. Bugs are however found by carefully analyzing the code which is done by the senior programmers and testers.
- There possibilities where a code may operate correctly but may not be written to meet the specification standards. This is similar to writing English which is grammatically correct, but may not convey the correct meaning.

- Guidelines are the best practices and recommendations which are preferred to be followed. Standards are rules which must be adhered to, whereas guidelines are instructions which enable a person to follow a set of standards.

Coding Standards and Guidelines

Reasons for adhering to standards and guidelines :

- ***Reliability:*** It has been observed that a code which is being written for a particular standard with formal guidelines is more reliable and secure than the ones that are not.
- ***Readability or Maintainability:*** Codes which have been written based on standards and guidelines are easier to understand and maintain, when compared to the ones which are not.
- ***Portability:*** Codes written by programmers must be portable enough to run on different hardware and also different compilers. When standards and guidelines are followed, it becomes easier for people to access the code. Sometimes, project requirements may demand to meet the international standards and guidelines.

Code Review Checklist

- Code reviews are performed in addition to the general process of comparing the code against the standards and guidelines. This ensures that the design requirements of the software project are met.
- To conduct code reviews in detail, some amount of programming experience is required

Code Review Checklist

Example

- Does the code do what it has been specified in the design specifications?
- Does the software module have another similar existing module, so that it could be reused?
- Does the module have a single entry point and single exit point (As multiple entry and exit points can be tedious to test)

Errors

- Errors discovered while testing
- ***Data Reference Errors:*** Data reference errors relate to the errors which are caused due the usage of variables, constants, arrays, strings, or records which are not properly declared or initialized to use and refer them.
- When looking at data declaration errors
- Check if any un-initialized variables are referenced
- Check if the arrays and the string subscripts integer values are within the array's bounds or string dimension
- Check if there are any "off-by-one" errors in indexing operations or references to arrays
- Check if a variable is used where a constant would work better
- Check if a variable is assigned a value that's of a different type than the variable
- Check if memory is allocated for referenced pointers
- Check if the data structures are referenced in different functions defined identically

Data reference errors are the primary cause for buffer overruns - the main bug concerned with security issues

Errors

- ***Data Declaration Errors:*** Data declaration errors occur due to improper declaration of variables and constants.
- When checking for data declaration errors
  - Check if the variables are assigned the correct length, type, storage class
  - Check if a variable is initialized at the time of declaration, and also analyze if it is properly initialized and consistent with its type
  - Check if there are any variables with similar names
  - Check if there are any variables declared which are never referenced or just referenced once (should be a constant)
  - Check if all variables are explicitly declared within a specific module

Errors

- ***Computation Errors:*** Computational errors arise due to errors in calculations -- where the expected results are not obtained due to erroneous calculations.
- Check if any calculations use variables which have different data types
  - Eg Adding integers and floating point numbers
- Check if any calculations use variables which have the same data type but vary in size.
  - Eg Adding long integers to short integers
- Check if the compiler's conversion rules for variables of inconsistent type or size understood and considered while calculating
- Check if overflow or underflow in the middle of a numeric calculation possible
- Check if it is ever possible for a divisor/modulus to be zero
- Check if a variable's value goes outside its meaningful range

Errors

- Example The probability of a result being less than zero percent or greater than 100 percent.

- Check if the target variable of an assignment is smaller than the right-hand expression
- Check if parentheses are needed for clarification
  - Example For expressions containing multiple operators, there is confusion about the order of evaluation.
- Check if for cases of integer arithmetic, the code handles some calculations, particularly division, which results in loss of precision

Errors

- ***Comparison Errors:*** Comparison errors occur during boundary conditions such as 'less than, greater than, equal, not equal, and true or false'.
- Check if the comparisons are correct
  - Eg Using < instead of <=
- Check whether there are any comparisons between floating-point values
  - Checking for precision problems when comparing.
  - (It is similar to checking whether 7.00000007 is close enough to 7.00000008 to be considered equal)

Errors

- Check whether the operands of a Boolean operator are correct
  - Checking, if an integer variable containing integer values is used in a Boolean calculation
  - In the 'C' language, zero is considered as true and non-zero is considered as false.
- Check if each Boolean expression states what it is supposed to state, works as expected and if there are any doubts about the order of evaluation

Errors

- ***Control Flow Errors:*** Control flow errors arise due to improper behavior of loops and control structures in a language. They are caused by direct or indirect computational or comparison errors

a) Check whether each switch statement has a default clause
b) Check if the nested switch statements are in loops
c) Check the possibility of a loop not executing
d) Check if the compiler supports short-circuiting expression in evaluation
e) Check if the language contains statement groups such as begin...end and do...while, are the 'end's explicit and do they match their appropriate groups
f) Check if there is a possibility of premature loop exit
g) Check if there are any "off by one" errors which may cause unexpected flow through the loop

Errors

Subroutine Parameter Errors: Subroutine parameter errors occur when incorrect data is passed to and from subroutines.

- Check if constants are passed to the subroutine as arguments or are they accidentally changed in the subroutine
- Check whether the units of each parameter matches with the units of each corresponding argument
- Check the types and sizes of the parameters received by a subroutine match with those sent by the calling code
- Check if a subroutine alters a parameter that's intended only as an input value

Errors

- ***Input/Output Errors:*** Input/output errors relate to errors such as reading from a file, accepting input from a keyboard or mouse, and writing to an output device such as a file or screen.
- (a) Check whether the software strictly adheres to the specified format of the data being read or written by the external device
- (b) Check if the error condition has been handled
- (c) Check if the software handles the situation of the external device being disconnected
- (d) Check if all the exceptions are handled by some part of the code
- (e) Check if all error messages have been checked for correctness, appropriateness, grammar, and spelling

Errors

- 8. Other Checks: The ones which do not fit into the above mentioned categories fall under the other checks. In this category of error checks, we will check for the following:
- Check if the code is portable to the other OS platforms
  - The GCC compiler warnings(GCC is the GNU Compiler Collection used to compile C programs)
- Check whether the code handles ASCII and Unicode
  - Checking whether the software will work well with languages other than English
- Check whether your code passes the lint test
- Check issues related to internationalization
- Check whether the code relies on deprecated APIs
- Check whether the code ports to architectures with different byte orderings

Dynamic White Box Testing

- Dynamic white box testing is a validation check which involves testing and running the test cases with input values. The dynamic white box testing ensures that the application works according to the specification throughout the execution. The results of the execution give proof about the actual behavior of the code. A software tester collects ample information which enables the tester to analyze what the code does, how it works, what to test, what not to test, and thus determines the approach of the testing process.

Dynamic White Box Testing

- Dynamic white box testing also known as structural testing, as it enables the tester to view the structure of the code and its design and run the tests.
- four prominent areas that dynamic white box testing encompasses are:
- Testing low-level functions, procedures, subroutines, or libraries directly.

- Testing the software at the top level, as a completed program, by adjusting the test cases based on what we know about the software operation.
- Gaining access to read variables and state information from the software and enabling the tester to determine whether the tests are doing what they were designed to do. Forcing the software to do things that would be difficult, if they are tested normally.
- Measuring how much of the code and which particular code a tester "hits" when he/she runs the tests and also how he/she adjusts the tests to remove the redundant test cases and add the missing ones.

Dynamic White Box Testing vs. Debugging

- Dynamic white box testing and debugging are two techniques which appear similar, since they deal with the code and the bugs in software. They do overlap when bugs are found. However, they are different in their goals. Dynamic white box testing finds the bugs in the software while debugging deals with the process of fixing the bugs that have been found during the testing phase.
- As a tester In white box testing, you narrow down to the information pertaining to the number of lines of code which look erroneous. The programmers debug the erroneous lines of code which are reported to them. They find the reason for the cause of bug and fix it.

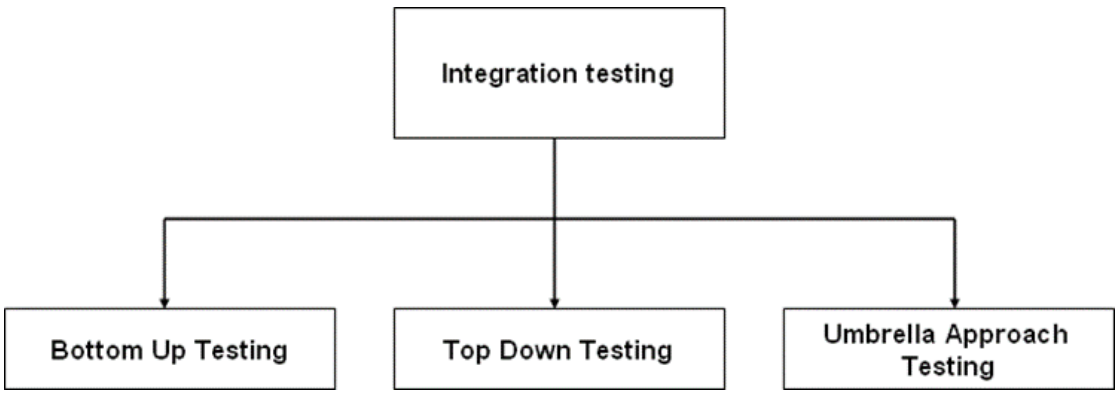
Testing the Pieces

- In dynamic black box testing, a bug is found at the final stages of testing. It is difficult and at times impossible to figure out the occurrence of bug and hence a tester will be unable to learn which part of the software is causing the problem.
- There are also possibilities of some bugs which hide the existence of other bugs and these bugs might cause the software to fail. Although, a programmer fixes the problem, when the tester re-runs the software, it will fail again. As a result, bugs keep accumulating and it becomes impossible to get to the core of the defect.

Unit and Integration Testing

- To overcome the problem of accumulating core defects, the concept of testing the software in pieces and gradually integrating these pieces into larger portions to form the system came in to existence.
- The lowest level of testing known as unit or module level testing.

Integration Testing Strategies



Integration Testing Strategies

- ***Bottom Up Testing:*** The bottom up testing is conducted from the sub module to the main module. In the absence of the main module (when the module is not developed), a temporary program known as 'Drivers' (Drivers are the written code that are used to execute the modules that are being tested. These drivers act as real modules by sending test-case data to the modules, reading the results and verifying their correctness) to simulate the main module. This approach helps in creating easy test conditions and easy observation of test results.

Integration Testing Strategies

- ***Top Down Testing:*** The top down testing is conducted from the main module to the sub module. In the absence of the sub module (when the module is not developed), a temporary module known as 'Stub' (Stubs are small pieces of code which act as interface modules by feeding "fake" data) is used to simulate the sub module. This type of testing boosts the morale of the program by eliminating the flaws. The top down method of testing relates to testing all the higher-level modules by using stubs as low-level interface modules. This type of testing is found to be very useful, as we can quickly run through the numerous test values and also validate the operations of a module.

Integration Testing Strategies

- ***Umbrella Approach Testing:*** The umbrella approach testing is a combination of both, the bottom up and top down testing approaches. This method focuses on testing the modules which have high degree of user interaction. The input modules are integrated in the bottom up approach and the output modules are integrated in the top down approach. It is considered to be a beneficial approach for the early release of GUI (Graphical User Interface) based applications used for enhancing the functionality.

Integration Testing Strategies

- The umbrella approach of testing ensures:
  - Early release of limited functionality.
  - Reduction in the need for stubs and drivers.

Data Coverage

- Data coverage focuses on data which is tracked completely through the software. It can relate to the values stored in variables, constants, arrays, and data structures. It includes the keyboard and mouse input, files, screen input and output and those that are sent to modems, networks, and so on.
- the code is divided into data and states, relating to program flow which is similar to black box testing.
- Data coverage deals with the aspect of generating test data sets which cover all the required tests before the software component is certified as reliable and shipped to the customer.

Data coverage testing

- ***Data Flow:*** Data flow involves tracking the data through the software. During the unit testing phase, tracking is done only in the small modules, as tracking all the modules in a system becomes a tedious process. When a function is tested at the low level, a debugger along with a watch variable is used to view the program during execution.
- white box testing, one cannot view the program execution, but can only view the value of the variable at the beginning and at the end.
- dynamic white box testing gives the provision of checking the intermediate values during program execution.

Data coverage testing

- ***Sub-Boundaries:*** Sub-boundaries in software are the most common places where bugs can be found.
- Example
  - An operating system which is running on low RAM may move the data to a temporary storage on the hard drive. In this case, the boundary may not be fixed, as it depends on the space remaining on the disk. There are more possibilities of errors occurring at the sub-boundaries.

Data coverage testing

- ***Formulas and Equations:*** Generally formulas and equations are not visible and are embedded deep in the code. Hence, we cannot know their effect and presence.

Data coverage testing

- ***Error Forcing:*** This is a method of data testing, where a software tester can force a variable to hold a value in order to check how the software handles it. When the software runs in a debugger, there is a privilege of not just watching the variable and the values that they hold, but you also have the privilege of adding required specific values. (A debugger is used to force a variable value to zero)

Code Coverage

- Black box is just the half work done and the rest needs to be completed with the white box testing. While testing, testers need to assure that they test the entry and exit of every module, every line of code, and follow logical and decision path of the product. This methodology of testing followed is known as code coverage testing.
- Code coverage testing is a dynamic white box testing where testing is done by executing the test and the tester has complete access to the code. He/she is also aware of the parts of the software that have passed and the parts that have failed.

Code Coverage

- Code based testing, which involves detecting errors by following execution oriented methods, is also known as white box testing. Here, a tester is expected to know and understand the low level design and identify the code-based approach and to apply the techniques in the code that is written.

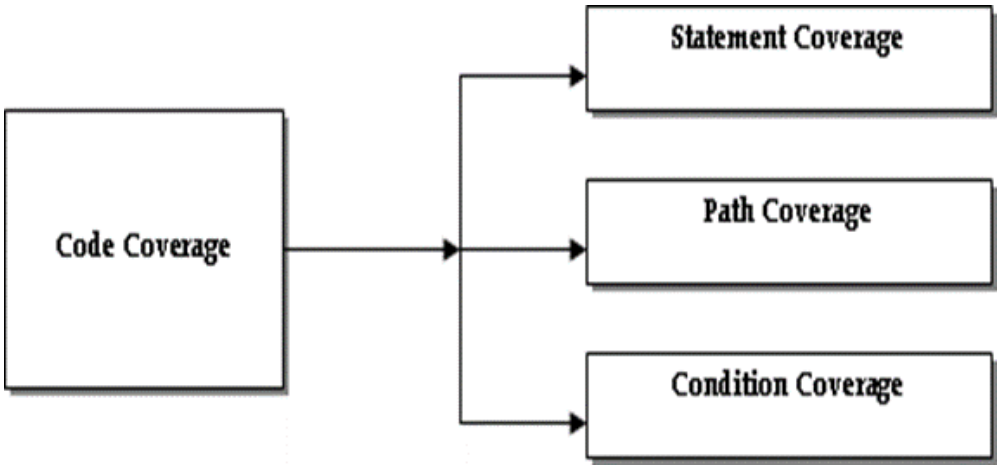
Code Coverage

- white box testing is to ensure that the internal structure of the program has a logical flow. The need for code-based testing is to:

a) Understand the objective and arrive at test cases.
b) Check programming styles and compliance with coding standards.
c) Check for logical errors and incorrect assumptions introduced during coding.
d) Find incorrect assumptions about execution of paths.
e) Find typographical errors.

A tester is also required to maintain a checklist for code walkthrough, code inspection, traceability matrix, and record bug findings in the bug report and in the review records.

Different types of code coverage testing



Types of code coverage testing

- ***Statement Coverage:*** The simplest form of code coverage is known as statement coverage or line coverage. A software tester needs to ensure that all the statements are executed at least once.
- Example
- 1: PRINT “Good Morning” 2: PRINT “Be Positive”
- 3: PRINT “Today is a Great Day”
- 4: END
- *Sometimes statement coverage can be misleading. This is due to the fact that statement coverage can provide information relating to the execution of each statement only, and not relating to the paths traversed in the software.*

Types of code coverage testing

- ***Path Coverage:*** Path coverage relates to covering all the paths in the software. The simplest form of path testing is known as branch coverage testing.
- Loops are the important parts of a structured program. The direction in which the control proceeds in a program and the number of times the statement gets executed is based on the initiation and termination condition of loops. Testing such types of loops is a challenging task, as an error may be revealed only after exercising some of the sequence of decisions or particular paths in a program.

Types of code coverage testing

- ***Path Coverage***
- ***Example***
- An 'If statement' that creates a condition 1: PRINT "Good Morning"
- 2: IF Date$ = "25-12-2008" then
- 3: PRINT "MERRY CHRISTMAS"
- 4: END IF
- 5: PRINT "The date id:"; Date$
- 6: PRINT "The time:";Time$
- 7: END

Types of code coverage testing

- ***Condition Coverage:*** Condition coverage is a complicated path testing.
- This is done by adding an extra condition to the IF statement.
- Condition coverage is a combination of branch coverage and more detailed conditions.
- Branches alter the flow of sequence in a program and jump from one part of the program to another. Both conditional and unconditional transfers can occur.
- In this type of coverage, every branch of the control flow graph (A control flow graph is a graphical representation of all the paths which the program has traversed through during execution) is executed at least once and the values of the conditions are also exercised at least once. The simplest condition criterion called the basic or elementary condition coverage requires each elementary condition to be covered, which implies that each elementary condition should have both True and False outcomes at least once during the execution of test set.

Types of code coverage testing

$$\text{Basic Condition Coverage} = \frac{\text{Number of executed conditions}}{\text{Total Number of conditions}}$$

Types of code coverage testing

- Multiple conditions in the ‘if statement ‘can create more paths through the code.
- 1: PRINT "Good Morning"
- 2: IF Date$ = "25-12-2008" AND Time$ = "00:00:00" THEN
- 3: PRINT "MERRY CHRISTMAS"
- 4: END IF
- 5: PRINT "The date is: "; Date$ 6: PRINT "The time is: "; Time$
- 7: END

- line 2 both date and time are checked. Thus, condition coverage takes the extra conditions. This will require four test sets of test cases which assure that the each IF statement is covered.

Types of code coverage testing

- Test cases for a Full Condition Coverage Date$ Time$ Line # Execution
- 25-12-200711:11:11 1, 2, 5, 6, 7
- 25-12-2007 00:00:00 1, 2,5,6,7
- 25-12-2008 11:11:11 1, 2,5,6,7
- 25-12-200800:00:00 1,2,3,4,5,6,7
- All the four cases are considered to be important, since they traverse through different conditions of the IF statement such as False-False, False-True, True-False, and True-True.

Types of code coverage testing

Write the statement coverage test cases for the following program.

```
Void function eval (int X, int Y, int A)
{
If (X>1) and (Y=0)
then A=A/X;
if(X=2) or (A>1)
then A=A+1;
}
```

(Note: X=2,Y=0, A=3 (A can be any assigned value)

Types of code coverage testing

1. Based on statement coverage, write a simple program to print the day, date, time, month and year.
2. Based on condition coverage, write a simple program to print the marks you obtained in each subject and the class to which those marks belong. (If mark is greater than 60 implies class 1, if marks greater than 50 but less than 60 implies class 2, if marks less than 50 but above 35 implies class 3, if marks less than 35- implies fail)

Questions