UNIT-1 REVIEW OF SOFTWARE ENGINEERING

S/w is more than program. It consist of a program, documentn & the procedure used to set up & operate the s/w system. The component of s/w system are – Program, Documentn, Procedure. Any program is a subset of s/w & it is a set of instructn & it become s/w only if the documentn & the operating procedure manuals are prepared. Program is the combination of both sourcecode & object code.

Software Engg.

The establishment & use of sound engg. principles in order to obtain economically developed s/w that is reliable & works efficiently on a real m/c.

A discipline whose aim is the prodn of quality s/w that is delivered on a time, within a budget & that satisfy the customer reqto.

Both the defn are popular & acceptable. However, due to ↑oe in the maintenance cost, objective is now shifting to produce the quality s/w that is maintainable, delivered on time, within the budget & also satisfy the customer reqts.

SDLC

SDLC is a process followed for a s/w project. It consist of a detailed plan describing how to develop, maintain, replace & alter or enhance specific s/w. The lifecycle define a methadology for improving a quality of s/w & the overall development process.

SDLC includes the following stages – Feasibility study → RA → Design → Coding → Testing → Deployment.

S/w Evolution

The process of developing a s/w project using SE principles & methods is known as S/W evolutn. This includes that initial develop^t of the s/w & its maintenance & update, till desired S/W product is developed, which satisfy the expected reqt.

S/W evolutn is the set of activities both technical & managerial that ensures that s/w continues to lead the orgn & business process in the cost-effective manner.

Evolutn starts from reqt. gathering process after which developr create a prototype of intended s/w & show it to the user to get their feedback at the early stage of S/W product develop^t. The users suggest the changes on which the several consecutive updates & the maintenance keep on changing too. This process changes to the original s/w till the desired s/w is completed. Even after the user has the desired s/w in hand, the advancing technology & the changing reqt. force the s/w product to change. Recreating a s/w from scratch & to go one on one with the reqt. is not feasible. The only feasible & the economical soln is to upgrade the existing s/w, so that it fulfill the latest reqt.

10/8/16

Software Testing

It is the process of executing the s/w in controlled manner in order to answer a question 'Does the s/w does as specified?'

The process of analyzing the s/w to detect the difference b/w the existence & required conditions, i.e. bug & to evaluate the features of s/w items.

The process of analyzing the program with intention to find the bug or error.

Testing the program consists of set of test inputs, i.e. test cases

& observing 'Does the program behave as expected', if a program fails to behave as expected, then the condⁿ under which the failure occurs are listed for later debugging & correction.

Testing Limitations

1) You can't test a program completely.
2) We can test only against the s/w reqt.
   - may not detect the errors in the reqts.
   - Incomplete or ambiguous reqt may lead to incorrect testing.
3) Exhaustive testing is impossible in present scenario
4) Time & the budget normally require very careful planning of testing effort.
5) You can't test every bug.
6) Can't test every valid & invalid i/p.

Error - It is the deviation from actual & expected value. It represent mistakes made by people.

fault - It is an incorrect step, process or data defⁿ in a computer program which causes the program to behave in an unintended manner. It is the result of error.

Bug - It is a fault in the program which causes the program to behave in an unintended manner. It is an evidence of a fault in the program.

failure - It is the inability of the system or a component to perform its required functions within specified performance reqts. failure occurs when fault executes.

Verification & Validation

Verification - It is the process of evaluating the system or component to determine whether the product of a given development phase satisfy the cond^{n} imposed at the start of this phase.

We apply the verifcn activities from the early phase of the s/w development & check or review the documents generated after the completion of each phase. Hence, it is the process of reviewing the reqt. document, design document & the other related document of the project.

Validation - It is the process of evaluating a system or a component during or at end of the development process to determine whether it satisfy the specified reqts. It requires the actual ex^{n} of a program. It is dynamic testing & requires a computer for ex^{n} of the program.

Verification	Validation
1) Ensure that the s/w system meets all the functionality as per the specifi^{n}.	1) Ensure that the functionalities meet the intended behaviour.
2) It takes place first & include the checking for the document^{n} & coding, etc.	2) It occurs after the verif^{n} & involve checking the overall product.
3) Done by developer.	3) Done by tester
4) Includes the static techniques like s/w review, code walk-through.	4) It includes the dynamic activities, i.e., executing the s/w against the customer reqts.
5) It is an objective process.	5) It is a subjective process.

Test Case & Test Suite

A Test case consist of inputs & its expected o/p. When we pass

testing, giving selected i/p to the program & note the observed o/p. We compare the observed o/p with the expected o/p & if they are the same, the test case is successful. If they are different, i.e, the failure condⁿ with the selected i/p & this should be recorded properly in order to find the cause of failure. A good test case has high probability of showing the failure condⁿ. Hence, test case either should identify weak areas of the program.

The test case is a triplet (ISO) where the I stands for i/p, S stands for state of a system at which the data i/p & the O stands for the expected o/p.

Test suite is a collectⁿ of set of all test cases with which a given s/w product is to be tested. We may have test suite of all test cases, test suite of all successful test cases & the test suite of all unsuccessful test cases. Any comboⁿ of a test cases will generate a test suite. All the test suites should be preserved as we preserve the source code & other documents. They are equally valuable & useful for the purpose of maintenance of the s/w.

Deliverables & Milestones

Different deliverables are generated during the different phases of a s/w developᵗ like SRS, SDD, coding, etc.

The milestone are the event that are used to ascertain the status of the project. For instance, finalizⁿ of SRS is a milestone.

Verification Methods

The objective of any verificⁿ method is to review the document with the purpose of finding the fault. Verificⁿ help in prevent

of potential fault, which may lead to failure of s/w. Verific^{n} & valid^{n} activities may be performed after the implement^{n} or coding phase. However, ~~only~~ verific^{n} is possible in the phases early to the reqt. phase & design phase.

Many methods are commonly used in practice are as follows -

1) Peer Review - This is the simplest way of reviewing the document or program to find out faults during the verific^{n}. We give the document or program to someone else & ask to review the document.

Our force should be to find the fault in a program not in the person who develop them. The activities involved may be the SRS document verific^{n}, s/w design document verific^{n} & the program verific^{n}. The reviewer may prepare the report of observat^{n} & finding or may inform verbally during the discussion. This is an informal activity to be carried out by the peer & may be very effective if the reviewer have domain knowledge the good programming skills.

2) Walk through - These are the more formal & systematic than the peer review. In this, the author of a document presents it to the small group of 2 or 7 persons. Participants are not expected to prepare anything. Only the presenter who is the author prepare for the meeting. The documents are distributed to all the participants. During the meeting, the author introduce the material in order to make them familiar with it. All the participants are free to ask questions. All the participants may write the questions, so that everyone may see & give the views. After the review, the author write a report about findings & faults pointed out in the meeting.

The disadvantages of this system are - the non-preparation of

participants & the incomplete of a documents presented by the author The author may hide some critical areas & emphasis on some specific area of his/her interest. Walkthrough may help us to find potential faults & may also be used for sharing the documents with others.

3) Inspection- Many names are used for this verific^n like formal review, technical review, inspect^n, formal technical review (FTR), etc. This is the most structured & the most formal type of verific^n method & is commonly known as inspect^n. In this type of verific^n, the presenter is not the author but some other person who prepare & understand the document being presented. This forces that person to learn & review the document before the meeting. The documents are distributed to all the participants in advance in order to give sufficient type for prepar^n. Rules for such meeting are fixed & communicated to all the participants. A team of 3-6 participants are considered lead by presenter & the recorder. A presentor & the recorder are also added to the screen to assure that rules are followed & ~~used~~ views are documented properly.

Every person in the group participates openly, actively & follows the rule about how such a review is to be conducted. Everyone may get a time to express their views, potential faults & the critical areas. Important points are displayed by some display mechanism, so that everyone can see & note them.

After the meeting, a report is prepared by the presentor & circulated to all the participants. They may give their views again if any. A final report is prepared after including the necessary suggestions. Inspections are very effective to find the faults & problems in the document like SRS, SDD, coding, etc.

SRS Document Verification

The outcome of the reqt. analysis phase of the SDLC is the SRS document. This describe "what do we expect from the system? However, it does not carry any detail about "How do we achieve these expectations". The SRS also become a Legal document to resolve any conflict b/w the customer & the org^n.

The SRS document should cover both functional & the non-functional reqts. functional reqt. are the expect^n from the proposed s/w. They explain what the s/w has to do. These are also known as product features. Non-functional reqts. are the quality reqts, that include how well the s/w does, what it has to do. Some of the non-functional reqts. are reliability, portability, maintainability, etc

16/8

Nature of SRS

SRS should include the following-

1) Expect^n from s/w - The SRS document should clearly specify what do we expect from a s/w? & broadly describe the fn^c of the
2) Interfaces of s/w - The s/w will interact with many persons, h/w & other external s/w. These interfaces should be written & the form for the interact^n may also be provided.
3) Non-Functional Reqts - These reqts. are very important for the success of the s/w. They may help us to design a performance criteria in the terms of reqts like response time, speed, availability, recovery of the various s/w fn^c, etc. These reqts. should also be placed properly in the SRS reqts.
4) Implement^n Difficulties & limitations - There may be some limit^n of P/L & the database integrat^n, etc. All the constraints of project implement^n including resource limit^n & the operating environment should also be specified in the SRS

The SRS should be written in clear & un-ambiguous language, simple which may be understandable to all the developers & the customer.

Characteristics of SRS

The SRS document act as a contract b/w the developer & the customer. This document should have the following characteristics as specified in the IEEE std for a SRS-

1) correct 2) Complete 3) Consistent 4) Unambiguous
5) Modifiable 6) Tracable 7) Verifiable 8) Stable

SRS Document Checkpoints

The SRS document is reviewed by the testing persons by using any verific^n method. We may use inspect^n due to their effectiveness & capability for producing the good result. A checklist is a popular verific^n tool which consist of a list of critical info^n content that a deliverable should contain. Checklist are used during the review & may make a review more structured & effective. An SRS document checklist should address the following issues-

1) Correctness - Every reqt stated in the SRS should correctly represent an expect^n from the proposed s/w. If the expect^n is that the s/w should respond to all the button presses within 2 seconds, but the SRS states that s/w shall response to all the button presses within 20 sec, then the reqt is incorrectly documented.

2) Ambiguity - There may be an ambiguity in the stated reqt, if the reqt conveys more than one meaning, then it is a serious problem. Every reqt must have a single interpret^n only. Hence,

the reqt; statement should be short, precise & clear.

3) Completeness - The SRS document should contain all the functional & the non-functional reqts. It should also have forms, diagrams, tables & references of all keywords.

4) Consistency - Consistency of a document may be maintained if the stated reqt. do not differ with the other stated reqt. within the SRS document.

5) Verifiability - An SRS document is said to be verifiable iff every reqt stated in the SRS is verifiable. Non-verifiable reqts include stmts like good interface, excellent response time, well etc. These stmts shouldn't be used. An eg. of verifiable stmt is marksheet shall be displayed on a screen within the 10 seconds.

6) Modifiability - The SRS document should include modifn w/o disturbing its structure & the style. It is very important characteristic due to the frequent changes in the reqt.

7) Tracability - The SRS document is tracable if the origin of each reqt. is clear & may also help for the future development.

8) Feasibility - Some of the reqts. may not be feasible to implement due to technical reasons or lack of resources. Such reqts. should be identified & removed from the SRS document. A checklist may also help to find non-feasible reqts.

The SRS document is the source of all future problems. It must be reviewed effectively to improve the quality of a s/w. Every review process will add to improvement of its quality which is dependant upon the characteristics discussed above. A checklist should be designed to address the above mentioned issues. A well designed & meaningful checklist

may help the objective of producing good quality maintainable s/w that deliver on time within the budget.

17/8

Source Code Review

A source code review involves one or more reviewers examining the source code & providing the feedback for developers, both +ve & -ve, reviewers shouldn't be from the develop^t team.

Code review in the most org^n are painful experience for every one involved. The developer often feel like, it is a harsh session designed to beat out their will. And the other developers that may be involved often use this as a chance to show how much better they can be by pointing out possible issues in someone code.

We may review the source code for syntax, standards defined, readability & the maintainability. Review will have std. checklist as a guide for finding the common mistake & to validate the source code against established coding std. The source code review always improve the quality & find all the faults. The fault may be due to poor structure, breaking of the business rules, etc.

Issues Related to the Source Code Review

Soura code reviews help us to achieve good quality maintainable s/w within the budget. Some of the recommended SE practices for designing a source code are as follows -

1) Always use meaningfull values variables.
2) Avoid confusing words in names. - eg Number — Nox Numx
3) Do not abbreviate so to Number to 'NO'. 'Num' is better choice.
4) Declare the local variables & avoid global variables to the extent possible, thus minimize the scope of variables.
5) Donot overload variables with the multiple meanings.

6) Define all the variables with the meaningful, consistent & clear names.
7) Use comments to ↑se the readability of the source code.
8) Generally comments should describe what the source code does & not how the source code work.
9) Always update the comments while changing the source code.
10) All the divisors should be checked for the 0 or garbage va
11) Always remove unused lines of the source code.
12) Minimize the module buffering & maximize the module cohes
13) Source code filename should be all lowercase & contain a-z, 0-9, _ (underscore), . (dot).
14) All the loops, branches & the logic should be complete & correct & properly nested & also avoid the deep nesting.
15) The reason for declaring the static variables should be given
16) Always ensure that loop iterate the correct no. of times.
17) When m/m is not required, it is essential to make it fre
18) Release all allocated m/m & resource after use.
19) We may add many issues which are to be addressed during the reviewing. A good checklist may have the reviewers to organize & structure the review process.

User Document Verification

We prepare many documents during the SDLC. Some are for the users like installation guide, user manuals, system administration guide, etc. Some are prepared for the internal purposes like SRS, SDD, source code, etc & are known as documentation manu Verific^n of internal documents is essential for the success of imp- lement^n & quality of final product. The document which are given

to the customer are also important for the overall success of the project - These are part of the s/w supplied along with other variables. User documents may be provided as user manual in e-form, as a printed booklet or in the form of online help.

Review Process Issue

These documents should be reviewed thoroughly & all the consistency should be maintained in all the documents. The documents should be written in a simple & clear & short sentences. Install^n procedure of a s/w must be explained step-by-step with proper justific^n. All the tables, figures & the graphs should be numbered properly.

19/8

Software Project Audit

Audit of a s/w project is imp. activity & may be carried out at anytime during a SDLC. Generally, auditors are appointed by top mgt. to review the progress of project. The auditors are different from from the developers & the testers & may not have any involvement in the project. They may examine the progress of the project & the quality of the process wrt the many attributes like project planning, mgt, resources, development approaches, testing, appl^n architecture & the tech^l architecture for auditing process is the continuous activity & may be carried out many times during SDLC. We may audit SRS, s/w design doc & other related docs including the project. The audit process is a verific^n activity & the auditor prepare audit reports, after examine the records & the document. This report may help the mgt to initiate timely corr^n, if required if any in the develop^t the mgt may get to know about the delay implement^n of S/w engg. practices & stds, status of risk assessment activities, etc.

1) Relevance Scale - It has been given in the project audit & the review

checklist who measure the importance of any attribute at the time of auditing the project. Many attributes may have been define in the check-list. Auditor has to find their importance to the project of the state when the audit is being conducted.

2) Theory & Practice Scale - We may have further indicat? to the strength & weakness of the attribute given in the project audit. An attribute may be important moderatity at one pt. of time & maynot be imp at another pt. of time. The theory & practice scale is very useful & indicates the implement^n status of an attribute.

This type of qualific^n is very useful to monitor the progress of project process. Auditor should always be non-judgemental & having the good comm^n skill. They also need to behave in a +ve way in order to get the failure & the correct picture of the project. Project audit must be carried out many times during the develop^t. They will definitely improve the performance, quality & progress of project

2/8

Configuration Audit:

A configue^n rigt process that confirms the integrity of system product prior to delivery. There are 2 type of configue^n audit -

1) Functional Audit - The objective of fn^c audit is to provide an independant evaluat^n of a s/w product. Verifying its configue^n items actual functionality & the performance is consistent with the rqt specific^n. This audit is held just prior to the s/w delivery to verify that all the rqt. specific^n have been met.

2) Physical Audit - The objective of physical audit is to provide an independant evaluat^n of a s/w product items to confirm that all the components in the as-built version map to their specific^n. Especially, this audit is held to verify that the s/w & its document^n are internally consistent. So

Configuⁿ may be conducted by SQA Team, the configuⁿ mgt team or by the verificⁿ & validⁿ fnc.

Configuⁿ audit are conducted at the end of each life cycle phase they verify that

1) All required configuⁿ items have been produced.
2) All the configuⁿ items verify with the specified reqt.
3) Technical documents completely & accurately describe the configuration items.
4) All approved change requests have been resolved.
5) At the completⁿ of developt, the s/w or system product is ready for delivery.

UNIT- 2 FUNCTIONAL TESTING & STRUCTURAL TESTING.

Functional Testing

s/w Testing is very important but is an effort consuming activity. A large no. of test cases are possible & some of them may make a s/w fail. As we all know if observed behavior of the s/w is different from the expected behavior, we treat this as a failure. Failure is a dynamic condn that always occurs after the execution of s/w.

Functional testing techniques attempt to design those test cases which have higher probability of making a s/w fail. These techniques also attempt to test every possible functionality of a s/w. Test cases are designed on the basis of functionality & the internal structure of a program is ignored. Observed o/p are compared with the expected o/p for the selected i/p. The s/w is treated as a black box & therefore it is known as black-box testing.

The test cases are designed on the basis of the user reqt without considering the internal structure of a program. This black box knowledge is sufficient to design good no. of testcases.

In the functional testing techniques, execut of program is necessary & hence, these testing techniques come under the category of the validation. These techniques can be used at all the levels of s/w testing like unit, integration, system & the acceptance testing. They also help the tester to design the efficient & the effective test cases to find the fault in a s/w.

Boundary Value Analysis

This is a simple but the popular functional testing. Here we concentrate on the i/p values & design the test cases with the i/p values that are on or close to the Boundary line values. Experience has shown that such test cases have higher probab

of detecting a fault in a s/w. In the BVA, we select the values on or close to boundaries & all the o/p value may have one of the following -

1) Minimum value
2) Just above the min. value
3) Maximum value
4) Just below the max. value
5) Nominal value (avg).

Max. test cases in the BVA is $4n+1$ where n is the no. of inputs.

Q. Consider a program for determin^n of division of a student based on the marks in the 3 subjects. Its I/p is the triplet of positive integers say mark1, mark2, mark3 & the value are from interval 0-100. The division calculated according to the following rules

mark obtained (Avg)	Division
75 - 100	First (Hons.)
60 - 74	First
50 - 59	Second
40 - 49	Third
0 - 39	Fail

Total marks obtained are the avg. of marks obtained in 3 subjects. The program o/p may have [Fail, III, II, I, I(Hons)]

Sol:→ mark 1 - 0, 1, 50, 99, 100 = mark 2 = mark 3

Test case	mark 1	mark 2	mark 3	O/p
1	0	50	50	Second Fail
2	1	50	50	Second Fail
3	50	50	50	First second
4	99	50	50	First
5	100	50	50	First
6	50	0	50	Fail
7	50	1	50	Fail
8	50	99	50	First
9	50	100	50	First
10	50	50	0	Fail
11	50	50	1	Fail
12	50	50	99	First
13	50	50	100	First

Q. Consider a program for determining the day of week. Its o/p is a triple of day, month & year such that $1 \le month \le 12$, $1 \le day \le 31$, $1950 \le year \le 2$ the possible o/p would be the day of a week or invalid day. Design BV test cases.

Sol:- Month - 1, 2, 6, 11, 12 / Day - 1, 2, 15, 30, 31 / Year - 1950, 51, 85, 2014, 1

Test Case	Day	Month	Year	O/p Expected
1	1	6	1950	Valid
2	2	6	1951	valid
3	15	6	1985	Invalid
4	30	6	2014	Invalid
5	31	6	2015	Invalid
6	15	1	2014	Invalid
7	15	2	1985	valid
8	30	6	1985	Invalid
9	31	11	1985	Invalid
10	30	12	1985	Valid
11	31	2	1985	Invalid
12	30	2	1985	Invalid
13	2	11	2014	Valid

Note - We also consider single fault assumption theory of reliability which states that failure are rarely the result of simultaneous occurence of 2 or more faults. Normally one fault is responsible for one failure. With this theory in mind we select one o/p value for boundary values & the other n-1 o/p values as avg values.

Robustness Testing

This is the extension of BVA. Here we also select the invalid o/p values & see the responses of the program. Invalid values ar

also important to check the behavior of the program. Hence, 2 additional states are added, i.e., just below the min. value & just above the max. value. This extended form of BVA is known as robustness testing. Thus, the total cases in the robustness testing are $6n+1$, where n is the no. of i/p values. All the i/p values may have one of the following values - min value, just below the min value, just above the min value, max value, just above the max value, just below the max value & the avg. value.

Q. Consider the program of addition with i/p values x & y & it gives add^n of $x+y$ as o/p. The range is $100 \leq x \leq 300$ & $200 \leq y \leq 400$. Design the robustness testcases for above.

Sol^n → x : {99, 100, 101, 200, 299, 300, 301}

y : {199, 200, 201, 300, 399, 400, 401}

Test case	x	y	o/p Expected
1	99	300	Invalid
2	100	300	400
3	101	300	401
4	200	300	500
5	299	300	599
6	300	300	600
7	301	300	Invalid
8	200	199	Invaid
9	200	200	400
10	200	201	401
11	200	399	599
12	200	400	600
13	200	401	Invalid

Worst Case Testing

This is a special form of BVA test cases where we don't consider the single fault assumpt^n theory of reliability. Now failures are due to occurence of more than one fault simultaneously. The restrict^n of one I/p value & the other I/p value must be nominal is not valid in the worst case testing. The I/p may have one of the following values min., just above the min, max, just below the max & the avg. This will be the no. of test cases from $4n+1$ to 5^n test cases where n is the no. of I/p values.

Q. Implement worst case testing for previous question.

Soln

Test case	x	y	Expected O/p
1	100	200	300
2	100	201	301
3	100	300	400
4	100	399	499
5	100	400	500
6	101	200	301
7	101	201	302
8	101	300	401
9	101	399	500
10	101	400	501
11	200	200	400
12	200	201	401
13	200	300	500
14	200	399	599
15	200	400	600
16	399	200	499
17	399	201	500
18	399	300	599
19	299	399	698
20	299	400	699
21	300	200	500
22	300	201	501
23	300	300	600
24	300	399	699
25	300	400	700

Robust Worst Case Testing

In robustness testing, we add 2 more states, just below the min. value & just above the max. value. We also give the invalid inputs

& observe the behavior of the program. A program should be able to handle invalid i/p values, otherwise it may fail & give unexpected o/p values. There are the 7 states, so the total no. of test cases is 7^n where n is the no. of i/p This will be the largest set of test cases & requires the max. effort to generate such test cases.

Note - BVA is a simple technique & may prove to be effective when used correctly. Here, the i/p values should be independant with restricting its applicability in many programs. This technique does not make a sense for boolean variables where i/p values are true & false & no choices is available for the avg. value, just above the boundary values, just below the boundary values, etc.

26/8

26/0

Equivalence Class Test

A large no. of test cases are generated for any program. It is neither feasible nor desirable to execute all such test cases, we want to select a few test case & still wish to achieve a reasonable level of coverage. We may divide into domain into various categories with some rel'ship & expect that every test case from a category exhibits a same behaviour.

We may assume that if one representative test case work correctly, others may also in the same result. This assump^n allow us to select exactly one test case from each category. Each category is c/d as equivalence class & type of testing c/d equivalence class testing.

Q. Consider a prog^n for a square of a no. x (range 1-100), identify the equivalence class test cases o/p & i/p domain $1 \leq x \leq 100$. o/p outcome is range then square, otherwise invalid i/p.

Sol^n :- o/p domain

Test case	Input x	Expected o/p
O_1	50	2500
O_2	0	Invalid i/p

I/p domain

Test case	Input x	expected o/p
I_1	50	2500
I_2	0	Invalid I/p
I_3	101	"

Equivalence class : I_1 : {$1 \le x \le 100$} valid I/p

I_2 : {$x < 1$} invalid I/p , I_3 : {$x > 100$} invalid I/p.

Q. Consider a prog^m of addition with 2 I/p x & y & it gives the o/p as 'x+y'. Write the equivalence class testing for above. The range of both I/p are $x \rightarrow \{100 \le x \le 300\}$ & $y \rightarrow \{200 \le y \le 400\}$.

Soln. o/p domain → Test case x y Exp. o/p

O_1	200	300	500
O_2	0	300	invalid I/p.

Equivalence classes → $I_1 = \{100 \le x \le 300, 200 \le y \le 400\}$ for all valid I/p

$I_2 = \{x < 100, 200 \le y < 400\}$

$I_3 = \{x > 300, 200 \le y \le 400\}$ } Either x or y is invalid

$I_4 = \{100 \le x \le 300, y < 200\}$

$I_5 = \{100 \le x \le 300, y > 400\}$

$I_6 = \{x < 100, y < 200\}$

$I_7 = \{x > 300, y > 400\}$ } Both invalid I/p.

$I_8 = \{x < 100, y > 400\}$

$I_9 = \{x > 300, y < 200\}$

I/p domain table will contain x & y value for all the classes with respective o/p.

The equivalence class testing is applicable at every Test level like . The basic reqt. is that I/p or o/p must be partitioned based on the reqt. & every partition will give a test case. If one test case catches a bug, the other probably will too. If 1 test case doesn't find a bug, the other test case of a same equivalence class may also not find any bug.

29/8 ## Decision Table Based Testing

Decision tables are used in many engg. incidents to represent the complex, logical represent^n. An o/p may be dependant on different i/p cond^n & decision table give a pictorial view of various combos of i/p cond^n. There are 4 portions of decision table. The decision table provides set of conditions & their corresponding actions.

	Stubs	Entries
Cond^n	c_1 c_2 I c_3	II
Action	a_1 a_2 III a_3	IV

Decision Table.

The 4 parts of decision tables are known as -

1) Condition Stub - All the cond^n are represented in this upper left sect^n of the decision table. These cond^n are used to determine a particular act^n or a set of act^n.

2) Condition Entry - In the cond^n entry portion of the decision table, we have no. of columns & each column represent a rule. Values entered in the upper right portion of the Table are known as i/ps.

3) Action Stub - All the possible act^n are listed in the lower left portion of the decision table.

4) Action Entries - Each entry in this portion have some associated act^n or set of act^n in this lower right portion of the table. These values are known as o/p depending upon the functionality of the program.

Decision table testing technique is used to design a complete set of test cases without using complete structure of the prg. Every column is associated with the test rule & generate a test case.

DO Not Care Condition - This cond^n is represented by the dash '-' sign. This cond^n has no effect on the o/p. If we donot do so & represent all T/F values, the no. of problems in decision table.

will be unnecessarily. This is nothing but a represent^n facility to reduce the no. of cols in the decision table. Ideally, each column has one rule that leads to a test case. A column in the entry portion of a table is known as rule. A term is used as 'rule count' with the don't care entries in the decision table & has a value 1. If don't care cond^n are not there, but it doubles for every don't care entry. Hence, rule count can be calculated as RuleCount = $2^{\text{no. of don't care cond}^n}$

Impossible conditions - Decision table are very popular for the generat^n of test cases. Sometimes, we may have to make a few attempts to reach the final sol^n. Some impossible cond^n are also generated due to the combination of various i/ps & an 'impossible' stub included into the action stub to show such cond^n.

Applicability - Decision tables are popular in circumstances where an o/p is depend upon the various i/p cond^n & a large no. of cond^n are required to be taken. They may also include complex business rules & use them to design the test cases.

Q. Consider the program for determining the largest amongst the 3 numbers such that $1 \leq x \leq 300$, $1 \leq y \leq 300$, $1 \leq z \leq 300$. Identify the test cases using the decision table based testing.

Sol^n → The test cases are-

$x > y$
$> z$

15 columns & 15 rows

Cond^n	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9	R_{10}	R_{11}	R_{12}	R_{13}	R_{14}	R_{15}	
$x \geq 1$	F	T	T	T	T	T	T	T	T	T	T	T	T	T		Cond^n Stub
$x \leq 300$	-	F	T	T	T	T	T	T	T	T	T	T	T	T		
$y \geq 1$	-	-	F	T	T	T	T	T	T	T	T	T	T	T		
$y \leq 300$	-	-	-	F	T	T	T	T	T	T	T	T	T	T		
$z \geq 1$	-	-	-	-	F	T	T	T	T	T	T	T	T	T		
$z \leq 300$	-	-	-	-	-	F	T	T	T	T	T	T	T	T		
$x > y$	-	-	-	-	-	-	T	F	T	F	T	T	F	F		
$y > z$	-	-	-	-	-	-	T	F	T	T	F	F	T	F		
$z > x$	-	-	-	-	-	-	T	F	F	T	T	F	F	T		
Rule count	256	128	64	32	16	8	1	1	1	1	1	1	1	1		
Invalid o/p	X	X	X	X	X	X										
x is largest									X			X				Action Stub
y is largest										X			X			
z is largest											X			X		
Impossible							X	X								

Q. Consider a program for the determ^{n} of division of a student based on the marks in 3 subjects. Its i/p is the triple of +ve integers say mark1, mark2, mark3 & the values are from interval [0-100]. The division is calculated according to the following rules avg.

Avg	division
0-39	fail
40-49	3rd div
50-59	2nd div
60-74	1st div
≥75	1st div with distinct^{n}

Design the test cases using the decision table based testing.

Sol^{n} →

Limited Entry Decision Table

Cond^n	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9	R_{10}	R_{11}	R_{12}	R_{13}	R_{14}	R_{15}	R_{16}	R_{17}	R_{18}	R_{19}	R_{20}	R_{21}
mark1 ≥ 0	F	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T
mark1 ≤ 100	-	F	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T
mark2 ≥ 0	-	-	F	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T
mark2 ≤ 100	-	-	-	F	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T
mark3 ≥ 0	-	-	-	-	F	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T
mark3 ≤ 100	-	-	-	-	-	F	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T
0-39	-	-	-	-	-	-	T	T	T	T	T	F	F	F	F	F	F	F	F	F	F
40-49	-	-	-	-	-	-	T	F	F	F	F	T	T	T	T	F	F	F	F	F	F
50-59	-	-	-	-	-	-	-	T	F	F	F	T	F	F	F	T	T	T	F	F	F
60-74	-	-	-	-	-	-	-	-	T	F	F	-	T	F	F	T	F	F	T	T	F
≥ 75	-	-	-	-	-	-	-	-	-	T	F	-	-	T	F	-	T	F	T	F	T
Rule count	2^{10}	2^9	2^8	2^7	2^6	2^5	2^3	2^2	2^1	1	1	2^2	2^1	1	1	2^1	1	1	1	1	1
Invalid i/p	X	X	X	X	X	X															
Fail											X										
3rd div															X						
2nd div																		X			
1st div																				X	
1st div with distin			✓																		✓
Impossible							X	X	X	X		X	X	X		X	X		X		

Extended Entry Decision Table – I_1 : {A_1 : $0 \le marks1 \le 100$}

I_2 : {A_2 : $marks1 < 0$} I_3 : {A_3 : $marks1 > 100$}

I_4 : {B_1 : $0 \le marks2 \le 100$} I_5 : {B_2 : $marks2 < 0$}

I_6 : {B_3 : $marks2 > 100$} I_7 : {C_1 : $0 \le marks3 \le 100$}

I_8 : {C_2 : $marks3 < 0$} I_9 : {C_3 : $marks3 > 100$}

I_{10} : {D_1 : $0 \le avg \le 39$} I_{11} : {D_2 : $40 \le avg \le 49$}

I_{12} : {D_3 : $50 \le avg \le 59$} I_{13} : {D_4 : $60 \le avg \le 74$}

I_{14} : {D_5 : $avg \ge 75$}

Entries	1	2	3	4	5	6	7	8	9	10	11
mark 1 in	A1	A1	A1	A1	A1	A1	A1	A1	A1	A2	A3
mark 2 in	B1	B1	B1	B1	B1	B1	B1	B2	B3	-	-
mark 3 in	C1	C1	C1	C1	C1	C2	C3	-	-	-	-
avg	D1	D2	D3	D4	D5	-	-	-	-	-	-
Rule count	1	1	1	1	1	5	5	15	15	45	45
I^st div method					X						
I				X							
II			X								
III		X									
IV	X										
Invalid						X	X	X	X	X	X

No. of equ. classes for a new elements cartesian product = rule count

19

Decision Count Graph Technique

Extended Entry Decision Table- I_1 : {A_1 : $0 \leq marks1 \leq 100$}

I_2 : {A_2 : $marks1 < 0$} I_3 : {A_3 : $marks1 > 100$}

I_4 : {B_1 : $0 \leq marks2 \leq 100$} I_5 : {B_2 : $marks < 0$}

I_6 : {B_3 : $marks2 > 100$} I_7 : {C_1 : $0 \leq marks \leq 100$}

I_8 : {C_2 : $marks < 0$} I_9 : {C_3 : $marks3 > 100$}

I_{10} : {D_1 : $0 \leq avg \leq 39$} I_{11} : {D_2 : $40 \leq avg \leq 49$}

I_{12} : {D_3 : $50 \leq avg \leq 59$} I_{13} : {D_4 : $60 \leq avg \leq 74$}

I_{14} : {D_5 : $avg \geq 75$}

Entries	1	2	3	4	5	6	7	8	9	10	11	
mark 1 in	A1	A1	A1	A1	A1	A1	A1	A1	A1	A2	A3	No. of equ. classes for a row elements cartesian product = rule count
mark 2 in	B1	B1	B1	B1	B1	B1	B1	B2	B3	-	-	
mark 3 in	C1	C1	C1	C1	C1	C2	C3	-	-	-	-	
avg	D1	D2	D3	D4	D5	-	-	-	-	-	-	
Rule count	1	1	1	1	1	5	5	15	15	45	45	
1st divi method					X							
I				X								
II			X									
III		X										
IV	X											
Invalid						X	X	X	X	X	X	

1/9

Cause-Effect Graph Technique

This technique is a popular technique for a small program & considers the combin^{n} of various inputs which are not available in the earlier techniques like BVA, integration, system & equv. class testing. 2 terms are used here causes & the effects which are nothing but i/p & the o/p respectively. The steps for the generation of test cases are as follows-

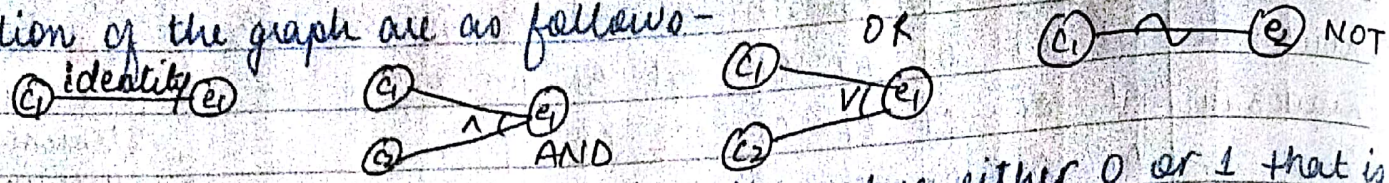
Identify the causes & effects → Design cause-effect graph → Apply constraints if any → Design limited decision table → Design test cases using rule of decision table

Identific^{n} of Causes & Effects-

The SRS document is used for the identific^{n} of causes & the effects which are inputs to a program & the o/p of a program respectively. The causes & the effects can be easily identified by reading the SRS document.

Design of Cause-Effect graph- The rel^{n}ship amongst the causes & the effects are established by using the cause-effect graph. The basic nota-

tion of the graph are as follows-



Assume that each node having the value either 0 or 1 that is the absent state or present state respectively.

1) The identity fnc state that if c_1 is 1, then e_1 is one else e_1 is 0.
2) The OR fnc state that if c_1 or c_2 is 1, then e_1 is 1 else e_1 is 0.
3) The AND fnc states that if both c_1 & c_2 are 1, then e_1 is 1 else e_1 is 0.
4) The NOT fnc states that if c_1 is 1, then e_1 is 0, else e_1 is 1.

Note- The AND, OR, NOT fnc are allowed to have any no. of inputs.

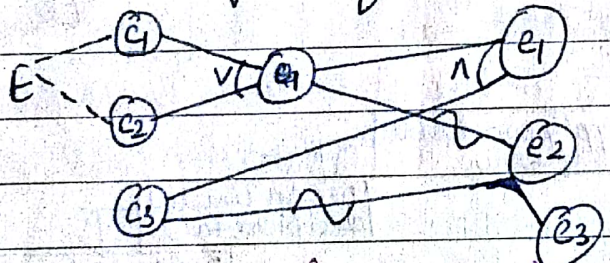
Q. The character in a column 1 must be 'A' or 'B'. The character in col 2 must be a digit. In this situation, the file is updated. If the col 1 is incorrect, then message 'X' is issued. If the character in col 2 is not a digit, then message 'Y' is issued. Draw cause-effect graph.

Soln- c_1: col 1 must have 'A'. c_2: COL 1 must have 'B'

c_3: col 2 must have digit

e_1: file update e_2: message 'X' if col 1 doesn't have A or B

e_3: message 'Y' if col 2 doesn't have digit.



c_3 goes to e_3 directly. No edge b/w e_2

Use of Constraints in Cause-Effect graph- There may be no. of inputs in any program. We may like to explore the rel^n ship amongst the causes & this process may lead to some impossible combin^n of causes. Such impossible combin^n of situation are represented by the constraints. Symbols are as followed-

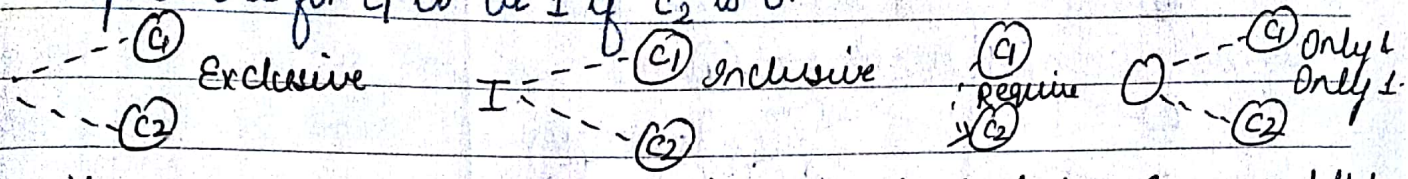
1) Exclusive (E)- This constraint state that atmost one of c_1 or c_2 may be 1 (c_1 or c_2 can't be 1 simultaneously). However both c_1 & c_2

can be 0.

2) Inclusion (I) - This constraint state that atleast one of c_1 or c_2 must always be 1. Hence, both can't be 0 simultaneously. However, both can be 1.

3) Only & Only 1 - It state that one & only one of c_1 & c_2 must be 1.

4) Require (R) - This constraint state that for c_1 will be 1, c_2 must be 1. It is impossible for c_1 to be 1 if c_2 is 0.



Q. Consider the eg. of keeping the record of marital status & no. of children. The value of marital status must be 'U' or 'M'. The value of no. of children must be digit or null in the case of citizen is unmarried. If the info^n entered by the user is correct, then an update is made. If the value of marital status of the citizen is incorrect then the error message one is issued. Similarly if the value of no. of children is incorrect, then errmessage 2 is issued. Draw the cause-effect graph.

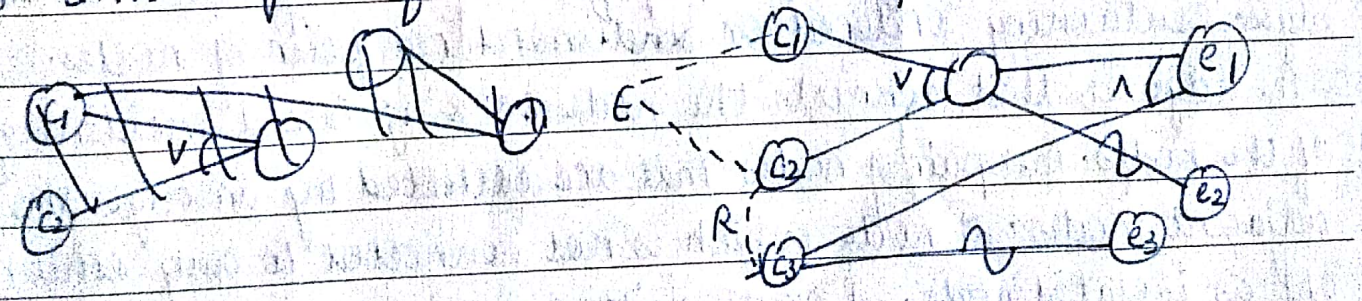
Sol^- c_1 : col1 must have 'U' c_2 : col1 must have 'M'

c_3 : col2 must have digit ~~& col2 must have null if col1 has 'U'~~.

e_1 : file update if info^n correct

e_2 : errmessage 1 if col1 has incorrect info^n.

e_3 : errmessage 2 if col 2 have incorrect info^n.



19

Q. A tourist of age greater than 21 year & having a clean driving record is supplied a rental car. An amount is also charged if the tourist is on business otherwise, it is not charged. If the tourist is less than 21 years old or does not have a ~~clean~~ ~~valid~~ record, the system will display ~~key~~ ~~display~~

clean drivij

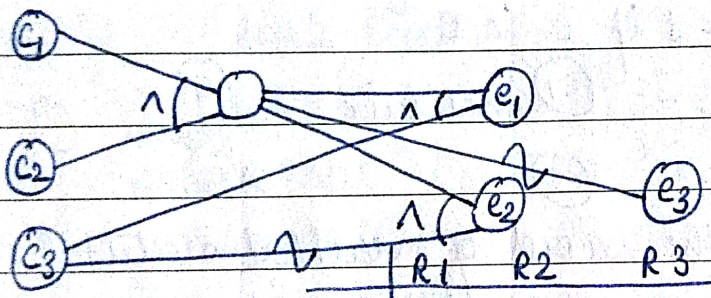
"Car can't be supplied"

the following message: Draw the cause-effect graph & generate the test cases.

Sol^n :→ C_1: Tourist is of age greater than 21 years. C_2: Tourist has driving record.

C_3: on business e_1: car supplied with amount.

e_2: car supplied with no amount e_3: car can't be supplied.



Decision table →

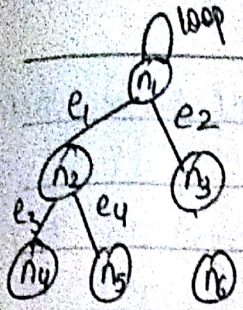
	R1	R2	R3	R4
C_1	F	T	T	T
C_2	–	F	T	T
C_3	–	–	F	T
Rule count	2^2	2^1	1	1
e_1				X
e_2				
e_3				

Structural Testing

Graph

A graph has a set of nodes & edges that connect these nodes. A graph $G = \{V, E\}$ consist of non-empty finite set V of nodes & set E of edges containing ordered or unordered pair of nodes.

The edge e_i that connects the node n_i & n_j is c/d incident of each of the nodes. The pair of nodes that are collected by an edge are c/d adjacent nodes. A node which is not connected to any other node is c/d as isolated node. A graph with only isolated node is c/d null graph. If in a graph $G = \{V, E\}$, each $e_i \in E$ is associated with an ordered pair of nodes, then graph G is c/d directed graph. If each edge is associated with an unordered pair of nodes, then graph G is c/d undirected graph.

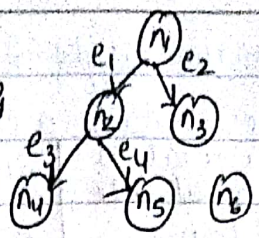


Undirected graph

$V = \{n_1, n_2, n_3, n_4, n_5, n_6\}$

$E = \{e_1, e_2, e_3, e_4\}$

$= \{(n_1, n_2), (n_1, n_3), (n_2, n_4), (n_2, n_5)\}$



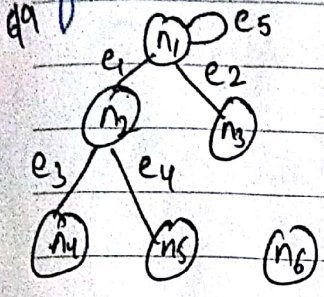
Directed graph

$V = \{n_1, n_2, n_3, n_4, n_5, n_6\}$

$E = \{e_1, e_2, e_3, e_4\}$

$= \{\langle n_1, n_2\rangle \langle n_1, n_3\rangle \langle n_2 n_4\rangle \langle n_2 n_5\rangle\}$

An edge of a graph having the same node at its end point is c/d a loop. The direction of the edge is not important because the initial & final nodes are one & the same.

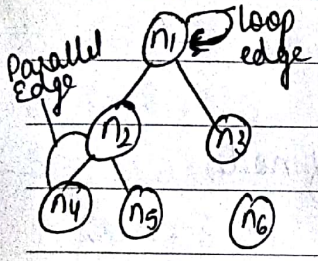


$V = \{n_1, n_2, n_3, n_4, n_5, n_6\}$

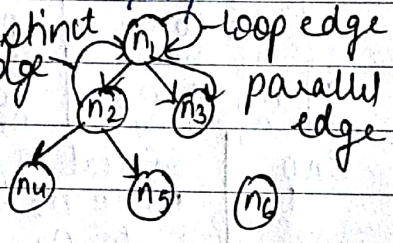
$E = \{e_1, e_2, e_3, e_4, e_5\}$

$= \{(n_1, n_2)\ (n_1, n_3)\ (n_2\ n_4)\ (n_2\ n_5)\}$

If a certain pair of a nodes are connected by more than one edge undirected & the undirected graph are c/d parallel edge.



Undirected Graph



Directed Graph

If a graph is directed graph & the parallel edges are in opp. direct", such edges are considered as distinct parallel edges.

A graph that has neither loops no || edges is c/d as simple graph.

A graph with one or more || edges is c/d as a multigraph.

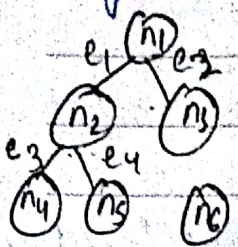
Degree (n) = No. of incoming + outgoing edge.

In undirected graph, loop contributes twice to a degree of that node & the degree of an isolated node is 0.

Path & Independant Paths

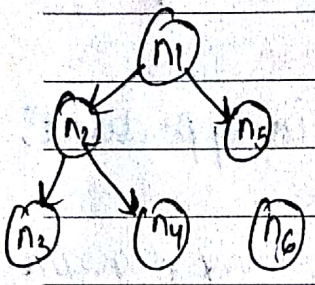
A path is a sequence of adjacent nodes where the nodes in a

sequence share common edges or a sequence of adjacent edges where edges in a sequence share a common node.



Undirected Graph

SNo	Initial to final node	Sequence of nodes	Sequence of edges
1	n_1 to n_4	$n_1 n_2 n_4$	$e_1 e_3$
2	n_1 to n_5	$n_1 n_2 n_5$	$e_1 e_4$
3	n_1 to n_3	$n_1 n_3$	e_2
4	n_1 to n_2	$n_1 n_2$	e_1
5	n_2 to n_4	$n_2 n_4$	e_3
6	n_2 to n_5	$n_2 n_5$	e_4
7	n_3 to n_1	$n_3 n_1$	e_2
8	n_2 to n_1	$n_2 n_1$	e_1
9	n_4 to n_2	$n_4 n_2$	e_3
10	n_5 to n_2	$n_5 n_2$	e_4
11	n_4 to n_1	$n_4 n_2 n_1$	$e_3 e_1$
12	n_5 to n_1	$n_5 n_2 n_1$	$e_4 e_1$
13	n_2 to n_3	$n_2 n_1 n_3$	$e_1 e_2$
14	n_3 to n_2	$n_3 n_1 n_2$	$e_2 e_1$
15	n_4 to n_3	$n_4 n_2 n_1 n_3$	$e_3 e_1 e_2$
16	n_3 to n_4	$n_3 n_1 n_2 n_4$	$e_2 e_1 e_3$
17	n_5 to n_3	$n_5 n_2 n_1 n_3$	$e_4 e_1 e_2$
18	n_3 to n_5	$n_3 n_1 n_2 n_5$	$e_2 e_1 e_4$
19	n_4 to n_5	$n_4 n_2 n_5$	$e_3 e_4$
20	n_5 to n_4	$n_5 n_2 n_4$	$e_4 e_3$



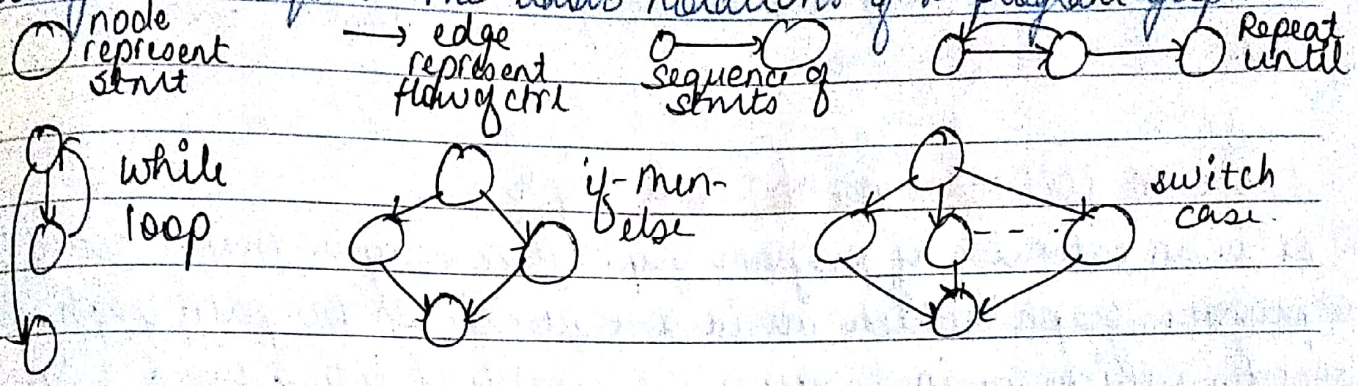
SNo	Initial to final node	Sequence of nodes
1	n_1 to n_2	$n_1 n_2$
2	n_1 to n_3	$n_1 n_2 n_3$
3	n_1 to n_4	$n_1 n_2 n_4$
4	n_1 to n_5	$n_1 n_5$
5	n_2 to n_3	$n_2 n_3$
6	n_2 to n_4	$n_2 n_4$

A Path represented by the sequence of nodes is more popular represent^n of sequence of edges. An independant path in a graph that has atleast one new node or one new edge in its sequence from initial to its final node.

Generation Of Graph From A Program

Graphs are extensively used in the testing of a computer program

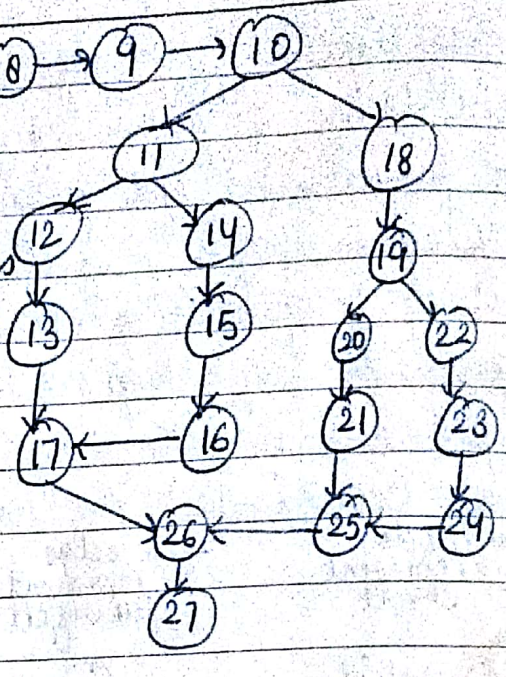
We may represent flow of data & flow of control of any program in the terms of directed graph. A graph representing the flow of control of the program is c/d program graph.
A program graph is a directed graph in which nodes are either stmts or fragments of stmts and edges represent the flow of control. Pm program graph help us to understand the internal structure of the program which may provide the basis for designing the testing techniques. The basic notations of a program graph are-



7/9
Q: Draw the program graph for the following program

```
1. void main() {
2. float A, B, C;
3. clrscr();
4. printf("A=");
5. scanf("%f",&A);
6. printf("B=");
7. scanf("%f",&B);
8. printf("C=");
9. scanf("%f",&C);
10. if(A>B) {
11. if(A>C)
12. printf("A is largest");
13. }
14. else {
15. printf("C is largest");
16. }
17.
18. else {
19. printf("C is largest"); if(C>B) {
20. printf("C is largest");
21. }
22. else {
23. printf("B is largest");
24. }
25. getch();
26. }
```

Soln- 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9 → 10



Node 2 - 9 are the sequence nodes.
Nodes 10, 11, 18, 19 are the decision nodes or the predicate nodes i.e. having 2 outgoing edges.
Loop no, 17, 25, 26 are the junction nodes, i.e., 2 incoming & 1 outgoing node.

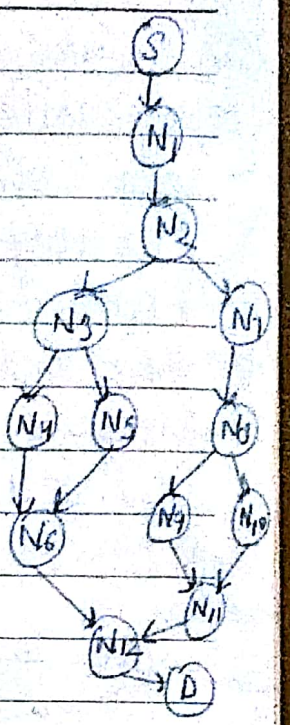
Decision to Decision (DD) Path Graph

It is an extension of program graph. There may be many nodes in a program graph which are in a sequence. In DD path graph, such nodes which are in sequence are combined into a block & are represented by a single node. Hence, the DD path graph is a directed graph in which nodes are sequences of stmts & the edges are control flow amongst the nodes. All the programs have an entry & an exit & the corresponding program graph has source node & the destination node.

We prepare a mapping table for the program graph & the DD graph nodes. A mapping table maps a node of a program graph to the corresponding nodes of the DD path graph. This may combine the sequential of nodes of a program graph into a block & represented by a single node in a DD path graph. This process may reduce the size of program graph & convert it into more meaningful DD path graph.

Program graph nodes	DD path graph nodes	Remark
1	S	Source node
2 to 9	N_1	Sequential node
10	N_2	Predicate node

11	N_3	Predicate node
12, 13	N_4	Sequential "
14, 15, 16	N_5	"
17	N_6	Junction node
18	N_7	Intermediate
~~1 7~~ 19	N_8	Predicate node
20, 21	N_9	Sequential node
22, 23, 24	N_{10}	"
25	N_{11}	Junction node
26	N_{12}	"
27	D	Destinat" node.



8/9

Identification Of Independant Path

An independant path should have atleast one new node or new edge which is to be traversed. We should identify every independant path of a program & pay special attention to these paths during testing. A few concepts are used in the testing techniques which may help us to identify the independant paths.

Cyclomatic Complexity - This concept involves using the cyclomatic no. of graph theory which has been re-defined as the cyclomatic complexity. This is nothing but the no. of independant path through a program. The cyclomatic complexity can be calculated by 3 methods -

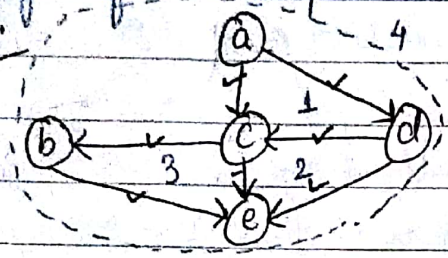
1) $V(G) = e - n + 2P$, where $V(G)$ is cyclomatic complexity, e is no. of edges in DD graph, n is no. of nodes in DD graph & P is no. of connected components. The program graph is directed graph with a single entry node & single exit node. A connected graph is a program graph where all the nodes are reachable from entry node & the exit node. Such program graphs will have connected component (P) value = 1. If there are the parts of the program graph, then value will be the no. of path to of a program graph, where one part may represent the main program & other part may represent the subprogram

36

2) Cyclomatic complexity = no. of regions of a program graph.
3) Cyclomatic complexity $V(G) = \pi + 1$, where π stands for the no. of predicate nodes in a program graph.

The only restrictⁿ is that the every predicate node should have 2 outgoing edges, i.e., one for true & other for the false condⁿ. If the predicate node contain more than 2 outgoing edges, the structure is required to be changed in order to have only 2 outgoing edges. If it is not possible then this method ($\pi+1$) is not applicable.

Q.



$S = \{a\}$, $D = \{e\}$

(i) $V(G) = 7 - 5 + 2(1) = 4$

(ii) $V(G) = 4$

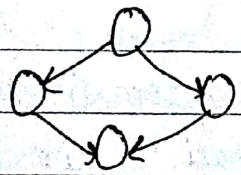
(iii) $V(G) = 3 + 1 = 4$

Nodes with 2 outgoing edges are c/d predicate nodes (a, c, d).

Paths - a, c, e / a, d, e / a, d, c, e / a, c, b, e / a, d, c, b, e

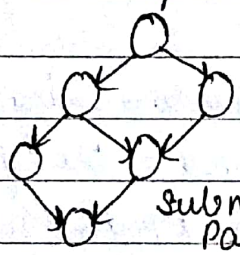
Independant paths are the ones with new node in each path. So, a, d, c, b, e is not an independant path.

Q.



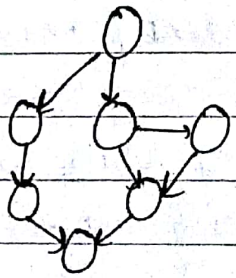
Main Module
Part (I)

$V(G) = 2$



Submodule I
Part (II)

$V(G) = 3$



Part 2

$V(G) = 3$

$V(G)$ overall $= 19 - 17 + 2 \times 3 = 2 + 6 = 8$

9/9

Structural Testing

It is considered more technical than the functional testing. It attempt to design the test cases from the source code not from the specific^. The source code becomes the base document which is examined thoroughly in order to understand the internal structure

of the source code.

Structural testing also c/d white box testing due to consideration of internal structure of a source code.

Control Flow Testing - This technique is very popular & effective. We identify the path of a program & write the test cases to execute each path. As we know that, path is a sequence of stmts that begins at an entry & exit at an end. There may be too many paths & it may not be feasible to execute all the path.

Every path covers a portion of a program. We define coverage as a % of a source code that has been tested w.r.t. the total source code available for the testing. The most reasonable level may be too test every stmt of a program atleast once before the completion of testing. If we do so, we have some satisfaction about the reasonable level coverage. Some of such techniques are discussed below which are the part of control flow testing-

1) Statement Coverage- We want to execute every stmt of a program in order to achieve 100% stmt coverage.

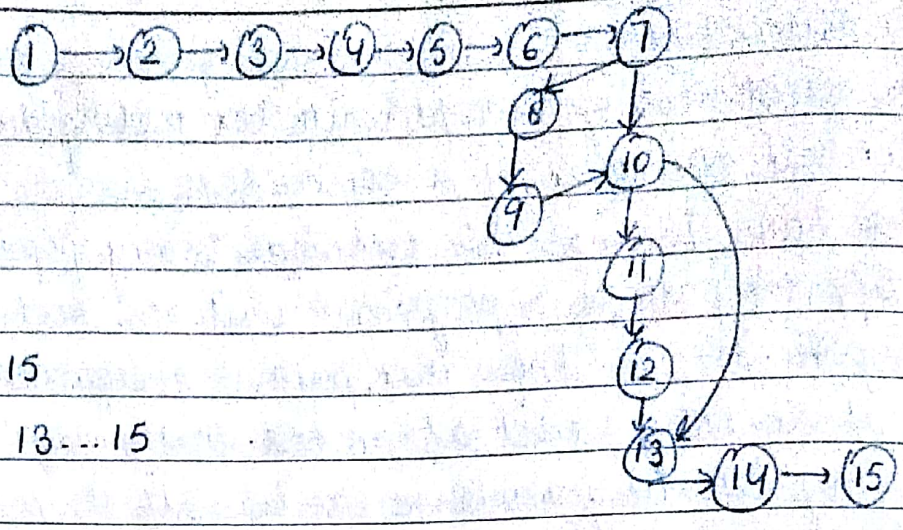
```
1. void main()
2. {
3.   int a, b, c, x=0, y=0;
4.   clrscr();
5.   printf
6.   scanf
7.   if((a>b)&&(a>c));
8.     x = a*b + b*b;
9.   }
10.  if (b>c) {
11.    y = a*a - b*b;
12.  }
```

When we give i/p as test case

~~1 2 3~~
9 8 7
5 4 3
} only these

full program executes in one time & follow a one path, i.e, 1....15.

13. printf
14. getch();
15. }



. V(G) = 3

Path

1) 1 --- 7, 10, 13 --- 15

2) 1 --- 7, 8 --- 10, 13 --- 15

3) 1 ---- 15

4) 1 --- 7, 10, 11 --- 15

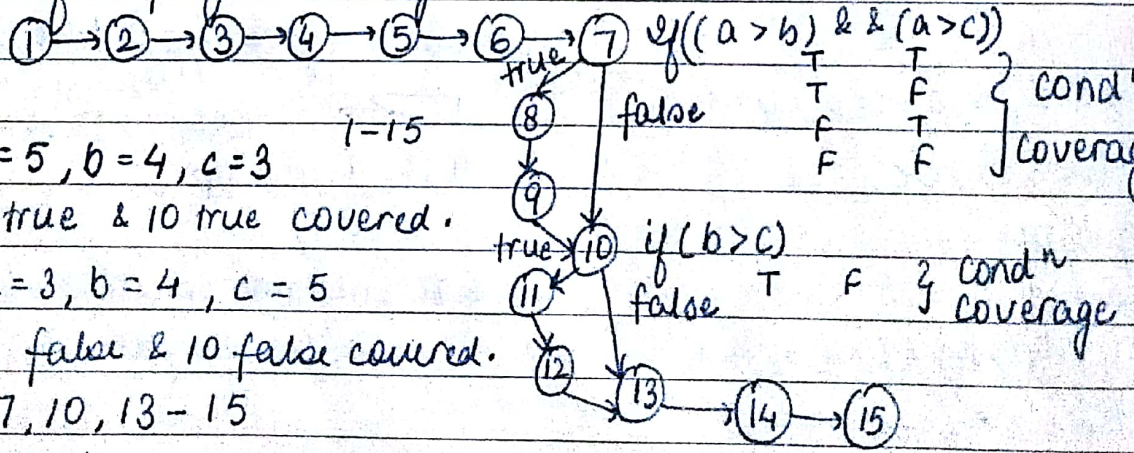
Independant paths are 1, 2 & 4.

Only one test case a=5, b=4, c=3 may cover all the stmts but will not execute all the possible 4 paths & not even cover all the independant path.

The objective of achievement of 100% stmt coverage is difficult in practice.

5/9/19

2) Branch Coverage - In the branch coverage we want to test every branch of the program. Hence, we wish to test every True & false cond^n of the program. For Eg.



1-15

1) a=5, b=4, c=3

7 true & 10 true covered.

2) a=3, b=4, c=5

7 false & 10 false covered.

1-7, 10, 13-15

Note - The branch coverage does not guarantee 100% path coverage but it does guarantee 100% stmt coverage.

3) Condition Coverage - Cond^n coverage is better than the branch coverage because we want to test every cond^n atleast one

However, branch coverage can be achieved without testing every cond^n.

Test cases - 1) a = 5, b = 4, c = 3 4) a = 3, b = 4, c = 5

2) a = 5, b = 4, c = 6

3) a = 5, b = 7, c = 4 // Not valid cond^n since F & T is not a possible cond^n.

4) Path Coverage - In this coverage criteria, we want to test every path of a program. There are too many paths in any program due to loops & the feedback connect^n. It may not be possible to achieve this goal of executing all the path in a program. If we do so, we may be confident about the correctness of the program. If it is unachievable, atleast all the independant path must be executed.

Test cases - ~~1) a = 3~~

S.No.	Path	a	b	c	O/p
1)	1-15	5	4	3	
2)	1-10, 13-15	5	3	4	
3)	1-7, 10-15	3	5	4	
4)	1-7, 10, 13-15	3	4	5	

A path testing guarantee stmt coverage, branch & cond^n coverage. However, there are many paths in any program & it may not be possible to execute all the path. We should do enough testing to achieve a reasonable level of coverage. We should execute atleast all independant path which are also referred to as a basic path to achieve the reasonable coverage.

16/9 # Data Flow Testing

In a control flow testing, we find the various path of a program & design the test cases to execute those paths.

Q. Consider the following program

```
void main()
{
int a, b, c;
```

The given program will give unpredictable & garbage value since value of b & c are not defined.

```
a = b + c;
printf("%d", a);
}
```

Data flow testing may help to minimize such mistakes. It is based on the variables, their usage & their def^n in a program. The main pts of concern are-

1) stmt where the variables receive a value (def^n).
2) stmt where these values are used (reference).

Data flow testing focus on the variable def^n & the variable usage. Hence, this technique concentrate on how the variable is defined & used at a different place of the program. Some of defined / reference anamalies are as follows-

1) A variable is defined but never used.
2) A variable is used but never defined.

Defined / reference anamalies may be identified by the static analysis of the program, i.e., analysing the program without executing. This technique is used to understand & define used cond^n of all variables.

Definition- A program is first converted into a program graph. As we all know, every stmt of a program is replaced by a node & flow of control by an edge to prepare program graph.

1) Defining Node- A node of a program graph is a defining node for a variable say 'u', iff the value of variable 'u' is defined in the stmt corresponding to that node. It is represented by DEF(u, n), where 'u' is the variable & 'n' is the node corresponding to that stmt in which 'u' is defined.
2) Usage Node- A node of a program graph is a used node for a variable 'u', iff the value of variable 'u' is defined used in the stmt corresponding to that node. It is represe-

nted by USE(u,n) where 'u' is the variable & 'n' is the node corresponding to the stmt where variable 'u' is used.

A use node is a predicate used node denoted as P_use() iff the stmt corresponding to node end is a predicate stmt.

3) Defination Use Path- A def^ use path for a variable 'u' is a path b/w the 2 nodes 'm' & 'n', where m is the initial node in a path but the defining node for a variable u & 'n' is the final node in the path but the used node of the variable 'u'. It is denoted as du-path(u, m), du-use path(u, n)

4) Definition Clear Path- It is denoted by 'dc-path' for a variable 'u' is a def^ used path with initial & final node such that no other node in the path defining the node of variable u.

Identific^ of DU & DC-Path- The various steps for the identific^ of du & dc-path are given as-

Step-1 Draw the program graph of the program.

Step-2 Find all the variables of a program & prepare a table for defined & the used statis of all variables using the following format - | S.NO | Variables | Define at node | Use at node |

Step-3 Venkate all the du-path from define & use value in a table using the following format- | S.NO | Variables | Du-paths begin, end |

Step-4 Identify those du-pam which are not dc-path.

Eg. from date 7/9/16, scanf is def^ stmt.

∴ def(a,5), def(b,7), def(c,9) Defin node.

∴ use(a,10), use(a,11), use(b,10), use(c,11) Use node.

2019 Testing strategies Using the DU Path- We want to generate the test cases which place every def^ to each of its use & every use is faced to each of its def^.

Some of the testing strategies are as follows-

1) Test of all DU path- All the DU path generated for all the variable

are tested. This is the strongest data flow testing strategy covering all the possible DU path.

2) Test all uses - Find atleast one path from every defⁿ of every variable to every use of that variable which can be reached by the defⁿ.

3) Test all the defⁿ - Find the path from every defⁿ of every variable to atleast one use of that variable.

We may choose any strategy for testing. As we go from test all the du path to test all the defⁿ, the no. of path are reduced. However, it is best to test all the du path & give the priority to those du path which are not defⁿ clear path.

21/9

22/9

Mutation Testing

The process of changing a program is known as mutation. This change may be limited to one, two or very few changes in the program. We prepare a copy of a program under test & make a change in a stmt of the program. This changed version of a program is known as mutant of the original program. The behavior of the mutant maybe different from the original program due to the introduction of change. However, the original program & the mutant are syntactically correct & should compile correctly. To mutate a program means to change a program. We generally make only one or 2 changes in order to assess the effectiveness of the test suite. We may make many mutants of a program by making small changes in the program. Every mutant will have different change in program.

The mutants generated by making only one change are known as first order mutant. We may obtain the second order mutant by making 2 simple changes in a program & so on. The second order mutant & above are called higher order mutants. Generally in prac-

Paperkraft

tice we prepare to use only first order mutant in order to simplify the process of mutation.

Mutation Operator - Mutants are produced by applying the mutant operator. An operator is essentially a grammatical rule that changes a single expression to another expression. The changed expression should be gramatically correct as per used language. We may be able to generate large set mutants. We should measure the degree to which the program is changed. If the original expression is $x+1$ & the mutant for the expression is $x+2$. That is considered as a lesser change as compared to the mutant where the expression is $y \neq 2$ by changing both operator & operands

Higher order mutants become difficult to manage, control & replace. They are not popular in practice & the first order mutants are recommended to use. Some of the mutant operator of OOL are C++ & Java are -

1) Changing the access modifier, i.e., public to private.
2) Static modifier change.
3) Super keyword change.
4) Operator Change
5) Any operand change by numeric value.

Mutation Score - Mutation Score = $\frac{\text{no. of killed mutant}}{\text{total no. of mutant}}$

$(0 < \text{mut}^n \text{ score} < 1)$

When we execute the mutant using the test suite, we may have any of the following outcomes -

1) The result of the program are affected by the change & any test case of the test suite detect it. If this happens, then mutant is c/d the killed mutant.

2) The result of the program aren't affected by the change & any test case of the test suite doesn't detect the mutation. This mutant is c/d the live mutant.

The mutat^ score measures how sensitive the program is to change & how accurate the test tool is. A mut^ score always lies b/w 0 & 1. A higher value of mut^ score indicates the effectiveness of a test suite.

The live mutants are important for us & should be analyzed thoroughly. The test case that identify the changed behavior of a live mutant should be preserved & transferred to the original test suite. Hence, the purpose of mutation testing is not only to access the capability of test suite but also enhand the effectiveness of the test suite.

Date - 30/9/2016

UNIT-3 REGRESSION TESTING

Develop^t of s/w may take up few years, but the same may have to be maintained for the several years. s/w needs changes, whatever well-written & design initially it may be. There are many reasons for some changes -

1) Some errors may have been discovered during the actual use of the s/w.
2) The user may have requested for additional functionality.
3) s/w may have to be modified due to change in external policies & principles.

This list is endless but the message is out & clear, i.e, "change is necessary". This changed s/w is required to be retested inorder to ensure that changes work correctly & these changes have not affected the other part of the s/w.

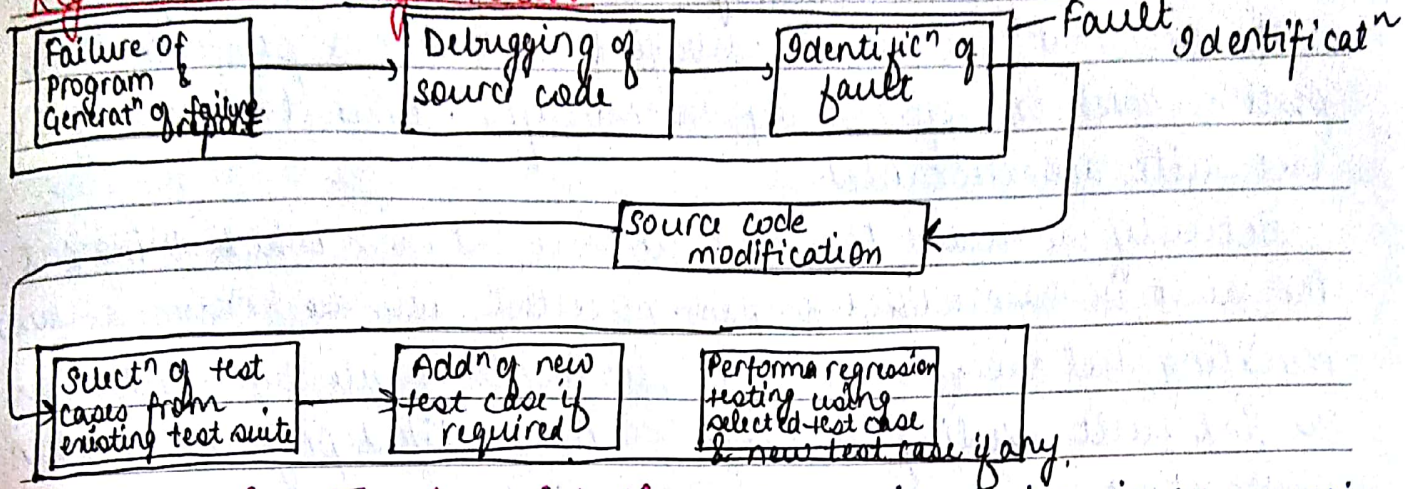
When we develop a s/w, we also develop^t testing to obtain the confidence in the correctness of the s/w. When we modify s/w, we typically retest it & also check changes does not affect the other part of the s/w. This process is c/d regression testing.

Hence the regression testing is the process of retesting the modified parts of the s/w & ensuring that no new error have been introduced into previously tested sourcecode due to these modific^n.

Paperkraft

Therefore, regression testing test both modify source code & other part of source code that may be affected by the changes.

Regression Testing Process -



4/10 1) **Regression Test Case Selection** - Test suite design is an expensive process & its size can grow quite large. Most of the times, running an entire testsuite is not possible as it require a significant amount of time to run all the test cases. Many techniques are available for the selection of test cases for the purpose of regression testing.

1) Select all the test cases - This is the simplest technique where we do not want to take any risk. We want to run all the test cases for ably change in a program. A program may fail many times & everytime you will execute the entire test suite. This technique is practical only when the test suite is small. For large size test suite, it becomes impractical to execute all the test cases.

2) Select test cases Randomly - We may select the test cases randomly to reduce the size of test suite. We decide how many test cases are required to be selected depending upon the time & available resources. When we decide a no., the same no. of test cases is selected randomly. If the no. is large, we may get good no. of testcases for execution & testing may be of some use. But if the no. is small, testing may not be useful at all. In this technique, one assump" is that all the test

cases are equally good in their fault detect^{n} ability. Many random selected test cases may not have any rel^{n}ship with the modified portion of the program.

3) Select Modific^{n} traversing test case - We select only those test cases that execute the modified portion of a program & the portion which is affected by the modific^{n}. Other testcases of the test suite are discarded.

Actually we want to select all those test cases which bring out the fault in the modified program. These test cases are known as fault revealing test cases. There is no effective technique by which you can find fault revealing testcases for the modified program. Another lower objective may be to select those test cases that bring out the difference b/w o/p of original & modified program. These test cases are known as modific^{n} revealing test cases. Unfortunately, we do not have any effective technique to do this. The reasonable objective is to select those testcases that traverse the modified source code & the source code affected by the modific^{n}. These testcases are known as modific^{n} traversing test cases.

Test case reduction - Test case reduction is an essential

Reducing the no. of test cases - Test case reduction is an essential activity & we may select those test cases that execute the modific^n & the portion of a program affected by the modific^n. We may minimize the test suite or prioritize the test suite in order to execute selected no. of test cases.

1) Minimiz^n of Test Cases - We select all those test cases that traverse the modified portion of the program & the portion that is affected by the modific^n. If we find the selected no. very large, we may still reduce it using any test case minimiz^n technique. These test case minimiz^n technique attempt to find the duplicate test cases. A duplicate test case is one which achieve an objective which

has already been achieved by another test case. The objective may be source code, branch, reqt coverage, etc

5/10
2) Prioritiz^n of Test Cases - We may indicate the order with which a test case may be exist. This process is known as prioritiz^n of test cases. A test case with the highest rank has the highest priority & the test case with the second highest rank has the Ω highest priority & so on. This method does not start any test case. The efficiency of regression testing depends upon the priority of the test cases.

Prioritiz^n guideline should address 2 fundamental issues-

1) What fn^c of the s/w must be tested - Identify the essential features that must be tested in any case.

2) What are the consequences if some fn^c are not tested - Identify the risk or the consequences of not testing some of the features.

Every reduc^n activity has an associated risk. All the prioritiz^n guideline should be designed on the basis of risk analysis. All risky fn^c should be tested on higher priority. The risk analysis may be based on the complexity, impact of area, etc.

The simplest priority category scheme is to assign a priority code to every test case. The priority code may be based on the assump^n that 'test case of priority code 1' is higher than the 'test case of priority code 2' we may have the priority codes as follows-

Priority Code 1 : Essential test case
Priority Code 2 : Important test case
Priority Code 3 : Execute, if time permit
Priority Code 4 : Not important test case
Priority Code 5 : Duplicate test cases.

There may be other ways for assigning the priorities based on customer reqt or market cond^n like

Priority Code 1 : Important for customer
Priority Code 2 : Required to ↑se customer satisfaction
Priority Code 3 : Help to ↑se markets share based on product.

Risk Analysis -

Risk is a problem that may cause some loss on the success of the project but which has not been happened yet. Risk is defined as the probability of occurence of an undesirable event & its impact of occurence on that event. Risk may delay & overbudget the project. Risky projects may also not be specified quality levels. Hence, there are the 2 things associated with the risk are as follows-

1) Probability of occurence of problem i.e., an event
2) Impact of that event on the problem.

Risk analysis is a process of identifying the problems & then assigning the probability of occurence of problem value & the impact of that problem value for each identified problem. Both of these values are assigned on a scale 1 to 10, i.e., low to high. A factor risk exposure is calculated for every problem which is the product of probability of occurence of that event & the impact of that event. The risk may be ranked on the basis of its risk exposure. These values may be calculated on the basis of historical data, past experience, intuitions & criticality of the problem. We should not confuse with the mathematical scale of probability values, i.e., 0-1. Here the scale of 1-10 is used for assigning the values to both the components of risk exposure. The risk analysis table is as follows-

S.NO.	Problem or Event	Probability of occurence of event (I)	Impact factor of event (II)	Risk exposure = I × II

UNIT-4 S/W TESTING ACTIVITIES & AUTOMATED TEST DATA GENERN

S/w Testing activities

We start the testing activities from the first phase of SDLC. We may generate the test cases from the SRS & the s/w DD & use them during the system & the acceptness testing. Hence develop & the testing activities are carried out simultaneously in order to produce good quality maintainable s/w within the time & budget.

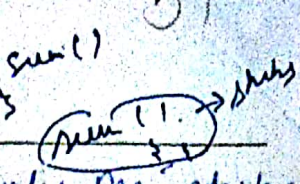
Levels Of Testing - s/w Testing is generally carried out at different levels. There are 4 such levels namely, unit, integration, system & the acceptness testing. Each level has specific testing objective. For eg, at unit testing level, independant units are tested using the structural or functional testing technique. At the integrat^n testing level, 2 or more units are combined together & testing is carried out to test the integrat^n testing issues. At the system testing level, the system is tested as a whole & the functional testing techniques are used to test the system. Non functional reqts. like performance, reliability are also tested at this level. The last level, i.e., accptness testing is done by the customer or user for the purpose of accepting a final product.

Unit Testing - We develop the s/w main parts or units & every unit is expected to have a defined functionality. We may call it a component, module, fn^, etc, which will have a purpose & may be developed independantly & simultaneously. A unit is a smallest, testable piece of a s/w which may consist of 100s or even just few lines of source code & generally represent the result of work of 1 or 2 developer. The unit testing's main purpose is to ensure that the unit satisfy its functional specific^. A unit may not be completely independent. It may be calling a few units & also be c/d by one or more units. The additional source code is added to that unit which handle the activities of called unit is c/d driver

& the additional source code which handle the activities of calling unit is c/d stub. The complete additional source code which is written for design of stub & a driver is c/d scaffolding. Many WB testing techniques may be effectively applicable at the unit level. We should keep stubs & the driver simple & small in size to reduce the cost of testing. We can only minimize the regt. of stubs & the driver depending upon the functionality & its division in various groups.

Integration Testing - A s/w program may have many units. We test the units independently during the unit testing after writing the required stubs & drivers. When we combine the 2 units, we may like to test the interface b/w the units. We combine 2 or more units because they share some rel^ship. This rel^ship is represented by interface & is known as coupling. The coupling is a measure of degree of interpedence b/w these units. Highly coupled units are more dependent to each other compared to the loosely coupled units. Coupling ↑ses the no. of calls among units or the amt of shared data ↑ses. A design with high coupling may have more errors.

When we design the test cases or the interfaces, we should be very clear about the coupling among the units & if it is high, a large no of test cases should be designed to test that particular interface. In integrat^ testing, we focus on the issues related to the interfaces among the units. There are several integrat^ strategies - 1) Top-Down - The top-down integrat^ start from the main unit & keeps on adding all sdd units on the next level. This portion should be tested thoroughly by focusing on interface issues. After complet^ of integrat^ testing at this level, next level unit is added & so on till we reach the



lowest level unit. There will be no requ. of driver only the stubs will be designed.

2) Bottom-up - In this integrat^n, we start from the bottom & keep on adding upper level units till we reach the top. There will be no need of stubs.

3) Sandwiched Integrat^n - This strategy start from the top & bottom concurrently, depending upon the availability of units & may meet somewhere in the middle. In practice, sandwiched integrat^n is more popular. This can be started as at when 2 related units are available. We may use any structural or functional testing technique to design the test case.

7/10 System Testing - We perform the system testing after the complet^n of unit & integrat^n testing. We generally use functional testing technique although few structural testing technique may also be used.

System testing ensure that each system fn^c works as expected & it also test for the non-functional requ like performance, security, stress, load, etc. This is the only phase of testing with functional & non-functional requ. of the system. In this testing, we also review all the associated documents & the manuals of the s/w. This verific^n activity is equally important & may improve the quality of final product.

Most care should be taken for the defects found during the system testing phase. A proper impact analysis should be done before fixing the defect. Sometimes, if the system permits, instead of fixing the defect they are just documented & mentioned as known limit^n. This may happen in a situation when fixing is very time consuming or technically it is not possible in the present design. Progress of a system testing also build confidence in the

developt team as this is the first phase in which the complete product is tested with the specific focus on the customer reqt.

Acceptness Testing - This is the extension of system testing. When the testing team feels that the product is ready for the customer, they invite the customer for demo". After demo" of the perbduct, customer may like to use the product to measure their satisfaction & the confidence. This may range from Adhoc used to the systematic well planned use of the product. This type of use is essential before accepting the final product. ~~This may be~~ the testing done for the purpose of accepting a product is known as acceptness testing this may be carried out by the customer or the person authorized by the customer.

13/8

Debugging - whenever a s/w fails, we would like to understand the reason for such a failure. After knowing the reason, we may attempt to find the soln & may make necessary changes in the source code accordingly. These changes will hopefully remove the reasons for that s/w failure. The process of identifying & correcting a s/w error as known as debugging. It starts after receiving a failure report & complete after ensuring that all the correct" have been rightly placed & the s/w does not fail with the same set of i/p. The debugging is quite difficult phase & may become one of the reason for the s/w delay.

Why debugging is so difficult? - Debugging is a difficult process. This is probably due to the human involvement & their psychology. Developers become uncomfortable after receiving any

request of debugging. It is taken against the professional pride. "It is the one of the most frustrating part of programming. It has the element of problem solving coupled with annoying recognition that we have made a mistake. The unwillingness to accept the possibility of errors for the task difficulty. Fortunately, there is a sigh of relief & lessening of the tension whenever it is soon ultimately corrected."

The debugging process attempt to match the symptom with the cause, thereby leading to error correct^n. The symptom & the cause may be geographically remote i.e. the symptom may occur in one part of a program while the cause may actually be located in the other part. Highly coupled program structure may further complicate the situation. In some cases, symptoms may be due to the causes that are distributed across the no. of tasks running in different processes.

There may be many reasons which may make the debugging process difficult & time consuming. However, psycological reasons are more prevailant than the mechanical reasons.

Debugging Process- Debugging means detecting & removing the bugs from the program. Whenever, program generate an unexpected behavior, it is known as failure of program. This failure may be annoying, disturbing, serious & extreme. Depending on the type of failure act^n are required to be taken. The debugging process starts after receiving the failure report from the testing team or from the user. The steps of debugging process are-

1) Replicat^n of bug 2) Understanding a bug 3) Locating a bug
4) Fixing a bug & retesting.

14/8 1) Replicat^n of bug - The I step in fixing a bug is to replicate it.

this means to create undesired behavior under the controlled cond^{n}. the same set of i/p should be given under the similar cond^{n} to the program & the program after the execut^{n}, should produce a similar unexpected behavior. If this happens, we are able to replicate a bug.

In other cases, replicat^{n} may be very difficult & sometimes may be nearly impossible. Some of the reasons for non-replication of a bug are -

1) the user incorrectly reported the problem.

2) the program has failed due to h/w problems like poor n/w connectivity, deadlock cond^{n}, etc.

3) the program has failed due to system s/w problems. The reason may be the use of different type of OS, compiler, device driver, etc.

Our effort should be to replicate a bug. If we can't do so, it it advisable to keep the matter pending till we are able to replicate it.

2) Understanding the bug - After replicating the bug, we may like to understand the bug. This means we want to find the reasons for this failure. There may be one or more reasons & is generally the most time consuming activity. We should understand the program very clearly for understanding the bug. If we are the designer & the source code writer, there may not be any problem for understanding the bug. If not then we may have serious problem. If the readability of the program is good & the associated documents are available, we may be able to manage the problem. If readability is not good & associated documents are not proper & completed then the situation becomes very difficult & complex.

We may have to put effort in order to understand

the program. We may start from the first stmt of the source code to the last stmt with the special focus on the critical & the complex area of source code. The main pt. is that in order to understand a bug, program understanding is essential.

3) Locate the bug - There are the 2 portions of the source code which need to be considered for locating the bug. The first portion of the source code is one which causes the visible incorrect behavior & the II portion of a source code is one which is actually incorrect. In the most of the situation, both portions may overlap & sometimes both portions may be in different parts of a program. We should I find the source code which causes the incorrect behavior. After knowing the incorrect behavior & its related portion of the sourcecode, we may find portion of source code which is at fault.

The most useful & the powerful way is to inspect the source code. This may help us to understand the program, understand the bug & finally locate the bug.

Sometimes the bug may not be in the program at all. It may be in the library routine or in the OS or in the compiler. These cases are very rare but there are the chances & if everything fails, we may have to look for such options.

4) Fix the bug & retest the program - After locating the bug, we may like to fix the bug. The fixing of a bug is the programming exercise rather than debugging activities. After making necessary changes in the source code, we may have to retest the sourcecode in order to ensure that the correction have been rightly done at the right place. Every change may affect other portion of the sourcecode too. Hence an impact analysis is required to identify the affected portion & that portion should also be

19/80 retested thoroughly.

Debugging approaches – There are many popular debugging approaches, but the success of any approach is dependent upon the understanding of the program. If the person involved in the debugging understand the program correctly, they may be able to detect & reshape the bug. The following approaches are - trial

1) Trial & Error Method- This approach is dependent on the intuitability & the experience of the debugging person. After getting the failure report, it is analyzed & the program is inspected. Based on the experience & the intelligence & also using the hit & trial technique, the bug is located & the sol^n is found. This is a slow approach & becomes impractical in a large programs.

2) Backtracking- This can be used successfully in a small programs. We start at the pt where the program gets an incorrect result such as an unexpected o/p is predicted. After analyzing the o/p, we trace backward the source code ~~m/m~~ manually until a cause of failure is found. The source code, from the stmt where the symptoms of failure is found, to the stmt where the cause of failure is found is analyzed properly. This technique brackets the loc^ of the bug in a program.

Another obvious variat^n of backtracking is forward backtracking, where we use the print stmts & analyze the intermediate results to determine at what pt the result first become wrong. These approaches may be useful only when the size of program is small. As the program size ↑es, it becomes difficult to manage these approaches.

3) Brute-force Approach- This method is most common & least efficient for isolating the cause of s/w error. We apply this method when all else fail. In this method, a printout of all registers & relevant m/m loc^ is obtained & analyzed. All dumps m/m loc^ should be well documented & retained for possible use on the subsequent problems.

21/10.

Automated Test Data Generation

Test Data - This is the data that is used in the test of the s/w system. In order to test a s/w appl^n, you need to enter some data for testing most of the features of the s/w. Any such identified data which is used in the test is known as test data.

You can have test data in Excel sheet which can be entered manually while executing the test cases or it can be read automatically by the automat^n tools.

Some test data is used to confirm the expected result, i.e., when test data is entered, the expected result should come & some test data is used to verify the s/w behaviour to invalid i/p data.

Test data is generated by the test term or by the automat^n tools with support testing.

Generating the test data requires a proper understanding of SRS, SDD & the source code of the s/w. Automated test data generation is an activity that generates the test data automatically for the s/w under test. The quality & the effectiveness of the testing is heavily dependent upon the test data generation.

"The assurance of s/w reliability partially depends upon the testing. However, it is interesting to note that testing itself also needs to be reliable. Automating testing process is the sound engg. approach which can make a testing efficient, cost effective & the reliable."

The simplest way is to generate the test data randomly, meaning, without considering any internal structure or the functionality of the s/w. However, this way may not be proper to generate the test data automatically.

Test Adequacy Criteria - Some of the way to define the test adequacy criteria are given as-

1) Every stmt of the source code should be executed atleast once.
2) Every branch of the source code should be executed atleast once.
3) Every condⁿ should be tested atleast once.
4) Every path of the source code should be executed atleast once.
5) Every independent path of the source code should be executed atleast once.
6) Every stated reqt should be tested atleast once.
7) Every defⁿ-use path & the defⁿ-clear path should be executed atleast once.

Effectiveness of the testing is dependent on the defⁿ of test adequacy criteria because it sets std. to measure the thoroughness of the testing. Many times, it may be difficult to generate large no. of test data manually to achieve the criteria & the automatic test data generatⁿ procds may be used to satisfy the defined criteria.

Static & Dynamic Test Data Generation - Test data can be generated either by statically evaluating the program or by the actual exⁿ of the program. The technique which are based on the static evaluatⁿ are c/d static test data generatⁿ technique. Static test data generatⁿ technique donot require the exⁿ of the program. They generally use the symbolic execut to identify the constraints on the input variables for the particular test adequacy criteria. The program is examined thoroughly & its paths are traversed without the exⁿ of the program.

The techniques which are based on the actual exⁿ of a program for the generatⁿ of test data are c/d dynamic test data generatⁿ techniq. Test data is generated during the exⁿ of a program. If during the exⁿ, a desired path is not executed, the program is traced back to find the stmt which has diverted the desired flow of a program.

28/10

Approaches To Test Data Generation - The approaches to the test data generation can be divided into 2 categories, i.e., static & the dynamic test data generation. One needs execut^n of a program (dynamic) & other does not need the ex^n of the program (static). We may automate any functional testing technique or the structural testing technique for the generat^n of test data. The program will execute automatically & the test data will be generated on the basis of selected technique. The program ex^n will continue till the desired test adequacy criteria is achieved.

1) Random Testing - It generates test data randomly & execute the s/w using that data as i/p. The o/p of s/w is compared with the expected o/p based on the inputs generated using the random testing. It is the simplest & the easiest way to generate the test data. For a complex test adequacy criteria, it may not be a proper technique because it does not consider any internal structure & the functionality of the source code - Random testing is not expensive & needs only a random no. generator along with the s/w to make it functional. The disadvantage of this technique is that it may not even generate test data that execute every stmt of the source code., for any reasonable size s/w, it may be difficult to attempt 100% stmt coverage which is one of the easiest test adequacy criteria. It is the fast technique but does not perform well as it merely generate the test data based on the probability & has a low chances of finding small bugs. However, in the absence of any other technique, it is the only other technique which is commonly used in a practice for the test data generat^n.

2) Symbolic Execution - It is a s/w testing technique, i.e., useful to generate test data & improving the program quality.

Symbolic execut^n isn't the execut^n of a program in its true sense but that rather the process assigning the expressions to the program variables

as a path is followed through the code structure. Steps to use the symbolic execut^n -

1) The execut^n requires a select^n of a path that are exercised by the set of data values. A program which is executed using the actual data results in the o/p of a series of values.
2) In the symbolic execut^n, the data is replaced by the symbolic values with the set of expressions, one expression per o/p variable.
3) The common approach for the symbolic execut^n is to perform & analysis of a program resulting in the creation of a flow graph.
4) The flow graph identify the decision points & the assignments associated with each flow. By traversing the flow graph from an entry point, a list of assignment stmts & the branch predicate is produced.

3) Dynamic Test Data Generation - This technique requires the actual execut^n of a program with some selected i/ps. The values of a variable are known during the execut^n of a program. We also determine the program flow with such selected i/ps. If the desired program flow or a path is not executed, we may carefully examine the source code & identify the node from where the flow took the wrong direction. We may use the different types of search methods to alter the flow by changing the inputs till the desired path is achieved. When we change a flow at a particular node, some other flows at different nodes may also change accidently.

3/11

Test Data Generation Tools - The use of a tool for test data is still in its earliest ~~stag~~ state. Continued in next line

although some s/w industries have been using their own ~~auto~~ tools for generation of test data. The o/p of such tools is a set of test data, which include the sequence of inputs to the system under the test. Some tools

are also available that accept manually created, automatically generated, pre-defined test sequences & execute the sequences without the human intervent^{n} & supervision. A few egs. of such tools are- winrunner, Loadrunner, rotational Robot, etc. The purpose of these tools is to execute already generated test cases & not to generate the test data automatically. The automated test data generat^{n} tools are different & designed for specific purpose of test data generat^{n}. Some of the popular test data generat^{n} tools are-

S.No.	Tool Name	Language/platform	Description
1.	T-VEC	Java/C++	Test cases are generated by applying the branch coverage.
2.	GS-Data Generator	ERP/CRM & datawarehouse	Test cases are generated for the acceptness testing.
3-	CONFORMIQ	C, C++, VB, Java	Test cases are generated for the functional testing using BB technique.
4.	CADATAMACS Test Generator	Mainframe	Used for testing the mainframe programs.

Software Testing Tools- Here are the 3 main categories of s/w testing tools which are as follows-

1) Static S/W Testing Tool- These are those that perform the analysis of a program w/o executing them at all. They may also find the source code which will be hard to test & maintain. Some of the static s/w testing tools are-

i) Complexity Analysis Tool- Complexity of a program plays a very important role while determining its quality. This gives us the idea about the no. of independent path in a program & is dependent upon the no. of decisions in the program. A higher value of

cyclomatic complexity may indicate the poor design & the risky implementⁿ. There are other complexity measure, also which are used in practical life are Halstead s/w size measure. These tools may take a program as I/p & process it & produce the complexity value as O/p. This value may be a indicator of quality of design & implementⁿ.

ii) Syntax & Semantic Analysis tool- These tools find the syntax & semantic error. Although the compiler may detect all the syntax error during the compilation, early detectⁿ of such errors may help to minimize the associated errors. semantic errors are very significant & compiler may help us in finding such errors. There are the tools in the market that may analyze the program & find such errors. Non-declarⁿ of variable, double declarⁿ of variable, divide by unspecified I/ps are some of the issues which may be detected by semantic analysis tool. These tools are the platform are the platform dependent & may parse the source code.

iii) Flow Graph Generator Tool- These tools are language dependent & take a program as I/p & convert it to its flowgraph. The flowgraph may be used for many purposes like complexity calculatⁿ, path identificⁿ, generatⁿ of dyⁿ- used path, etc. These tools help us to understand the risky & the poor design areas of a source code.

iv) Code Inspector- Source code inspector do the simple job of enforcing stds. in a uniform way. They inspect the program & force to implement the guideline of good programming practices. Although they are the language dependent, most of the guideline of good programming practices are similar in many languages. These tools are simple & may find the critical & weak areas of the program. They also suggest possible changes in the source code for the improvement.

4/11 2) Dynamic S/W Testing Tools- Dynamic s/w testing tools select the test cases & execute the program to get the results. They also analyze the results

& find the cause of failure of the program. They will be used after the implementⁿ of the program & may also test the non-functional requirements like efficiency, performance, etc. The following tools are involved in the dynamic s/w testing -

i) Coverage Analysis Tool- These tools are used to find the level of coverage of a program after executing the selected test cases. They give us an idea about the effectiveness of the selected test cases. They highlight the unexecuted portion of a code & force us to design special test cases for that portion of source code. There are many levels of coverage like stmt coverage, branch coverage, condⁿ coverage, path coverage & the independent path coverage, etc. We may like to ensure that every stmt coverage must be executed once. This minimum level of coverage may be shown by the tool after executing the set of test cases.

ii) Performance Testing Tool- We may like to test the performance of a s/w under the stress or load. A tool may help us to stimulate the situations & test these situatⁿ in various stress condⁿ. These tools simulate the multiple user on a single computer. We may also see the response time for a database when 10 users access the database, when 100 users access the database simultaneously. Performance testing include the load & the stress testing. Some popular tools are Load Runner, Apache Jmeta, rational performance tester, etc.

64

S/w Test Plan - It is a document to specify the systematic approach to testing activities of the s/w. If we carry out testing as a well-designed systematic test plan document, the effectiveness of a testing will improve & that may further help to produce a good quality product.

This document addresses the scope, schedule, milestone & the purpose of various testing activity. It also specify the items & the features to be tested & which are not to be tested.

A test plan document is prepared after the completⁿ of the SRS document & may be modified along with the progress of a project. We should clearly specify the test coverage criteria & the testing techniq to achieve this criteria. We should also specify who will perform testi at what level & when. Roles & the responsibilities of the tester must be clearly documented.

UNIT-5 OBJECT ORIENTED & WEB APPLICATION TESTING

Web Applications Testing - The quality of web appln must be assured in terms of response time, easy to use, no. of users, ability to handle a traffic, provide accurate infon, etc. Testing these web pages are real challenge because the conventional testing technique may not be directly applicable. The main challenge of testing a web appln isn't only to find the common s/w errors but also to test the associated quality related risk that are specific to the web appln. We should know the architecture & the key areas of a web applⁿ to effectively plan & execute the testing.

Web Application Vs Client-Server Architecture - In the client-server architecture, client program is installed on each client m/c that provide the user interface. The clients are connected with the server m/c by providing the requested infon. In the client-server architecture, the functionality of an applⁿ is distributed b/w the client & the server. For eg, business logic may reside on the server m/c, user interface may reside on the client m/c & the database may reside either on the client or the server m/c. Business Logic is the procedure that are to be followed by an appln based on the inputs given by the user. This architecture is known as 2-tier client-server architecture. Any upgrade at the server side would require upgrading of the client s/w that has been installed on the client m/c.

The web-browser is a s/w program that retrieve & present the infoⁿ to the client in the desirable format. A web-applⁿ includes a 3-tier architecture. It includes client m/cs, web server that has the business logic & the database server for handling the data storage. In 3-tier architecture, the applⁿ are divided into the 3 seperate layer, hence, changes in one-tier donot have any affect on other layer. The advantage of 3-tier architecture inclu-

does less disk space at the client m/c.

Key Areas in Testing Web Application - Web appl^n are difficult & complex as compared to the traditional client-server architecture. They are required to be tested on different browsers. It is important & critical to identify the areas that need special focus while testing a web-appl^n. A web appl^n needs to be tested for-

1) Functionality 2) Usability 3) Security
4) Performance (load & stress) 5) Database Testing

Functional Testing - It involves checking of the specified functionality of a web appl^n. Functional test cases for the web appl^n may be generated using any blackbox testing technique. Eg.

test id	Descript^n of f^n/c	i/p	Expected o/p

12/11

UI Testing - It test the user interact^n features work correctly. These feature include the hyperlinks, tables, frames, forms & the UI items like list, radio buttons, checkboxes, dialog boxes, etc.

UI testing ensures that the appl^n handles mouse & the keyboard events correctly & displays the hyperlinks, tables, frames, buttons, menus & error message boxes.

Types of UI Testing - 1) **Navigation Testing** - It investigate the functioning of all internal & the external links. This must ensure that the website provide the consistent, well-organized links & should also provide the alternative navigation schemes such as search option. The placement of the navigation link on each page must be checked. Search based navigation facility must also be thoroughly tested & search items should be consistent across one page to another.

2) **Form-Based Testing** - Website that includes the forms need to ensure

that all the fields in the form are working properly. Form based testing involve the following issues-

i) Proper navigation from one field to another field by using the tab key.
ii) Ensures that the data entered in the form is in valid format.
iii) Check that all the necessary fields are entered in the form.

Q.

About	Product	Contact Us	Registration	Order	Search

Navigation testing in the above web page must ensure that About, Product, Contact Us, Registration & Order, on click, navigate to their respective pages directly. Search, on click must give the images related to the phrase given.

Usability Testing - Usability is one of the quality attribute that the web appln must possess. It is important to develop a website that is easy to use. Usability is concerned with the degree to which the s/w fulfill the user specifn & the expectn. It is the measurement of the amount of satisfaction of the user. It also specify the user find the s/w easy to use & understand. Usability can be divided into one or more attributes -

1) Accuracy - It specify the degree to which the website meet the user needs with the precision & the correctness.
2) Efficiency - It refers to correctness, easiness & the quickness of the use of the website by the user. It is generally measures in the terms of time.
3) Completeness - the extent to which a website implements specified functions.
4) Satisfaction - It refers to the user feeling & the opinion about the

website. It is usually captured through the survey. Users are more likely to use the website that satisfy their needs as compared to other ones.

5) Clarity & Accuracy of Online Help & the written Documentation - The extent to which the Online help & the documentn is clearly & accurately written.

Performance Testing- To ensure that the web applⁿ can bear the load during the peak hours along with the response to the user in timely & reliable manner. Performance test including the load & the stress test need to be connected.

The performance of the applⁿ during the peak hours must be tested & monitored carefully. Several factors that may influence the performance of a web applⁿ are as follows-

1. Response time
2. N/w Bandwidth
3. No. of users
4. Type of users
5. Time to download

The goal of performance testing is to evaluate the applⁿ performance w.r.t. the real world scenario. The following issue must be addressed during the performance testing-

1) Performance of system during the peak hours.
2) Pt at which the system performance degrade or the system crash.
3) Impact of the degraded performance on the customer.

Load Testing- It involve testing the web applⁿ under the real-world scenario by simulating no. of users accessing the web applⁿ simultaneously. It test the web applⁿ by providing its max. load. It may follow the following steps in order to ensure reasonable performance during the peak hours-

1) Defining the environment for the load test.

ii) Defining the testing strategy & determining the no. of users.
iii) Choosing the right tool & executing the load test.
iv) Examining the results.

Stress Testing- It involve the exn of web applⁿ with more than the max. load for a long period. Unlike performance & the load testing, stress testing evaluate the response of a system when the system is given the load beyond its specified limits. It is also used to monitor & check the reliability of the web applⁿ when the available resources are on beyond the max. usage. The behavior of a system is monitored to determine when a system under stress test fails & how does it recover from the failure.

The system performance is expected to degrade when a large no. of users hit the web-applⁿ simultaneously. After the complet^n of stress test, the testing team must analyze & note the system performance degrad^n pt. & compare them with the acceptable performance of a system.

15/11

Database Testing - In the web applⁿ, many applⁿ are database driven. For eg. E-commerce websites or B2B applⁿ. It is important to these applⁿ work properly & provide the security to the user infoⁿ like the personal details & the banking details. Testing a data centric web-applⁿ is important to ensure their error-free operatⁿ & to the customer satisfactⁿ. Important issues in a database testing may include-

1) Data validatⁿ
2) Data Consistency
3) Concurrency control & the recovery
4) Data manipulatⁿ operatⁿ like insertⁿ, deletⁿ, updatⁿ & retrieval of data.

A database must be tested for the administrative level operatⁿ such as addition, deletⁿ, updatⁿ in a database & the user operatⁿ such as searching & providing the personal details.

In database testing, the following correctness must be ensured -

1) Are the database operat^{n} performed correctly?
2) Is the concurrent users' access to the database handled correctly?
3) Are the performance rqt. such as throughput & the response time met?
4) Does the database restore to the consistent state after the crash recovery?
5) Are backup & the recovery procedures designed & ensure uninterrupted services to the user?
6) Does the database have enough space & the memory to store the records & handle the multiple administrative & the user operat^{n}?

Database testing may include generat^{n} of new records, monitoring of system performance & the verific^{n} of the performance.

Post Deployment Testing - It may reveal those problems which went ~~when~~ undetected before the deployment of the web appl^{n}. Despite all the planning & the testing carried out before the deployment, obtaining the user view is important for the improvement of a website & it ensures that the website adapt to the need of the user. User feedback may come in various forms, ranging from reporting of faults to the suggestions for improvement of the website. The effective way to obtain the user view is to get the survey filled by the user. The survey can be used to detect trends & may provide the valuable infon for the improvement of the website. The response obtained from the survey may help the developer or the owner of the website to improve the website.

Once the user view is obtained, it is important to identify useful fault reporting, suggestions & the recommendations. The

following criteria is used to identify which suggestion needs attention-

1) Frequency of suggest^{n} - How many users have given the same suggestion or recommend^{n}? If a small no. of users are making the same request then we must think twice before implementing the suggestions.

2) Source of feedback - who is providing the suggestion? It is vital to make sure that the suggestion comes from the regular user or the accidental user.

3) Cost of implementing the suggestion - Is the suggested idea worth implementing? The correctness of the proposed change & its impact on the cost & the schedule must be analyzed carefully. The benefits of implementing the suggested idea to the business must be determined.

4) Impact of implementing the suggestion - Will implementing the suggest^{n} add the complexity of the website? Will the change be compatible with the other functionalities of the website? It is important to obtain the answers of these questions as a result of implementing a change are sometime unpredictable.

Security Testing - Security is the procedure used to protect the info^{n} from the various threat. It is very important to protect the sensitive & critical info^{n} & the data while communicating over the n/w. The user wants implement^{n} of a safe guard to protect personal, sensitive & the financial info^{n}. We want data to be accurate, reliable & protected against the unauthorized access.

Security involves the various needs such as unauthorized access, unauthenticated users, message send to an unintended user, etc.

The primary reqt. of security include-

1) Authentication - Is the info^n sent from the authorised user?
2) Access control - Is the data protected from the unauthorised user?
3) Integrity - Does the user receive exactly what is sent?
4) Delivery - Is the info^n delivered to the intended user?

A web appl^n must fulfill the above mentioned primary security environment. Testing the threats in the web appl^n is the important activity. The tester must check the web appl^n against all known internet thieves.

Virus that are the most common & the sophisticated kind of threat to the web appl^n that may enter from the n/w.. A virus is a program that modify the other program by attaching itself to the program, which may infect the other programs when the host program execute. The virus may perform any fnc such as deleting the files & the programs. The goal of testing the web appl^n against the virus threat is to verify method of virus prevent^n, detect^n & the removal.

Security testing requires an experienced tester having through knowledge of Internet related security issues. The security expert needs to check the security issue including the authentication, unauthorised access, virus & the recovery from the failure.

16/11

Object Oriented Testing

OOP concepts are different from the conventional programming & have become a preferred choice for the large scale system design. The fundamental entity is the class that provide an excellent structured mechanism. It allows us to define the system into well-defined units which may be implemented seperately. We still do the unit testing although the meaning of the unit has changed. We also do the integrat^n & the system testing to

test the correctness of the implementn. We also do the regression testing in order to ensure that the changes have been implemented correctly. However, many concepts & the techniques are different from the conventional testing techniques.

Unit - In a conventional programming, a unit is the smallest part of the program that can be compiled & executed. In OOS, we have 2 options for a unit. We may treat each class as a unit or a method within a class as a unit.

Unit testing of a class with a super class may be impossible to do without the super class's methods or variables. ~~This may solve~~ One of the soln is to merge the super class & the class under test so that all the methods & the variables are available this may also solve the immediate testing problem & is c/d as 'flattening of the classes.'

Levels of Testing - The various testing levels are -

1) class testing (Unit) 2) Integratn testing 3) System testing

In order to test a class, we may create an object of a class & pass the proper parameters to the constructor. We should also examine the internal data of the object. The encapsulation plays an important role in the class testing because the data & functn (operatn) are combined in the class. We concentrate on each encapsulated class during the unit testing but each fnc may be difficult to test independently.

Inter-class testing (Integratn testing) considers the parameter passing b/w the 2 classes & is similar to the integratn testing.

System testing consider the whole system & the test cases

BVA

are generated using the functional testing techniques, i.e; ~~BB~~ testing.

Integration Testing - Integrat^n testing in an OOS is also c/d inter-class testing. We don't have hierarchical control structure in the object orient^n. Thus, the conventional integrat^n testing techniques like top-down, bottom-up & the sandwiched integrat^n can't be applied. There are the 3 popular techniques for the inter-class testing in the OOS, which are as follows-

1) The first is thread based testing where we integrate the classes that are needed to response to an I/p given to the system. Whenever we give I/p to the system, one or more classes are required to be executed that respond to that I/p to get the result. We combine such classes which execute together for a particular I/p or a set of I/p & this is treated as a thread. Thread-based testing is easy to implement & has proved to be an effective testing technique.

2) The second is the use-case based testing where we combine the classes that are required by the one use-case.

3) The third is the clustered testing where we combine the classes that are required to demonstrate one collaborat^n.

In all the 3 approaches, we combine the classes on the basis of a concept & execute them to see the outcome. Thread-based testing is more popular due to its simplicity & easy implement^n. The advantage of OOS is that the test-cases can be generated earlier in the process even when the SRS & the SDD is being designed. Both the teams including the tester & the developer may review the SRS & the SDD thoroughly in order to detect many errors before coding. However, testing of the source code is still the very important part of the testing & all generated test cases

will be used to show their effectiveness & the usefulness.

17/11.

Class Testing - A class is very important in the OOP. Every instance of a class is known as object. Testing of a class is very significant & critical in the OOT where we want to verify the implementn of a class w.r.t. the specifican. If the implementn is as per the specifican, then it is expected that every instance of a class may behave in the specified way. Class testing is similar to the unit testing of the conventional system. We require the stubs & the drivers for testing a unit & sometimes it may require significant effort. Similarly, classes also can't be tested in isolatn. They may also require additional source code, similar to the stubs & the drivers for testing independently.

Validatn & the verifican techniques are equally applicable to test a class. We may review the source code during the verifican & may be able to detect good no. of errors. Reviews are very common in the practice, but their effectiveness is heavily dependent upon the ability of the reviewer.

Another type of testing is the validatn where we may execute a class using a set of test cases. This is also common in the practice but the significant effort may be required to write the drivers & the stubs & sometimes this effort may be more than effort of developing a unit under the test. Classes should be tested by its developers after developing the drivers & the stubs. Developers are familiar with the internal design, complexities & the critical issues of a class under test & this knowledge may help to design effective test cases. Class should be tested w.r.t. its specifican.