Unit - IV

Testing & Maintenance

Software testing fundamentals - internal & external views of testing - white box testing - basis path testing - control structure testing - black box testing - Regression testing - Unit testing - Integrating testing - validating testing - System testing and debugging - Software implementation techniques: coding practices - Refactoring. Maintenance & Reengineering - BPR Model - Reengineering process Model. Reverse & Forward Engineering.

Testing & Implementation

Introduction to testing:

Testing is the process of finding the errors in the software before delivering to the end user.

The main objective of testing is to uncover errors with a minimum of effort and time.

Characteristics of a Good test:

A good test has a high probability of finding errors.

The good test is not redundant

A good test is neither too simple nor too complex.

A good test should be "best of breed".

Testability

Software testability is simply how easily a computer program can be tested.

Characteristics of testable Software:

Operable - The better it work, the easier it is to test.

Observable - Incorrect output is easily identified; internal errors are automatically detected.

Controllable - The states and variables of the software can be controlled directly by the tester.

Decomposable - The SW is built from independent modules that can be tested independently

Simple - The less there is to test, the more

- The program should
  - Functional Simplicity
  - Structural Simplicity
  - Code Simplicity

Stable:
- If the changes are less, then the disruptions to testing are also less.
- changes to the Software during testing are infrequent and do not invalidate existing tests.

Understandable:-
- The more information exists, the smarter to test.
- The architectural design is well understood, documentation is available and organized.

Internal & External views of testing:-

Black-box testing (External view of testing)
- Knowing the specified function that has to be performed, if we test to check whether the function is fully operational and error free then it is termed as Black-box testing.
- Black box testing will not test the internal logical structure of the software.

White box testing checks whether the internal operations perform according to the specification.

This is also referred as [glass box testing]

This involves test that concentrate on close examination of procedural details. Logical paths are also tested.

This uses the control structure of component design to derive the test cases.

The test cases:

- Guarantee that [all independent paths] with in a module have been exercised at least once.
- Exercise all logical decisions on their true and false sides.
- Execute all loops at their boundaries and with in their operational bounds.
- Exercise internal data structures to ensure their validity.

White - Box testing:

Basis path testing:

- One of the white-box testing techniques
- This helps to derive a logical complexity measure of a procedural design
- Test cases derived to exercise the

Flow graph

- It is a pictorial representation of control flow.
- It is also referred as a **Program graph**
- A circle in a graph represents a **node** which stands for procedural statements.
- A node containing the condition is referred to as a **predicate node**
  - ⇒ Each **compound condition** is a conditional expression containing one or more Boolean operators represented by a **predicate node**
  - ⇒ A predicate node has **two edges** leading out from it (True or false)

SEG: → seq. in which the different instructions of a program get executed.

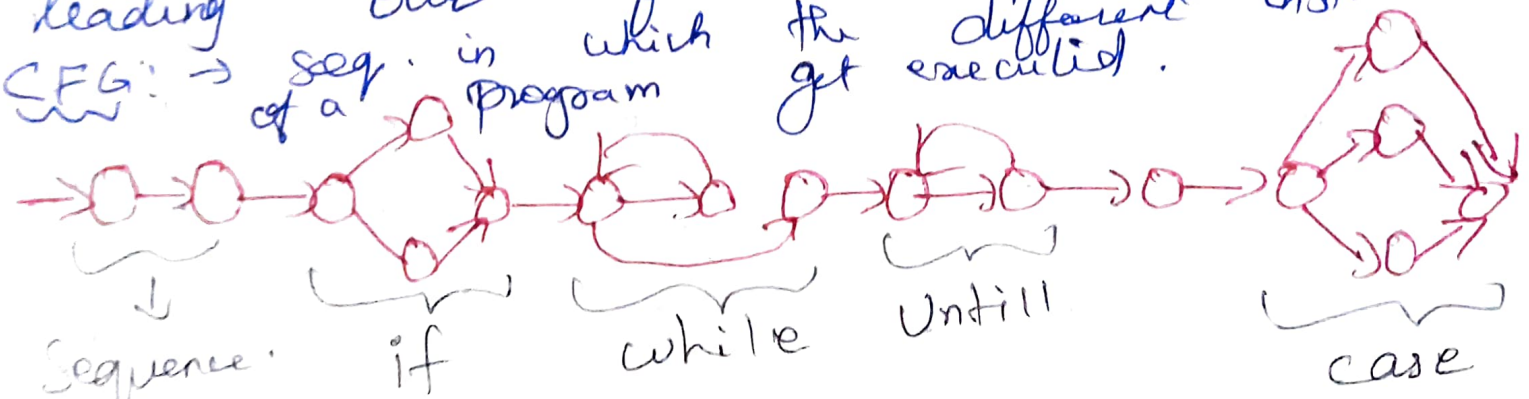
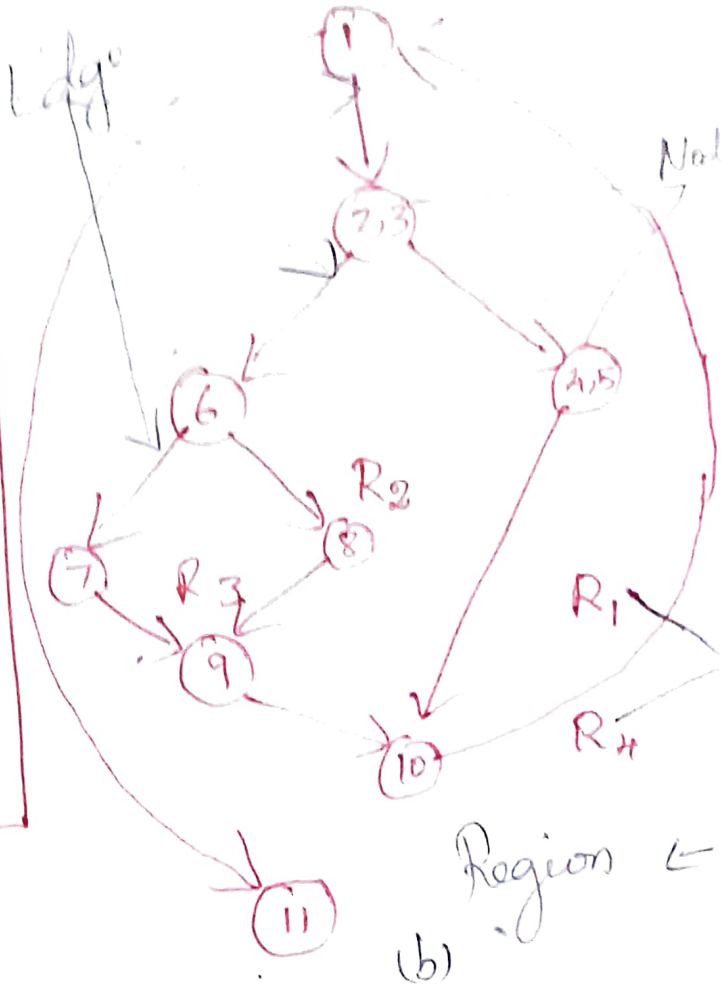


fig: Flow graf notation

- An **edge** or a link is a arc arrow representing the control flow in a specific direction.
- Areas bounded by a set of edges and nodes are called **Regions.**
- When counting regions, include the area outside the graph as a region, too

(a)
Flow chart



(b)
Flow graph.

Region ←

Independent Paths

- Any path from the start node to en node that introduce atleast one new set of processing statements or a new condition.
- An independent path must move along at least one edge that has not been traversed before by a previous path.
- Basis set of above flow graph:

Path 1 : 0 - 1 - 11
Path 2 : 0 - 1 - 2 - 3 - 4 - 5 - 10 - 1 - 11
Path 3 : 0 - 1 - 2 - 3 - 6 - 8 - 9 - 10 - 1 - 11
Path 4 : 0 - 1 - 2 - 3 - 6 - 7 - 9 - 10 - 1 - 11

The no. of. paths is the basis set is determined by the

- Provides a quantitative measure of the logical complexity of a program.
- It defines no. of independent paths in the basis set.
- It provides upper bound for the no. of test must be conducted to ensure all statements have been executed atleast once.
- This can be computed three ways.
  - The no. of regions
  - $V(G) = E - N + 2$, $E \rightarrow$ no. of edges, N – no. of nodes, G – graph
  - $V(G) = P + 1$, $P \rightarrow$ no. of predicate nodes

Example: cyclomatic complexity for the flow graph.

No. of regions = 4

$V(G) = 14$ edges $- 12$ nodes $+ 2 = 4$

$V(G) = 3$ predicate nodes $+ 1 = 4$

Deriving Basis set & test cases:

Following steps:

1. Using the design or code as a foundation draw a corresponding flow graph.
2. Determine the cyclomatic complexity of the resultant flow graph.
3. Determine a basis set of linearly independent paths.
4. Prepare test cases that will force execution of each path in the basis

set is empty and

Control Structure testing:

Testing testing

It is a test case design method used to check the logical conditions in the Module. This tests each condition in the program to ensure that its error-free.

A simple condition is a boolean variable or a relational expression, possibly preceded with one NOT (¬) operator.

A relational expression takes the form

$E_1 <$ relational-operator $> E_2$.

arithmetic expressions ↑ one of the following (<, >, =, <=, etc)

A condition without relational expression is referred to as a boolean expression.

If a condition is incorrect, then atleast one component of the condition is incorrect

Types of errors in a condition include boolean operators errors, boolean variable errors, boolean pattern errors, relational operators errors and arithmetic expression errors.

Data flow Testing :-

The DFT method select test paths of a program according to the locations of definitions and uses of variables in the program.

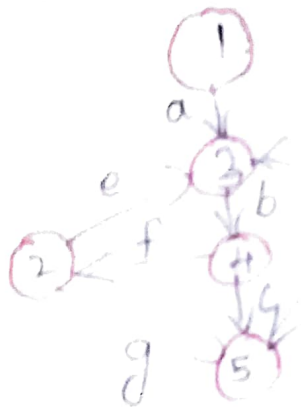
Assume that each statement in a program is assigned a unique statement number and that each function does not modify its

Graph matrix

It is a DS (Data structure) that assists developing a software tool that basis path testing.

A graph matrix is a square matrix whose size is equal to the no. of nodes on the flow graph. Each row & column corresponds identified node & matrix entries correspond connection between nodes.

Eg:



flow graph

A [link weight] can be added matrix entry to provide additional information about control flow.

If the [link] weight is 1 (connection exists) or 0 (a connection does not exist)

```
int function(void)
{
   int x = 0;
   int y = 19;
A: x++;
   If (x > 979)
     go to D;
   if (x % 11 == 0)
     Goto B;
   else
     goto A;
B: if (x / y == 0)
     goto C;
     else goto A;
C: Printf("%d\n", x)
   goto A;
D: Printf("End of list\n");
   return 0;
}
```

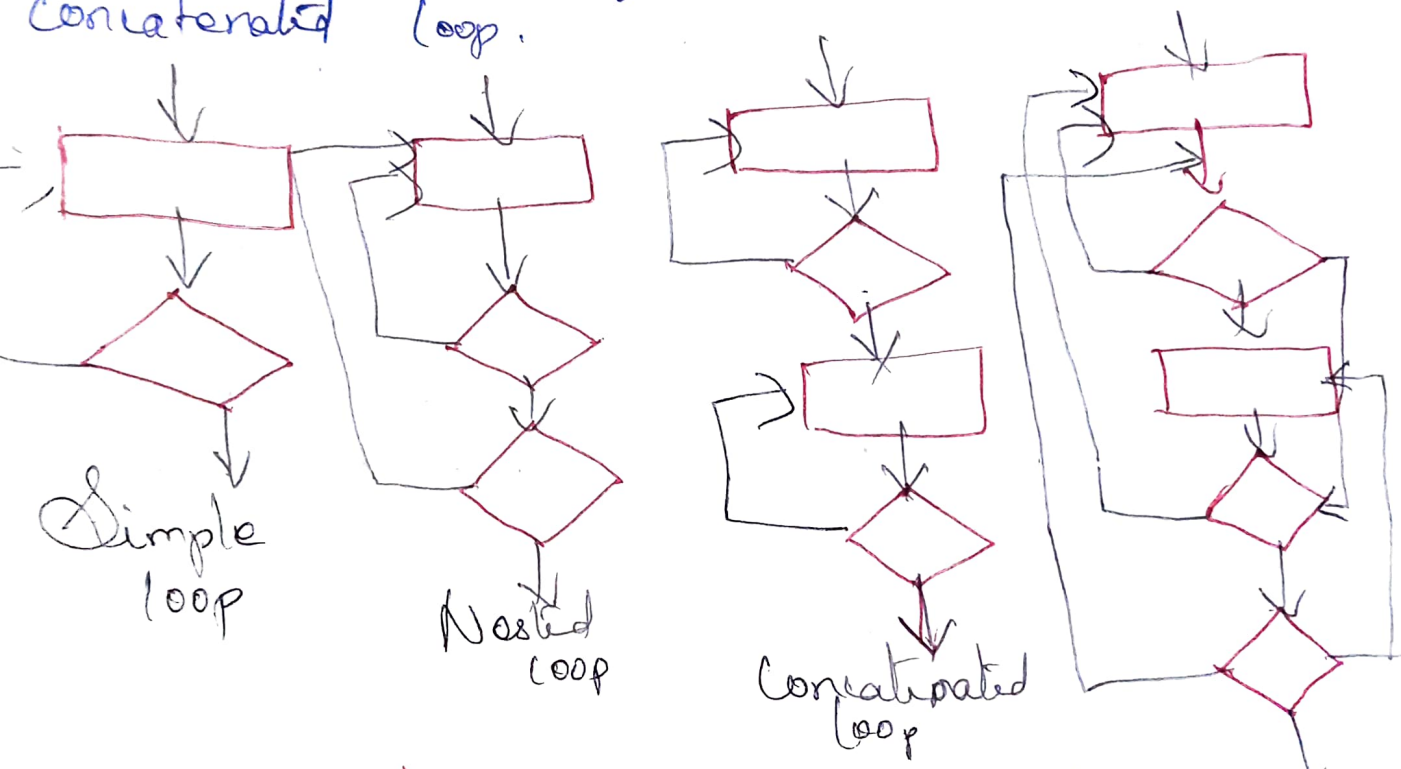
its TRUE set is empty that is
is based on the condition of statement
② A Definition - Use (DU) chain of variable
X is of the form [X, S, S'] where S and S'
are statement numbers, X is in DEF(S)
and USE(S') and the definition of X in
statement S live at statement S'.
② Every DU chain must be covered
at least once. this strategy is the DU
testing strategy.
~~Structural testing~~:
Loop testing:
② Loop testing is a white box testing
technique that is used to test the validity
of loop constructs.
② 4 different class of loops :- simple,
concatenated, nested, Unstructured loop.
② Testing occurs by varying the loop
boundary values
Testing of simple loop:
⇒ where n is the max. no. of allowable passes
② Skip loop entry
② Only one pass through the loop
② Two passes through the loop
② m passes through the loop, m<n
② n-1, n, n+1 passes through the loop
Testing of Nested loop
② Start innermost loop, set all other loop to min value
② Conduct simple loop for inner most loop while
holding the outer loop at their min iteration
parameter. add other test for out of range or
excluded values.
work outward conducting tests for the next loop
but keeping all other outer loop at min value

Testing Concatenated loop

for independent loop, Use the same approach as for simple loop otherwise Use approach applied for nested loop.

Testing of Unstructured loop:-

1. Redesign the code
2. Reflect Use of program structure
3. Depending on the resultant design
4. Apply testing for simple, nested or Concatenated loop.



Testing types:

Manual testing:
→ Testing Manually. without any tool & any script.
→ over the role of end user & identify the error (or) bugs.

Automation testing:
→ Unit testing, system testing, & User acceptance testing.
→ It saves Money & time.
→ Also called as test automation

Automated

Structural testing

Develop test cases based upon structure of code

May follow stronger testing or complementary testing.

Stronger ⇒ Strategy B ⇒ {errors}
Strategy A ⇒ {Strategy B errors + some additional errors}

complementary ⇒ two strategy are different with respect to detects errors at least some type of errors.

① Statement coverage:

- weakest form of testing
- Every statement executed at least once.

Branch coverage:

- Test cases are designed to make each branch condition assume true and false in turn.
- Also known as edge testing. each edge traversed atleast once.

condition coverage

Every branch must be involved atleast once.

All possible combination in decision must be exercised.

Not capable of capturing faults associated with decision carried out in presence of Multiple condition.

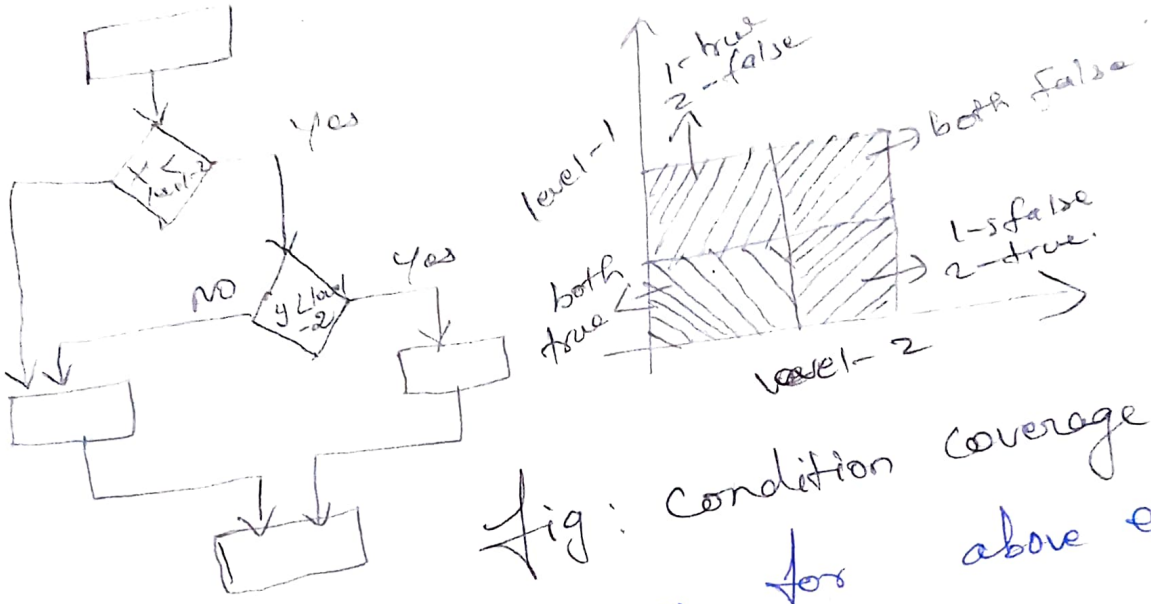


Fig: Condition coverage.

- ⊗ It tests cases for above ex.
- ⊗ n subcondition → 2n test cases.
- ⊗ if n get high → not feasibile
- ⊗ if n is small → then it feasible.

Path coverage:

Program are linearly independent paths in the executed atleast once.

It is defined in term of CFG.

White box testing intended to uncover

- ↳ logical errors
- ↳ typographical errors are random.

Consider all possible logical Paths in the program.

Test cases aimed at exercising a program along each path.

impractical → loops leads to very high no. of. paths.

Black-Box Testing:

- Behavioral testing
- focus on functional requirements of the software.
  - * incorrect or missing functions,
  - * Interface errors
  - * Errors in data structures (or) External database access
  - * Behavior or performance errors.
  - * Initialization & termination errors.
- A set of i/p conditions are identified to check all the functional requirements for a program.

Graph-based testing method:

- The First step is to Understand the objects and their relationships.
- Series of tests are defined to verify whether all the objects have the expected relationship with each other.

eg

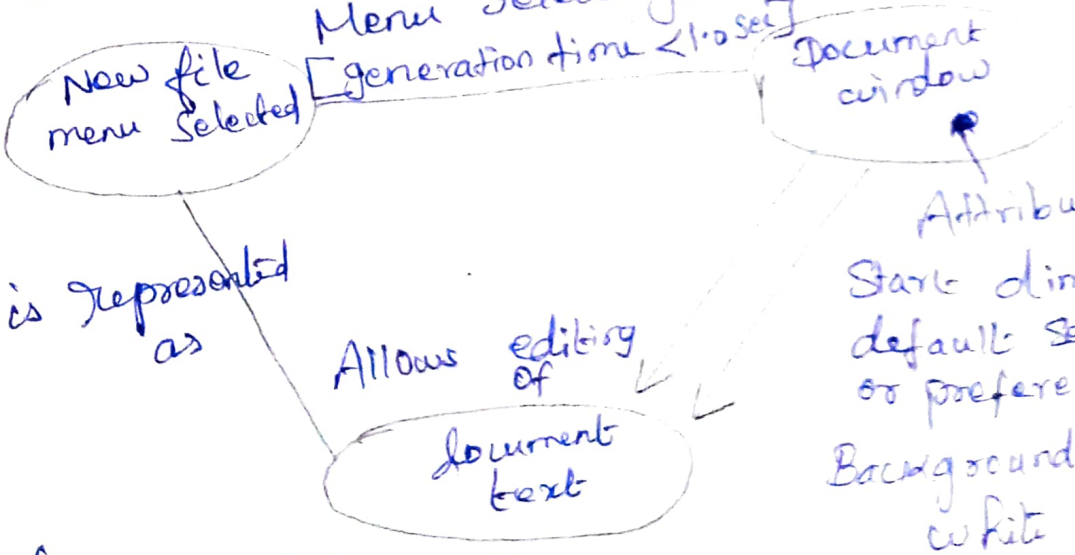


fig: Graph notation (b)

Nodes represent the object and links represents the relationship between objects.

Node weight describe the properties of the node and link weight describes the characteristics of the link.

- Directed link – relationship moves in only one direction.
- Bidirectional link – Symmetric link, relationship applies in both direction
- Parallel link – Used when a number of different relationships are established between graph nodes.

Behavioral testing methods that can make use of graphs:

Transaction flow modeling:

- node → steps in some transaction.
- link → logical connection between steps.
- DFD diagram can be used to assist in creation of graph of this type.

Finite State Modeling:

- node → User-observable states of the software
- links → transaction that occur to move from state to state.
- State diagram can be used to assist in creation graphs of this type.

~~Data flow~~ Timing modeling:

- node → ~~Program~~ objects
- links → sequencial connections between those objects.
- link weight → Used to specify the required execution times.

orthogonal Array testing

Data flow modeling

node - data objects,
link - translate one data object in to another.

Equivalence partitioning:

② Divides the input domain into classes of data and then derives the test cases from it.

② Ideal test case single - handedly uncovers a complete class of errors and reduce the total number of test cases that has to be developed.

② The case design is based on an evaluation of equivalence classes for an input condition.

② represent a set of valid or invalid states for input conditions.

② for each equalence class, test cases are selected so that the large number of attributes of an equivalence class are exercise at once.

Guidelines of Defining Equivalence class.

If an i/p condition specifies a

Range one valid and two invalid equivalence classes are defined.

i/p range : 1-10 Eq classes : {1..10}, {x<1}, {x>10}

Specific value one valid and invalid equivalence class are defined.

i/p value : 250 Eq classes : {250}, {x<250}

Member of a set → one valid and one invalid equivalence class are defined.

i/p {-2.5, 7.3} Eq class {-2.5, 7.3} {}

i/p : [Boolean value] - one valid & one invalid class are defined.

I/p : {true condition} Eg: classes : {true condition}
{false condition}

Boundary value analysis:

⊗ greater no. of errors occur at the boundries.

⊗ BVA is a test case design method that complements equivalance partitioning.

⊗ Select test cases at the edge of class.

⊗ Derive test cases from both i/p & o/p domains.

Guideline for Boundary value Analysis:

⊗ if i/p condition specifies [range] bounded by a & b then test case should be designed with value a & b as well as just above and below a & b.

⊗ if i/p ⇒ no. of values ⇒ test case should be developed exercise min & max numbers. values just above & below of min & max.

⊗ Apply guidelines 1 & 2 to o/p conditions. Produce o/p with min & max values as well as test the values just above & below.

⊗ if internal program data structures have prescribed boundaries design a test case to exercise the ds at its min & max.

Orthogonal array testing:

- Input domain is relatively small but too large to accommodate exhaustive testing.
- finding errors with region faults.
- L_q orthogonal array of testcases is created.
- It has balancing property.
- Test case dispersed uniformly throughout the test domain.
- Orthogonal array approach enables us to provide good test coverage with fewer testcases than the exhaustive Stratergy.

Comparison testing:

- Redundant software & hardware are often used to minimize the errors.
- each version is tested with same test data → all should provide identical output.
- independent version from the basis of a black box technique called comparison or back to back testing.
- If all o/p is same then all the implementation is correct.
- If o/p is different, each application is investigated to determine if a defect is in one or more version.
- It is done by automated tool.

Problems:

- Not fool proof
- condition testing is fail

Syntax driven testing:

- Good for compilers & Syntactic pattern classifiers.
- Specification expressed in standard BNF or Production rules.
- Straight forward approach.
- Each production rule is applied atleast once.

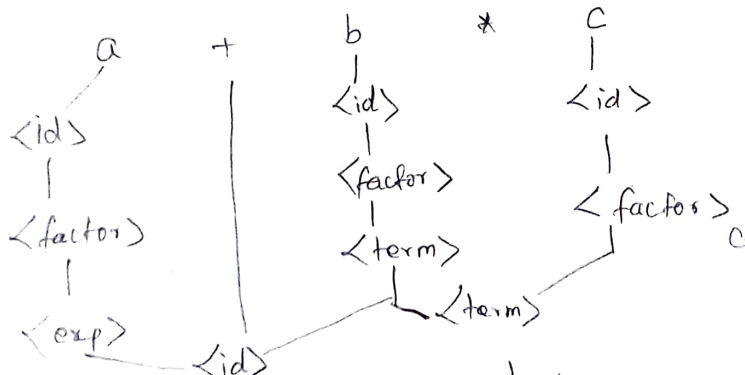
<exp> ::= <exp> + <term> | <exp> - <term> | <term>

<term> ::= <term> x <factor> | <term> - <factor>

<factor> ::= <identifier> | (<expression>) | <factor>

<id> ::= |a|b|c|d|e..... |2.

- each test cases that exercise the above rule.
- Depending upon the production rules each statement will be tested.



Decision table

Conditions	
Con. 1	
Con 2	
Action	
act. 1	
act. 2	

Decision table - based testing:

- If - then Statement

form a rule - If (cond 1 and cond 2 and condn) then action i

Table => no. of. columns has all test reqr.

Upper part => conditions

lower portion => actions.

n = 2

then columns = 4

... used in functional testing.

② Disadvantage of Decision table method all i/p are considered eventhough requirement suggest another way of handling the problem.

② independance of i/p is assumed in boundary value analysis and equivalance partitioning

② These are all overcomed in cause-Effect graph

② combination of specific i/p & o/p r/s.

② Input - cause, output - effect

② Intermediate nodes linking cause and effect forming logical expression.

> cause and effect of ATM:

Cause: C_1 : Credit
C_2 : Debit
C_3 : Account number is valid
C_4 : Transaction amount is valid.

Effect E_1 : Print invalid command
E_2 : Print invalid account number
E_3 : Print debit amount not valid
E_4 : Debit account
E_5 : credit account.

① Construct the cause and effect

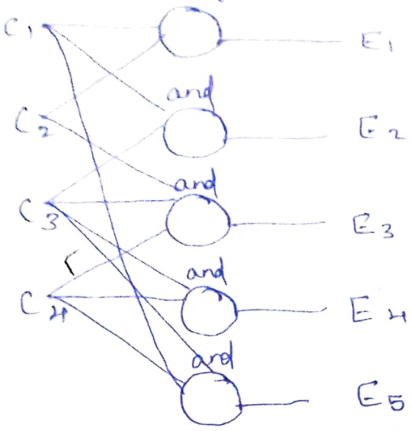
② no. of. nodes depends upon cause & effect

② nodes connected by "and" or "or" operator

② negative symbol is placed → effect is true

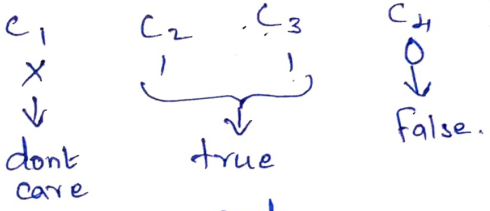
Once associated node is false

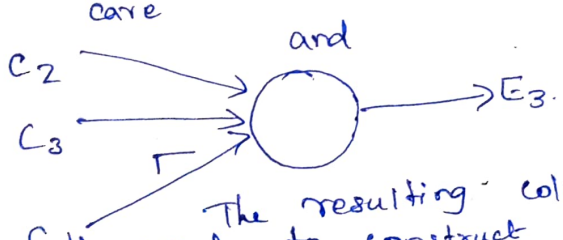
and → if all the i/p are true (1)
or → if atleast one i/p is true (1)
→ if I/p are false (0)



② E_3 is executed only if C_2 & C_3 are true C_4 is false does not bother about C_1.

② Cause has no impact on the effect can be regarded as dont care (x).





The resulting column in decision table used to construct test cases.

C_1	0	1	x	x	1
C_2	0	x	1	1	x
C_3	x	0	1	1	1
C_4	x	x	0	1	1
E_1	1	0	0	0	0
E_2	0	1	0	0	0
E_3	0	0	1	0	0
E_4	0	0	0	1	0
E_5	0	0	0	0	1

fig: ATM - cause effect decision table.

⇒ if decision table has to be reduced then back tracking is followed

modified ... made in requir... & D2 is

Misunderstood

Regression Testing:

- Recent change in the program affect existing feature
- Partially select of already executed testcase
- re execute to ensure existing functionalities
- ensure better performance.
- early identification of bugs and errors.

Types of regression testing:

Corrective regression testing:
- no changes in the Software
- Existing testcases are easily reused.

Retest-all regression testing:
- all aspect of testing of a product
- reusing all testcases
- not need if change is small.

Selective regression testing:
- new code is added to the already existing code.
- Subset of existing test case is tested.
- reduce effort and cost.

Progressive regression testing:
- Changes done in specification
- no ~~exist~~ feature is exist in previous version.

Complete regression testing:
- Multipile changes done to existing code
- new change make impact in node node

Need of regression testing:
- when change is made in requirement and code is modified.
- when new feature is added.
- Defect fixing.

Regression testing technique:

Regression test selection

Reusable test case: Used in Succeeding regression cycle.

Obsolete test case: Can't be used in succeeding regression-cycle.

Prioritization of test case: business, Critical, frequently used

- Selection of test case based on priority.

Effective regression test done by:

- Integration test case
- complex test case
- Boundary value test case
- Sample of successfull test case
- Sample of failure test case

Regression ~~Regres~~ testing tool:

Selenium: browser based regression testing.

Quick test professional: automated Software designed to automate functional & regression test case.

Rational functional tester (RFT): Primarily Used for automating regression test case.

Unit testing: Test individual components independently, to ensure the s/w Quality.

- conducted after Module has coded & Successfully reviewed.
- Important control paths are tested to uncover errors with in the boundary of Module

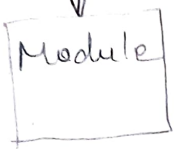
Unit test considerations:

- Module interface is tested to ensure information properly flows into and out of the program.
- local data structure is examined to ensure that data stored temporarily maintain its integrity
- All independent path through the control structure is exercised.
- All statement in the module tested
- all error handling paths are tested

Common errors:

1. Misunderstood
2. Mixed mode operation
3. Incorrect initialization
4. Precision inaccuracy.
5. Incorrect symbolic representation.

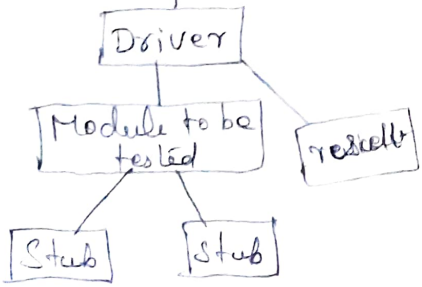
Good design dictates that error conditions be anticipated and error-handling path set up to reroute when an error occurs. this approach is called antibugging.



interface
local data structure
Boundary condition
Independent paths
Error handling paths

Unit test procedure:

- Source code developed → reviewed → Verified ⇒ Unit test begin.
- Component is not stand-above program driver, stub, s/w developed for each unit.
- To test single Module we need a complete environment, necessary for execution of the Module.



interface
local DS
Boundary condition
independent path
Error handling path

② Stub & Drivers are designed to provide complete environment.

② Driver → non-local data structure accessed by the Module under test & code to call the function of the Module under test.
→ It is a program that accept test data, print relevant results

② Stub → dummy procedure that has same I/p parameters but simplified behaviour.
→ Sub program used to manipulate data

② Simplified when component high cohesion

② One function addressed no. of test. case reduced & errors easily predicted & uncovered

Integration testing: Group of dependent components are tested together to ensure their quality of their integrated Unit ② Test Module interfaces to ensure there are no errors in parameter passing.

② Construct program structure

② Conduct test to uncover error associated with interfacing

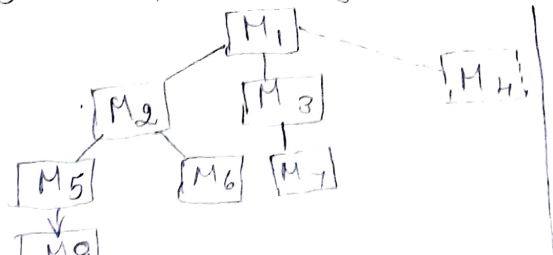
② Resources & environmental integration

* Non incremental integration ⇒ big bang approach
Components combined in advance, entire program is tested as a whole. Once set of errors corrected, more errors
* Incremental integration ⇒ Pgm is constructed & tested in small increments, errors are easier to isolate and correct, interfaces likely to be tested completely.

Top down integration:

Depth first integration: M1, M2, M5 & M8 or M1, M2, M6 & M8

Breadth first integration: M2, M3, and M4



big bang (simple approach)
→ Components combined in advance
→ Entire pgm is tested as whole
→ Result in chaos

① Main control Module → Test driver
Stubs are substituted for all components.
② Subordinate stubs are replaced one at a time
③ test are conducted as each component is integrated.
④ on completion, another stub is replaced.
⑤ Regression testing conducted to ensure new errors have not been introduced.

② logical Problem occurs when Processing lower levels
① Delay many test until stubs are replaced
② Develop stub that perform limited functions
③ integrate the software form the bottom of the hierarchy upward.

Bottom-up integration:
① Low level components are combined in to clusters. that perform a specific s/w sub fn.
② A driver (pgm is written to co-ordinate) test case i/p & o/p.
③ whole cluster is tested.
④ Driver is removed & clusters are combined moving upward in program structure.

① components are combined & form cluster 1, 2, 3
cluster tested using driver
② cluster 1, 2 are subordinate to Ma after testing, Drive D1 & D2 is removed & clusters are interfaced directly to Ma.
③ Similarly D3 removed & integrate with Module Mb
④ both Ma & Mb integrated with Mc

Regression testing:

Automatically rerun some of the product has been made.

a Software whenever a slight change to

(1) capture a test for repay

(2) comparing new O/p with old ones to make sure that there are no unwanted changes.

Smoke testing:

integration testing designed as a pacing mechanism for time critical projects. allowing the S/w team to assess its project on a frequent basis.

(1) S/w components translated in to codes are integrated into a 'build'.

build → all files, libraries, reusable Modules & engineered components that are required to implement one or more product functions.

(2) A series of tests is designed to expose error that will keep the build from properly performing its function.

(3) The build is integrated with other builds and the entire product is Smoke tested daily. The integration approach may be either top-down or bottom up.

Benefit: applied on complex, time critical, S/w engineering projects.

- Integration risk is minimized
- The quality of end-product is improved.
- Error diagnosis & correction are simplified.
- Progress is easier to assess.

contain test plan, test procedure, (15)
work product of S/w process.

Specific tools are documentation as test specification.

i) ~~User inter~~ Interface integrity
ii) Functional validity
iii) information content
iv) performance.

Validation testing: (performance)

The integrated software is tested based on requirements to ensure that the desired product is obtained.

Ensure that

- all functional requirement are satisfied.
- all behavioral requirement are achieved.
- all performance requirement are attained.
- Documentation is correct
- usability and other requirements are met

After each validation test.

- function conforms to specification and is accepted.
- A deviation from specification is uncovered & a deficiency list is created.

Configuration review: (or) (audit)

A review is taken to ensure that all the elements of S/w configuration are developed as per requirements

Alpha testing:

Alpha test:

- conducted at developers site by end user
- conducted in control environment.
- under the supervision of developer.

Beta test:

- conducted at end users site.
- tested by the customer without developer
- after this test engineer make modification in the S/w & prepare for release of S/w product

System Testing:

Series of tests conducted to fully computer based system (s/m).

types:

1. Recovery testing
   - check s/m ability to recover from failures.
   - Software is forced to fail and then it is verified whether the system recovers properly or not.
   - For automatic recovery, then reinitalization, checkpoint mechanism, data recovery & restart are verified.
2. Security testing
   - verifies s/m protection mechanism
   - Prevents improper penetration or data alteration.
   - Prevents intrusion in) unauthorized access or willful damage.
   - Tester plays role of the hacker to acquire password, may purposely cause errors to attempt to system error

goal ⇒ penetration cost is more than value of information

(c) Load testing
(i) Determine breakpoint of a sys
to establish max service level.
(ii) Demand resources in abnormal quantities
frequency & volume.
(iii) Variation of Stress testing is a
technique called sensitivity testing is tried to
uncover data from a large class of valid
data that may cause instability or improper
processing.
(d) Performance testing
(i) Evaluates run time performance
of a sys. (ii) resource utilization (cpu load,
throughput, response time, memory usage) measured by
(iii) for big sys (ex) banking sys
many users connecting to servers so this testing
is very difficult.
(iv) Beta testing is useful for
performance testing.

Debugging:

Debugging is a process of removal
of a defect. It occurs as a consequence
of successful testing.
(i) Select with execution of test case
(ii) Actual test result is compared with
expected result.
(iii) debugging find the cause of correspondence
between actual & expected results.
(iv) Suspected cause are identified

memory dumps & traces are examined & program with suitable statement is located to obtain clues to error causes

In this method " Let computer find the error" approach is used.

This is least efficient method of debugging

Applicable to small programs

Source code is examined by hand from symptoms of potential causes of errors

Other methods:

Test driven development:-

Programmer first write the test Script then write the code to pass the test.

whole process done incrementally

Pair programming:-

More than one programmer implements the code and thus algorithm are checked in a better way.

Advantages:

- Common ownership of programs
- Informal view by both.
- Support Refactoring

Refactoring:

- Improve existing code caused by
- prevent design decay with time changes
- change made in internal structure of Software to make is easier to understand.
- Done on Source code for 'improve design'

Main risk is existing code may break due to changes

Rule: *Refactor in small parts
*Have test Scripts to test existing fn.

Refactoring become continuously improving ⇒ Design decay evolves & improve with time.

Need: Duplicate code exist
Method become long
class are large
Parameter list are long
Too many Swith Statement
Too many communication blw objects.
Passes call or message chaining.

System Testing between

Deployment testing:

- also called configuration testing.
- s/w should execute on variety of platform under more than one OS environment
- examine all installation procedure & specialized installation used by customers.

Software Maintenance:

Software Implementation technique:

Coding process:

- starts from some form of design has been done.
- programming team have a designated leader, well defined organization structure & thorough understanding of the duties & responsibilities of each team member.
- Implementation team must be provided with a well defined set of software requirements, an architectural design, specification & detail design description.
- each team member must understand the objective of implementation.

Incremental coding process:

- coding written for implementing only part of the functionality of the module.
- compiled & tested with some Quick test to check.
- when it is passed, developer proceeds to add further functionality.

Advantages:

- easy identification of error
- test script prepared and run each time

a) Extracting functionality cohesive code
b) Add/remove parameters as per req.

Improving class:
a) moving method to appropriate class
b) move field
c) Extract class from large class
d) Replace data value with object.
   => if it used frequently.

Improve hierarchies:
a) Replace conditional with polymorphism.
b) Pull up field/method => if a field exists in most of the subclass it pulled up to superclass.

Software Maintanance & RE-engineering:

Software Maintanance:-
Software Maintanance is an activity in which program is modified after it has been put in use.
either modifying the existance or add new component.

Types of maintenance:
Corrective maintenance - problems are corrected
Adaptive maintenance - Keep S/m up to date
Perfective maintenance - Usable for long period. improve Reliability & perfor-mance
Preventive Maintenance - prevent the future pro-blem

Phases
Identification & tracing - identify requirement for maintenance.
Analysis - Cost of Maintenance is analyzed.
Design - Analyze new modules & test cases are created for validation & verification
System Testing - Integration testing is done out between new module & system

Acceptance testing - Complaints are addressed to address in next iteration.

Delivery - System is organized to delivered & final listing takes place.

Maintanance Management - Configuration Management
Version control tool
Semi version (or) patch Management

Software Re engineering:

- Redesign of business process to achieve improvement in cost, quality, service.
- Rewrite parts of s/w without impact its functionality.
- Makes easier to maintain

Process:

- Decide a part of s/w to re-engineer
- perform reverse engineering
- Restructure the program
- Re structure the data

Advantages

- Reduced risk
- Reduced cost

BPR Model (Business process Re engineering)

- Business process redesign.
- Analyse and design of workflow and business process.
- Set of related work activities performed by employees to achieve business goal.

BPR Principle :-

- Organize around outcomes
- Who use o/p of the process perform the process
- Information Processing into Real work that

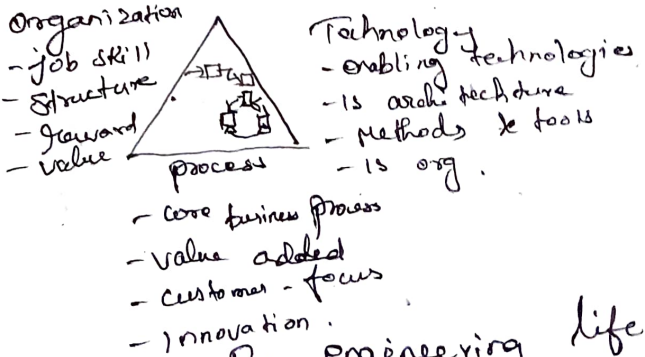
- Treat resources as they centralized (or)
- link parallel activities instead of integrating
- Put decision point where work is performed & build control into the process.
- capture information once and at the sources.

Objective of BPR:

- To improve process
- reduce costs
- improve customer service
- improve productivity.

BPR frame work:

- Business process framework is Critical component of framework.
- comprehensive, industry agreed, multi-layered view of the key business process required to run an efficient, effective and agile digital enterprise.



Business Process Re-engineering lifecycle:

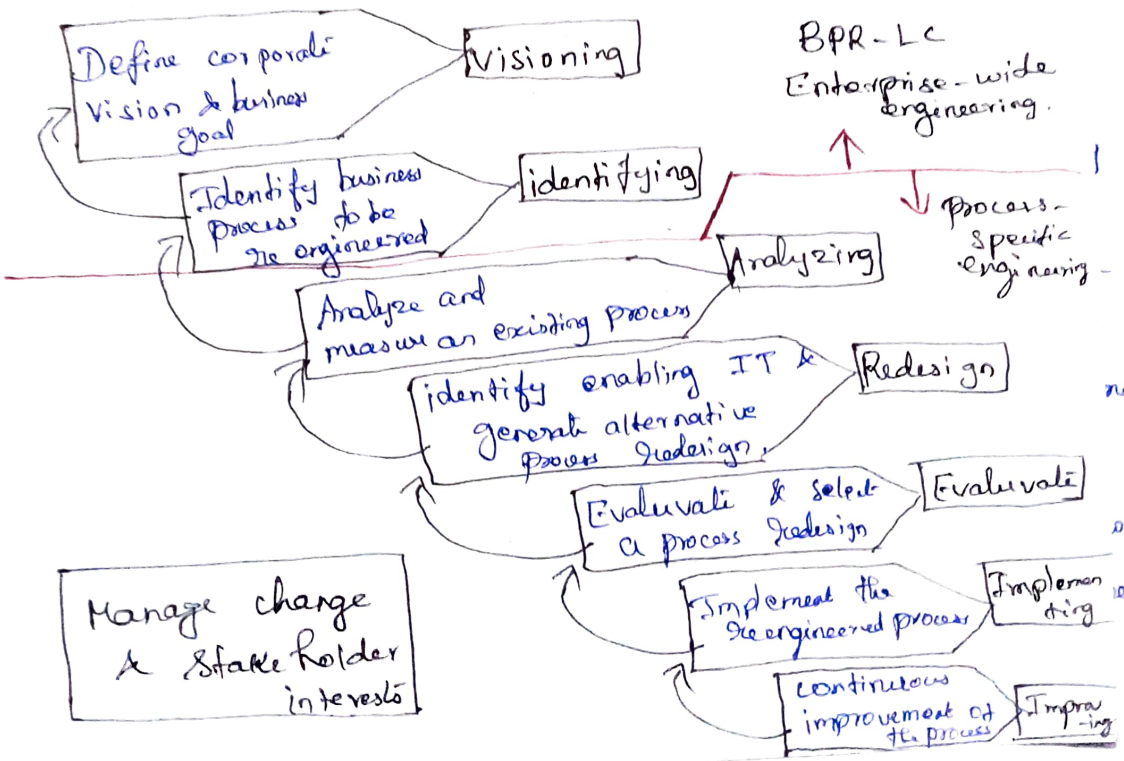
- BPR lifecycle Methodology
- proposed by process reengineering lifecycle
- Six-stage methodology
- follow during BPR projects by companies

BPR steps:

1. Identify & communicating the need for change
2. Put together a team of experts.
3. Find the inefficient process and define key performance indicators (KPI)
4. Re-engineer the process and compare KPI

Benefit of BPR:

- core expected results.
1. Increase effectiveness
2. Identify core function & inefficient
3. Reduce overall cost
4. Meaningful work for staff
5. Process promote greater staff involvement.



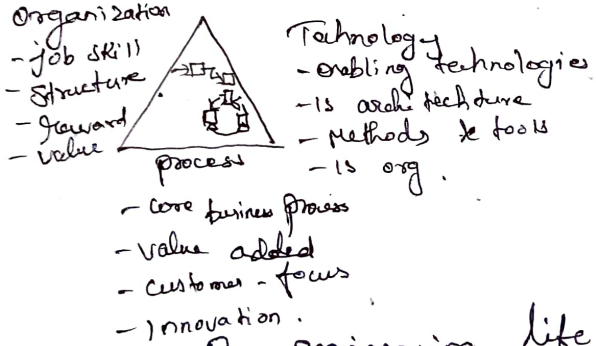
- Treat resources as they centralized
- link parallel activities instead of integrating
- Put decision point where work is performed & build control into the process.
- capture information once and at the sources.

Objective of BPR:

- To improve process
- reduce costs
- improve customer service
- improve productivity.

BPR frame work:

- Business process framework is critical component of framework.
- comprehensive, industry agreed, multi-layered view of the key business process required to run an efficient, effective and agile digital enterprise.



Business Process Re-engineering lifecycle:

- BPR lifecycle Methodology
- Proposed by process reengineering lifecycle.
- Six-stage methodology
- follow during BPR projects by companies
- Process Identification