## Behavioural Modeling:

### Interaction diagram:

→ It is a part of dynamic view of the system.
→ It is represented in 2 diagrams.
1. Sequence ⇒ depend on time ordering message
   It doesn't described object but define it.
2. Collaboration ⇒ depend on structural organisation.
   It describes object.

### Purpose of interaction diagram:

→ to capture the dynamic behaviour of the system.
→ describe the message flow.
→ describe the structural organisation.
→ describe the interaction among objects.

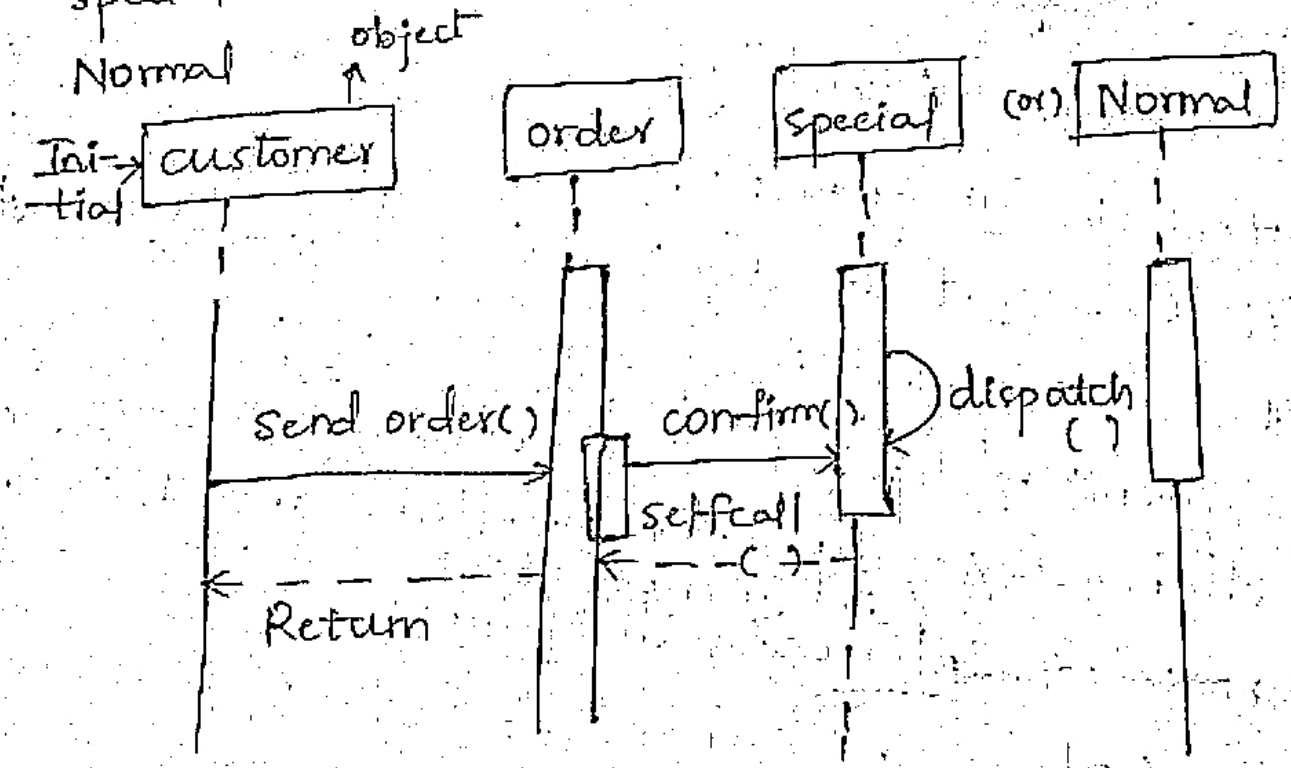
How to draw interaction diagram:

→ The object is a part of interaction.
→ message flow.
→ The sequence in which messages are flowing.
→ object organization.

Sequence diagram:

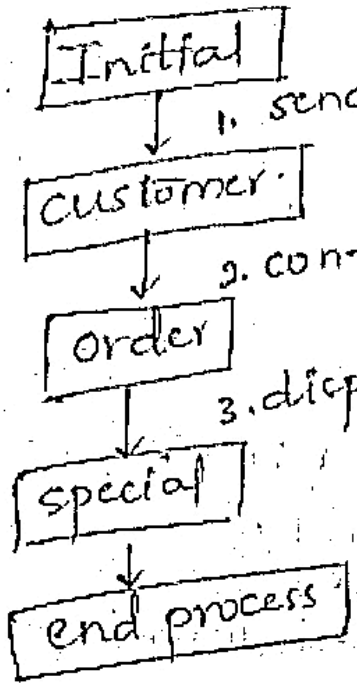
→ It describes the dynamic part of the system.
→ It depends on time order messages.
→ There are 4 objects.

customer
order
special
Normal




Collaboration diagram:

→ It describes the structural organization.
→ The sequence number how the method are called one after another.




Usage of interaction diagram:

→ The main purpose is it captures the dynamic view of the system.

→ to the model flow of control by time sequence.

→ to the model flow of control by structural organization.

→ It is used for forward and reverse Engineering.

Use case diagram:

→ It is a most important aspect of capturing the behaviour.

→ It consists of

1. Actor
2. Usecase
3. Relationship.

→ It includes 4 diagrams.

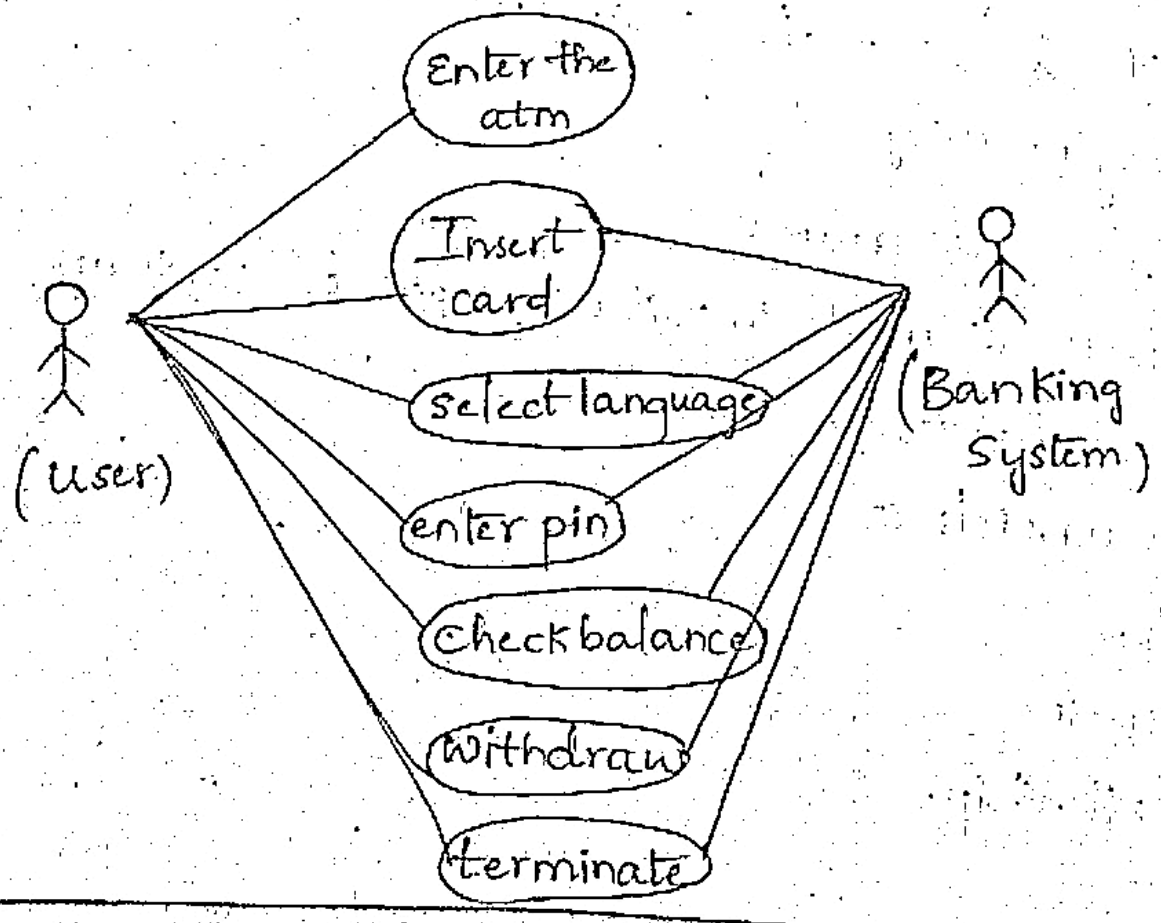
1. Activity
2. Sequence
3. Collaboration
4. statechart.

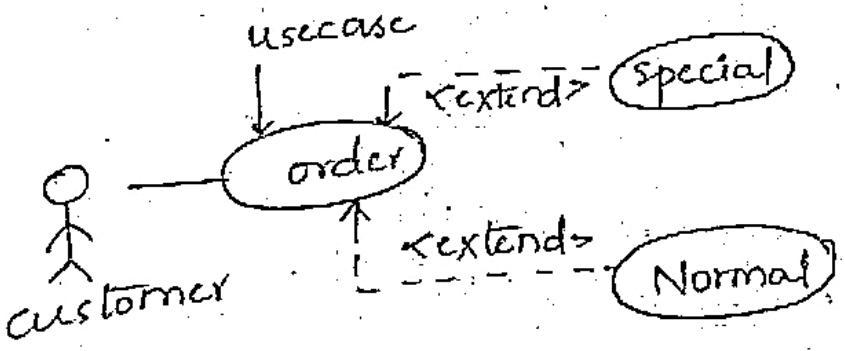
purpose:

→ It is used to gather the requirements and
→ get the outside view of the system.
→ Identify internal and external factors.

How to draw usecase diagram:

→ It is considered for high level of require-ment analysis of a system.
→ The requirements of a system are analysed and functionality captured in use cases.
→ The actor can be a human user, some internal (or) external application.






Back to

Where to use usecase diagram:

→ The main purpose of the usecase diagram is to gather the requirement.

→ The well-structured usecase diagram defines

1. precondition
2. postcondition
3. exception.

Activity diagram:

→ It is basically a flow chart.

→ It shows the flow from one activity to another activity within the system.

→ It describes operations of system.

→ This flow can be,

1. Sequential
2. Branched
3. Concurrent.

purpose:

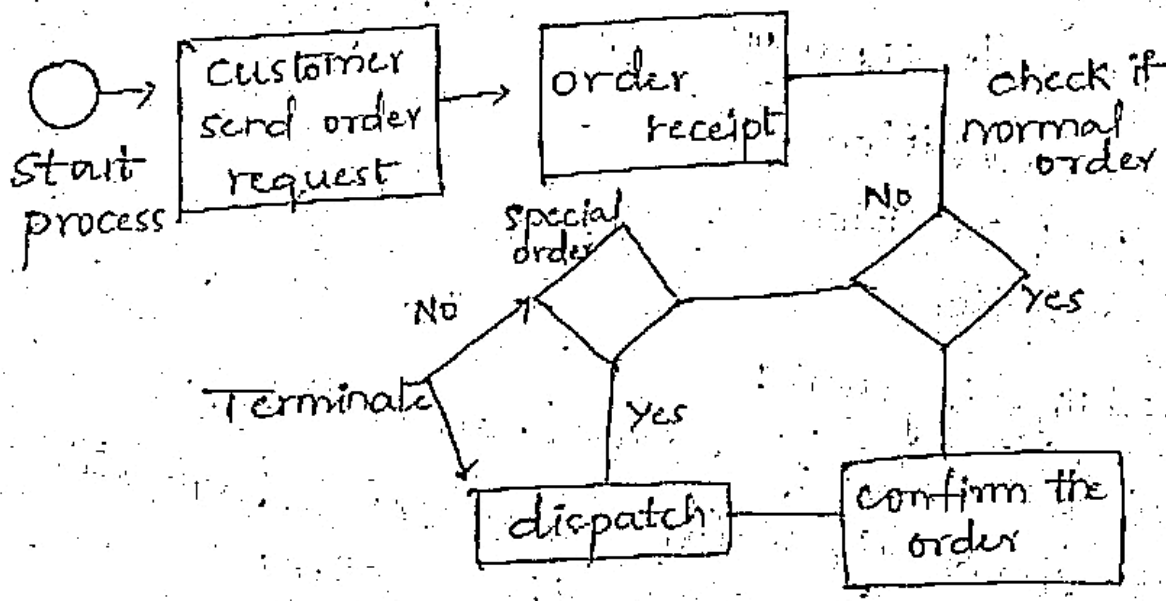
→ Activity diagram not only used for visualiz-ation of dynamic system but also used to construct the executable system.

*→It doesnot show any message flow from one to another activity.

How to draw the activity diagram:

→ Before drawing the activity diagram we should identify following elements.

1. Activity
2. Association
3. condition
4. constraint.




Usage of Activity diagram:

→To create an impact of business requirement

→It is a high level of understanding system functionality.

Statechart diagram:

→ It describes the different states of components in a system.

→ It describe state machine.

→ The machine can defined as different state

of an object and these states are controlled by internal (or) external event.

purpose:

→ It describe the flow of control from one state to another state

→ The main purpose of state chart diagram to model object lifetime from creation to termination.

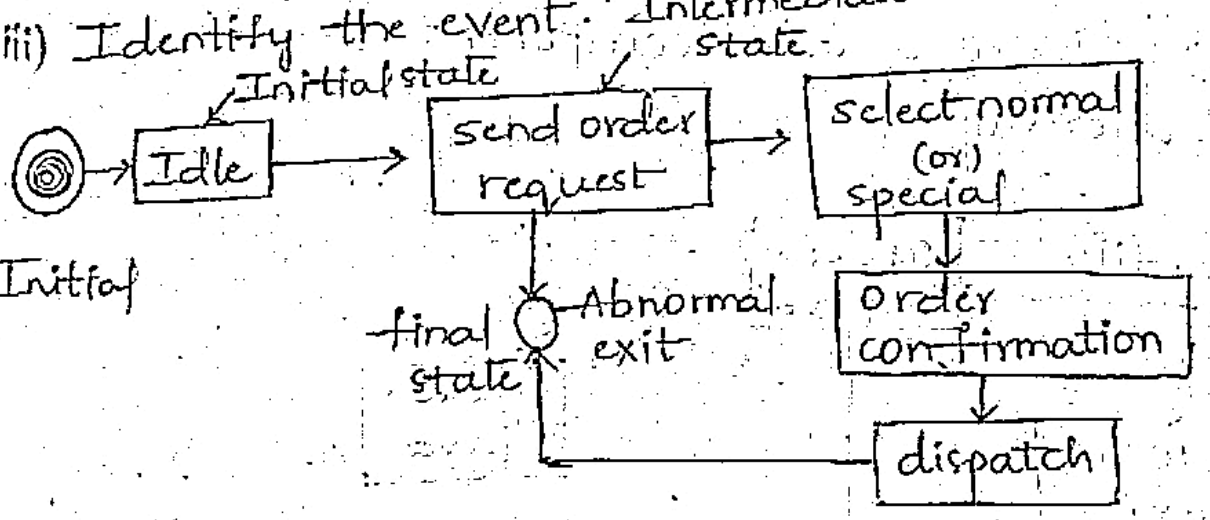
How to draw state chart diagram:

→ The statechart diagram, identifies the following points.

i) Identify the important object to be analysed.

ii) Identify the state

iii) Identify the event.




Usage:

→ It is used to clarify different states of an object.

→ Identify the event responsible for state changes.

Component diagram:

→ It is used to visualize the organization and relationship among the components in a system.

software

→ It is used to make executable system.

Purposes:

→ It is a special kind of UML diagram.

→ It is used to describe the software components.
↓
ex: libraries, packages, files etc.

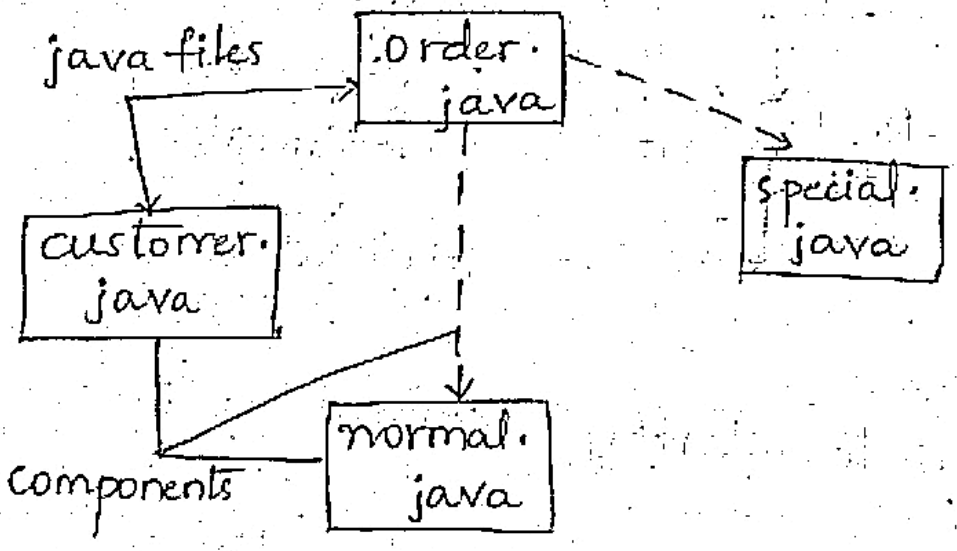
How to draw component diagram:

→ The component diagram is used to implementation phase of an application.

→ It is used to get idea of implementation.

→ The following artifacts to be identified for the following points.

1) files used in the system.
2) Libraries and other component
3) Relationship




Usage:

→ It is used to model the component of system, executable files and source code.

Deployment diagram:

→ It is used to visualize the hardware components of a system where the software components are already deployed.

→ It describe the static view of a system.

→ It consists of node and relationship.

purpose:

→ It is the use to visualize the hardware topology.

How to draw deployment diagram:

→ Identify the following parameters

1) Performance
2) Scalability
3) Maintainability
4) Portability.

# Unit-4 : Testing

## Testing:

⇒ A strategic Approach for software Testing

## Program Testing :

→ It consists of set of inputs (or) test cases and observing if the program behaves as expected.

→ If the program is failed to behave as expected then, the conditions under the failures occured in the project.

→ Some common terms used in testing are,

1. Failure → finding errors occured in the project.
2. Test case → ISO - Input Statement Output
3. Test suite. →

→ The software testing includes the following points.

1. Before testing starts to identify the requirements of the project.
2. It specifies the objectives of testing.
3. Identify the user category and develop a profile for each user.
4. Developing the test plan and focus on rapid cycle testing.
   ↓
   improve the quality
5. The Robust software is developed and designed.
6. Conduct formal review.
7. Conduct formal review to evaluate quality

(or) ability of test strategy and test cases.
8. For testing process, developing an approach for continuous development.

Testing strategy for conventional software:
→ There are many strategies which can be used to test software. One approach is you can wait until the system is fully constructed and conduct testing.
2) Second approach is conduct the test on daily basis. It is a part of system which is constructed and given for testing.

Types of testing:
1. Unit testing
2. Integration testing
3. Validation and
4. System testing.

Unit testing:
→ Individual units (or) components of software are tested.
→ A unit may be individual function, method, procedure (or) module.
→ Errors can be known in the early stage
→ It helps to save → It saves time and investment.
→ The reason to perform unit testing;
1. It helps to fix the bugs early in development cycle.

2. It helps the developer to understand the code and make the changes quickly.

Integration testing:

→ In this testing, the individual module (or) component are combined in the single group and the testing is conducted.

→ The goal of integration testing is to check correctness of communication among the modules.

*Types of integration:

1. Incremental testing
2. Non-Incremental testing
3. Hybrid testing.

1) Incremental testing:

→ The modules are added in ascending order one by one according to the need.

→ The selected module must be logically related.

There are 2 approaches/methods used in Incremental testing they are:

1. Top down approach
2. Bottom up approach.

Top down approach: temporary storage location: stub

→ In this process, the higher level modules are tested with low level modules until the successful completion of testing of all

modules.

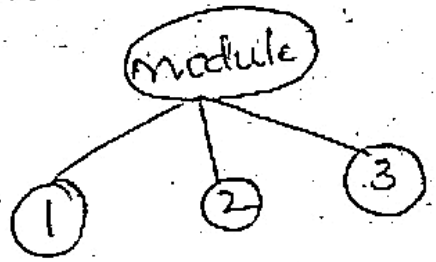
Bottom up approach:

temporary storage location : driver

→ It is the process of lower level modules tested with higher level modules until the successful completion of testing of all modules.

2) Non-Incremental testing:

→ In this method, the data flow is very complex and difficult to find who is the parent node and who is the child node.




3) Hybrid testing:

→ In this approach, both top down and bottom up approaches are combined for testing.

→ In this process the top level module are tested with lower level module and, lower level module are tested with high level module.

Validation testing:

→ In this type of testing there are 2 approaches. They are:

1. Verification
2. Validation.

1) Verification testing:
→ It include different activities like business requirement, system requirement, design review and source code while developing a product.
→ how to reach path/destination.

2) Validation testing:
→ This testing perform functional and non-functional testing.
→ Functional testing include Unit testing, integrating testing and system testing.
→ Non-functional testing include User Acceptance Testing (UAT).

System testing test for
→ It performs the entire system.
→ It is fully integrated software system.
→ A computer system consists of group of software to perform various tasks.
→ It is also called as end to end testing.
→ It includes the following steps:
1) The verification of input function of the application to test whether it is producing expected output (or) not
2) The testing of integrated software
3) It is testing the whole system.
→ It is a type of Black-box testing.

## Difference between verification and validation:

| Verification | Validation |
|---|---|
| → It focus on procedure to acheive the result | → It focus on result itself. |
| → It is a static activity | → It is a dynamic activity |
| → The activities like requirement review, architecture review, design and code review. | → The activities like unit testing, integration testing, system testing and acceptance test. |
| → It comes before validation. | → It comes after verification once the executable software is available. |
| → The Quality assurance team conducts verification | → The software testing team conducts validation |

## Software testing:

→ There are 2 major categories of software testing. They are:

1. Black box testing.
2. White box testing.

## * Black box testing:

→ It is a software testing method in which the functionalities of software application

are tested without having the knowledge of internal code structure, implementation details and internal path.
→ It mainly focus on input and output of software application.

Need of Black-box testing:
→ It reduces the test cases.

How to do Black-box testing?
→ Initially, they check requirement and specification.
→ The tester choose valid input to check whether SUT (System Under Test) passes correctly (or) not.
→ To determine the expected output for all inputs.
→ The software tester construct the test cases with selected input.
→ The test cases are executed.
→ If any defects are found, they are fixed and retested.

Types of Black-box testing:
1. functional
2. Non-functional
3. Regression

Functional:
→ All the requirements of the system comes under functional.

Non-functional:
→ The characteristis / properties of the requirements are non-functional.

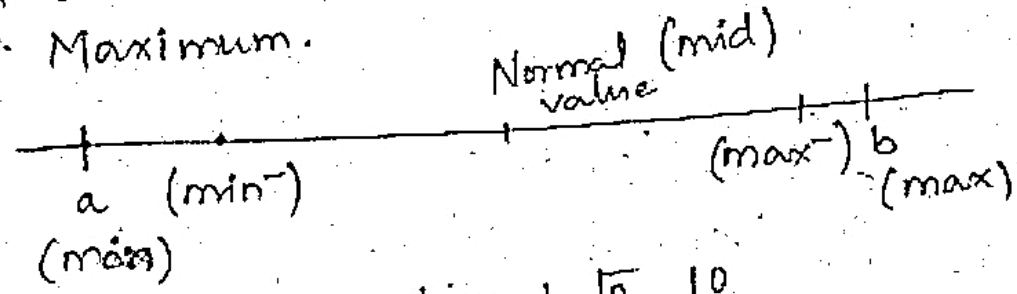
Regression:
→ Any updations in functional (or) non-functional is Regression..

Techniques of Black-box Testing:
→ Equivalence class partitioning
→ Boundary value analysis.
→ Decision table.

Equivalence class partitioning:
→ It is used to minimise the no. of possible test cases to an optimum level while maintaining reasonable test coverage.
→ The input value provided to system is divided in to different classes / groups based on similarity.

Boundary value analysis:
→ It is a process of testing between extreme end (or) boundaries between the partition of input value.
→ The basic ideas of bourdary values are,
1. Minimum
2. just above minimum

3. Normal value
4. just below maximum
5. Maximum.




ex: The input value 1 to 10

| I/p value | expected o/p |
|---|---|
| Boundary value = 0 | The system doesn't accept/ not accepted |
| B.V = 1 | The system accepted |
| B.V = 11 | not accepted |
| A | not accepted |
| -1 | not accepted |

→ We can't test all the possible values because if it is done, the no. of test cases will be more than 100.
→ To address this problem we use Equivalence class partition and Boundary value analysis.

1 to 10 ⇒ input value

| Invalid | valid | Invalid |
|---|---|---|
| → 0 | 1 10 | 11 99 |
| ↓ | ↓ | ↓ |
| partition - 1 | partition - 2 | partition - 3 |

→ The hypothesis behind this testing is, if condition is true then all the conditions are also true.
→ Otherwise, if one condition fails, all other conditions also fail.

Decision table:

→ It is a tabular representation of input (or) test condition.
→ It is used to check the logical relationship between 2 (or) more input.

for ex: Gmail account

Email.ID [ ]

password [ ]

| T/F | | | |
|---|---|---|---|
| username | T | F | T |
| password | F | T | T |
| Error | (E) | (E) | (H) → Homepage |

White box testing:

→ It is a testing technique to execute the program structure and test data from the program logic (or) code.
→ It is also used to test internal structure and implementation detail
→ The other name of White box testing is Glass box testing / open box testing / structure testing.

→ White box testing verify the internal Security code and flow of specific input, loop, decisions and conditions and expected output.

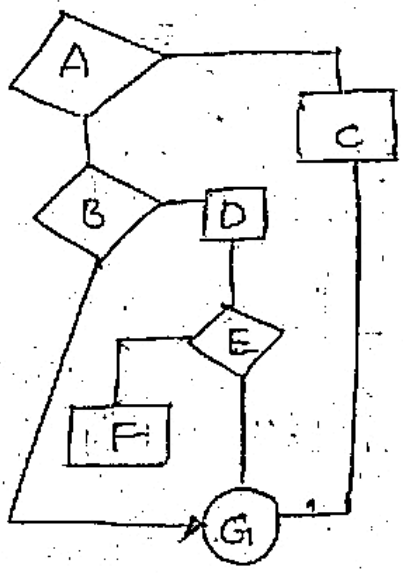
Techniques:

→ Statement coverage
→ branch coverage
→ path coverage

Statement Coverage:

→ In this technique, to execute all the statements atleast once and each line of code is tested.

ex:




Branch coverage:

→ In this technique, the test cases are designed so that each branch from all the decision points are tested atleast once.

ex:
```
If x Then
    If y Then
        A
        B
    Else
        If z then
            C
    else
        D
```

Explanation:

→ A, C, D are condition branches because they occurs only if the condition is satisfied.

→ B is an unconditional branch because it always execute after A.

→ In branch coverage it is used to identify all conditional and unconditional branches.

Path coverage:

→ In this technique, the control flow graph made from code (or) flowchart and cyclomatic complexity is calculated which define the no. of independent paths.

Flow graph notation:

→ It consists of nodes and edges.

→ Each node represents sequence of statements and predicate node represents a decision point that contains a condition after graph splits.

Cyclomatic complexity:

→ It is measure of logical complexity of software and used to define no. of independent paths and no. of decisions in given code.

→ For graph G, V(G) is a cyclomatic complexity.

Calculating V(G):

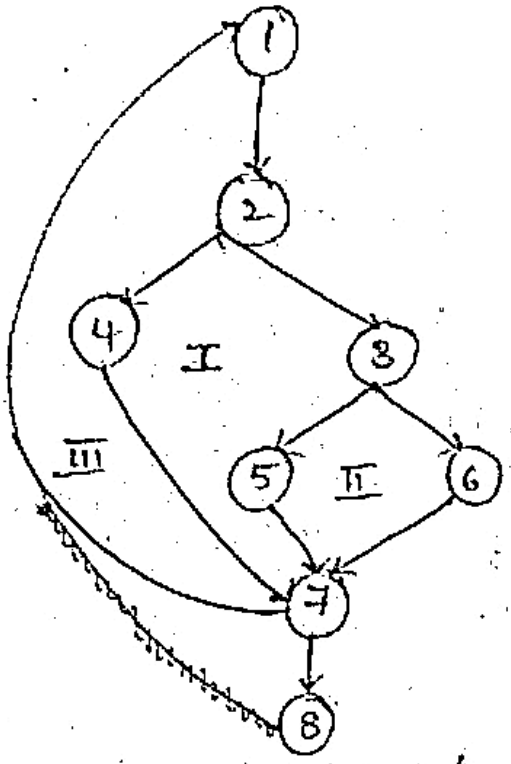
→ V(G) = P + 1

P = no. of predicate nodes.

→ V(G) = E − N + 2

E = no. of edges

N = no. of nodes.




P = no. of predicate nodes = {2, 3, 7} = 3

E = no. of edges = 10

N = no. of nodes = 8

V(G) = P + 1 = 3 + 1 = 4

V(G) = E − N + 2 = 10 − 8 + 2 = 2 + 2 = 4

#Path 1 (P1) = 1 → 2 → 3 → 6 → 7 → 8

#path 2 (P2) = 1 → 2 → 4 → 7 → 8

#path 3 (P3) = 1 → 2 → 3 → 5 → 7 → 8

#path 4 (P4) = 1 → 2 → 4 → 7 → 1 → - - -

Loop testing:

→ Loop is widely used for many algorithm.

→ The error occurs at the beginning and end of the loop.

*Simple loop
*) Nested loop

## The art of debugging:

→ debugging is the procers of fixing the bug in the softwaare.
→ It is used to refer to identifying, analysing and removing the error.

## Differences between debugging and testing:

| Testing | Debugging |
|---|---|
| → The testing focus on finding the error. | → Debugging focus on where the error is and how to solve it. |
| → It is used to find the minimum success rate. | → Debugging is used to identify the error. |
| → Testing levels are unit testing, integration testing, validation testing, System testing. | → It require knowledge and skill. |

## Debugging process:

→ The process involves 2 stages.
1. Problem identification
2. Report preparation.

## Debugging approaches:

→ There are 3 approaches
1. Brute force method
2. Back tracking
3. Cause elimination method.

1) Brute force method:

→ In this method, the program is loaded with print statement to print intermediate value

→ Based on the printed value, it helps to identify the statements having error.

2) Backtracking:

→ In this approach, beginning from the statement which error symptoms has been observed and source code dragged from backward to forward process.

→ It is applicable for small programs.

3) Cause elimination method:

→ The list of causes which could possibly contribute to error symptom are developed and tested to eliminate the error.

Debugging tools:

→ Lot of public domain like GNU gdb, dbx are available for debugging.

→ some automative tools are Tracers, profilers and interpreters.

Part-2

Process and product Metrics:

↓

Software quality:

→ software quality product is defined in terms of fitness.

→ The quality product focus on the user

requirement according to SRS document.
→ Quality is associated with software product.
→ Several quality methods like 1) portability, 2) usability and 3) reusability 4) correctness and 5) Maintainability.

Product Metrics:
→ In software development process, a working product is developed at the end of each successful phase
→ Each product can be measured at any stage of development.
→ The product metrics assess the internal product attribute.
1) Analysis, design and code model.
2) Test cases.
3) Overall quality of the software under development
→ The product metrics process are listed below
1) Metrics for analysis model
2) Metrics for design model
3) Metrics for source code
4) Metrics for testing
5) Metrics for maintenance

1) Metrics for analysis model:
→ In this analysis model, we test (or) predict the size of final system.
→ The size indicates coding, integration and testing

→ There are 2 common methods involved in software design.

1. Line of code (LOC)
2. Function Point (FP)

Line of Code:

→ It calculates the size of computer program and also used to calculate no. of lines in a project.

→ It doesn't include blank spaces and comment lines.

→ It is 'Language dependent.

ex:

| project | LOC | cost $ | effort of process | Doc. Time | error |
|---|---|---|---|---|---|
| A | 10,000 | 118 | 18% | 365 | 39 |
| B | 12,000 | 125 | 20% | 370 | 42 |

Function point:

→ It focus on the size of the program created and count the no. of functionalities used in software applications.

→ It includes external input/output, internal logical file and no. of external interfaces.

→ It is requirement dependent.

## Metrics for specification quality :

→ to evaluate the quality of analysis model and requirement specification, the set of characteristic has been proposed.

→ This characteristic include,

1. Specificity
2. Completeness
3. Correctness
4. Verifiability
5. Internal & external consistency
6. Reusability

ex: $n_r$ is a set of requirements. It can be calculated by following equation,

$$n_r = n_f + n_{nf}$$

$n_r$ = no. of requirements

$n_f$ = no. of functionality requirements

$n_{nf}$ = no. of non-functional requirements.

→ This metric is based on consistency of reviewers understanding of each requirement has been proposed.

→ It is represented by, $Q_1 = \frac{n_{ui}}{n_r}$

$Q_1$ - specification.

$n_{ui}$ = no. of requirements understood by reviewers

$n_r$ = no. of requirements

→ The complete of the functional requirements can be calculated by, $Q_2 = \frac{n_u}{(n_i \times n_s)}$

$n_u$ – no. of unique functions

$n_i$ – no. of inputs/out

$n_s$ – no. of states

→ The above eq$^n$ consider only the functional requirements and not include non-functional requirements.

→ In order to consider non-functional requirements use the following eq$^n$,

$$\alpha_3 = \frac{n_r}{n_c + n_{nv}}$$

$n_c$ = valid requirement

$n_{nv}$ = non-valid requirement

Metrics for software design:

→ The success of the software project depends on quality and effectiveness of software design.

Architecture design metrics:

→ This metrics focus on program architecture and effectiveness of components within the architecture.

→ There are 3 types of methods applied for design metrics.

1. Structural complexity:

→ The structural complexity is calculated by the eq$^n$: $S(j) = f^2 out(j)$

$f out(j)$ → fan-out

→ fan-out means no. of modules subordinating the module j.

2. Data complexity:

→ It indicate the complexity in internal interface for module j.

→ It is defined as, $D(j) = \frac{V(j)}{fout(j)+1}$

$V(j)$ = no. of inputs & outputs passed to the module

3. System complexity:

→ It is the sum of structural complexity and data complexity.

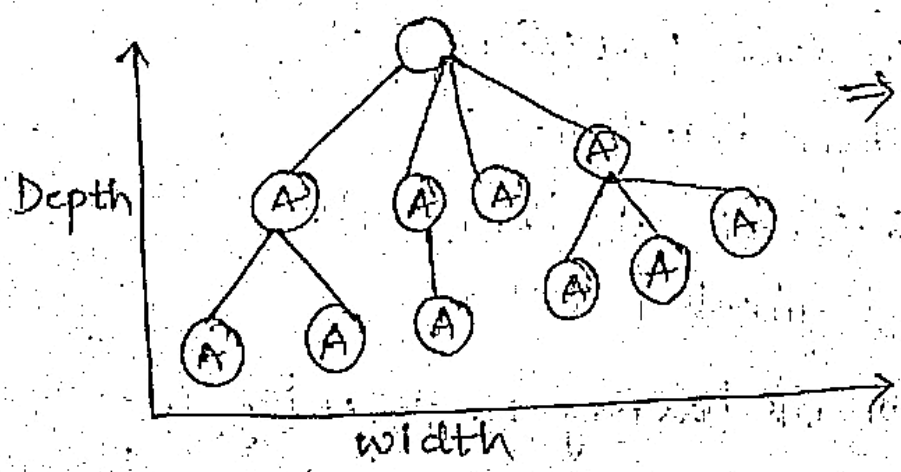
→ It is calculated by $C(j) = S(j) + D(j)$

→ A metric can be developed by referring call and return architecture.

→ It is calculated by, $size = n + a$

Here, n = no. of nodes

a = no. of arcs.




$\Rightarrow$ n = 11
a = 10
size = n + a
= 11 + 10
= 21

→ depth is defined as the longest path from top (root) node to leaf node.

→ Width is defined as the maximum no. of nodes at any level

→ This architecture is indicated by arc to ratio.

→ This ratio is calculated by, $r = \frac{a}{n}$

a = no. of arcs
n = no. of nodes.

→ Quality of architecture design plays an important role to determine the overall quality of the software.

→ One of the major characteristic is DSQI.

DSQI :- Design Structure Quality Index

→ to calculate DSQI the no. of steps involved are,

1. to calculate DSQI the following values must be determined

i) The no. of components in program architecture (S1)

ii) The no. of components whose correct function is determined by source input. (S2).

iii) The no. of components whose correct function depends on previous processing (S3)

iv) no. of database items (S4)

v) no. of different db items (S5).

vi) no. of db segments (S6).

vii) no. of components having single entry and exit. (S7)

2. Once all the values from S1 to S7 are known and intermediate values are calculated.

i) program structure (D1)

ii) In this design where D1 = 1, orelse D1 = 0

ii) module independence (D2)

$$D_2 = 1 - (S_2/S_1)$$

iii) if module is not dependent on previous processing ($D_3$)

$$D_3 = 1 - (S_3/S_1)$$

iv) Database size ($D_4$)

$$D_4 = 1 - (S_5/S_4)$$

v) Database compartmentation ($D_5$)

$$D_5 = 1 - (S_6/S_4)$$

vi) Module entry and exit ($D_6$)

$$D_6 = 1 - (S_7/S_1)$$

3. Once all the intermediate values are calculated. DSQI is calculated by the formula,

$$DSQI = \sum w_i D_i$$

Where, i = 1 to 6

w = the weightage of the importance of intermediate value

→ The various metrics are developed for component level design are listed below.

i) cohesion metrics:

→ It follows 5 concepts:

a) Data slice

b) Data token
c) Glue token
d) Super Glue
e) Stickiness

Dataslice:
→ It is completely dependent on code
→ It is a process of checking the values holded by module.
→ It is backward to forward process.

Data token:
→ They were It is a set of variables.

Glue token:
→ It is a set of data tokens.

Super Glue:
→ It is related to Glue token.

stickiness:
→ It defines 4 views like.

a) population:
b) population value.
c) length
d) function.

ii) Coupling metrics:
→ It indicates the control flow, global coupling and environmental coupling.

$$M = d_i(a * c_i) + d_o(b * c_o) + g_d + (c * g_c) + \underset{w+r}{}$$

Here, $d_i$ = total no. of input parameters.
$c_i$ = total no. of input control parameters.
$d_o$ = total no. of output parameters.
$c_o$ = total no. of output control parameters.
$g_d$ = no. of global variable utilized as data.
$g_c$ = no. of global variable utilized as control.

$\boxed{\$ 1 \S}$ → Global (or) environmental flow

→ By using the above formula, module coupling indicator (MC) is calculated by,

$$\boxed{MC = \frac{K}{M}}$$

K = proportionality constant
M = module.

Complexity metrics:

→ different types of software metrics can be calculated and many metrics have been proposed for VI design.
→ One of the most important metric is LA.
LA - Layout Appropriateness.

Metrics for object-oriented design:

→ Inorder to develop the metrics for object-oriented design the following metrics are considered and listed below.
→ complexity
→ coupling
→ Sufficiency

→ cohesion
→ Completeness
→ primitiveness
→ Similarity
→ Volatility
→ Size

Metrics for coding:

→ 'Halstead' proposed the first analytic law for computer science by using the set of primitive measure which can be derived once the design phase is complete and code is generated.

→ This measurement are listed below

1. program length: $N = n_1 \log_2 n_1 + n_2 \log_2 n_2$

$n_1$ = no. of different operators
$n_2$ = no. of different operands.
$N_1$ = total no. of operators
$N_2$ = total no. of operands.

2. program volume: $V = N \log_2 (n_1 + n_2)$

→ volume depends on program language.
→ It is required to specify a program.
→ The volume ratio (L) is calculated by,

$$L = \frac{\text{volume of most compact form of program}}{\text{volume of actual program}}$$

3. Program difficulty level (aD) and Effort(E):

$$D = \left(\frac{n_1}{2}\right) * \left(\frac{N_2}{n_2}\right) \quad ; \quad E = D * V$$

## Metrics for software testing:

→ The testing focus on testing process and the technical characteristics of test like error discovered and, no. of test cases needed and testing effort and so on.

→ By using the program volume(V) and program level (PL) of Halstead effort(e) is calculated by $\boxed{e = \frac{V}{PL}}$

→ The set of metrics proposed for testing are listed below:

1. (LCOM) Lack of cohesion Method
2. Percent public (or) protected (PAP)
3. Public access to Data members (PAD)
4. Number of Root classes (NOR)

## Metrics for maintenance:

→ IEEE proposed Software Maturity Index(SMI) which provide the stability of software product

→ SMI is calculated by following parameters.

1. no. of module in current Release (MT)
2. no. of module have been added in current Release (Fa)
3. no. of module have been deleted in current Release (Fd).
4. no. of module have been changed in current Release (Fc).

→ SMI is calculated by;

$$SMI = \frac{[MT - (F_a + F_d + F_c)]}{MT}$$

Metrics for process and product:

*Software measurement:

→ It indicates the ① size, ② quantity and amount (or) ③ dimensions of a particular attribute of product (or) process.

*Classification of software measurement:

→ There are 2 types of measurement

1. Direct measurement
2. Indirect measurement

Characteristics of software metrics:

1. Quantitative
2. Understandable
3. Repeatable
4. Economical.

Metrics for software quality:

→ This metrics is a subset of software metrics and focus on the quality aspect of product, process and project.

→ It is divided in to 3 categories:

1. Product Quality metrics
2. In process Quality metrics
3. Maintenance quality metrics.

## Product Quality metrics:

→ This metrics includes the following:

1. Mean Time to failure
2. defect density
3. Customer problem
4. Customer satisfaction.

## In-process Quality metrics:

→ Tracking of defect arrival during machine testing.

→ This metrics include:

1. Defect density during machine testing
2. Defect arrival pattern
3. phase-based defect removal
4. Defect removal effectiveness.

$$DRE = \frac{\text{defect removal deployment phase}}{\text{defect latent phase}} \times 100\%$$

## Maintenance Quality metrics:

→ It cannot be done to alter the product quality during this phase

→ It includes:

1. Fix backlog and backlog management Index.
2. Fix Response time and fix Responsiveness.
3. Percent delinquent fixes
4. Fix Quality:

error

# Unit – 5

## Risk Management

→ Risk is a problem, it might happen (or) might not.

→ There are 2 major characteristics

1. Uncertainity.
2. Loss

### Reactive Vs Proactive risk

→ Reactive strategy moniters the projects and deal with the resource, it becomes actual problem.

→ Proactive strategy begins long before the technical workers initiated and identify the potential risk and a probability and assessment ranked by importers.

### Software risk:

→ It involves 2 characteristics

1. Uncertainity
2. Loss

→ Systematic attempt to specify the threads to the project plan

→ There are 2 different types

i) Generic risk

ii) Product specific risk

### Generic risk:

→ It is a thread to every software project.

### Product specific risk:

→ It can be identified by clear understanding technology & user environment.

→ Risk projection is also called as estimation.
→ Eash risk in 2 ways
1. Livelihood
2. Probability
→ Based on the risk projection in developing risk table.
→ It include :
1. List out all the risk
2. The risk is assessed
3. There are 4 components of risk like performance, support, cost, Schedule
4. High probability of risk are noted in top of table and low probability risk noted in bottom of the table.

| Risk | probability | Impact | RMMM |
|---|---|---|---|
| | | | |

← format

Risk refinement :
→ It represent the risk in CTC (condition transmission & consequence) -format sub condition1, sub condition 2, sub condition 3.

Quality Management :

Quality :
→ It is a systematic approach to ensure the software to meet the highest standard in terms of functionality / reliability and user satisfaction.

## Quality concepts:

→ The Quality management approach.
→ Effective software technology.
→ Format Review
→ Testing strategy.
→ software documentation.
→ Measurement and reporting mechanism.

## Quality control:

→ We will do some Assessment and we can know did we acheive goals (or) not.

## Cost of quality:

→ The quality cost is divided into
1. Prevention cost
2. Apprasal
3. Failure.

## Software Quality Assurance:

→ To know the quality of software and explicit stated functional requirement.
→ Explicited documented and implicit characteristic developed by software.
→ There are 3 important points,
i) foundation of requirement quality, is measured
ii) Specified standard development
iii) The set of implicit requirement is unmentioned.

SQL Activity :

→ Prepare for SQL plan.
→ Participate in development of project
→ Review software engineering.
→ Verify software
→ To ensure software work.

Software Review :

→ The review applied at various points in the software development to identify the error and defect can be removed.
→ There are 3 types of software review.

1. Informal Review
2. Formal review
3. Formal Technical review

Formal Technical review : (FTR).

→ It is a software quality assurance performed by software engineer.
→ The objectives of FTR are :

1. Review meeting
2. Review reporting
3. Review Guidelines

Software reliability :

→ It is defined as probability of failure-free operation.

1. Measurement of reliability :

It is calculated by, MTBF = MTTF + MTTR

MTBF = Mean Time Between Failure

MTTF - Mean Time To Failure

MTTR: - Mean Time To Repair.

Software safety:

→ It focus on identification and assessment of error and defect to the entire system.