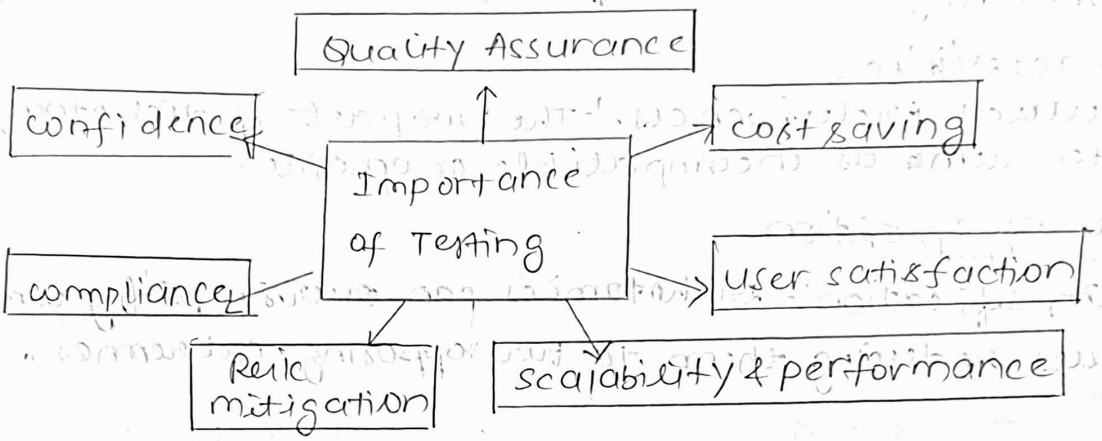
16/04/2025 ~~Soft skills 2nd class~~

1) Software testing:- Testing is crucial in SDLC, for ensuring software quality identifying defects early and reducing the future cost. Or software testing is a process used to identify the correctness, completeness and quality of a developed software

* Software testing is a process of verifying and validating whether the s/w product is working as expected.

Why testing is important

(1)




1) Quality Assurance:- Testing helps to ensure that software meets the required quality standards and it is free from defects.

2) cost saving:- Fixing the defects earlier in the development process is significantly cheaper than fixing them after s/w is released.

(3) user satisfaction:- Testing helps to ensure that s/w is usable and functionally meets user expectation leading to increased user satisfaction.

(4) Scalability and Performance:- Testing ensures that s/w can handle increased load and still can maintain optimal performance.

(5) Risk mitigation:- Testing helps to identify potential risk early in SDLC allowing time for developing mitigation strategies.

(6) compliance:- Testing ensures that s/w adheres/obeys/follows to relevant standards and regulation.

(7) confidence:- Rigorous (extensive) testing provides ~~testing~~ confidence that the s/w works as intended/expected/desired.

Date : 25/05/25

→ Dichotomies in STM

Dichotomy is a division or contrast between two things that are considered to be complete different or mutually exclusive.
ex:- good vs evil. → classic example of dichotomy where two opposing moral values are presented.

(2) Nature vs Nurture :- It's a debate in psychology which tells about relative contribution of genetics (nature) and environment to human development.

→ characteristics

1) Mutual Exclusiveness :- The two parts of dichotomy are often seen as incompatible or opposite.

2) Binary opposition

3) Simplification :- Dichotomies can oversimplify complex issues reducing them to two opposing extremes.

| TESTING | DEBUGGING |
|---|---|
| 1) Testing starts with known condition - it uses predefine procedures and has a predictable outcome. | (1) Debugging starts from possibly unknown initial conditions and the end can't be predicted. except statically |
| (2) Testing can and should be planned designed and scheduled | (2) Debugging is the deductive process |
| In debugging ~~procedure and can be constraint~~. | (3) Debugging is ~~rectifying~~ / ~~Testifyer~~ programmers Justification. |
| (3) Testing is a demonstration of errors ~~and rectifying the~~ | |
| (4) Testing proves a programmers failure. | |

| Functional Testing | Structuring Testing |
|---|---|
| 1) The program or system is treated as a black box. | 1) It does look at the implementation details |
| (2) It is subjected to inputs and the desired output and are verified for the specified behaviour. | (2) Both structural and functional test are useful both have limitations. and both target at different kind of bugs. |

| | |
|---|---|
| (3) functional testing takes the users point of view and bothers about fool functionality and features and not about the programs implementation | (3) structural test are inherently finite but can't detect all the errors. even if completely executed. |

25/05/26 (Monday)

(III) Designer vs Tester

| Designer | Tester. |
|---|---|
| (1) The one who creates a blue-print/plan for actual implementation. | (1) A tester is a person who tests the actual software. |
| (2) During functional testing the designer and the tester are probably two different persons. | (2) During unit testing the tester and the programmer preferrably are the same person. |
| (3) | (3) |

(4)

(IV) Modularity and Efficiency.

↳ that can be divided into sub modules.

1) A module is a discrete, well defined, small, components of a system.

2) Smaller the module, it is difficult to integrate; Larger the module, it is difficult to understand.

3) Moore both tests and system can be modular.

4) Testing can and should be organised into modular components.

5) Small independent test cases can be designed for independent modules.

(V) Small V/s Large.

1) Programming in Large means constructing programs that consist of many components written by many different programmers over years.

2) Programming in small: it is what we do for ourselves in own office/privacy.

3) Qualitative and Quantitative changes occurs with size and so for testing methods.

⑥ Builder vs Buyer (clients)
(developer).

1) Most of the softwares are written and used by the same organization there is a problem of accountability.
2) If there is no seperation b/w builder and buyer there can be no accountability.

→ Models for Testing

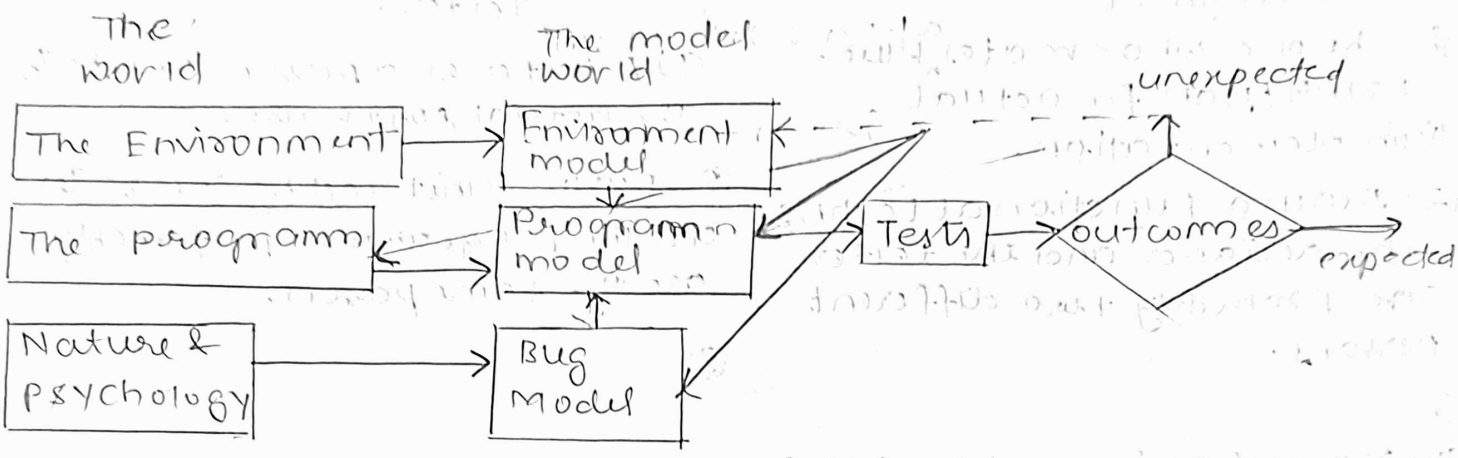



fig: Models for testing

Above figure is a model of testing process, it includes three models

1) The model of the environment
2) A model of the program
3) A model of expected bug

1) Environment Model:- This A program's environment is the hardware and software required to make it run.

- for online system the environment may include communication lines, terminals and operators.
- The environment also includes all the programs that interact with and are used to create the program under test. such as operating system, linkages, editors, loader & compiler.

2) program:-

Most programs are to complicated to understand to in detail. The concept of the program is to be simplified in order to test it.

If simple model of the program doesn't explain the unexpected behaviour we may have to modify that

model to include more facts and details.

(3) Bugs :-

Bugs are more in numbers than ever we expect, bugs are usually unable to detect test effectively and unable to certify most of the test programs.

(4) Tests :-

Tests are formal procedure inputs must be prepared to predict the output

Test should be documented and outcomes should be recorded

we do 3 different kind of testing.

(5) Unit testing

A unit is the smallest testable piece of a software that can be compile, assembled, links, loaded etc

unit testing is the testing done to show that the unit. unit testing is the testing done to show that the unit doesn't satisfy its functional specifications or implementation structure.

doesn't match with expected design. component doesn't satisfy its functional specification or implementation structure doesn't match with the expected design.

component is an integration of 1 or more units.

(6) component testing

It is a testing that we do to show that component doesn't satisfy its functional specification or implementation structure

② Integration

Integration is the process by which the components are aggregated to create larger components.

③ System Testing

A system is a big component. System testing is aimed at revealing the bugs that can't be attributed to components. It includes testing for performance, security, accountability, configuration, sensitivity and recovery.

## ⇒ BUGS AND TAXONOMY OF BUGS

1) Bug:- S/w bugs are defects or errors in the software code that lead to incorrect or unexpected program behaviour.

causes:-

(1) Coding mistakes, (2) Syntax errors, (3) logical errors, (4) incorrect data handling, (5) Design flaws/errors., (6) Errors due to h/w synchronization

Examples:- syntax errors, memory leaks, infinite loops etc.

importance of identifying bugs:-

→ finding and fixing the bugs is crucial for ensuring S/w stability, reliability, and a positive user experience.

→ The process of identifying and fixing the bug is called as debugging.

→ Taxonomy of bugs:-

The Taxonomy isn't rigid, a given bug can be put into one or another category depending on its history and programmers mindset.

The major categories of bugs are.

(1) Requirements, features, functionality bugs
(2) Structural bugs
(3) Data bugs
(4) Coding bugs
(5) Interface, integrations and system bugs
(6) Design bugs

^
Test.

(1) Requirement & specification bugs :-
→ Requirements and specification developed from them can be incomplete / ambiguous, or self contradicting.
sometimes requirements can be misunderstood or impossible to understand.
→ These bugs are earliest to invade the system and the last to leave.

1.b
(2) feature bug :-
→ Specification problem usually creates corresponding feature problem.
→ A feature can be wrong, missing, or superfluous (serving no purpose).
→ A missing feature or a case is easier to detect and correct

1.c
(3) feature interaction bug :-
↳ feature usually comes in groups. The features of each group and the interaction of the features within the group should be well tested.
↳ The problem is unpredictable interaction between feature groups or even between individual features.

1.d
↳ specification and feature remedies
most feature bugs are rooted in human to human communi-cation problem. One solution is to use high level formal specification language.

(2) Structural bugs :-
(2.1) control and sequence bugs :- control and sequence bugs include unreachable code, improper nesting of loops, loop back, loop termination criteria incorrect, missing process steps, duplicated processes unnecessary processing or ill-conceived switches (improperly planned)

Date :- 28/05/25
(2.2) :- Logic bugs :- Bugs in logic especially those creating mis-understanding how the case statement and the logic operator will behave in unit / combination.
- If bugs are part of logical processing they are characterized as processing bugs
- If the bugs are part of logical expression (control flow -

statements then they are characterized as control flow bugs.

3) Processing bugs: It includes arithmetic bugs mathematical functions evaluations and algebric bugs.
ex: Improper use of greater than or equal to, incorrect conversion of data.

(4) Initialization bugs: Typical initialization bugs include forgetting to initialize variables, initilization in wrong format. etc.
- Initialisation bugs can be in proper and super-flaws.
- ~~most initialisation bugs are up~~

(5) Data flow bugs and anamolies: most initialization bugs are special case of data flow anamolies.

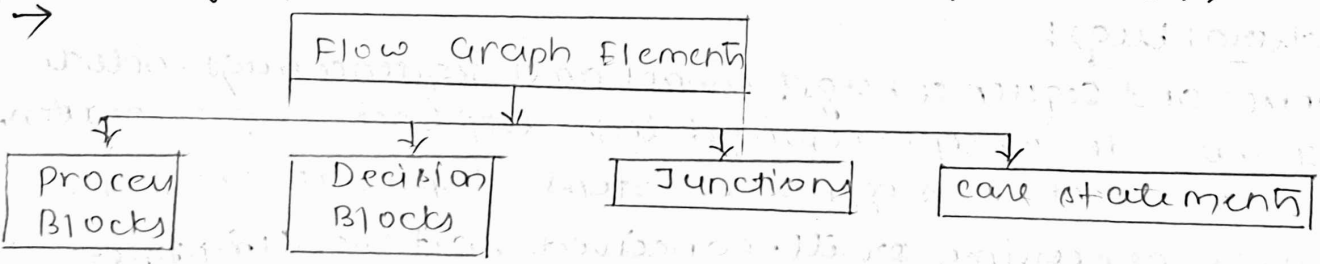
---

19 June 2025:

Path Testing: Path testing is the name given to a group of test techniques for selecting a test path to test a program
- path testing requires complete knowledge of the program structure. it is
↳ It is more often used by the programmers to unit test their own code.
↳ The effectiveness of path testing rapidly (degraded) deteriorates as the size of software increases.

↳ control flow Graphs: (nodes, linkages)
The control flow graph is a graphical representation of program control structure. It uses a elements name process block, decisions and functions.

→ flow graph are similar to the earlier flowcharts.
→



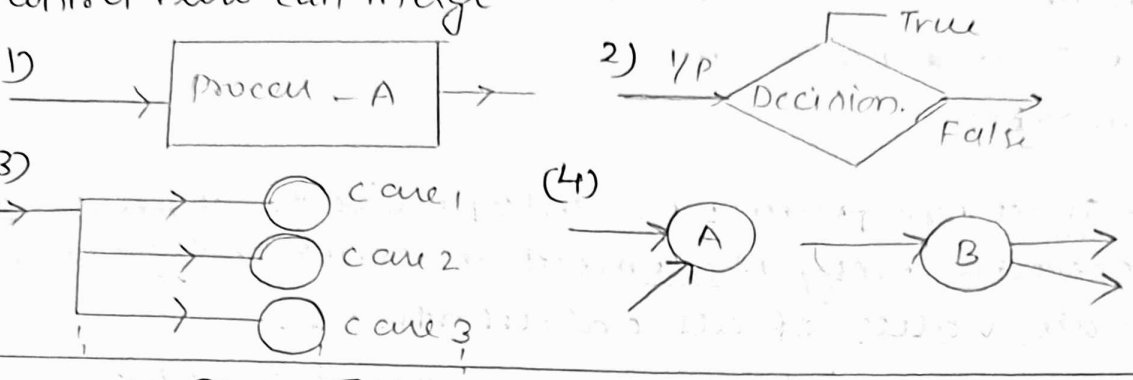

1) Process Blocks:
→ sequence of statements / instructions, if any one statements is missing in the process block it'll result in failure of that process block.
→ A process has one entry and one exit, a macro/function call

2) Decision Blocks: A Decision is a program point at which the data flow or control can diverge.

→ conditional branching / conditional stmt conditions are examples of decisions.

→ Case statements: Multiway branching / decision

→ Junctions: A junction is a point in the program where the control flow can merge




Path → 20/JUNE/25 :-

Path predicate Expression

↳ A path predicate expression is a set of boolean expressions all of which must be satisfied to achieve a selected path

(Ex) A: $x = 20$, B: $x + 3x2 + 20 > 0$; C: $x > 17$, D: $3x2 < 17$

path predicate expression for above example is. $(ABC + CD) = A + B$

path: path refers to a sequence of statements or instructions executed within a program from a starting point to destination.

path testing: path testing is a white box testing-based technique based on control structure of a program/module.

→ Here a control flow graph is prepared and various possible paths present in graph are executed as a part of testing.

→ path testing process: It requires complete program knowledge.

To design test cases using these techniques four steps are to be followed.

1) Constructive control flow graph
2) computing cyclomatic complexity of the graph, (counting no. of cond/decision)
3) Identifying independent paths used in program
4) Design test case from independent paths. to be continue!

Date: 21 June 2025 Sa

29 Path Instrumention :- selecting a specific path

Path instrumentation is what we have to do to confirm that the outcome was achieved by the intended path.

The types of instrumentation methods include.

(1) Interpretive Traced Program

(2) Traversal marker

(30) → Interpretive Trace Program : An interpretive trace program is one that executes every statement in order and records the intermediate values of all calculations.

- They trouble with interpretive trace program is that they gave us far more information that is required.

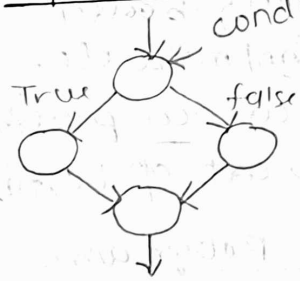
→ Traversal Marker : It is a simple & ~~in~~effective form of instrumentation.

It will be a ~~work~~ naming every link by lower case letter.

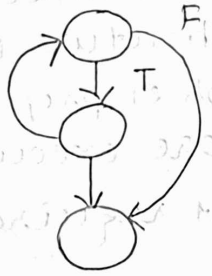
Instrument the links so that links name is recorded when the link is executed.

Date: 23 June 2025 Representation of conditioning/looping in CFG

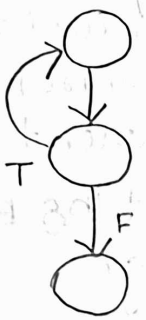
1) if else




(2) while




(3) do-while

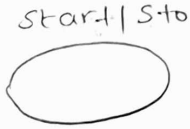



(1) perform the operation

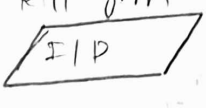
(2) check the condition

~~example~~ : ~~Sub Algorithm~~ flowchart symbols

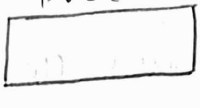
(1) start/stop



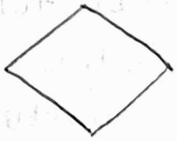
(2) I/P



(3) process



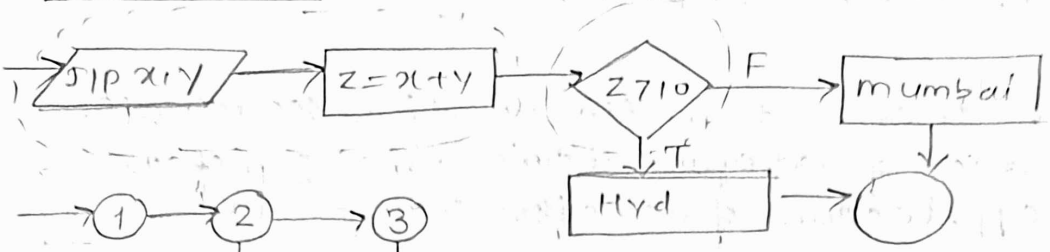
(4) condition.

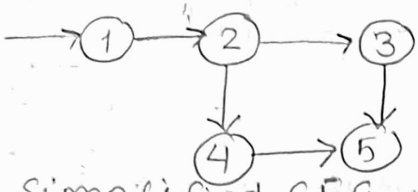


(5) → flow

(6)

## Flowchart

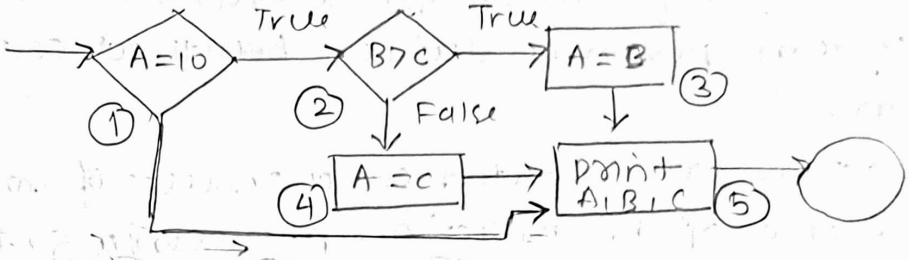


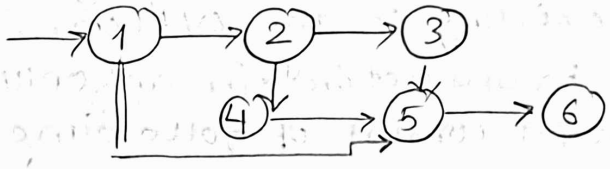

Simplified CFG

Ex (11) If A=10 Then
If B > C
A = B
else A = C
endif
endif
print A, B, C

1) Flowchart




2) CFG




⇒ Advantages of CFG:

(1) clear visualization of program flow.
(2) Basis for software Testing (e.g. white-box testing)
(3) Helps in program Analysis and optimization (Dead code, loop optimizations, constant propagation).
(4) Detects infinite loops or unreachable code
(5) Aids in Debugging & maintenance
(6) support security Analysis (unsafe path, unchecked inputs, Buffer overflows.
(7) support compiler construction.