1.1. The Purpose of Testing:-

Testing:- It is the process of executing a program (or) system with the intent of finding errors.

* Testing is basically a task of locating errors.

> Software Testing = Software Verification + Software Validation.

Software testing is an activity that is used to evaluate the capability of a system and determine if it is in accordance with the specifications. The purpose of testing is more than just debugging and detecting bugs. Testing is usually performed for the following ~~steps~~ purposes. They are as follows,

1) For improving and assuring software quality.
2) For verification and validation.
3) For estimating reliability.

1) Software Quality Assurance (SQA):-

Quality is used to determine if the software meets all the requirements that were specified during design phase.

Software quality assurance is used to monitor the Software engineering processes and methods for ensuring its quality.

2) Verification and Validation:-

Once the product is developed the software has to be tested to ensure that it meets the requirements of the customer and also the internal requirements. Validation is done to check whether the software conforms to all the specifications.

Both Verification and Validation is done to test the quality of a software. Based on the test results, testers make decisions whether the product will work properly or not.

Software quality has the following three aspects such as functionality, engineering and adoptability.

1) Functionality:- It is an exterior quality factor. It is determined by considering correctness, reliability, usability and integrity of a product.

ii) Engineering:- It is an interior quality factor. It is determined by considering efficiency, testability, documentation and structure of a Product.

iii) Adaptability:- It is a future quality factor. It is determined by considering flexibility, reusability and maintainability of a product. When a validation test is performed on a product, it is referred to either as a clean or positive tests.

The disadvantage of validation is that the software works only for some particular test cases. The finite number of test cases cannot ensure that the software works properly under all circumstances. On the other hand, only one failed test shows that the software doesn't work. Negative tests are performed to show that the software fails to meet the expected requirements of the user.

3) Reliability Estimation :-

Testing is an important factor to quantify software reliability. Software reliability is related to the different aspects of software such as structure, documentation etc. It is difficult to estimate software reliability only by considering its related aspects.

1.1.1 Goals for testing :-

The two major goals of testing are as follows,

1) Bug Prevention

2) Bug Discovery

1) Bug Prevention :- Which is considered as the primary goal for testing. When a bug is detected, appropriate method should be used to remove it. And if the bugs are not Prevented, then certain ~~Sympots~~ Symptoms are discovered that are the major causes for the occurence of bugs. When a particular bug is Prevented there's no need to perform testing again to confirm

accuracy of the Program. Also a prevented bug won't affect the execution schedule of the Program.

2) Bug Discovery:- Which is considered as the secondary goal for testing. It is performed when the primary goal fails to prevent the bugs. Since bugs are not static, they always change their states. Even if all the information regarding the expectations, Procedures and output of test are maintained, there is a Probability of errors to occur.

1.1.2 phases in a Tester's Mental Life:-

A tester considers five phases of thinking. these phases are classified as follows,

i) phase 0 thinking
2) phase 1 thinking
3) phase 2 thinking
4) phase 3 thinking
5) phase 4 thinking.

1) Phase 0 (Thinking Criteria):-

Testing and debugging are similar. Testing is useful in debugging i.e it supports debugging. When testing came into existence, phase 0 thinking was the criteria for the software development. This thinking was applicable to an area from which the following resources can be determined which are as follows,

i) High-cost and inadequate computing resources
ii) Low-cost software resources
iii) Single Programmers.
iv) Small projects resources.
v) Irrelevant software resources.

2) Phase 1 (Thinking the software or program works):-

the aim of this testing is to display the working of a Program or software. Phase 1 thinking identifies the distinct relationship between testing and debugging. In this testing phase the number of tests performed on a software does not determine the execution of the software.

If a single test among the infinite number of available tests fails, then the tester concludes that the software will not be executed even if the subsequent tests make its execution possible.

3) Phase 2 (Thinking the program doesn't work):-

The aim of this type of thinking is to demonstrate that the program doesn't execute. The thinking of testers in phase 2 opposes the thinking of designers. If a test fails, then the goal of phase 2 thinking is accomplished. In the process of phase 2 thinking, bug can be detected by testers, programmers and designers.

i) The tester detects an error (bug).
ii) The programmer verifies and corrects the error.
iii) The designer designs test.
iv) The designer aims to perform different tests to identify different errors.

4) phase 3 (Thinking Test for Reducing the Risk) :-

the aim of testing to reduce the risks that are Predictable. The phase 3 thinking accepts the rules of statistical quality control to improve the quality of software. In the process of phase 3 thinking, the tyter detects the bugs and eliminates them.

5) phase 4 (Thinker's Knowledge) :-

the aim of phase 4 thinking is to reduce the risks of software with minimal testing effort. Software that require minimum testing in order to attain goals of lower phases testing is obtained by integrating the capability of testing with the knowledge of knowing the conditions under which software is made testable.

Testing is done for the following reasons,

i) To decrease the manpower required for testing.

ii) To ensure that a testable code has less number of bugs when compared to the code that is difficult to test.

1.1.3 Testing isn't Everything:-

Testing is an effective method for detecting errors. Alternatively, other methods are taken to detect errors and remove them. The other methods include reviews, inspections, walk through etc. After processing these methods the software is tested. The methods are as follows:

1) walkthroughs, Inspection and Review Methods:

Inspections, walkthroughs and review methods are performed to detect errors. These methods are capable of determining only some errors. The drawback of these methods is that they cannot detect all the errors present in the software.

2) Program Design Method:- the design model of a program determines whether the quality of the program is good or not. An improper design model degrades the quality of the software program.

3) Syntax checking Method:- During the analysis of source code, the syntax errors are checked by the compiler. The static analysis methods such as strong typing and type checking are considered.

these two methods detect all the errors in the program and correct them.

4) Software Development methods :-

A software development method is an internal process which uses various methods for developing a software. For example, a statistical control method, configuration control method and automatic distribution of information method are used to develop software. These methods detect and remove most of the errors in the software and the programmer is unaware of the changes that take place after removing those errors.

1.2 Dichotomies:-

1.2.1 Testing versus Debugging:-

Testing	Debugging
1) The goal of testing is to detect errors in a program.	1) The goal of debugging is to detect errors and correct them.
2) Testing is initiated with known conditions.	2) Debugging is initiated with unknown conditions.
3) The output can be anticipated.	3) The output cannot be anticipated.
4) It is necessary to have planned, designed and scheduled constraints.	4) While in debugging, it is not necessary to have these constraints.
5) Testing finds out the reason for program's failure.	5) Debugging is the Programmer's justification.
6) It is not necessary to have design knowledge while performing testing.	6) It is sufficient to have detailed design knowledge for debugging.
7) The test design and execution are done automatically.	7) Designing and execution can't be done automatically in debugging.

1.2.2 Function Versus Structure Testing:-

Structural Testing	Functional Testing
1) Structural testing is also known as white box, glass-box testing.	1) Functional testing is also known as black box or closed box testing.
2) Structural tests are ~~performed~~ performed based on the knowledge of internal structure of the source code.	2) Functional tests are performed without the knowledge of internal structure of the software.
3) The test cases take finite time but cannot detect all bugs.	3) The test cases take infinite time and detect all errors.
4) Structural testing is less effective when compared to functional testing.	4) Functional testing is more effective than glass-box testing.
5) Different methods for performing white box testing are as follows, i) Statement testing ii) Decision testing iii) Condition testing	5) Different methods for performing black box testing are as follows, i) Expected inputs method ii) Boundary values method. iii) Illegal values method.

1.2.3 The Designer Versus the Tester:-

Designer	Tester
1) Designer is based on the structural specification of the system.	1) Tester is based on functional specification of the system.
2) He/She depends on the implementation details.	2) He/She is independent of the implementation details.
3) A software designer is responsible for designing and executing the tests.	3) A tester is responsible for designing and executing the tests.

1.2.4 Small versus Large:-

Small	Large
1) Small programs have only few lines of code.	1) Large programs have more no of lines of code.
2) They consist of few components.	2) They consist of large number of components.
3) Small programs does not require any technique for testing.	3) They require different types of techniques for testing.
4) Small programs are more efficient.	4) Large programs are less efficient.
5) Small programs are written by single programmer.	5) Large programs are written by different programmers.

1.3 Model for Testing :-

1.3.1 A model Project for Testing Process :-

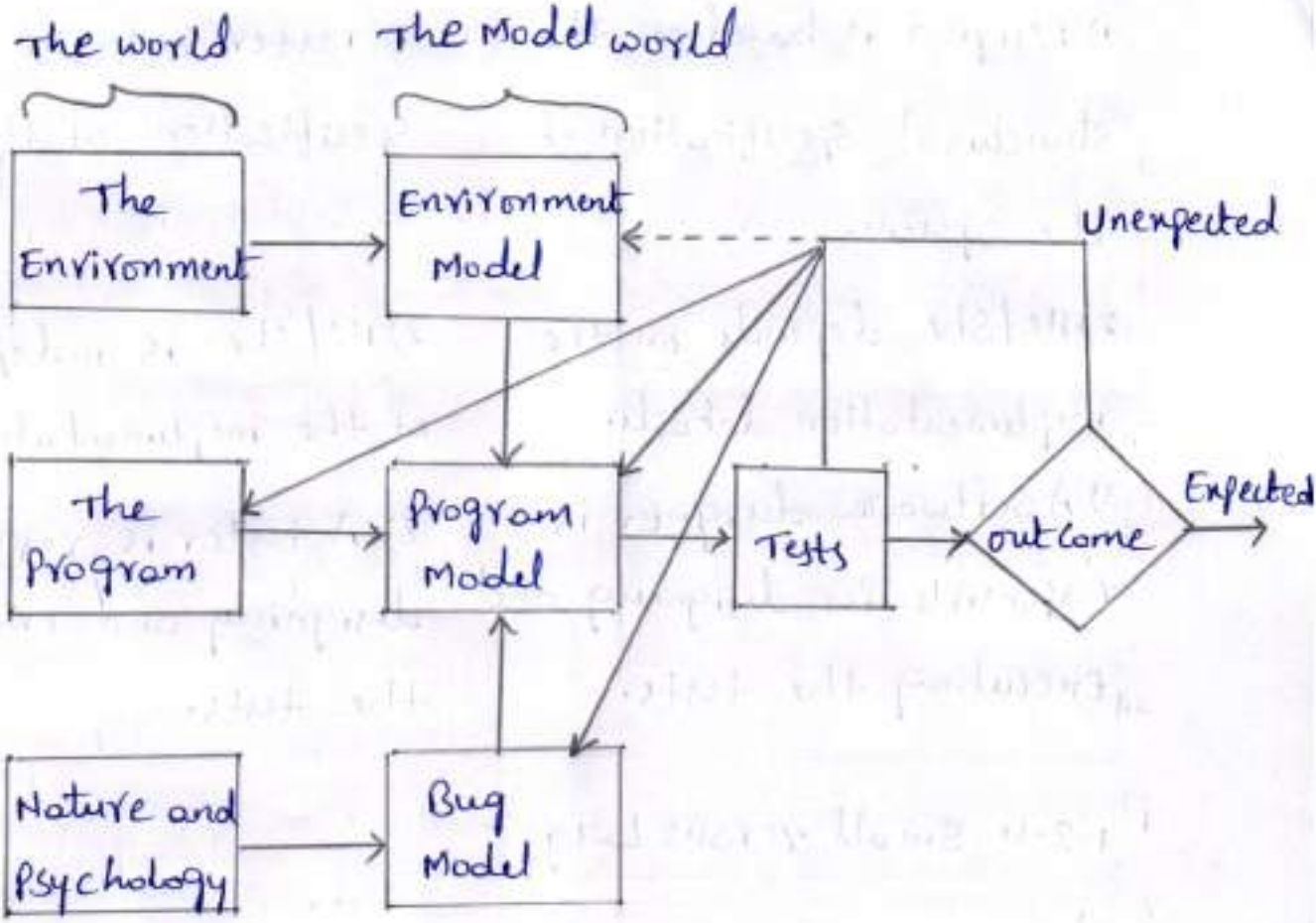

fig: A Model of Testing

Above figure is a model of the testing process. The process starts with a program embedded in an environment, such as a computer, an operating system, or a calling program. We understand human nature and its susceptibility to error. This understanding leads us to create three models, a model of the environment,

a model of the program, and model of the expected bugs, From these models we create a set of tests, which are then executed. The result of each test is either expected or unexpected. If unexpected, it may lead us to revise the test, our model or concept of how the program behaves, our concept of what bugs are possible or the program itself.

The program's environment is the hardware and software required to make it run. For online systems the environment may include communications lines, other systems, terminals and operators. The environment also includes all programs that interact with-and are used to create-the Program under test, such as operating system, loader, linkage editor, compiler, utility routines.

Programmers should learn early in their careers that it's not smart to blame the environment (i.e hardware and firmware) for bugs.

Hardware bugs are rare. So are bugs in manufacturer supplied software.

If testing reveals an unexpected results, we may have to change our beliefs (our model of the environment) to find out what went wrong. But sometimes the environment could be wrong, the bug could be in the hardware or firmware after all.

~~139~~

1.3.2 Software Testing Strategy:-

The overall strategy for software testing can be diagramatically represented in below figure using the Spiral model.

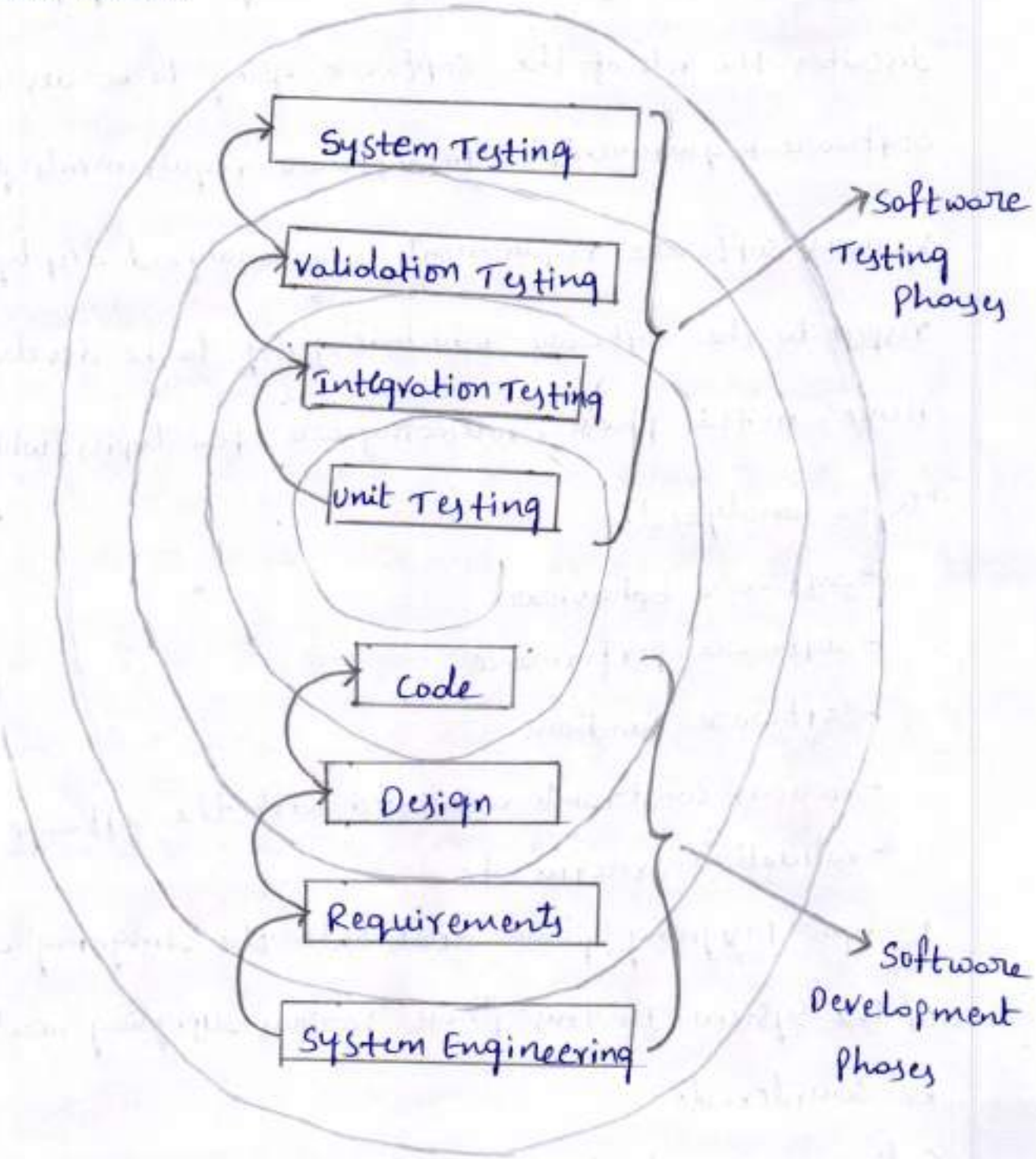

fig: Testing strategy using Spiral Model.

The spiral model depicting the software development process is diagramatically represented in above figure. the details of each entity in this model is as follows

System Engineering :- The system engineering phase describes the role of the software going to be developed.

Software Requirements :- In software requirements phase, various software requirements are analyzed deeply with respect to the software which is going to be developed. Hence, in this phase, following are few topics which are to be analyzed.

- Software behaviour
- Software performance
- Software functions
- Various constraints associated with the software
- Validation Criteria etc

Design :- Designing phase deals with the designing aspects of the system. In this phase various designing models can be considered.

Coding :- As usual the coding phase deals with writing of code for the software.

The four phases described above come under the Software development phases. Now Consider the Software testing phases which are associated with each phase of the Software development phases.

i) The Unit Testing phase:- As the name suggests in this phase each unit or module of the code is tested.

ii) Integration Testing:- Integration testing Concentrates on on testing the Software design.

iii) Validation Testing:- In validation Testing, The code is tested to verify that the Software satisfies all the requirements which were laid during the requirements phase of Software development.

System Testing:- In System testing the entire System is tested on a whole.

Each of these four phases are sequentially adopted in order to ensure that the Software which is delived to the given user satisfies all the functional, behavioural, performance, as well as the System requirements.

1.3.3 Testing and Levels:-

There are four different types of testing that can be performed on a software system. They are as follows.

1) Unit Testing:-
2) Component Testing
3) Integration Testing
4) System Testing

1) Unit Testing:- A unit is the smallest piece of source code that can be tested. It is also known as a module which consists of several lines of code that are processed by a single programmer. The main purpose of performing unit testing to reveal that a particular unit doesn't fulfill the specified functional requirements and also to show that the structural implementations is not similar to the expected structured designed.

Unit tests can be both static tests and dynamic tests. At first, static tests are performed followed by the dynamic tests to check the tests paths, boundaries and branches. Most of the unit tests are dynamic ✓

white box structural tests. These tests require either the execution of the software as a whole or parts of software.

2) Component Testing :- Component Testing is nothing but black box functional testing. This testing is used to test a single component or a group of components. A component is created by integrating one or more modules to form a single large component. A module is a component and the function it calls is also a component. Thus a component can either be an individual module or whole integrated system.

3) Integration Testing :- Integration is a process of combining smaller components to produce larger components. Integration testing is performed when the individual components undergo component testing successfully.

The main purpose of integration testing is to detect the inconsistency between these components. for example A and B are components that have gone through component testing successfully, but failed when integrated. Some of the situations where inconsistency arises are as follows,

i) When there is an improper call or return statement.

ii) When there is an inconsistent standard for data validation.

iii) When an inconsistent method is used for handling the data objects.

4) System Testing:- System testing exposes bugs that are not resulted from the components or from the inconsistency between the components. System testing is a black box functional testing to test the entire software system. It is performed to show the behavior of the system.

System testing is done either on the whole system that is integrated or only on the important parts of a system.

1.3.4 White Box Testing:-

White box testing deals with the internal logic and structure of the program code. It is also known as glass-box structural or open-box testing. In glass-box testing, test cases are derived based on the knowledge of software structure and its implementation. Using white box testing, the tester can detect which module or unit is not functioning properly.

White box tests can analyze data flow, control flow, information flow, exception and error handling techniques. these techniques are used to test the behavior of software.

A tester should have explicit knowledge about the internal working of the system being tested. Branch testing and path testing are the techniques used in white box testing. If the implementation details are changed, the tests should also be modified. The different methods used to perform white-box testing are as follows:

1) Statement testing
2) Decision testing
3) Condition testing.

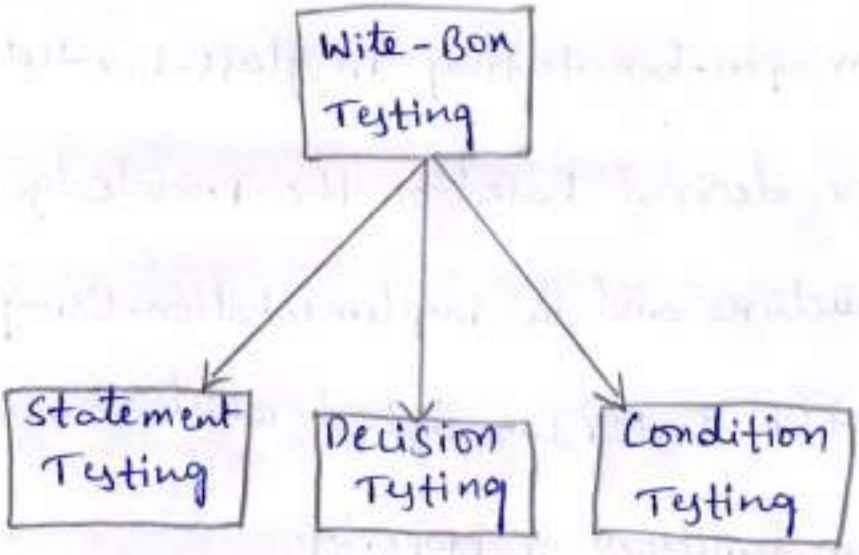

fig: White Box Testing

1) Statement Testing:- Test values are provided to check whether each statement inthe module is executed at least once. Statement testing executes statements when,

i) Optional arguments are available.
ii) User-provided parameters or procedures are available.
iii) planned user-actions are available.
iv) Defined error codes are available.

2) Decision Tyting:- Tyts are performed to check whether each branch of a decision is executed at least once. Decisiony requirey two valuey in standard Boolean decision tyting. But in cose of compound or nyted decisiony, the number of Boolean decision valuey can be greater than two.

3) Condition Tyting:- Tyts are performed to check whether each condition in a decision accepty all the neceSSary outputy atleast once. It alyo checky whether the entry pointy to the procedure or flow is ~~involved~~ invoked once. In cose of compound and nyted loopy, condition tyting is provided with multiple tyt valuey.

Advantagey of White Box Tyting:-

1) the applicotion is effective becouye of the internal knowledge of the program code.

2) The code is optimized.

3) the unneceSSary linyy of code which reyulty in hidden errory are removed.

Disadvantages of White Box Testing:-

1) White Box Testing is very expensive to perform since the tester should have the knowledge about the internal structure.

2) It is not possible to examine every unit for detecting hidden errors which results in application disaster.

1.3.5 Black Box Testing:-

Black Box testing is done without the internal knowledge of the program code. For example, in case of software engineering, test results with the known inputs and expected outputs but not with the functioning of a program. Because black box testing only deals with the specifications but not with the knowledge of the program.

Different methods used to perform black box testing are as follows,

1) Expected Inputs

2) Boundary Values

3) Illegal Values

1) Expected Inputs:- Expected inputs include values that are mostly expected at the time of executing a Procedure.

2) Boundary Values:- If the expected input value ranges from 1 to 123, tests performed on 1 and 123 values should ensure that results obtained are also within the boundary values of 1 and 123.

3) Illegal Values:- If an input value receives a value less than 1 or greater than 123, then those values are referred to as illegal values.

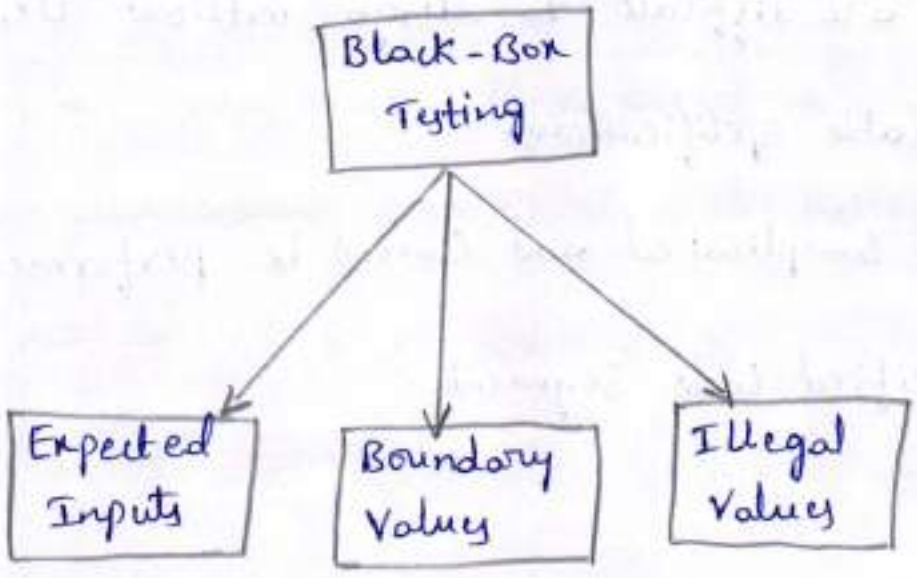

fig: Black Box Testing

Advantages of Black Box Testing:-

1) Black box testing is more effective than white box testing.
2) As no knowledge about the internal structure is required, the testers and programmers work independently of each other.
3) Tests are designed immediately after the completion of specifications.

Disadvantages of Black Box Testing:-

1) Black box testing takes long time to test each and every input.
2) Test cases are difficult to design without clear and understandable specifications.
3) Tests are complicated and cannot be performed directly to the specified code segments.

1.4 The Consequences of Bugs:-

1.4.1 The Importance of Bugs:-

Bugs are the errors that cease the program execution and sometimes generate an incorrect output. The importance of bugs depends on the following factors,

i) Frequency
ii) Correction Cost
iii) Consequential Cost
iv) Application Cost.

i) Frequency:- Frequency of a bug refers to the rate at which it occurs. The more frequently it occurs, the more will be its frequency.

ii) Correction Cost:- Once the bug has been detected, it need to be corrected. Correction cost is nothing but the cost that occurs during the error correction process. This cost depends upon the following two factors.

- The bug detection cost
- The bug correction cost

The cost of these bugs increased suddenly at the later stages in the development cycle, when the bug is discovered.

iii) Consequential cost :- It depends upon the consequences of bugs. There are many consequences of bugs which makes the system either to shut down or kill.

iv) Application Cost :- Application cost depends upon the number of installations i.e. this cost relies on the different applications that are used in the system. As the number of applications increases, the associated cost also increases. The application cost can control all the other costs.

A metric for the importance of bug is,

Importance of bug = Frequency cost * [Correction cost + Application cost + Consequential cost].

1.4.2 Various Consequences of Bugs:-

the various bug consequences are as follows,

1) Mild :- This consequence results in an incorrect, misspelled or wrongly aligned output.

2) Moderate :- this consequence effects the performance of the system and hence it results in a duplicate output.

3) Annoying :- Because of the presence of bugs in the system the performance of the system degrades. for example,

i) the names are ~~shot~~ shortened or changed.

ii) Bills for even null amount are sent.

4) Disturbing :- Because of this consequence, even the correct transactions are not executed. for example, an "automatic teller machine" refuses to process the 'withdrawal' transaction.

5) Serious :- the information about the transactions gets lost such as,

i) Tracking of transactions.

ii) Accountability of a transaction.

iii) Transaction occurrence.

When such information is lost the resulting bug is called a 'Serious bug'.

6) Very Serious:- This consequence results in reserving the operation of a transaction such as, deposit transaction is converted into ~~withdrawb~~ withdrawal transaction. Hence, the system is forced to perform incorrect transaction.

7) Extreme:- the consequence occurs frequently and is not limited to small number of users or transactions.

8) Intolerable:- When a large data is corrupted it is very difficult to perform the relovery process. In such situations shutting down the system is considered as the best option.

1.5 A Taxonomy for Bugs:-

The bugs are classified as follows.

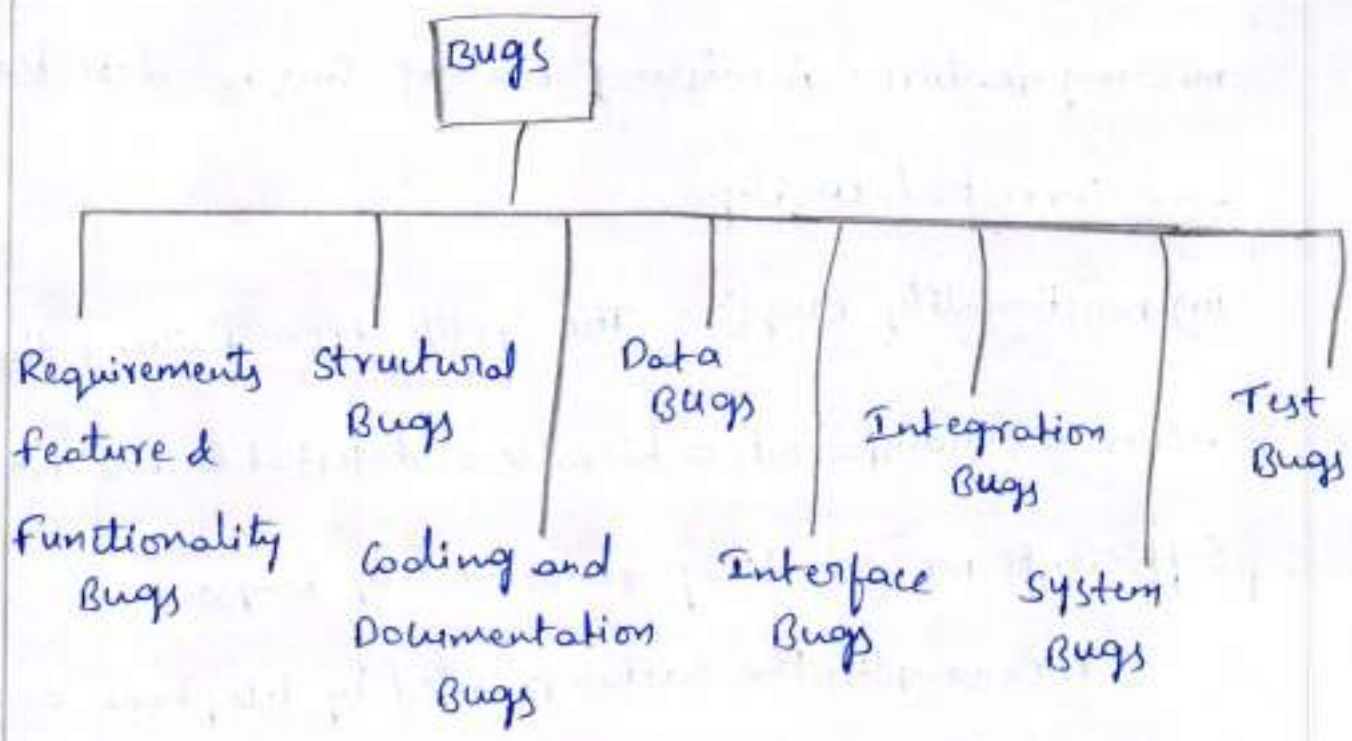

Fig: Different Types of Bugs

1.5.1 Requirements, features and functionality Bugs:-

i) Requirement Bugs: Requirements are expressed in the form of specifications. They are the major cause of bugs that are expensive to be fixed. The bugs rate varies depending on the type of application and the operating environment.

ii) Feature Bugs :- The difficulties that arise in feature bugs are due to the specification problems. A feature can either be an incorrect feature or a missing feature. A missing feature can be detected and corrected easily.

iii) Functionality Bugs :- The specification features that are accurate, transparent, achievable and testable are not sufficient for detecting functionality bugs.

For example, the services provided by telephone are call holding and call forwarding. Call holding provides an existing call to be on hold and receives the new call. Call forwarding redirects an incoming call to the other telephone. Based on this functionality, the above features have the following questions.

i) What happens to the third incoming call when there is an existing call on hold?

ii) What happens when a call is forwarded by an already forwarded call?

iii) What happens if a call is on hold when a forwarded call is active?

iv) What happens if a call is forwarded when a call is on hold?

1.5.2 Structural Bugs:-

Structural bugs consists of the following different bugs. they are as follows,

i) Control and sequence bugs

ii) Logic bugs

iii) Processing bugs

iv) Initialisation bugs

v) Data flow bugs and anomalies.

i) Control and Sequence Bugs:-

Control flow bugs are important because of the following reasons!

a) the control flow problems are popular in the area of software design and testing literature because of its adaptability. Therefore, it can be tested and detected easily in unit testing.

b) Researchers found that the inexperienced Programmers working on a software produce more control flow bugs than the experienced programmers.

ii) Logic Bugs:- Logic bugs show the behavior of the statements and operations. The logic bugs include design of test cases, incorrect explanation and combination of cases etc.

For example, after execution of few statements in a nested IF-THEN-ELSE statement, the truth value of the logical expression is determined. this stops the execution of further statements.

iii) Processing Bugs:- Processing bugs consists of algebraic, arithmetic, algorithm selection and mathematical functional bugs. these bugs occur if wrong data conversion methods are used for converting data from one format to another.

the other Problems associated with Processing bugs are, neglecting overflow of data, neglecting the difference ~~bet~~ between positive and negative zero, wrong use of greater than, greater than or equal, less than, less than or equal, expecting zero in floating point operations.

iv) Initialization Bugs:- Initialization bugs are detected by both experienced programmers and testers. Bugs are caused due to the wrong and unnecessary initializations. these bugs are frequent but less harmful and effects the Performance of the software.

v) Data flow Bugs and Anomalies:-

Data flow anomoliy arise when a data is attempted to be used for an unnecessary Purpose such as using an uninitialized variable, editing the data value but not using it attempting to use a non-existing variable, initializing a variable twice without using an intermediate value.

1.5.3 Data Bugs:-

Data bugs occur because of the bugs in the specification of data objects, the designs, the number of data objects and the initial values.

a) Static Data:- Static data are fixed and doesn't change its structure or content. Static data is analyzed based on the source code that is used to determine whether the piece of code is reachable or not.

b) Dynamic Data:- Dynamic data are not fixed and changes its content after a specified period of time. The life span of dynamic data is very short which is equivalent to the processing time of a single transaction. Dynamic data is analyzed based on the behaviour code that is used to determine whether the piece of code is reachable or not.

1.5.4 Coding Bugs and Documentation Bugs:-

Coding bugs create many other different types of bugs. the Syntax bugs are of no significance when the source language translator has enough capability for performing Syntax checking. But when a translator fails in detecting a Syntax error, a bug is occured in the software.

Documentation bugs are caused due to the small mistakes, wrong or incorrect comments in the statements

1.5.5 Interface Bugs:-

Interface bugs are as follows,

i) External interfaces
ii) Internal interfaces
iii) Operating System

i) External Interfaces:- The external interfaces makes the communication possible with the outside world. An important principle for interface design is that it should be robust. An external interface can either be a human or machine. The external interfaces use a protocol that is

Complex difficult to understand.

ii) Internal Interfaces:- Internal interfaces has the same principle as that of external interfaces but the actions related to internal interfaces are more controllable.

Internal interface bugs include inaccurate protection against corrupted data, improper function call, Protocol-design bugs, input and output design bugs and Parameter call bugs.

iii) Operating System:- operating System is associated with the program or software bugs. It includes both the hardware architecture bugs and interface bugs. The solution for operating system bugs are as follows,

a) Use specialists to write the interface programs.

b) Use declaration of macros explicitly for all system calls.

1.5.6 Integration Bugs:-

Integration bugs occur when the interfaces are integrated in between the assumed tested components.

The solution for integration bugs include methods like domain testing, Syntax testing and data flow testing.

i) Hardware Architecture:- Hardware architecture has both isolated and unreal nature during the processing of consecutive layers of operating system, compiler and other interrupting software.

The misunderstanding is because of the following reasons, incorrect paging mechanism, incorrect input/output device operation, incorrect input/output device location incorrect address generation, incorrect format, incorrect device-status code, neglecting hardware failures.

The solution for hardware architecture bugs are as follows,

a) Good knowledge of programming and testing is needed.

b) Centralized hardware program should be written by hardware specialists.

ii) Software Architecture:-

Software architecture bugs are the types of bugs that can interact with each other. Even if the procedures successfully pass unit and integration testing, they don't expose the software architecture bugs. These bugs occur when the system is loaded with many components. The causes for software architecture bugs are as follows,

a) Assuming that there are no interrupt signals.

b) Assuming that the program code is entered again.

1.5.7 System Bugs:-

System bugs are revealed when system testing is performed. System bugs occur when there is a complete interaction between different components such as program hardware, data and the operating system.

Transaction flow testing is the only technique that is applied directly to the system testing for detecting and eliminating system bugs.

i) Control and sequence Bugs:-

Control and sequence bugs are caused due to the following reasons.

a) Expecting that events arise in an explicit order.

b) Beginning a process before its requirements are satisfied.

c) Neglecting time.

d) Neglecting when the requirements have been satisfied.

e) Specifying incorrect priority level, Program level.

ii) Resource Management Bugs:-

Memory is fragmented into small resources which can be allocated dynamically such as queue blocks, buffer blocks, task control blocks. The bugs related with the resource management are due to the following reasons,

a) Required resources are frequent.

b) Incorrect resource used when different resources have the same design.

c) Resource assigned to the incorrect queue.

1.5.8 Test Bugs:-

Testers are not resistant to any sort of bugs. For performing system tests, complex conditions and databases are required. Due to this complexity, there are chances for the occurences of bugs while a code is being executed. Independent functional testing results in wrong understanding of specification.

The different solutions for test bugs are as follows,

1) Test debugging
2) Test quality assurance
3) Test execution automation.
4) Test design automation.

1) Test Debugging:- Debugging is the foremost step for testing bugs. Test debugging and Program debugging are different from each other because of the following reasons.

i) Test debugging is simple and easy if tests are properly designed.

ii) Test & debugging does not degrade the efficiency.

2) Test Quality Assurance:-

This remedy (solution) tests the quality of the software whether it satisfies the given specifications or not. Testystery assure that all the test actions are executed accurately.

3) Test Execution Automation:-

The bug removal and prevention are identical to the other automation techniques. To reduce the manual errors in the system, assemblers, loaders, computers have been developed. Test execution can either be done automatically or manually.

4) Test Design Automation:-

The test design can be automated since all the phases of software development cycle are automated. Because of this automation, bugs ~~are~~ rate decreases in both software as well as in test. It is difficult to perform data flow testing and domain testing manually.

Flowgraphs and Path Testing

2.1 Basic Concepts of Path Testing:-

Path Testing:- A sequence of statements which starts at an entry and ends at an exit by passing all the existing junctions, decisions etc. is known as path. Path may end at the same junction or the other or at any decision or exit.

2.1.1 Control Flowgraphs:-

A control flowgraph is a form of a flowchart which does not deal with the internal structure of the process rather it shows the data flow and the control flow between the processes. Every control flowgraph has some mandatory elements they are,

1) Process blocks
2) Decisions and Case statements
3) Junctions.

1) Process Blocks:- Process block is a block which consists of sequence of statements, which once initiated results in the