7/7

SE - Practitioners app, Roger Pressman
SD Testing principles and practices - Srinivasan D, Rajiv

Software Testing

Process of exercising the software products in predefined ways to check if the behaviour is the same as the expected behaviour

Goal of testing - To find errors

CO1 : Apply the principles and axioms of testing

Objectives of testing:

- To verify requirements
- To detect defects early
- Validate user experience
- Test integration
- Security and compliance
- To enable regression testing
- Measure software quality
- Support confident release
- Reduce post release cost
- Improve development process

Terminologies :

- Error mistake (By identified developers) misunderstanding / misconception
- Fault / Defect deviation b/w actual and expected result (By testers)
- Failure (By customers) observable incorrect behaviour

(Error leads to Fault; Fault leads to Failure)

- Incident
- Verification
- Validation compare with reqr

Testcase - specified input (input to test system), expected output, conditions

Software Testing:
Process to identify correctness

Ex: Validate password:
7 characters
1st ⇒ no
2nd to 7th ⇒ alpha numeric

1st ⇒ 10 ways
next chr ⇒ 62 combinations

Total combinations for entire password = 10 * 62 * 62 * 62 * 62 * 62 * 62 testcases

Types of error:
Syntactic
Calculation
Flow Control

Fault - representation of an error

Types of defects:
Latent defect
Masked defect
Defect Cascading

Incident: symptom associated with a failure that alerts

Are we building the system right? - Verification

Are we building the right system? - Validation

Testcase Template:

Pre condition

Post condition

Test suite - collection of testcases

Test script - step by step instructions that describe how a test case

Test characteristics / Attributes of a good test:

* Has a high probability of finding an error
* Not redundant
* Should be best of breed
* Neither too simple nor too complex

7 Principles of ST:

→ Exhaustive testing - not possible
→ Defect Clustering 80% issues comes from 20% modules
→ Pesticide Paradox
→ Testing shows a presence of defects
→ Early Testing
→ Testing is content dependent
→ Absence of error - fallacy

Verification:

Process of evaluating a system or component determine whether the products of a given phase satisfy the conditions imposed at the start of that phase.

Are we building the product right?

Validation:

Process of evaluating a system or component during or at the end of development process determine whether it satisfies the specified requirements.

Are we building the right product?

9/7

Test Process :

Planning and control
Analysis and Design
Implementation and execution
Evaluating exit criteria and reporting
Test closure activities

Planning and control:

Test planning - major tasks :

Analysis and Design :

To review test basis
To identify test conditions
To design test

Implementation and execution :

Test condition ⇒ Testcases

Test execution - major tasks :

Evaluating exit criteria and Reporting :

Test Closure Activities:

Done when software is delivered

When all the information has been gathered

Process Models :

Linear Model :

V-Model : V - Verification & Validation

Planning

Acceptance Testing

When to use V-model?

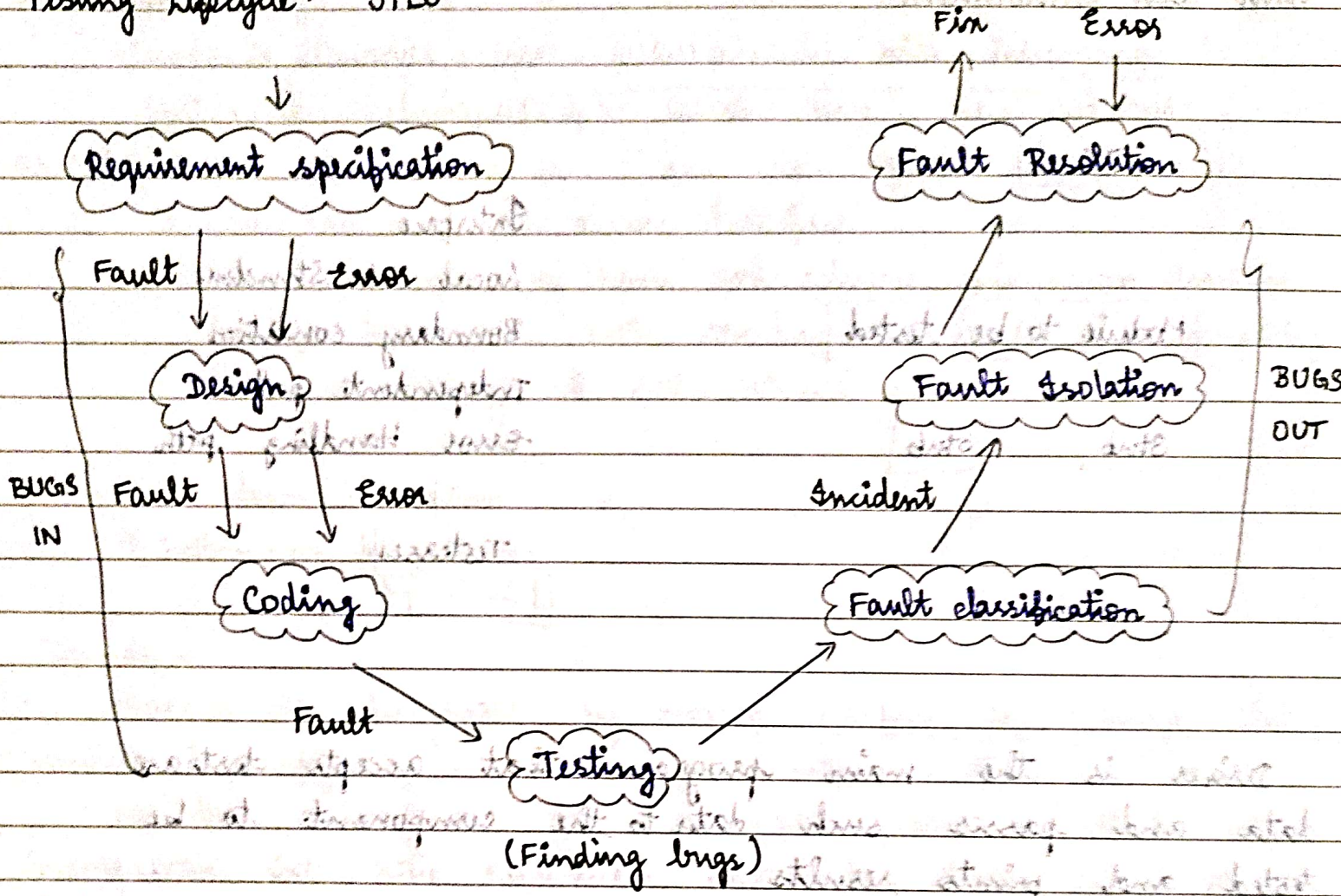
Systems requiring high reliability
All requirements are frozen ⇒ fixed, unchanged
Solution and Technology ⇒ known

Strength of V-model:

Test levels: (4)

Unit testing
Integration testing
System testing
Regression testing User Acceptance testing
In ll let carried out

Testing Lifecycle : - STLC



Unit Testing :

Testing done on the smallest unit of software design is Software component or module

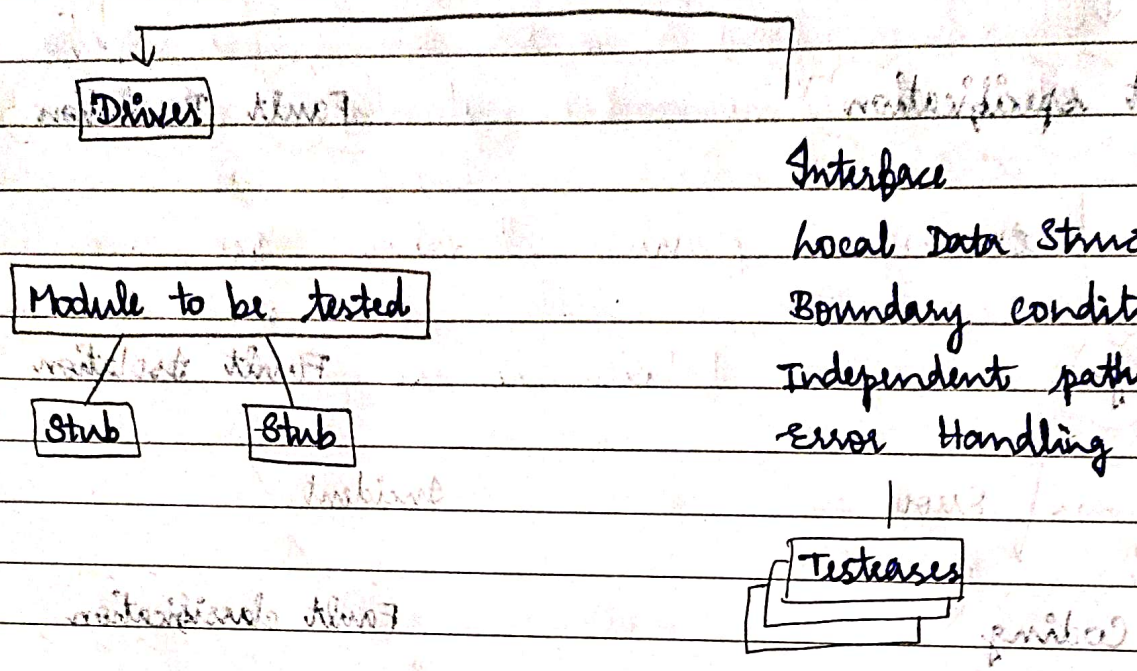
Modules

* I/f - Interface
* Local DS Data Structures
* Boundary condition
* Independent path
* Error Handling path

with the help of

Testcases

Unit Test environment



Driver is the main program that accepts testcase data and passes such data to the component to be tested and prints results

Stub is the subordinate or dummy sub program invoked by the component to be tested

Integration Testing:

Used to uncover errors associated with interfaces

Systematic technique for constructing the software architecture

Ex: Data can be lost across interface

: One component can have an adverse effect on another

Sub functions may not produce the desired functionality

Problems with global data structures

⇒ Top-down integration

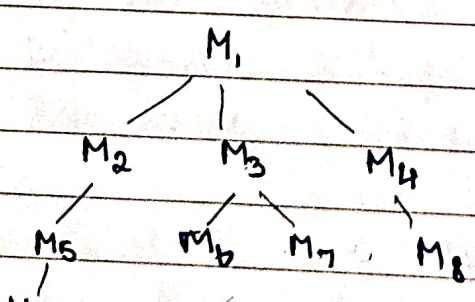
⇒ Bottom-up integration

Top-down:

Modules are integrated by moving downwards through the control hierarchy from the main control module.

Modules subordinate to the main control module are incorporated into the structure in either depth-first or breadth-first manner

level wise Verifies major control or decision points early in the test process



Tests are conducted as each component is integrated

Regression Test may be conducted

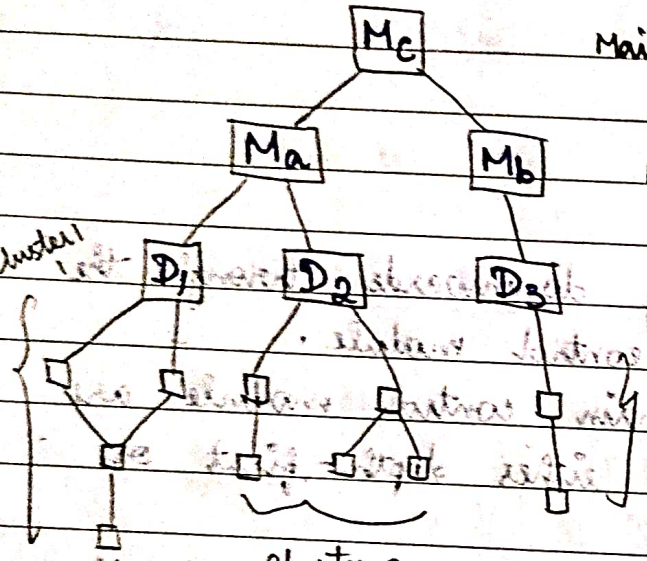
Bottom-up:

Testing with atomic modules ie components at the lowest level.

Low-level components are combined into clusters / builds

Clusters are combined moving upwards in the program structure

Need for stubs is eliminated



System Testing :

Testing conducted on the complete integrated system to evaluate the system compliance with its specified reqs.

Tests both functional and non-functional aspects of the product

Ensures all the reqs are met

System Testcases are developed

Done after unit and integration testing

Done to ensure that the product is ready for moving to user acceptance testing

- Performance testing
- Scalability testing
- Reliability testing
- Stress testing
- Interoperability testing

Functional testing :

Testing the product's features or functionality

Ex: Deployment testing

Beta testing

Non-functional :

Testing the products's quality factors like reliability, scalability, etc...

Testing aspects	Functional Testing	Non-functional Testing
Involves	Products features and functionality	Quality factors
Tests	Product behavior	Behaviors and experience
Result conclusion	Simple steps written to check expected results	Huge data collected and analyzed
Result varies due to	Product implementation	Product implementation, resources and configurations
Testing focus	Defect detection	Qualification of product
Knowledge required	Product and domain	Product, domain, design, architecture, statistical skills
Failures due to	Code	Architecture, design, code
Testing phase	Unit, component, integration, system	System
Testcase repeatability	Repeated many times	Repeated only in case of failures and for diff configuration
Configuration	One-time setup for a set of testcases	Configuration changes for each testcase

Regression testing :

Re-execution of some subset of test that have been already conducted to ensure that changes have not propagated unintended side-effects or additional errors

Ex: New I/O error
New control logic
New data flow path

Done ~~prod~~ periodically as proactive measures

⇒ Regular regression testing
done b/w test cycles

⇒ Final regression testing
done in final build before release

RT used to ensure : defect fixes works
No side effects or new errors
Should focus on impact of defect fixes than criticality of the defects

Regression test suite contains 3 classes of testcases:

- Sample tests to exercise all software functions
- Additional tests on functions likely affected by changes
- Test on software components that are changed

User Acceptance Testing (UAT): β-testing

The final test before releasing the s/w to end users.

Customer verifies if the s/w meet the needs and expectations

Verifies whether the s/w is ready for production / deployment

Validating the software against business requirements

Ensuring the s/w meets customer expectations

Getting final approval from customers

Testing should began during requirement phase - why?

Cost reduction

Reduced rework

Improve quality

Reduced Risk

To get clear requirements

Better collaboration b/w testers, developers, stakeholders

Psycological factors influencing software testing:

Mindset

Cognitive bias

Communication skills

Ability to manage stress and pressure

Characteristics of good testcase:

- Clear and concise
- Unique and Non Repitative
- Traceability
- Maximum coverage
- Consistency in results
- Reusability
- Aligning with customer requirements
- Context based

Sample Testcase Format / Template:

- Testcase id
- Testcase Description
- Test steps
- Pre-conditions
- Test data
- Expected result
- Actual result
- Status, comments

Test Policy:

Formal document that outlines the organization's approach to software testing

It sets the goals, responsibility, standards, methods to ensure software quality

It describes the roles and responsibility of the test team, test standards and processes, test tools and techniques, quality criteria and metrics

Steps to establish Testing Policy:

⇒ Define objectives:
- Ensuring product quality
- Early detecting of defects
- Risk reduction
- Ensuring regulatory compilance

⇒ Identify scope:
- Functional testing
- Non-functional testing
- Manual and automated system
- Testing in agile or traditional environment

⇒ Set responsibilities:
- Test manager:
  - Derives policy
  - Ensures compilance
- Q/A engineers execute test plans and report issues
- Developers conduct unit testing
- Stakeholders review quality outcomes

⇒ Define testing approach:
- Manual vs Automated testing
- Regression and Smoke testing

⇒ Establish entry and exit criteria:
- When testing can start and end

⇒ Select tools and standard:
- Testing tools like Selenium, JIRA, JMeter
- Frameworks like Agile, VModel

⇒ Determine metrics:
- Defect density
- Testcase pass or fail rate
- Code coverage
- Defect turn around time

⇒ Ensure communication:
- Test policy is communicated to all stakeholders

Structured Approach to testing: Refer to SDLC diagram

A systematic repeatable process to plan, design, execute and evaluate tests

Phases in a structured testing life cycle:

⇒ Test planning

What needs to be tested?

How the testing is to be performed?

What resources are needed?

Timelines, Risk

⇒ Test design

Deriving testcases from requirement specification using techniques like equivalence partitioning, boundary value analysis, etc.

⇒ Test environment setup

Test tools, data and configuration

⇒ Test execution

Running testcases, loggging results, reporting defects

⇒ Defect tracking

Exit criteria, Evaluation, Reporting

⇒ Test closure

Summary report

Lessons learned

Metrics

Verification	Validation
→ A static process of verifying documents, design and code	A dynamic process of validating testing the actual product
→ Does not involve executing the code	Involves executing the code
→ Human based checking of documents / files	Computer based execution of program
→ Target is requirements specification, application architecture, high level and detailed design and database design	Target is actual product - a unit a module, a set of integrated modules and the final product
→ Uses methods like inspections, walkthroughs, desk-checking, etc...	Uses methods like black-box, gray-box and white-box testing
→ It generally comes first - before validation	It generally follows verification
→ Can catch errors that validation cannot catch	Can catch errors that verification cannot catch
→ Are we building the product right?	Are we building the right product?

Test factors:

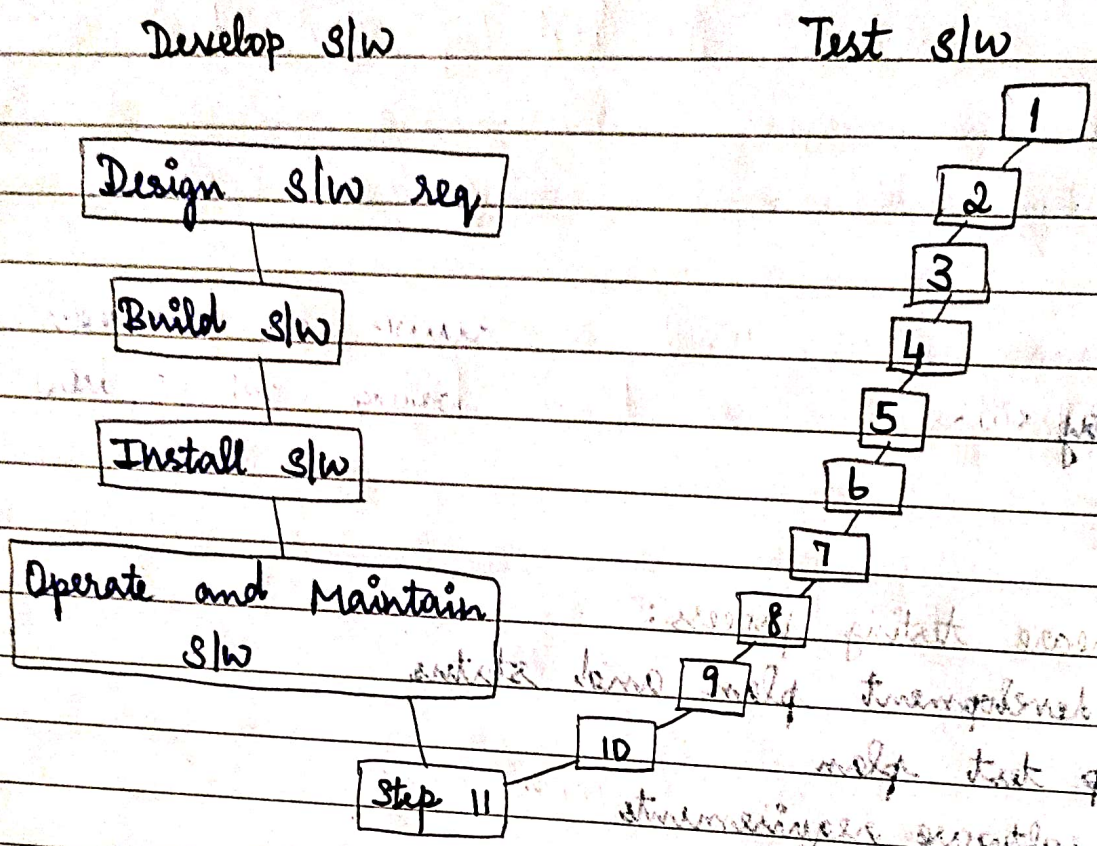
- Correctness
- Usability
- Performance
- Security
- Reliability
- Compatibility
- Integrity

11 steps in software testing process:

1) Assess development plan and status
2) Develop test plan
3) Test software requirements
4) Test software design
5) Build ~~face~~ phase testing
6) Execute and record result
7) Acceptance test
8) Report test results
9) Software installation
10) Test software changes
11) Evaluate test effectiveness

Software testing process includes V - concepts ; V represents both software development process and software testing process

V-Model:


Develop S/W
Test S/W
Design S/W req
Build S/W
Install S/W
Operate and Maintain S/W
Step 11
1
2
3
4
5
6
7
8
9
10

Risk - uncertainty (fear of unknown)

Problem - some event which has already occurred but risk is something that is unpredictable

Known Risks
Predictable Risks
Unpredictable Risks

Risk Exposure:

Quantified loss potential of business

$RE = P \times C$

P - Probability of occurrence of a risk

C - Cost to the project should the risk actually occur

P = 80% 18 of 60 software components will have to be developed

Cost for developing 18 components = $25,000

$RE = P \times C = 0.80 \times 25000 = \$20000$

Risk Assessment Matrix:

Severity: Impact of a risk and the negative consequences that would result

Likelihood: Probability of risk occurring

Severity:

→ Insignificant - Risk that bring no real

→ Minor

→ Moderate

→ Critical

→ Catastraphic

Likelihood :

→ Unlikely

→ Seldom

→ Occasional

→ Likely

→ Definite

Risk Management Matrix:

Risk Assessment Matrix | Risk Matrix | Risk Heat Map:

Tool used to evaluate and prioritize risks based on their likelihood and

Key Components:

Risk Identifier

Risk score = Likelihood x Impact

Colour-coded grid (GYO

representation

CO2 - Apply software testing techniques for testing software

White Box / Glass box testing:

Testcase design philosophy that uses control structure to derive testcases

All independent paths have to be exercised at least once

All logical decisions are true and false sides

All loops, all internal data structures

White box:

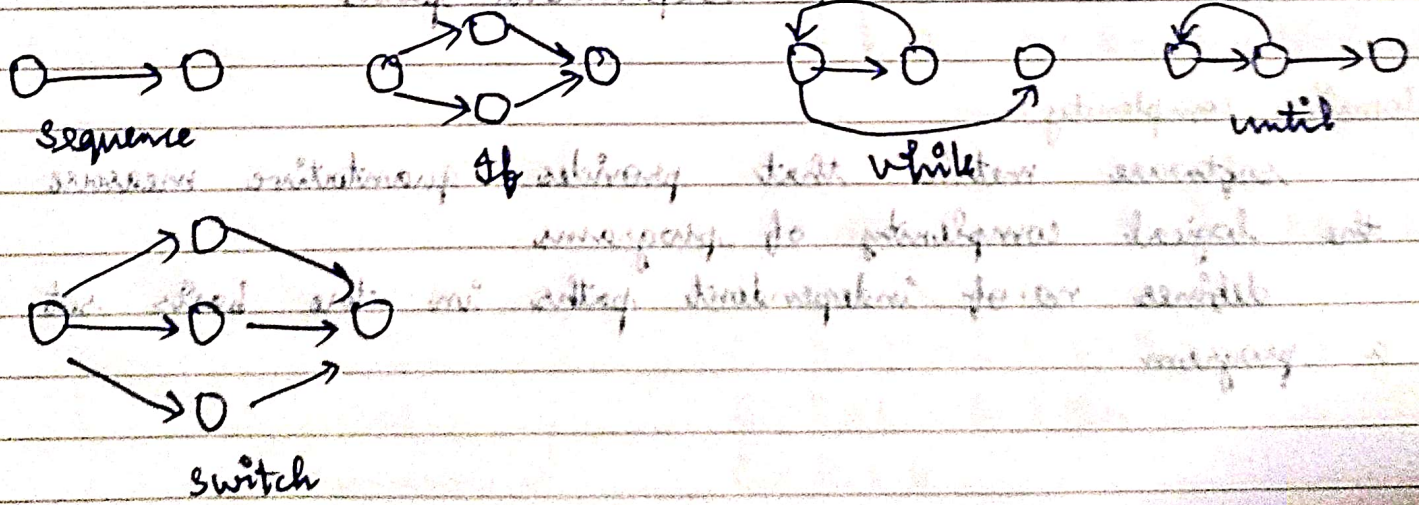
Basis path testing - helps to find logical complexities

control structure testing - to improve performance of white box

Basis path testing:

To determine the basis set of execution paths

Flow graph that depicts logical control flow is used



Flowgraph:

○ Node

○* Predicate node (multiple paths)
↳ contains condition

Cyclomatic complexity - no. of independent paths

$C = E - N + 2$

E - Edges N - Nodes

$C = P + 1$

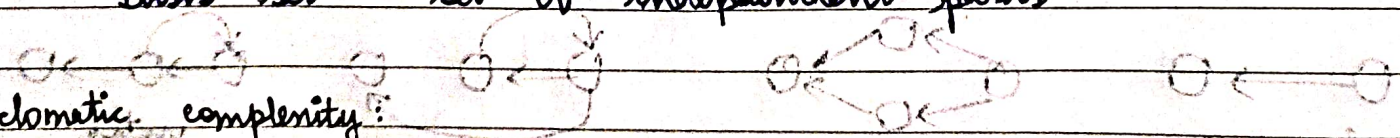
P - Predicate nodes

$C = R$

R - No. of regions - closed boundary in a flowgraph
↓
external env is also added

Independent program paths:

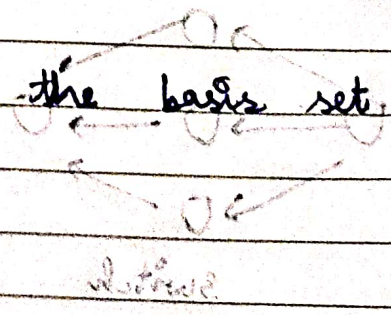
Basis set - set of independent paths

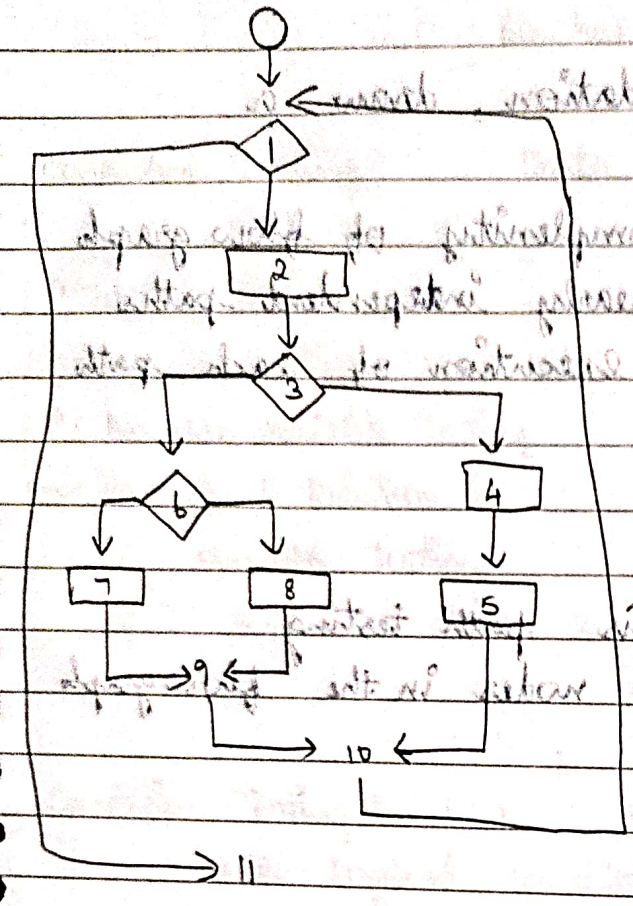


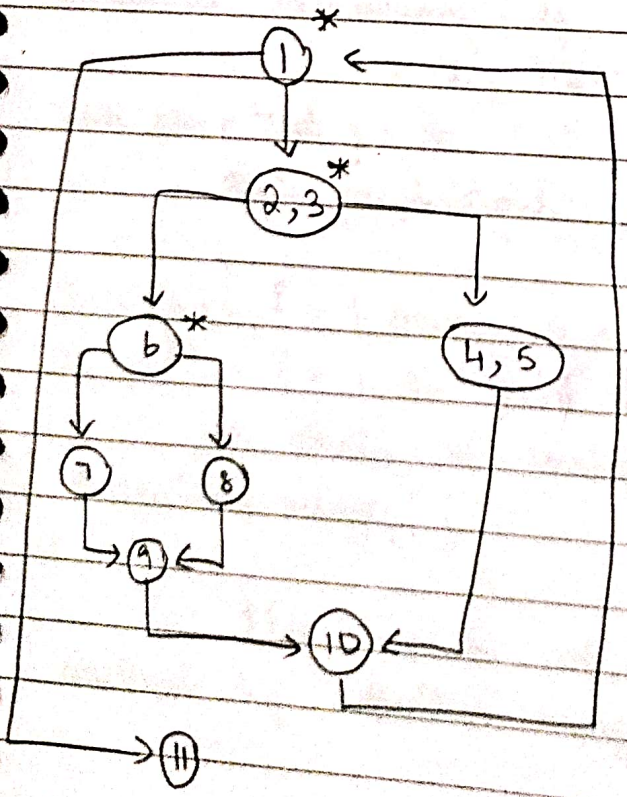
Cyclomatic complexity:

software metric that provides a quantitative measure of the logical complexity of programs.

defines no. of independent paths in the basis set of a program







Paths :

1 - 11

1 - 2, 3 - 4, 5 - 10 - 1 - 11

1 - 2, 3 - 6 - 7 - 9 - 10 - 1 - 11

1 - 2, 3 - 6 - 8 - 9 - 10 - 1 - 11

E = 11 N = 9 P = 3 R = 3 + 1 = 4

C = E - N + 2 = 11 - 9 + 2 = 4

C = P + 1 = 3 + 1 = 4

C = R = 4

Steps of basis path testing:

Using design or code as foundation, draw a corresponding flow graph

Determine the cyclomatic complexity of flow graph

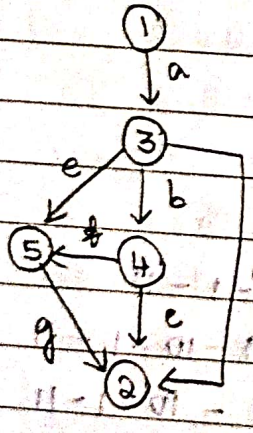
Determine a basis set of linearly independent paths.

Prepare testcases that forces execution of each path in basis set

Graph matrix:

Data structure to assist basis path testing

square matrix, size - no. of nodes in the flow graph



	1	2	3	4	5
1			a		
2					
3			d	b	e
4				e	
5				g	

Control Structure Testing

- Condition Testing
  - Branch Testing
  - Domain Testing
  - Boolean variable Testing
  - Branch & Relational operator Testing
- Data Flow Testing
  - Definition & Use
- loop Testing
  - Simple loops
  - Nested loops
  - Concatenated loops
  - Unstructured loops

Condition Testing:

tests logical conditions

A single condition is a boolean variable or a relational expression (eg: $a < b$)

Data flow Testing:

tests definitions and uses of variables

DEF(S) = {X | Statement S contains definition of X}

USE(S') = {X | Statement S' contains use of X}

DU chain of variables X = [X, S, S']

Definition usage

Else branch of if statement is not necessarily covered by definition usage (DU) testing

Loop Testing:

test the loop constructs

Skip loop entirely

Only one pass

2 passes

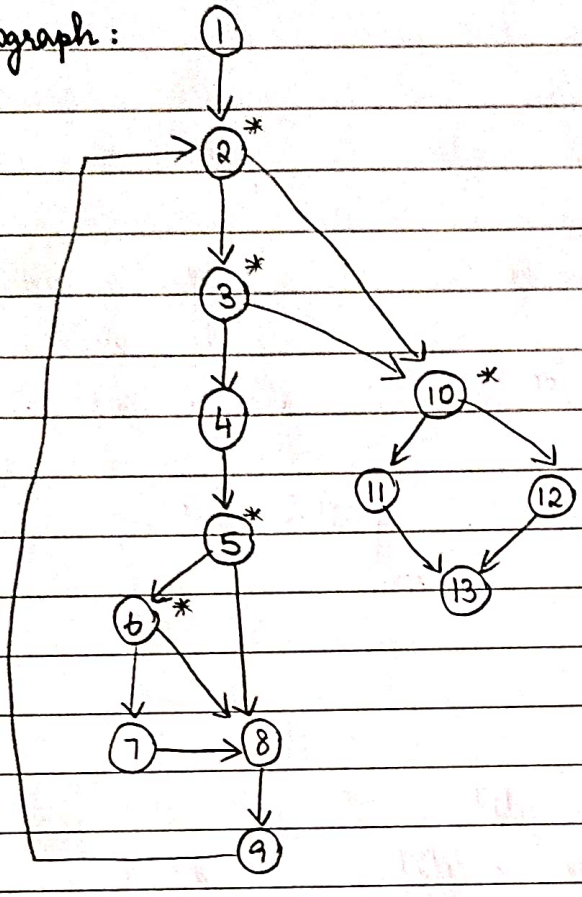
m passes where $m < n$

$n-1, n, n+1$ passes

Start with inner loops in case of nested loops

```
i = 1
total.input = total.valid = 0                    } (1)
sum = 0
DO WHILE value[i] <> -999 AND total.input < 100
    increment → (4)
    IF value[i] >= mini AND value[i] <= maxi
        THEN increment sum                       } (7)
        ELSE skip
    ENDIF                                        } (8)
    increment i
ENDDO - (9)
IF total.valid > 0                               (10)
    THEN avg = (11)
    ELSE avg = (12)
ENDIF
(13) END
```

Flowgraph :



Cyclomatic Complexity :

E = 17 N = 13 P = 5 R = 5 + 1 = 6

C = E - N + 2 = 17 - 13 + 2 = 6

C = P + 1 = 5 + 1 = 6

C = R = 6

Paths :

① 1 - 2 - 10 - 11 - 13

② 1 - 2 - 10 - 12 - 13

③ 1 - 2 - 3 - 10 - 11 - 13 total.valid = 0 so ⑩ T

~~1 - 2 - 3 - 10 - 12 - 13~~

④ 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 2 . . .

⑤ 1 - 2 - 3 - 4 - 5 - 8 - 9 - 2

⑥ 1 - 2 - 3 - 4 - 5 - 6 - 8 - 9 - 2

Testcases :

Path 1 : 1 - 2 - 10 - 11 - 13

value [1] = -999 , total.valid = 0

Path 2 : 1 - 2 - 10 - 12 - 13

value [1] = -999 , total.valid = 0

Path 3 : 1 - 2 - 3 - 10 - 11 - 13

value [1] ≠ -999 total.input = 0 , total.valid = 0

Path 4 : 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 2

value [1] ≠ -999 total.input = 0 + 1 value [i] = b/w min and max

Path 5 : 1 - 2 - 3 - 4 - 5 - 8 - 9 - 2

value [1] ≠ -999 total.input = 0 + 1 value [i] < min

Path 6 : 1 - 2 - 3 - 4 - 5 - 6 - 8 - 9 - 2

value [1] ≠ -999 total.input = 0 + 1 value [i] ≥ min but > max

Black Box Testing / Behavioural Testing :

test system against requirements from SRS

involves looking at specifications and does not require examining the code

from customer's viewpoint

Black Box Testing Techniques :

Requirement

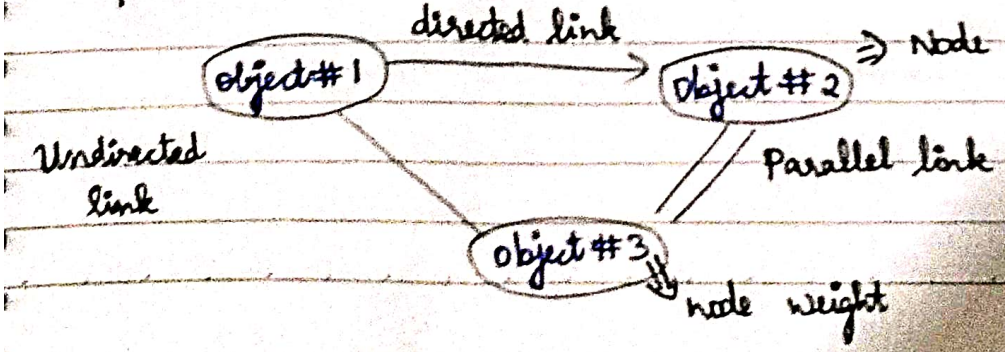
Black Box Testing:

Testing the functional requirements of the software
Encompasses end user perspectives
Handles valid and invalid inputs
Done without the knowledge of the internals of the system under test
Finds errors in
incorrect or missing fns
interface errors
errors in data structures or database access
behaviour or performance errors
initialization and termination errors

Techniques:

Graph based Testing Method
Equivalence Partioning *
Boundary Value Analysis (BVA) *
Orthogonal array Testing
Model based Testing
Requirement based Testing
Positive / Negative Testing *
Decision Tables
Compatability Testing
User documentation
Domain Testing

Graph-based:



Equivalence Partioning:

divide the input domain into classes of data

An equivalence cls has a state of valid / invalid states for input condition

It is a numeric values, range of values or Boolean cond

Guidelines for equivalence cls: EC

Input condition:

Range - 1 Valid 2 Invalid EC

Specific Value - 1 Valid 2 Invalid EC

Member of a set - 1 Valid 1 Invalid EC

Boolean - 1 Valid, 1 Invalid EC

Testcases are selected so that the largest no of attributes of equivalence cls are exercised at once

Boundary Value Analysis (BVA):

test bounding values of input domain

derives testcases from output domain also

Input condition:

Range (a, b) - (a, b, a-1, a+1, b-1, b+1)

No. of values - min, max nos, values above min, max

Apply these for output also

Data structures boundary

Orthogonal Array Testing:

Apply problems in which input domain is relatively small but too large to do exhaustive testing.

Useful in finding region faults

Testcases - dispersed uniformly throughout the testing domain

Provides good test coverage

Parameters : P_1 P_2 P_3 P_4

Values : 1 2 3

1 → Send it now

2 → Send it later

3 → send it after midnight

Orthogonal Array — No. of tests required = $3^4 = 81$

Testcases	Test Parameters			
	P_1	P_2	P_3	P_4
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

Detects all single mode, double mode and multi mode fault

Model Based Testing:

Uses UML state diagrams as basis for testcases

Useful when testing event driven applications

Helps to uncover errors in software behaviours

Requirements based Testing:

Validating the requirements gn in the SRS of the software system

Requirements are tracked by RTM - Requirements Tracability Matrix

RTM traces all requirements from design, development and testing

+ve and -ve testing:

+ve:

To prove that a gn product does what it is supposed to do

Ex: A product working as per specifications and expectations

A product delivering an error when it is expected to give an error

-ve:

To show that the product breaks with unknown

Ex: Product not delivering an error when it should and delivering an error when it should not

Decision Tables:

List various decision variables, conditions assumed by decision variables and actions to take in each combination of decisions

Variables that contribute to the decisions are listed as Columns

Columns of a decision table should all be boolean variables

Compatibility Testing:

When infrastructure parameters are changed (Processor, OS, database, servers), the product is expected to still behave correctly and produce correct results (expected)

User documentation Testing:

User doc. covers all lab manuals, user guides, setup guides, readme files, installation guides, software release notes, online help to help the end user to understand the software system

Done to check if doc. matches the product and what is there in the product explained correctly

Mutation Testing:

Type of software testing where we mutate (change) certain statements in the source code and check if the testcases are able to find the errors.

Changes in mutant programs are extremely small, so it does not affect the overall objective of the program.

Types:

Statement Mutation

Value Mutation

Decision Mutation

Ex: $x < y$ changed to $x > y$

Testing Web Applications:

Assessing Web Quality:

- Content
- Function
- Structure
- Usability
- Navigability
- Performance
- Compatibility
- Interoperability
- Security

Web App Test Strategy / Process:

- Content Testing
- Performance Testing
- Interface Testing
- Navigation
- Component
- Configuration
- Security

Content Testing:

To uncover syntactic, semantic and structural errors

Database Testing:

Web apps interact with DBMS systems and build dynamic content objects that are created in real time using data acquired from database

UI Testing:

Testing to ensure that design rules and aesthetics, visual content are to available to user without errors

Discover error related to interface

To uncover errors related to interface:

Forms
Links
Client side scripting
Dynamic HTML
Pop up windows
CGI scripts
Streaming content
Cookies

Ex: E-commerce websites

Regression Testing:

Enhancements or defect fixes work properly and doesn't affect the existing functionality.

Types:
- Regular
- Final

What?

Retesting done to ensure defect fixes work and no side effects

Why?

Defects creep in due to changes and they cause existing functionality to fail

When?

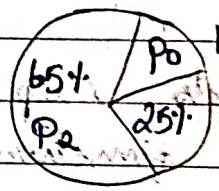
When defect fixes arrives, RT is performed in all the test phases

Steps in Regression Testing:

1) Perform initial Smoke or Sanity tests
2) Understanding criteria for selecting testcases
3) Classifying testcases into priorities
4) Methodology for selecting testcases
5) Resetting testcases for test execution
6) Concluding the results of regression cycle

Smoke Testing:
done to ensure the basic functionality of the product works.
Include testcases that have produced maximum defects in the past, basic functionality testing, etc...

Testcase classification:
Priority 0:
Sanity testcases which check basic functionality
Priority 1:
Deliver high project value to customers and development team
Priority 2:
Deliver moderate project value



~~Meet the~~
If the criticality and defect fixes are ~~low~~, very
Low - very few testcases of P0, P1, P2
Medium - All P0, P1 testcases
High - All P0, P1 and subset of P2 testcases

Resetting means setting not run or execute again in testcase database by suffering testcase result history

Results of regression

Results from regression :

Current result	Previous Result	Conclusion
Fail	Pass	Fail
Pass	Fail	Pass
Fail	Fail	Fail
Pass	Pass	Pass

Usability Testing :

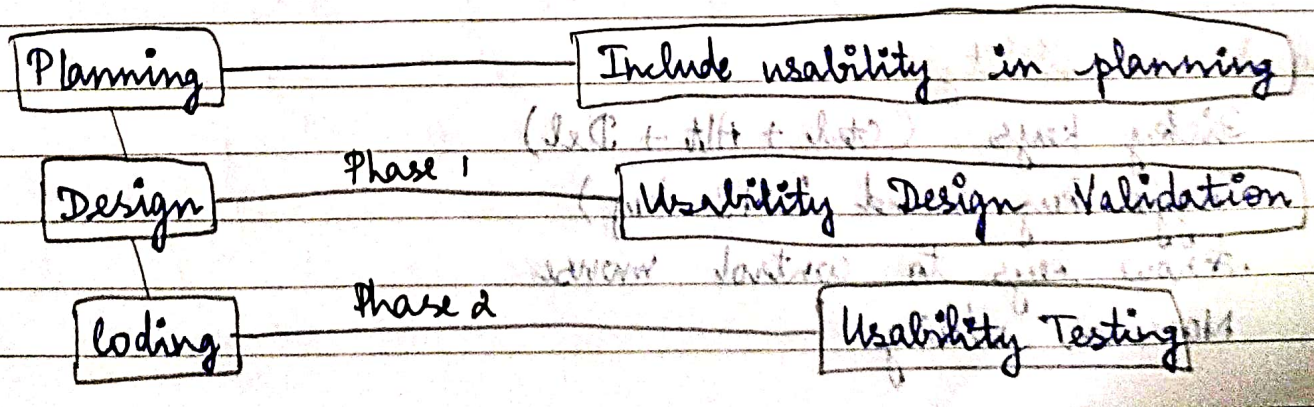
Testing done to check the product if it is easy to use

Test the "look and feel" and usage aspects of a product

Tests
- ease of use
- speed
- pleasantness
- aesthetics

from the user's point of view

Phases of Usability Testing



Results from regression :

Current result	Previous Result	Conclusion
Fail	Pass	Fail
Pass	Fail	Pass
Fail	Fail	Fail
Pass	Pass	Pass

Usability Testing :

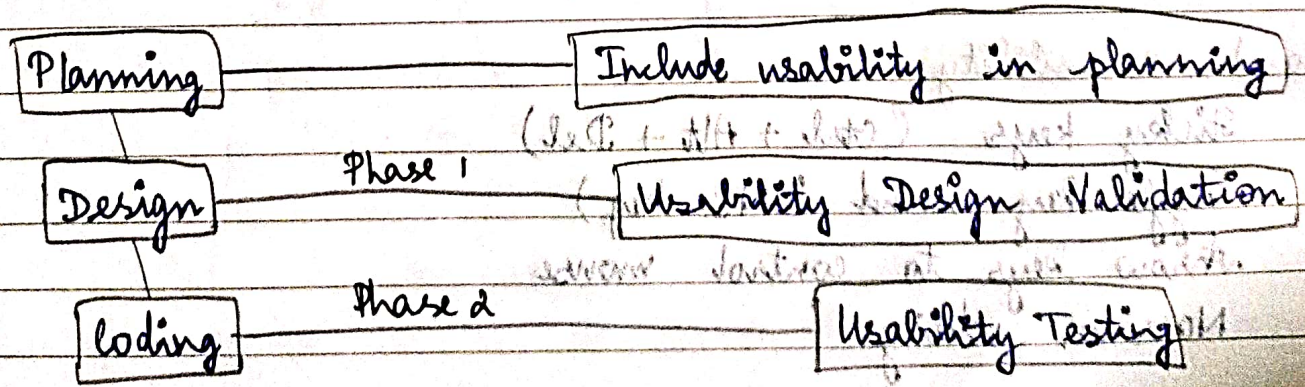
Testing done to check the product if it is easy to use

Test the "look and feel" and usage aspects of a product

Tests
- ease of use
- speed
- pleasantness
- aesthetics

from the user's point of view

Phases of Usability Testing :



Usability aspects should be reviewed by real users for maximum benefits

Users may be beginners, experts or novice

Usability design is verified through:

- Stylesheet
- Screen prototypes
- Paper design
- Layout design

Quality factors for usability

- Comprehensibility
- Consistency
- Navigation
- Responsiveness
- Aesthetics

Accessibility Testing

Verifying the product usability for physically challenged users (vision, hearing, mobility)

Keyboard accessibility:

- Sticky keys (Ctrl + Alt + Del)
- Toggle key sound (NUM key)
- Arrow keys to control mouse
- Narrator utility

Screen Accessibility:

- Visual sound
- Soft keyboard
- Captions for multimedia
- High constrast mode for vision impaired users

Product Accessibility:

- Text equivalence to be produced for audio, video
- Minimize the colour design
- Reduce speed of moving text and blinking text
- Reduce physical movement and timing pressure

Tools for usability and accessibility:

- Magnifier
- Narrator
- Soft keyboard
- HTML validator
- Stylesheet Validator