UNIT - I

Part - I : Introduction.

⇒ purpose of testing.
⇒ Dichotomies.
⇒ Model for testing
⇒ consequences of bugs
⇒ taxonomy of bugs
⇒ Flow graphs and path testing : Basics concepts of path testing.
⇒ predicates
⇒ path predicates and achievable paths
⇒ path sensitizing.
⇒ path instrumentation.
⇒ application of path testing.

Part-I: Introduction.

* purpose of Testing:-

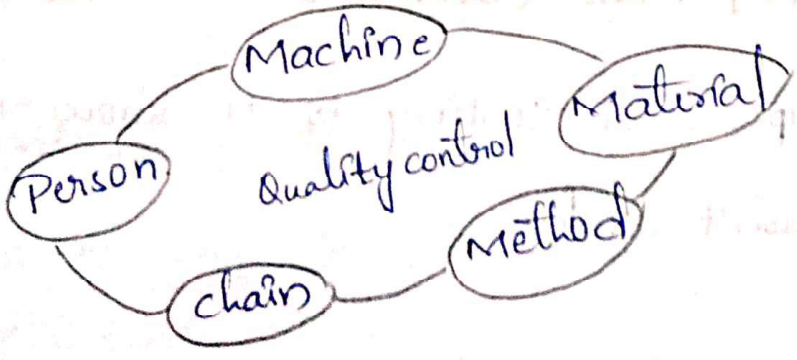
1). Testing consumes atleast half of the time and work required to produce a functional program.

2). History reveals that even well written programs still have 1-3 bugs per hundred statements.

* productivity and Quality in software:-

1). In production of consumer goods and other products, every manufacturing stage is subjected to quality control and testing from component to final stage.

2). If flaws are discovered at any stage, the product is either discarded (or) cycled back for rework and correction.



* Goals for Testing:-

1). Testing and test design are parts of quality assurance that should also focus on bug prevention.

2). Phases in testers mental life can be divided into the following 5 phases.

Phase 0: until 1956: (Debugging oriented)

1). There is no difference between testing and debugging.

2). phase 0 thinking was the norm in early days of software development till testing emerged as a discipline.

Phase 1: 1957 - 1978: (Demonstration oriented)

1). The purpose of testing here is to show that software works.

2). Highlighted during the late 1970's.

Phase-2: 1979-1982: (Destruction oriented).

1). The purpose of testing is to show that slw doesn't work.

phase-3: 1983-1987: (Evaluation oriented)

1). The purpose of testing is not to prove anything but to reduce the perceived risk of not working to an acceptable value.

phase-4: 1988-2000: (prevention oriented)

1). Testability is the factor considered here, once reason is to reduce the labour of testing.

2). Other reason is to check the testable and non-testable code.

* Test Design:-

1). software design is the process of implementing software solutions to one (or) more set of problems, one of the important parts of slw design is the software Requirements Analysis.

* Methods:-

1). Inspection Methods: Methods like walk throughs, desk checking, formal inspections & code reading appear to be as effective as testing, but the bugs caught do not completely overlap.

2). Design style:- while designing the software itself adopting stylistic objectives such as testability, openness and clarity can do much, to prevent bugs..

* Dichotomies :-

① . Testing Vs Debugging:-

S.NO	Testing	Debugging.
1.	Testing starts with known conditions, uses predefined procedures & has predictable outcomes	Debugging starts from possibly unknown initial conditions and the end can not be predicted except statistically.
2.	Testing can be done by an outsider	Debugging can be done by an insider

② . Function Vs structure:-

SNO.	Function	structure.
1.	It takes the user point of view - bother about functionality and features and not the programs implementation	It looks at the implementation details.
2.	In functional testing, the program (or) system is treated as a black box.	Things such as programming style, control method source language, database design, and coding details dominate

③. Designer Vs Testers:-

1). Test designer is the person who designs the test, whereas the tester is the one who actually tests the code.

2) During functional testing, the designer and tester are probably different persons.

④. Modularity Vs Efficiency:-

1). A Module is a discrete, well-defined, small component of a system.

2). If the system is modular, the tests can also be modular which results in the efficiently, of the system as well as in the efficiency of testing process.

⑤. small Vs Large:-

1). programming in the small, is what we do for ourselves in the privacy of our own offices.

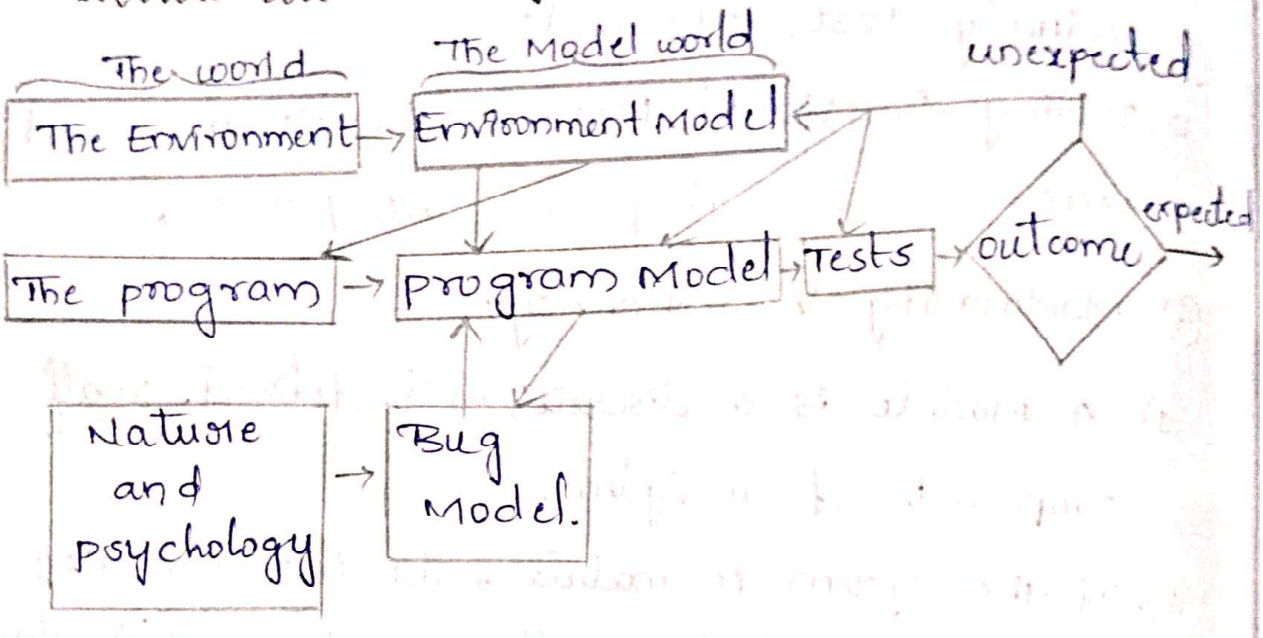
2). programming in the large, means constructing programs that consists of many components written by many different programmers.

⑥. Builder Vs Buyer:-

* Builder:- who designs the system and is accountable to the buyer.

* Buyer: who pays for the system in the hope of profits from providing services.

* Model for Testing:-



* Environments-

1). A programs environment is the hardware and software required to make it run.

2). The environment also includes all programs that interact with and are used to create the program under test such as OS, editor, compiler, etc.

* programs:-

1). Most programs are too complicated to understand in detail

2). The concept of the program is to be simplified in order to test it.

* Bugs:-

1). Bugs are more insidious than ever we expect them to be.

2). An unexpected test result may leads us to change our notion of what a bug is and our model of bugs.

* Tests:-

(1). Unit/component Testing:-

1). A unit is the smallest testable piece of sw that be compiled, assembled, linked, loaded etc..

2). A unit is usually the work of one programmer and consists of several hundred (or) fewer lines of code.

(2). Integration testing:-

1). Integration is the process by which components are aggregated to created larger components

2). Integration testing is testing done to show that even though the components were individually satisfactory.

(3). system testing:-

1). A system is a big component.

2). system Testing is aimed at revealing bugs that cannot be attributed to components.

* Consequences of bugs:-

1) Mild:- The symptoms of the bug offend us gently, a misspelled output (or) a misaligned printout.

2). Moderate:- outputs are misleading (or) redundant the bug impacts the systems performance

3). Annoying:- The systems behaviour, because of the bug is dehumanizing.

4). Disturbing:- It refuses to handle legitimate transactions.

5). Serious: It loses track of its transaction.

6). Very serious: The bug causes the system to do the wrong transactions.

7). Extreme:- The problems are not limited to a few users (or) to few transaction types

8). intolerable:- long term unrecoverable corruption of the database occurs and the corruption is not easily discovered.

9). Catastrophic:- The decision to shut down is taken out of our hands because the system fails.

10). Infectious:- what can be worise than a failed system? One that corrupts other systems even though it does not fail in itself; that

erodes the social physical environment, that melts reactors and starts a war.

* Taxonomy of Bugs:-

1). There is no universally correct way to categorize bugs.

2). The taxonomy is not rigid.

3). A given bug can be put into one (or) another category depending on its history and the programmers state of mind.

4). The major categories are:-

① Requirements Features & functionality bugs

↓

② structural bugs

↓

③ Data bugs

↓

④ coding bugs

↓

⑤. Interface, integration and system bugs.

① Requirements, Features and Functionality bugs:-

1) Requirements and specifications Bugs:- Requirements and specifications developed from them can be incomplete ambiguous, (or) self - contradictory.

2). Feature Bugs:- specification problems usually create corresponding feature problems

3). Feature Interaction Bugs:- providing correct, clear, implementable and testable feature specifications is not enough.

4). specification and Feature Bug Remedies:- Most feature bugs are rooted in human to human communication problems.

2. structural bugs:-

1) control and sequence bugs:- control and sequence bugs include paths left out, unreach -able code, improper nesting loops, duplicated processing, and pachinko code.

2). Logic bugs:-

1) Bugs in logic, especially those related to misunderstanding how case statements and logic operators behave singly and

combinations.

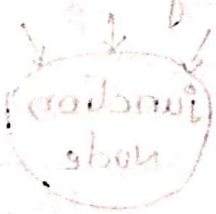
3. Data bugs:- Data bugs include all bugs that arises from the specification of data objects, their formats, the number of such objects, and their initial values.

4. Coding bugs:- coding errors of all kinds can create any of the other kind of bugs.

(5). Interface, integration and system bugs:-

1) External interfaces:- The external interfaces are the means used to communicate with the world.

2). Internal interfaces:- The internal interfaces are in principle not different from external interfaces but they are more controlled.



* Flow graphs and path testing:-

* Basics concepts of path testing:-

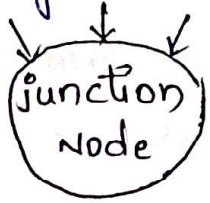
1). path testing is a method that is used to ~~testing~~ design the test cases.

2). To design test causes using this technique, four steps are followed.

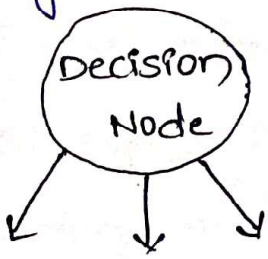
(1). control Flow Graph: A control flow graph is a directed graph which represents the control structure of a program (or) module.

2) A control graph can also have three Nodes. They are:-

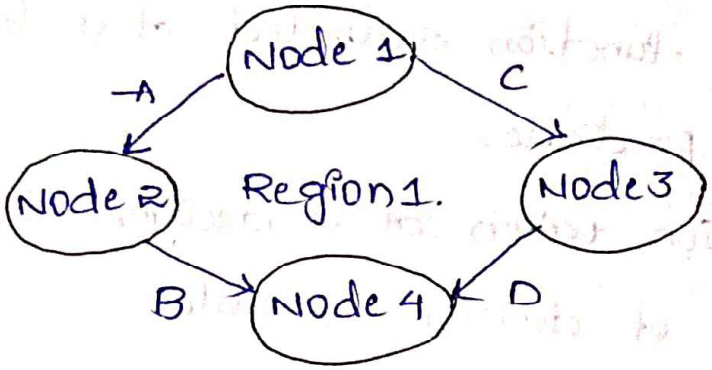
i). Junction Node: A node with more than one arrow entering it.



ii) Decision Node: A node with more than one arrow leaving it.



(iii) Region: Area bounded by edges and nodes.



2. cyclomatic Complexity:- The cyclomatic complexity is said to be a measure of the logical complexity of a program.

3. Independent paths: An independent path in the control flow graph is the one which introduces at least one new edge that has not been traversed before the path is defined.

4. Design Test cases from independent paths:- Finally, after obtaining the independent paths, test cases can be designed where each test case represents one (or) more independent paths. "

* predicates:-

1). The logical function evaluated at a decision is called predicate.

2). The direction taken at a decision depends on the value of decision variable.

* Example: A>0, x+y >=90.....

* path predicates:-

1). A predicate associated with a path is called path predicate.

* Example:- "X is greater than zero" is true

AND

"X+Y > =90" is false.

AND

"W is either negative (or) equal to 10" is true.

* Multiway Branches:-

The path taken through a multiway branch such as computed GOTO, case statement, (or) jump tables cannot be directly expressed in TRUE/FALSE terms.

* Inputs:- In testing, the word input is not restricted to direct inputs, such as variables in a subroutine call, but includes all data objects referenced by the routine whose values are fixed prior to entering it.

* Achievable paths:-

1). In software testing methodologies, "achievable paths" refer to the different paths (or) scenarios that can be taken during the testing process.

2) It helps identify the possible combinations of inputs and actions to ensure through testing coverage.

* Examples:-

1) successful login with correct username and password.

2) Failed login with incorrect username (or) password.

3) Login with a valid username but an expired password.

4) login with a valid username but an account that has been locked.,

* path sensitizing :-

1). Most of the normal paths are very easy to sensitize - 80% - 95% transaction flow coverage ($C_1 + C_2$) is usually easy to achieve.

2). The remaining small percentage is often very difficult.

3). sensitization is the act of defining the transaction.

4). If there are sensitization problems on the easy paths, then bet on either a bug in transaction flows (or) a design bug.

* path instruementation :-

1). Instrumentation plays a bigger role in transaction flow testing than in unit path testing.

2) The information of the path taken for a given transaction must be kept with that transaction & can be recorded by a central transaction dispatcher (or) by the individual processing modules.

3). In some systems, such traces are provided by the operating systems (or) a running log.

* Applications of path Testing:

Integration, coverage and paths is called components

↙

New code

↓

Maintenance

↓

Rehosting

(1). Integration, coverage and paths is called components:

1) path testing methods are mainly used in unit testing; especially for new software.

2) The new component is first tested as an independent unit with all called components and coreqursite components replaced by stubs.

(2). New code:

1). New code should always be subjected to

enough path testing to achieve C_2 stubs are used where it is clear that the bug potential for the stub is significantly lower than that of the called components.

2). Typically we will try to use the shortest entry/exit path that will do the task.

3. Maintenance:-

1). There is a great difference between maintanance testing and new code testing.

2). Maintenance testing is a completely different situation, it involves modifications which are accommodated in the system.

4. Rehosting:-

1) We get a very powerful, effective, rehosting process when C_1+C_2 coverage is used in conjunction with automatic (or) semiautomatic structural test generators.

2) software is rehosted because it is no longer cost effective to support the environment in which it runs the objective of rehosting is to change the operating environment.