STM

UNIT - I

Testing:

-> Testing is the process of verifying whether application works as per requirements or not.

-> The quality and productivity of a S/W is measured by testing.

-> The factors that have to be achieved while testing are,

- Correctness
- Reliability
- Usability
- Maintainability.

-> Once the source code has been generated, S/W must be tested to uncover as many errors as possible before delivery to the customer.

-> The goal of S/W testing is to design a series of test cases that have a high likelihood of finding errors.

Need for Testing

There is a need that every S/W product have to be tested because,

1) The development process cannot prepare a error free product.
2) Without testing, we cannot judge given product is error free or not.
3) Testing not only identifies detects but also quality of the product is measured which helps in deciding whether a product should be released or not.

→ Testing techniques provides systematic guidance for designing tests that,

a) Exercise internal logic of S/w components.

b) Exercise the i/p & o/p domains of the program to uncover errors in program function, behaviour and performances.

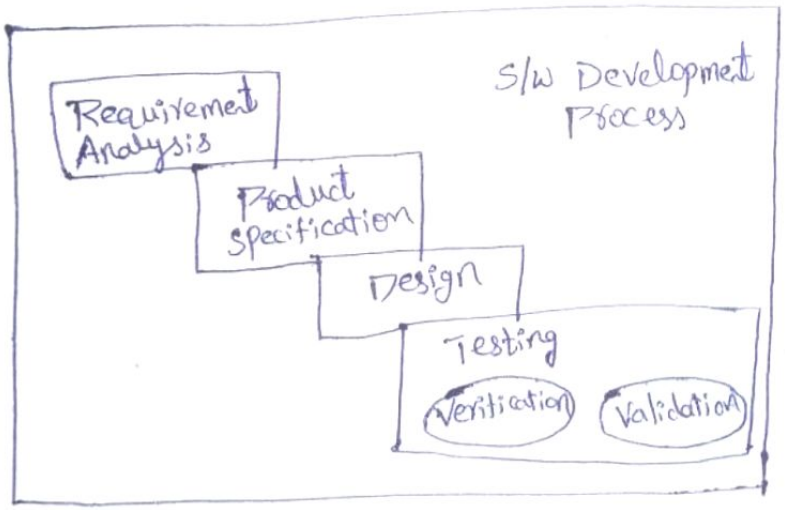
Testing As a process:

Testing is one of the process which is embedded in the S/w development process.

→ The testing is related to two processes.

1) Verification – Are we developing this product by following all design specification.

2) Validation – Are we developing the product which attempts all that users needs from S/w.



Process of Testing

Examples of Testing

Suppose if we consider the testing of a website or a web page. The few things that have to be tested are

1) Links i.e, internal, external, mail and broken links have to be tested.
2) In forms, the field validation, error message for wrong i/p, optional and Mandatory fields have to be tested.
3) The database testing will be done on database integrity.
4) Testing on cookies should be done on the client side on the temporary Internet files. and many other cases to be tested.

→ As explained, testing would be done on all possible cases of a given product or any s/w application.

Purpose of Testing

→ S/w Testing is nothing but a process of verifying and validating that the program or a s/w application or a s/w product which satisfies the business and technical requirements.

→ The most general purpose of s/w testing would be to get the confidence that the s/w product would work correctly without any errors.

→ The Major objectives of testing can be classified 5 factors.

1) verification
2) validation
3) Defect prevention
4) Quality Improvement
5) Reliability Estimation.

1) Verification:

→ The process of verification is done to chek whether the s/w Meets its specification or not.

→ A specification is nothing but the description of a function when an i/p is given & o/p is expected to be Measured.

→ Two types of verifications
   a) Static verification
   b) Dynamic verification

Static verification: The verification of static system to discover its problems is static verification.

Dynamic verification: Is the one where testing is performed by observing the product behaviour.

(4)

Levels of Verification:

There are four levels of verifications.

(i) Comoponent Testing: The testing is conducted to verify the implementation of one or more s/w components.

(ii) Integration Testing: The s/w & h/w elements are integrated and tested till the entire system has been integrated.

(iii) System Testing: The integrated entire system is verified to Meets its requirements.

(iv) Acceptance Testing: This testing is done to determine whether the user needs to accept the system or not.

(2) Validation:

The process of validation is done to check whether the s/w Meets its business requirements or not.

→ Validation involves execution of code where as verification does not. It can catch errors which cannot be caught by verification.

→ Disadvantage of validation is that the s/w works only for few particular cases. A finite no. of test cases cannot validate that the s/w works for all situations.

(3) Defect Prevention: Whenas/w is put to operation, the

variation b/n the expected result and actual result is known as defect.

→ As the defects increases the cost of s/w development, it is better to find the defects as early as possible in the development cycle.

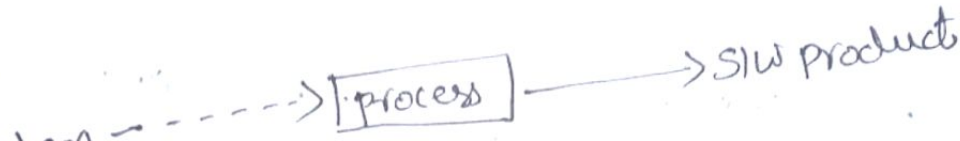
→ So testing should be done at every stage in order to Prevent the defects.

④ Quality improvement:

→ One of the most important thing that has to be considered is quality.

→ Quality is defined as a factor which is used to determine whether a s/w meets all the requirements that are specified in the design phase.

A product or process



s/w product (or) s/w process

→ Manufactured product is the one which can be seen physically where as s/w product cannot be seen.

→ Testing the quality of s/w is very expensive because it includes the cost of detecting and correcting the bugs, cost of developing and executed test cases.

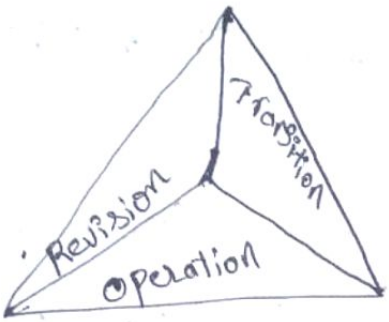
→ The quality of s/w cannot be tested directly, It should be tested based on the testing related factors.

→ J.A Mccall have developed few factors to test quality.

i) Product operation

ii) Product Transition

iii) Product Revision.



Quality Factors

All the factors in turn are considered concerned with other aspects like,

→ Product production deals with the quality factors are...

a) correctness

b) Reliability.

c) Efficiency

product transition deals with,

a) portability

b) Interpretability

Product Revision deals with

a) Maintainability

b) Testability

c) program Modification.

5) Reliability Estimation:

-> It is a method used to estimate the functioning of a failure free s/w for a particular period of time in a specific environment.

-> It is difficult to estimate s/w reliability by considering its related aspects.

-> An estimation model is used for performing data analysis for estimating the present and future reliability.

Differences between verification & validation

Verification	Validation
1) It is a static process of verifying documents, design and code.	1) It is a dynamic process of validating/testing the actual product.
2) It does not involve executing the code.	2) It involves executing the code.
3) It is human based checking of documents / files	3) It is computer based execution of program.
4) Target is requirements specification, application architecture, high level and detailed design, database design.	4) Target is actual product a unit, a module a set of integrated modules, final product

5) It uses Methods like inspections, Walk throughs, Desk checking etc.	5) It uses methods like black box, gray box, white box testing etc.
6) It generally, comes first done before validation.	6) It generally follows verification.
7) It can catch errors that validation cannot catch.	7) It can catch errors that verification cannot catch.

Productivity and quality in software

→ In production of a Product, there are several phases starting from requirement analysis to the phase of final testing before the delivery of product.

→ In all these phases, the product is subjected to the quality control and testing. If any defects are found at any stage, that component will be back for correction.

Measure for productivity:

→ Productivity is measured by the sum of the costs of resources, rework and failed components and cost of quality assurance and testing.

Productivity = cost(resources + rework + failed components + quality assurance + testing)

→ There is a trade-off b/n quality assurance costs and Manufacturing costs. If effort spent in quality assurance is not sufficient, the reject rate will be high and the next cost will also increases.

→ If inspection is so good, the reject rate will decrease but inspection cost will dominate, and again net cost will suffer.

→ The Manufacturing process designers attempt to establish a level of testing and quality assurance that minimize net cost for agiven quality objective.

→ The costs of testing and quality assurance for Manufactured item can be as low as 2% in consumer products or as high as 80% in products such as space ships, nuclear reactors & aircrafts etc.

→ The relation between productivity and quality for s/w is different from that for manufactured goods. s/w maintenance is different from hardware maintenance. S/w costs are dominated by development.

→ The Manufacturing cost of s/w is not important.

→ The cost of bugs is a part of s/w cost that includes the cost of detecting bugs, the cost of

correcting bugs, the cost of designing tests that discover them and the cost of running these tests.

→ The main difference bln productivity of Widget and SLW is that for HLW, quality is one of several productivity determinants, where as for SLW, quality & productivity are almost not distinguishable:

Goals for Testing

Bug prevention:

→ A Bug prevention is considered as testing's first goal. If a bug is present in the SLW, it is needed to be detected and corrected.

→ If such a bug is prevented, there is no need to correct it or detect it and the cost of testing will be decreased.

→ Hence we say bug prevention is better than bug detection and bug correction.

→ When a particular bug is prevented, there is no need to perform testing again to confirm the accuracy of the program.

→ To achieve this goal, testing should be performed at every stage of SLW development so that all the bugs could be discovered and prevented during the design phase itself.

Bug detection :

How much ever we try, the first goal cannot be achieved ideally because "to err is human". If we fail to achieve the first goal, atleast the second goal should be achieved mandatorily. i.e, bug discovery.

Phases in a Tester's Mental life

Tester's hay characterized by the five phases.

1) Phase 0
2) Phase 1
3) Phase 2
4) Phase 3
5) Phase 4

1) Phase 0 Tinking:

-> In this phase of thinking, testing and debugging are considered to be same. In fact testing supports Debugging.

-> In the initial stages of testing, this phase 0 thinking was the criteria for s/w development.

-> Phase 0 thinking was appropriate to an environment characterized by expensive and scarce computing resources.

- L) lowcost s/w
- L) lone programmers
- L) Small projects.
- L) Throwaway s/w.

-> In the initial days, this was the only criteria for development but now a days, it is considered as the barrier for good testing and quality software.

(2) Phase 1 Thinking - The s/w works

-> The purpose of testing is to show that the s/w works.

→ Phase 1 thinking represented progress because it recognized the distinction between testing and debugging.

→ In this ~~eas~~ Phase, to check the execution of s/w many number of tests are performed.

→ It is enough to prove that one test fails in case of failed s/w but to prove that s/w is executing, any number of tests are not sufficient.

→ s/w is tested, errors are also detected. i.e is this thinking is kind of reasoning to determine the working of a program without testing it.

③ Phase 2 Thinking – The s/w Doesn't work

→ The purpose of testing is to show that the s/w doesn't work.

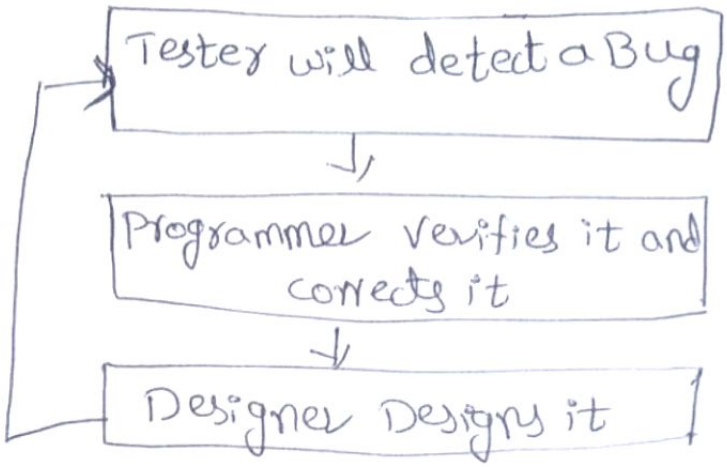
→ The differences b/n phase 1 and 2 thinking is illustrated by analogy to the differences between bookkeepers and auditors.

→ The bookkeeper's goal is to show that books balance, but auditor's goal is to show that despite the apperance of balance, the bookkeeper has embezzled.

→ Phase 2 thinking leads to strong, revealing tests.

(14)

→ While one failed test satisfies the phase 2 goal, phase 2 thinking also has limits.
→ The test reveals a bug, the programmer correct it. the test designer designs and executes another test intended to demonstrate another bug.
→ Bug can be detected by testers, programmers and designers in this phase.



Flowchart to show the process of Phase 2 Thinking

→ disadvantage in phase 2 thinking is that is is a never ending process.

④ Phase 3 Thinking - Test for Risk Reduction

→ The purpose of testing is not to ~~find~~ prove anything, but to reduce the perceived risk of not working to an acceptable value.
→ Its accepting the principles of statistical quality control.

→ To the extent that testing catches bugs and to the extent that those bugs are fixed, testing does improve the Product.

→ If a test is passed then the product's quality does not change, but our perception of that quality does.

→ Testing, Pass or fail reduces our perception of risk about a s/w product.

⑤ Phase 4 Thinking - A state of Mind

→ Testing is not an act. It is a Mental discipline that results in low-risk s/w without much testing effort.

→ Testability is the goal for two reasons.
  1) We want to reduce the labor of testing.
  2) testable code has fewer bugs than code that's hard to test.

Cumulative Goals:

→ The phases goals are cumulative.

→ Debugging depends on testing as a tool for probing hypothesized causes of ~~Syst~~ symptoms.

Test Design

1) Effective Fragmentation of the Available Tests:

→ Every test is fragmented into individual test modules that are easy to understand and mange.

→ The test modules are complete and even aggressive to detect bugs, before the system is fully designed.

→ Designing of the test case is an important factor for success in test automation.

→ Each module has a well-designed scope that is different from that of the other modules.

2) Accurate Testing Technique for Each Test Module:

→ Each of the fragmented test module is called as 'mini project'.

→ The scope of a test module determines various testing techniques that are used to develop an individual test module.

→ The testing strategies that are used to build the test cases are boundary analysis, decision tables etc.

→ Decisions need to be made regarding who should be involved in creating and evaluating the test cases.

→ Example: Consider a test module whose objective is to test the estimation of insurance premium.

It requires 'actuarial' department to get involved in Performing the test.

3) Specifying correct Level of Details for the Test:

→ This Principle, a correct "level" of abstraction should be determined. The high level details that are required for developing test cases should be specified and all the low level implementation details should be hidden.

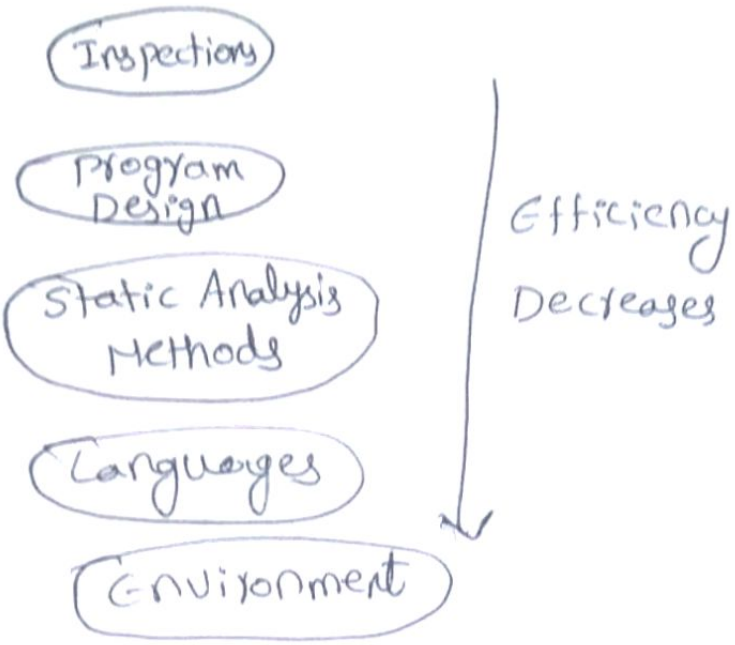
Example:

In an employee databases, test are performed using high level routine like 'check emp-status' and 'check-emp salary', but while testing a low level dialog routine, an action called "click" is used to determine whether an ok button is clicked or not on the appeared dialog box.

Testing Isn't Everything

→ The most effective method for detecting the errors is Testing.

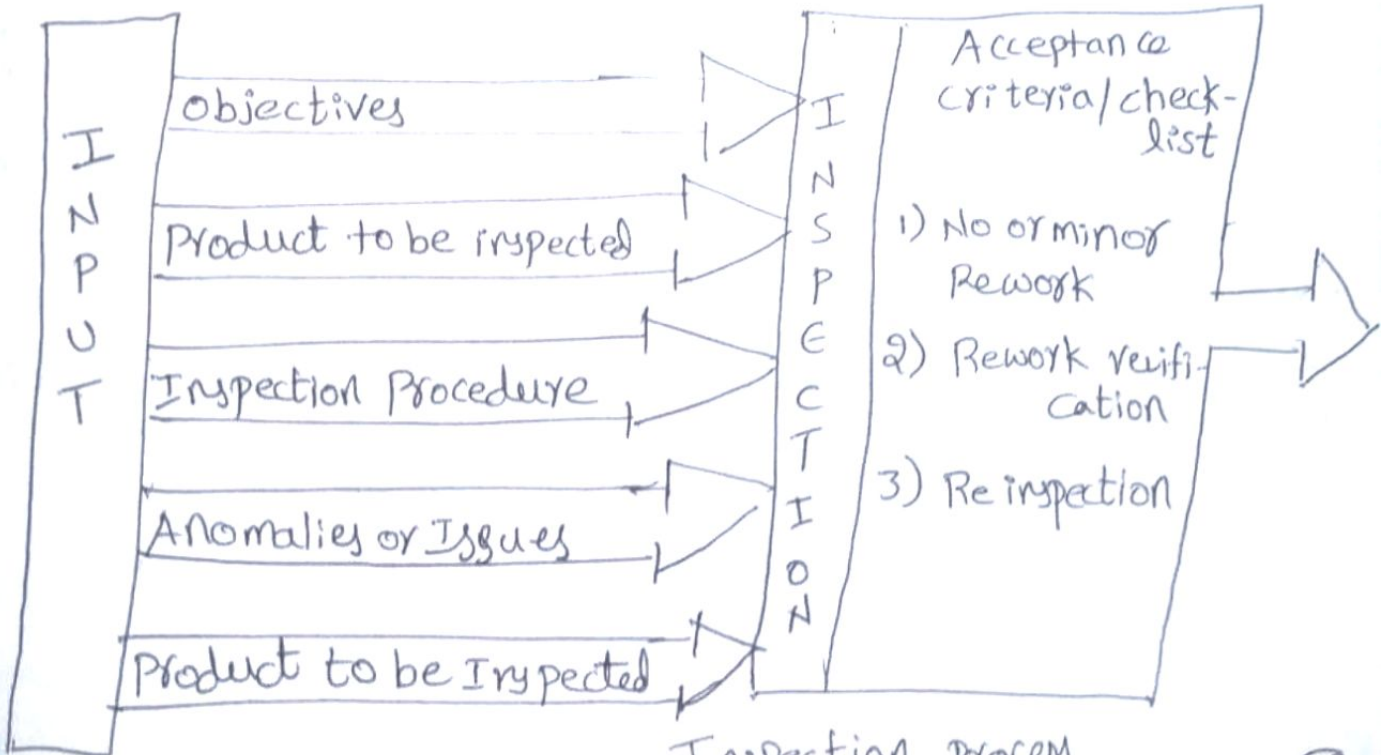
→ Apart from testing, there are other methods which can detect errors. The process of testing alone cannot make the s/w error free.



Efficiency order of other Methods of testing

Inspection Methods :

→ S/w inspection is one of the most important manual techniques which allows testing the program without running and comparing code or design of work product to set of pre established inspection rules



Inspection Process

Design style:

→ Design Model of a program determines whether the quality of the program is good or not.

→ An improper design model degrades the quality of the s/w program.

→ Design objectivies such as openness, clarity, testability are choosen in order to prevent bugs.

Static Analysis Methods:

→ It is a strong typing and type checking are considered static analysis is a part of research and development to find errors in data flow anomaly detection process.

Languages:

→ Languages are used in the programs results in preventing some errors.

→ A new type of error is detected in a new programming languages by the programmer, which makes the rate of bugs independent of the language being used.

Design Methodologies & development environment:

→ s/w development method is an internal proces, which uses various methods for developing s/w.

The Pesticide Paradox and the Complexity Barrier

Pesticide Paradox :

-) "Every Method used to prevent or find bugs leaves a residue of subtler bugs against which those methods are ineffectual".

-) This paradox states that running the same tests over and over again could not show any new defects because all defects found using the initial test cases could eventually be fixed.

Complexity Barrier :

-) "software complexity grows to the limits of our ability to manage that complexity".

-) This statement means that the complexity of the s/w grows to the limits till a human have the ability to manage it.

21

DICHOTOMIES

Testing vs Debugging

Testing

1) Testing is the process of executing S/w with the intent of detecting the presence of faults.
2) Error Detection is the goal of testing.
3) Testing always starts with known conditions.
4) The o/p is predictable
5) It is nessary to have planned, designed and scheduled Constraints
6) Testing is the process which demonstrates errors.
7) Testing Proves the Programmer's failure.
8) The objcctives of testing are predictable, dull constrained, rigid and in human.
9) Testing can be done without design knowledge.
10) Testing can be performed by an outsider.
11) Test design and execution are automated.

Debugging

1) Debugging refers to undertaking Measures to make programs free bugs.
2) Error detection and error correction are the goals of debugging.
3) Debugging starts from unknown conditions.
4) The o/p is unpredictable.
5) It is not necessary to have these constraints.
6) Debugging is the process of reducing the errors.
7) Debugging is programmers indication.
8) The objectives of debugging are intuitive leaps, conjectures, experimentation and freedom.
9) Debugging should have design knowledge.
10) Debugging should be performed only by the insider.
11) Debugging can't automated.

22

Function Vs structure

Structural Testing

1) In structural Testing, tests are designed from structural Point of view.
2) It is also known as white box, glass box, open box or clear box, testing.
3) Structural testing looks at the implementation details such as Programming style, control method, Source code, data base design and coding details.
4) Finite time is taken by the test cases but it cannot detect all bugs.
5) There is dependency b/n a tester and Programmer for test Process.
6) Tests are performed either from Programmers or designer's perspective
7) It is less effective than functional testing.
8) Various Methods that perform
   (i) Statment Testing
   (ii) Decision Testing
   (iii) Condition Testing
9) Unnecessary code which removes bugs is eliminated.

Functional Testing

1) In functional testing tests are designed from functional point of view.
2) It is also known as black box or closed box or opaque box testing.
3) This testing does not require any knowledge of internal structure of the s/w.
4) Infinite time is taken by test cases and all bugs are detected.
5) Tests are performed by the tester and programmer individually and they are independent.
6) Tests are done only from users perspective.
7) Functional testing more effective than structural testing.
8) Various Methods that perform
   (i) Expected I/P
   (ii) Boundary values
   (iii) Illegal values.
9) It reveals problems and inconsistencies in functional specifications. (A23)

The Designer vs Tester

Designer	Tester
1) Designer is based on the structural specification of the system.	1) Tester is based on the functional specification of the system.
2) Designer depends on implementation details.	2) Tester does not require any details of implementation.
3) Designer can design an efficient s/w when we have the knowledge of implementation.	3) As there are no details of implementation, the s/w developed is inefficient.
4) Designing and executing tests is job of s/w designer.	4) Tester is also responsible for designing and executing the tests.
5) Probability of fault design increases with the increase in the knowledge of design	5) Probability of eliminating unnecessary tests is increased with the increase in knowledge of test design.

Modularity Versus Efficiency

→ A Module can be defined as a distinct, small component that has a well defined purpose.

→ While constructing the system, it could be easy to understand it each components used in the system is small.

→ The system is made up of many smaller components, it could be affected with interface bugs. then its efficiency could fall down.

→ The process of testing will be done in the form of modular components. The smaller test cases are easy to execute and re execute.

→ Each test case in the process is designed to reduce the burden to designing, debugging and execution of the test.

Small versus Large

Small	Large
1) The Programs which contain only few lines of code are known as small programs.	1) The Programmer which contain large no. of lines of code are known as large programs.
2) They contain only few components.	2) They contain large no. of components.
3) No technique is required to test small programs.	3) Different types of techniques are used to test large programs.
4) These are more efficient	4) These are less efficient.
5) These programs can be written by a single programmer.	5) Large programmers are written by diff programmers.
6) Comparing small & large programs, small programs are of high quality.	6) Comparing small and large programs, large programs are low quality.

Builder versus Buyer

→ The s/w development class & functional class must sign an agreement with each other in the same organization.

→ For this reason, it is better to separate builder and buyer.

→ If the builder and buyer are not separated, there will be few problems such as "No Accountability".

→ There are many persons in the s/w development life cycle. like programmers and testers that can be merged to perform various test operations.

1) The builder, who designs for and is accountable to
2) The buyer, who pays for the system in the hope of profit from providing services to
3) The user, the ultimate beneficiary or victim of the system. The users interests are guarded by
4) The tester, who is dedicated to the builder's destruction. ~~and~~
5) The operator, who has to live with the builder's mistakes, the buyer's murky specifications, the tester oversights, and the user's complaints.

A Model for Testing

The Project:

→ Testing is applied to anything from subroutines to systems that consist of millions of ~~staffal~~ statements.

For any system or product to be produced a perfect planning and implementation is necessary.

→ A real world context characterized by the following model project.

(i) Application:

→ The specifics of the application are unimportant. It is a real-time system that must provide timely responses to user requests for services.

→ It is an online system connected to remote terminals.

(ii) Staff :

→ The programming staff consists of twenty to thirty programmers, big enough to warrent formality, but not too big to manage.

→ Big enough to use specialists for some parts of the system's design.

(iii) Schedule :

→ The project will take 24 months from the start of design to formal acceptance by the customer.

→ Acceptance will be followed by a 6 month cut over period. Computer resources for development and testing will be almost adequate.

(iv) Specification :

→ The specification is good. It is functionally detailed without constraining the design, but there are undocumented "under-standings" concerning the requirements.

(v) Acceptance Test :

→ The system will be accepted only after a formal acceptance test.

→ The application is not new, so part of the formal test already exist.

(vi) Personnel :

→ The staff is professional and experienced in programming and in the application.

→ Half of the staff has programmed that computer

before and must know the source language.

→ 1/3 Mostly junior programmers, have no experience with the application.

(vii) Standards:

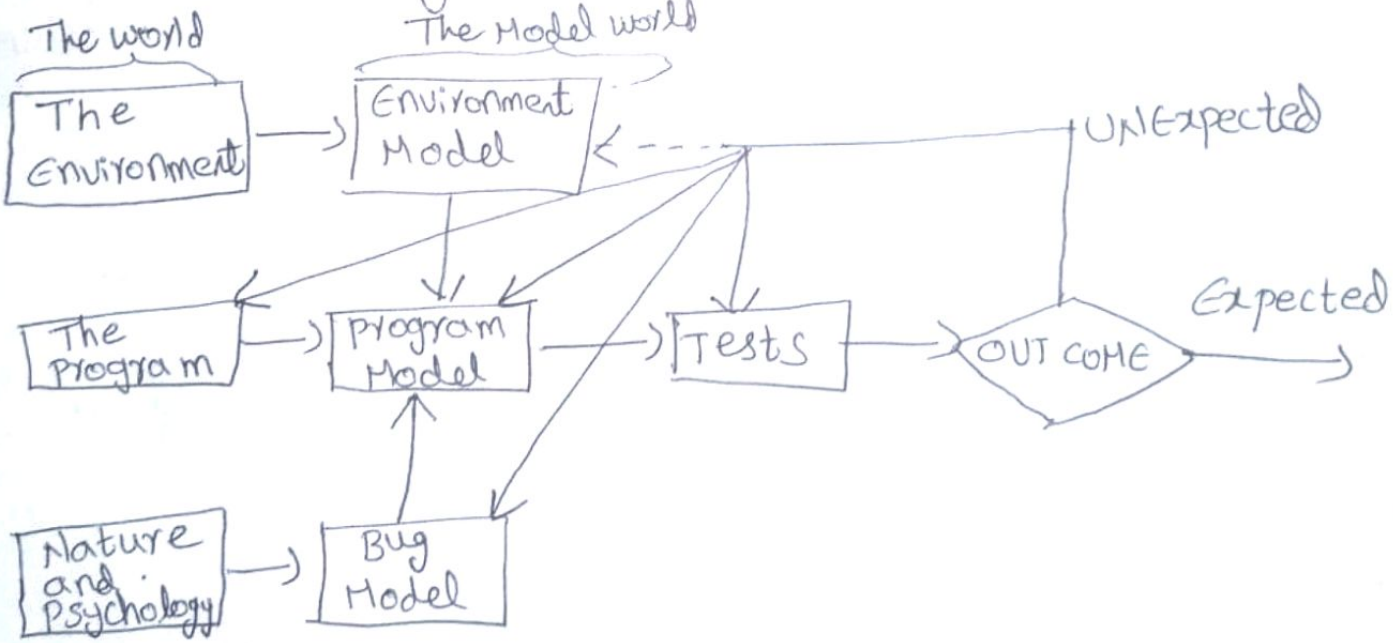
→ Programming and test standards exist and are usually They understand the role of interfaces and the need interface standards.

→ Documentation is good. There is an internal, semiformal, quality assurance function.

→ The database is centrally developed and adminstrated

(viii) objectives: The system is the first of many similar systems that will be implemented in the feture.

(ix) Source: 1/3 of the code is new, 1/3 extracted from a previous, reliable, but poorly documented system, and 1/3 is being rehosted.



A Model of Testing

Overview

-> The process starts with a program embedded in an environment, such as a computer, an os or a calling program.

-> Models can be created 3 types

1) Model of environment
2) Model of the program
3) Model of the expected bugs.

1) Model of environment :

-> Program's environment is the h/w & s/w required to make it run.

-> For online systems the environments may include communication lines, other systems, terminals and operators.

-> The environment also includes all programs that interact with and are used to create the program under test, such as operating system, loader linkage editor, compiler, utility routines.

2) Program :

-> Most programs are too complicated to understand.

-> We must simplify our concept of the program in order to test.

-> If the simple program unable to explain the unexpected behaviour, then the situation arises where you need to modify the program fore more details and if even that fails, the model itself have to be changed.

3) Bugs :

-> Bugs are more insidious than ever we expect them to be.

Bugs can be categorized 3 ways.

(a) Initialization
(b) call sequence
(c) Wrong variable

29

→ Unexpected result sometimes will change the model of bugs.

(i) Begin Bug Hypothesis:

→ The belief that bugs are nice, tame, and logical. only Week bugs have a logic to them and the ~~can~~ exposure by strictly logical means.

(ii) Bug locality Hypothesis:

→ The belief that a bug discovered within a component affects only that components behaviour.

→ Because of structure, language syntax, and data organization, the symptoms of a bug are localized to the components designed domain.

→ only Week bugs are so localized.

(iii) Control Bug Dominance:

→ It is a belief that the errors in the control structures could ~~documents~~ dominate the bugs, Many errors can be traced control flow but they cannot domains these bugs.

(iv) Code / Data separation:

→ This is a belief that the bugs would respect the code and data. But in real systems it is hard to differentiate the data and code.

(v) Lingua Salvator Est:

→ It is a belief that the language syntax and semantics would eliminate most bugs but there is no clear evidance to prove this.

(vi) Correction Abide :

→ It is a hypothesis where it is believed that a corrected bug will remain corrected.

→ But in real time many happen or may not. Subtle bugs May change even they are corrected many times.

(vii) Silver Bullets :

→ It is a Mistaken belief where we believe that the lang or design method or environment grants immunity from bugs.

→ It is considered true in case of easy to moderate bugs but could not be considered in case of subtler bugs.

(viii) Sadism Suffices :

The Most common belief that independent testers, low running & others are enough to catch all those bugs.

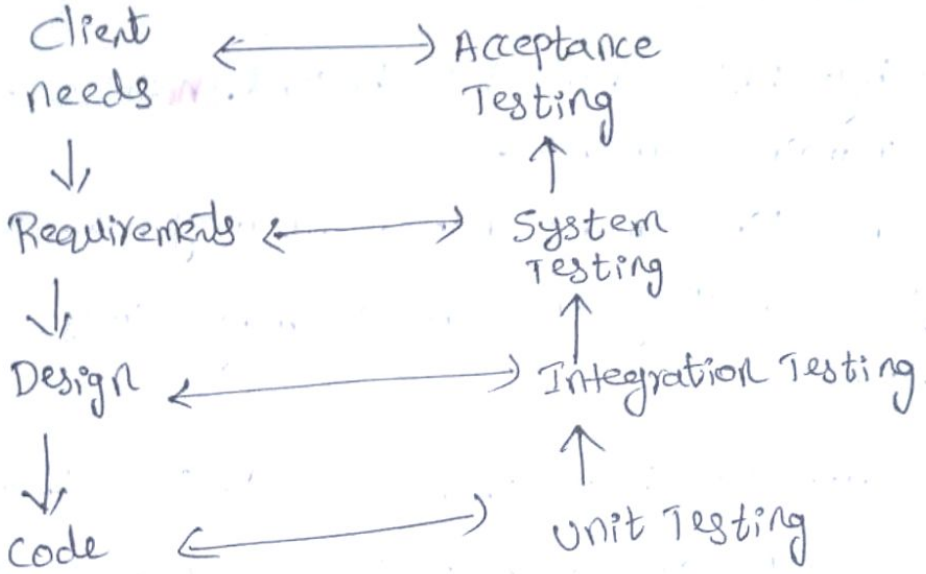
(ix) Angelic Testers : It belief that

"The testers are better at best design ~~that~~ than Programmers are at code design".

Testing and Levels

Three ~~the~~ different kinds of testing on a typical s/w system.

1) Unit / component testing
2) Integration testing
3) System testing



Levels of Testing

1) Unit Testing:

→ The first level of testing is called unit Testing. Unit Testing is essentially for verification of the code produced by individual programmers, and it is typically done by the programmer of the module.

→ The goal for unit Testing is insure that each individual s/w unit is functioning according to its specification.

→ Good testing practice calls for unit tests that are planned and public. planning includes designing tests to reveal defects such and ~~sequence defects~~. functional description, ~~def~~ algorithmic defects, data-defects, and control logic developed, using both white and black box test design strategies.

2) Integration Testing:

→ It is a process by which components are aggregated to create larger components.

→ Integration Testing is performed when the individual components undergo components testing successfully. But when components are integrated, component testing is either incorrect ~~or~~ inconsistent.

Ex: Components A and B have both passed their component tests. Integration testing is aimed as showing inconsistency b/n A and B.

→ A and B component testing successfully, but failed when integrated, some of the situations where inconsistency arises.

i) When there is an improper call or return statement.

ii) when there is an inconsistent standard for data validation.

iii) when an inconsistent Method is used for handling the data objects.

Real time Example

1) Assembling of Aeroplane: Each and every unit is Manufactured and tested separately, Then all the parts are assembled with each other and texted for functionality when two parts start communicating with each other. This is integration testing.

2) In Google:

→ Compose a mail and send it to some valid user id (Module 1)

(ii) Go check whether the sent mail is there in the "sent item" (module 2).

→ its there is data flow is corred otherwise wrong.

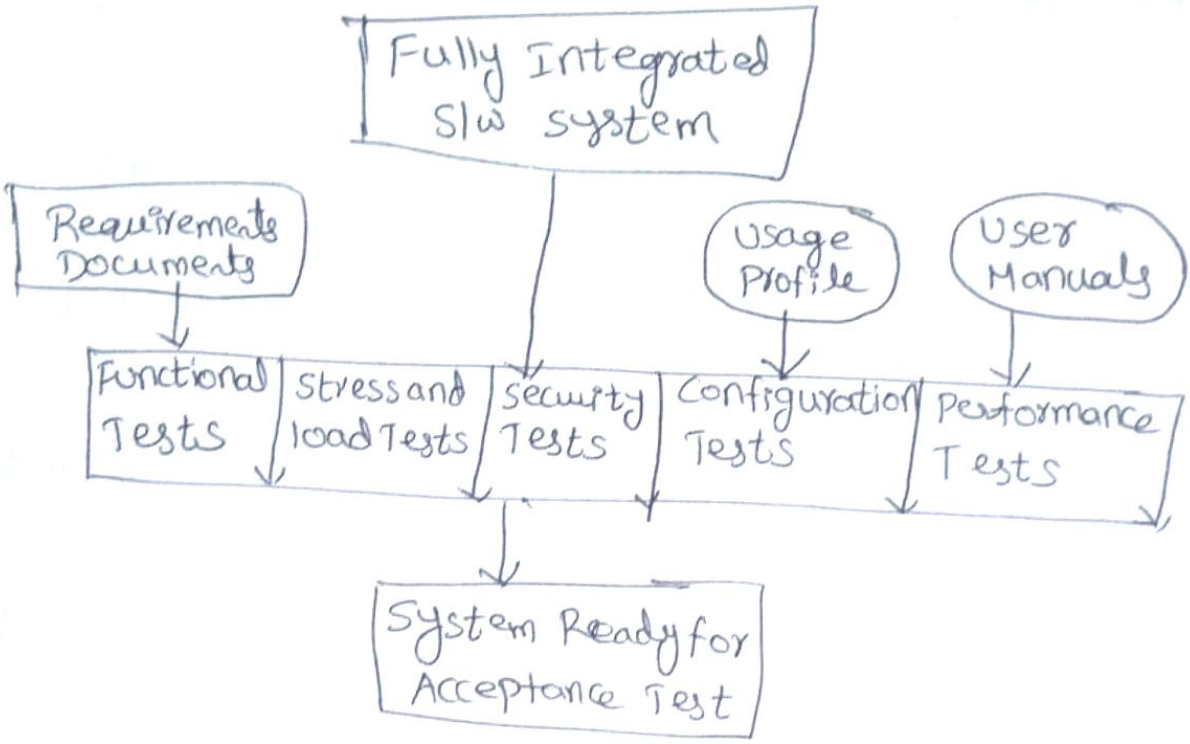
3) System Testing :

→ A system is a big component. system testing is aimed at revealing bugs that cannot be attributed to components as such to the inconsistances b/n components or to the planned interactions of components and other objects.

→ There are several types of system tests.

a) Functional Testing
b) performance Testing
c) stress Testing
d) configuration Testing
e) security Testing
f) Recovery Testing
g) Reliability Testing
h) Usability Testing

→ As the system has been assembled from its component parts, many of these types of tests have been implemented on the components parts and subsystems.



Types of system Tests

4) Acceptance Testing :

→ Acceptance tests are very important milestone for the developers. At this time the clients will determine if the slw meets their requirements.

→ Acceptance tests must be rehearsed by the developers/ testers. There should be no signs of unprofessional behaviour or lack of preparation.

→ clients do not appreciate surprises. clients should be received in the development organization as respected guests.

White box Testing

→ White box testing deals with the internal logic and structure of the program code.

→ It is also known as glass box, structual or open-box testing.

→ In glass box testing, test cases are derived based on the knowledge of sw structure and its implementation.

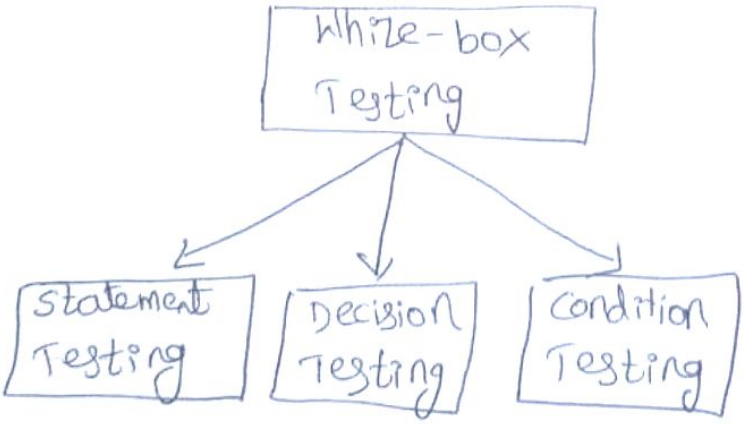
→ White boxing testing, the tester can detect which module or unit is not functioning properly.

→ White box tests can analyze data flow, control flow, information flow, exception and error handling techniques. These techniques are used to test the behavior of sw.

→ White box testing is done to check if the implementation is in accordance with the planned design.

→ The different methods used to perform white box testing are

1) Statement Testing
2) Decision Testing
3) Condition Testing



White box Testing (36)

1) Statement Testing :

→ Test values are provided to check whether each statement in the module is executed atleast once.

→ Stament ~~ex~~ testing executes statements
(i) optional arguments are available.
(ii) User provided parameters or producers are available.
(iii) Planned user actions are available.

2) Decision Testing :

→ Tests are performed to check whether each branch of a decision is executed atleast once.

→ Decisions require two values in standard boolean decision testing.

3) condition Testing :

→ Tests are performed to check whether each condition in a decision accepts all the necessary o/p's atleast ones.

→ It also checks whether the entry points to the procedure or flow is invoked once.

Advantages :

1) The code is optimized.
2) The unnecessary lines of code which results in hidden errors are removed.
3) The application is effective because of the internal knowledge of the program code.

Dis Advantage :

1) It is very expensive to perform since the tester should have the knowledge about the internal structure.

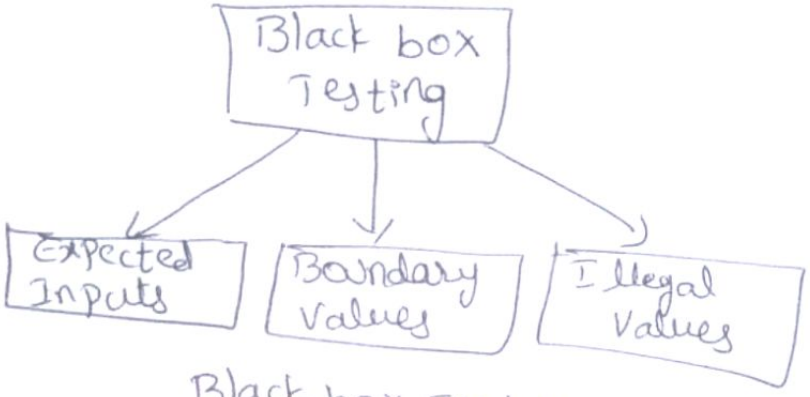
(37)

Black box Testing

→ Black box testing is done without internal knowledge of the program code.

Ex: s/w engineering test results with the known i/p's and expected o/p but not with the functioning of a program.

→ Because black box testing only deals with the specifications but not with the knowledge of the program.



Black box Testing

1) Expected Inputs:

It is the include values that are mostly expected at the time of executing a procedure.

2) Boundary values:

Expected i/p value ranges from 1 to 123, tests perform 1 and 123 values should ensure that result obtained are also within the boundary values of 1 and 123.

3) Illegal values:

If an i/p value receives a value less than 1 or greater than 123 then those values are reffered as illegal values.

Advantages

→ Black box testing is more effective than white box testing.
→ Test cases are performed from the user's perspective.
→ Tests are designed immediately after the completion of specifications.

Disadvantages

→ Black box testing take long time to test each and every i/p.
→ Tests are complicated and cannot be performed directly to the specified code segments.

The consequences of Bugs

Importances of Bugs:

1) **Frequency**: Frequency is the no. of times a particular kind of bug ~~error~~ occurs.

2) **correction cost**:
It is the cost required for correcting the detecting bugs.
   a) cost of Discovery
   b) cost of correction.

   Cost = cost of Discovery + cost of correction

3) **Installation cost**: It depends on the no. of installations. It would be small for single user program and large for PC OS ~~by~~ bug.

39

4) Consequences :

consequences can be measured by the mean size of the awards made by juries to the victims of your bug.

-> The Metric used to measure importance of bug is "$"

Importance ($) = frequency * (correction-cost + installation-cost + consequences - cost)

Consequences of Bugs

-> Bug consequences ranges from mild to catastrophic.

1) Mild : This consequence will affect ~~it~~ result in small errors such as misspellings, mis alignment etc. They will be affecting us aesthetically.

2) Moderate : The consequence will affect the performance of the system. Hence it results in duplicate output.

3) Annoying :

The presence of bugs in the system, the performance of the system degrades. for example

(i) The names are shortend or changed.

(ii) Bills for even ₹ null amount are sent.

4) Disturbing : This consequence, even the correct transcations are not executed. for example, an 'automatic teller Machine' refuses to process the 'withdrawal' transcation.

5) Serious : The information about the transcations gets
   i) Tracking of transcations
   ii) Accountability (responsibility) of a transcation
   iii) Transcation occurrence.
When such information is lost the resulting bug is called a 'serious bug'.

6) Very Serious :
This consequence results in reversing the operation of a transcation such as, deposit transcation is converted into withdrawal transcation.

7) Extreme : The consequence occurs frequently and is not limited to small number of users of transactions.

8) Intolerable : When a large data is corrupted it is very difficult to perform the recovery process.

9) Catastrophic : The system fails and hence the right to shut down the system is taken from the user.

10) Infectious : System that does not fail but corrupts the other systems results in the consequences.
   i) Detroys the physical conditions
   ii) Liquifies the nuclear reactors.
   iii) Kills the system
   iv) Prevents the system from normal functioning.

The Nightmare list and when to stop Testing

→ The nightmares for a organization can not be declared Particularly, each organization should define its own nightmares.

Step 1: All the worst slw nightmares should be listed and they should be listed in terms of the symptoms. It should be noted that how the usr will react to those symptoms.

Step 2: Convert the consequences of each nightmare into a cost. This will be the labour cost for correcting the nightmare.

Step 3: After calculating the cost, order the list from costliest to the cheapest and then the low cost and low concerned nightmares can be discarded.

Step 4: Based on your knowledge & experience, specify the kind of bugs corresponding to those symptoms expressed by each nightmare.

Step 5: As we have already developed a list of possible bugs, order the list by decreasing Probability.

→ This probability must be judged by persons intution, experience etc.

→ The importance of a bug type is calculated by multiplying the expected cost of the nightmare by the probability of the bug & summing across all nightmares.

$$\text{importance of bug type } i = \sum_{\text{all nightmares}} C_{ij} P(\text{bug type } i \text{ in nightmare } j)$$

Step 6: Using the formula, rank the bug types in the order of decreasing importance.

Step 7: Test and quality assurance inspection process must be designed using of the most effective methods.

Step 8: If a test is passed, there is no nightmares. If the test is failed, then there is a possibility of a nightmare. It goes away after correcting a bug.

Step 9: After correcting a bug, some of the nightmares would go away and now reorder the list and the entire process should be repeated.

Setep 10: Testing can be stopped when the probability of all nightmares has been inconsequential.

Taxonomy of Bugs

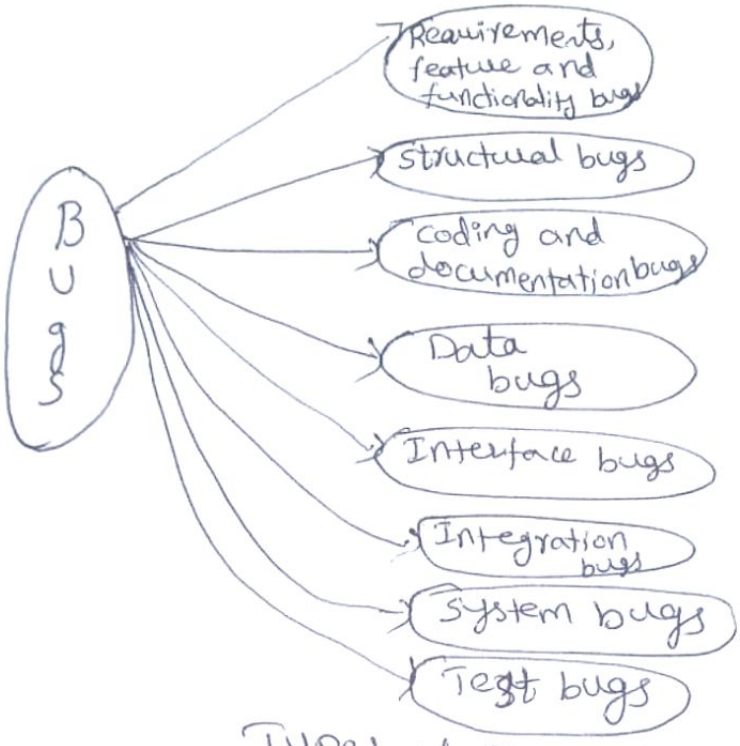
→ The taxonomy is not rigid. Bugs are difficult to categorize.

→ There are no accurate methods to categorize bugs. This taxonomy is not standard.

→ If bug is given, it can be put in the any one of the taxonomy of bugs.

→ It is not really that important whether you adopt a right taxonomy or not.

43



Types of Bugs

1) Requirements, Features, and functionality Bugs

(i) Requirements and specifications

-> The detailed description of the S/w requirements are given by specification.

-> As specifications are clearly defined, it is impossible to understand requirements are the major cause of bugs that are very expensive to be fixed.

-> bugs depends on type of application and operating-environment.

-> Disadvantage is the lifetime of requirements is from starting to ending of a process. these bugs easily pass through the development & beta tests.

(ii) Feature bugs:

-) It caused the problems in specifications.

-) Two possible bugs (a) Missing feature
(b) Wrong feature.

→ Missing feature can easily be detected and corrected.

(iii) Functionality Bugs:

→ For detecting functionality bugs, the specification features that are accurate, transparent, achievable and testable are not sufficient.

→ The similar features are combined into a single group.

→ The high level specification languages are
  ↳ short term support
  ↳ long term support

2) Structural bugs:

→ These bugs are caused due to errors in the structure of the program or code such as syntax error, logic error etc.

(i) control & sequence bugs
(ii) Logic bugs
(iii) processing bugs
(iv) Initialization bugs
(v) Data flow bugs and anomalies.

(i) control and sequence bugs:

→ The control and sequence bugs may be because of unreachable code, missing process steps, loop back or loop termination etc.

→ The importance of control flow bugs can be understood due to the ↳ the control problems are popular in the area of slw design and testing literature because of its adaptability. i.e is the reason it can be tested and detected easily.

→ Testing techniques are (i) structural Testing
(ii) planning Test.

Example: unreachable or dead code:

→ The part of the code which will never be executed is known as unreachable or dead code.

(ii) Logic bugs:

→ Logic bugs show the behaviour of the statements & operations.

(a) Incorrect design of test cases
(b) Incorrect explanation
(c) Complicated operations.

→ Testing Technique: The best way to elimate these bugs is to perform structural testing such as logic based Testing.

Ex: switch case with no default:

→ The behaviour of the program would be unpredictable if there is no difficult case in switch condition.

(iii) Processing Bugs:

→ The problems in Processing bugs occur due to incorrect conversion from one data representation to another data representation.

(a) Arithematic bugs
(b) Algebraic bugs
(c) algorithm selection
(d) generenal processing.
(e) Neglecting overflow data.

Testing Techniques

Unit testing and Domain Testing.

Ex: If the code contains a calculation that is incorrectly specified it causes processing bug.

(iv) Initialization Bugs

→ These bugs are caused mainly due to wrong and unitechniques initializations.

. Reasons ~~for~~ .

→ When a programmer forgets to initialize the working space, registers, intial variables etc.

Testing Techniques: avoid these bugs, data flow testing Methods are used.

Ex:
```
main()
{
  int a,b;
  c=a+b;
  Pf ("%d",c);
}
```

(v) Data flow bugs and Anomalies:

→ Data flow anomalies is a ~~present~~ superset of initialization bugs.

→ The data flow anomaly can be detected by the compiler during (a) Compile time
(b) execution time.

3) Data Bugs:

→ Data bugs occur because of the bugs in the specification of data objects, the designs, the number of data objects and the initial values.

Different types of data ~~are~~ bugs

(a) Static Data (b) Dynamic Data.

(a) Static Data:

→ Static data are fixed and doesn't change its structure or content.

→ Static data is analyzed based on the source code that is used to determine whether the piece of code is reachable or not.

→ for example: A huge communication n/w has several different parameters associated with it. The parameters specify the lines of code, the structure, the n/w topology, the features of lines, the user choices for those lines etc.

(b) Dynamic Data:

→ Dynamic data are not fixed and changes its contents after a specified period of time.

→ Dynamic data is analyzed based on the behaviour code that is used to determine whether the piece of code is reachable or not.

(48)

-) Dynamic data consists the following information.

(a) About the shared resources

(b) About the protective code that performs efficient data validation choices.

(c) Centralized resource management.

Ex: If the record in a database may be lacking a field then it causes data bug and if s/w reviews are conducted, the bug can be eliminated.

4) Coding bugs and documentation bugs

-> The bugs which are caused due to the errors in the coding or source code are known as coding bugs.

-> These bugs are capable of creating many other different types of bugs.

->A detected bug does not affect the test design and execution because the testing process begins only when such errors are corrected.

Testing Techniques.

No testing techniques can solve these bugs. the solution for these bugs can be through

(a) Inspection

(b) Automated data dictionaries

(c) Specification systems.

Ex:
```
main()
{
int a,b;
char c;
c= a+b;
}
```
C cannot take integer value as it is declared as char. It comes under wrong declaration of a variable.

5) Interface bugs :

a) External Interface

b) Internal Interface

c) Operating system.

(a) External Interface :

→ The external interface makes the communication possible with the outside world.

→ Important principle of interface design is that it should be robust. An external Interface can either be human or machine.

(b) Internal Interface :

→ Internal Interfaces has the same principle as that of external interfaces but the actions related to internal interfaces are more controllable.

→ The diff b/n the internal & external bug is that later one is static whereas the former one

consist of interfaces that are negotiated.

(c) Operating System:

→ OS is associated with the program or s/w bugs. It includes ~~the~~ both the h/w architecture bugs and interface bugs.

→ The OS grows, the bugs in it are detected or corrected.

→ The solution for OS bugs are

i) use specialist to write the interface programs

ii) use declaration of Macros explicity for all system calls.

Ex: User Interface description Methods are

Missing commands

Improper commands.

6) Integration Bugs:

→ Integration bugs occur when the interface are incorrect b/n the assumed tested components.

→ The communication Methods used for components integration can be ___ bugs.

(a) Hardware Architecture:

→ H/w architecture has both isolated and nature during the processing of consecutive layers of OS.

→ The solution for h/w architeture bugs
(i) Good knowledge of programming and testing is needed.
(ii) centralized h/w program should be written by h/w specifiers.

(b) Software Architecture:

→ s/w architeture bugs are the types of bugs that or interact with each other.
→ Even if the procedures successfully pass unit and integration testing, they don't expose the s/w architecture bugs.

Example: To integrate the first two modules but you are integrating first and third module and started the process of testing.

7) System Bugs:

→ system bugs are revealed when system testing is performed, System bugs occur when there is a complete interaction b/n different components such program h/w and data and the OS.

(i) Control sequence bugs:

→ Neglecting time
→ Expecting the events arise in an explicit error
→ Neglecting when the requirements have been satisfied.

7) Resource Management Bugs :

→ Memory fragmented into small resources which can be allocated dynamically such as queue blocks, buffer blocks, task control blocks.

↳ Required resources are frequent.

↳ Resource assigned to the incorrect value.

↳ Ignoring to sent back a resource.

→ Solution for resource management bugs

↳ Use simpler resources structure.

↳ Use limited types of resources and resources Pool.

8) Test and Test design bugs :

→ As testers are not resistant to bugs, complex condition and data bases are required for the process of testing.

→ These complexity there are chances of occurences of bugs while a code is executing. It is a difficult task to differentiate test bugs and s/w bugs.

(1) Test debugging :

→ Debugging is the foremost step for testing bugs.

→ Test debugging and program debugging are different from each other. ~~because of~~

53

→ Test debugging is simple and easy if tests are properly designed.

(ii) Test Quality Assurance:

→ The quality of the s/w whether it satisfies the given specifications or not.

→ There are various tools that are used to test the quality of the s/w among which s/w testing is the most widely used tool.

(iii) Test Execution Automation:

→ The bug removal and prevention are identical to the other automation techniques.

→ To reduce the manual errors in the system, assemblers, loaders, computers have been developed

→ The main purpose of automation tool is ~~the~~ to removal the bugs during the process of execution.

(iv) Test Design Automation

→ The test design can be automated since all the phases of s/w development cycle are automated.

→ Because of this automation, bugs rate decreases in both s/w as well as in test.

Flow graphs and Path Testing

→ Path Testing

→ Path testing has been one of the first test methods. It is typical with box testing proposed by Tom Mccabe.

Def:

→ A sequence of statements which starts at an entry and ends at an exit passing all the existing Junctions, decisions etc. is known as path.

→ The test cases created from the set allow the program to be executed in such away as to examine each possible through the program by executing each statement atleast once.

Reasons for Implementation

→ The Path testing is implemented is provide code with a level of test coverage, that is to find out how much of a piece of s/w has been examined for faults.

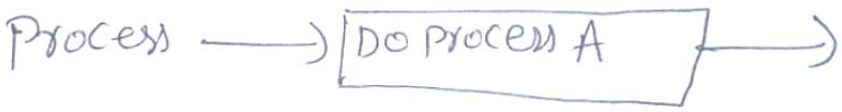
Control flow graphs

→ The control flow graph is a graphical representation of a program control structure.

1) Process Block :

→ Process block is a sequence of program statements uninterrupted by either decisions or Junctions.

→ It is a sequence of statements such that if any one statement of the block is executed, then all the statements thereof are executed.



→ A process has one entry and one exit. It can consist of a single statement or instruction, a sequence of statements or instructions, a single entry / or single exit subroutine, a macro function call, or a sequence

2) Decision and case statements

→ Decision is a program point at which the control flow can diverge.

→ Decisions can split in two-way / binary, occassionally they split in three-way branches.

→ Test case design is comparatively easy for two-way than with three way branches.

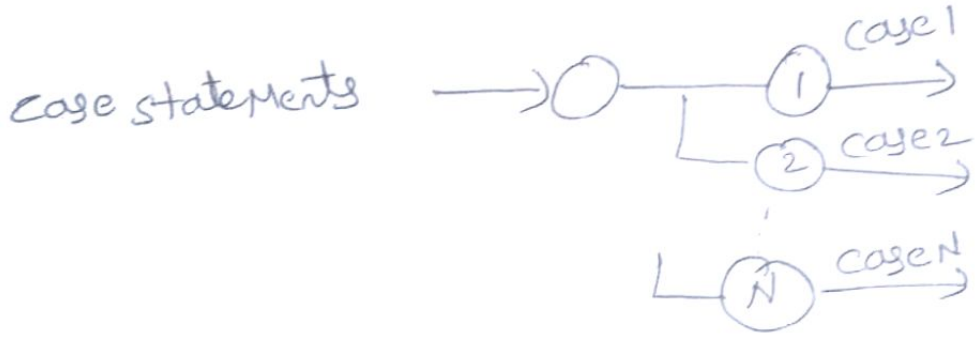
56

Case Statements :

→ Any decision can split the control flow into different way branches.

→ This multiway branches can be termed as case statements.

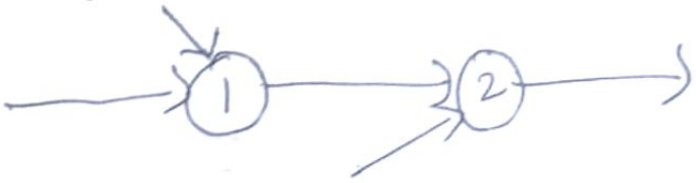
→ The designing of test cases for decision and case statements are same.



Junctions :

→ A Junction is just a contradict to decision. All the control flows can merge at a point in a program which can be termed as Junction.

→ Unconditional branches cannot be avoided despite they are unusable for programming.



Junctions

Control flow graphs versus Flow charts

Flow charts

1) Flowchart is a graph which represents the control structure of the program

2) It includes the internal structure of each and every process or process block.

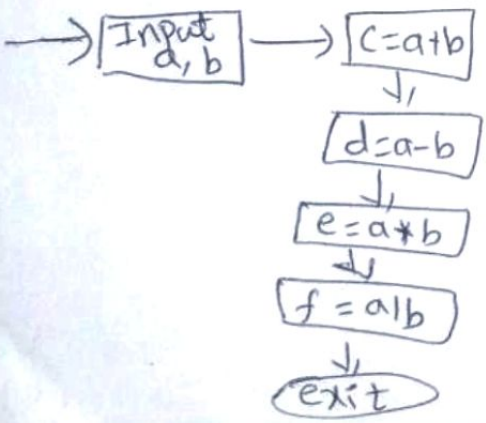
3) All the steps inside a process are shown using flowchart and also flow controls.

4) The each and every step of flow chart is represented by a rectangular box.

5) consider algorithm

```
begin
    int a,b
    c=a+b
    d=a-b
    e=a*b
    f=a/b
end
```

The flowchart for the program



Control flow graph

1) Control flow graph is also a graph which represents the control structure of the program

2) It does not include the detailed structure of process blocks.

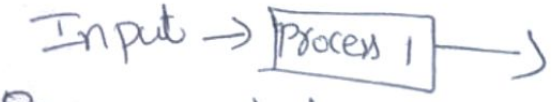
3) All the steps are considered as a single step and control flows.

4) The outline of the process block is retangular box in control flow graph.

5) consider an algorithm

```
begin
    input a,b
    c=a+b
    d=a-b
    e=a*b
    f=a/b
end
```

The control flow graph is



Process 1 includes all the five steps.

Path Testing

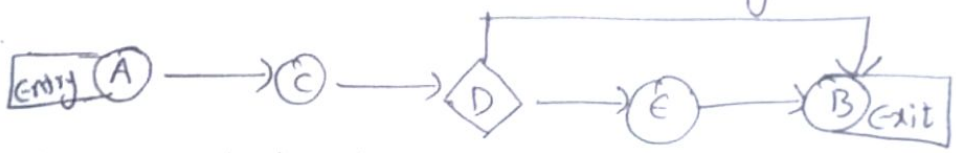
Paths :

→ A path through a program is a sequence of instructions or statements that starts at any entry, Junction or decision and ends at another or possibly the same, Junction, decision or exit.

Path segment : Several path segments combine to from a Path. It is a succession of consecutive links that belongs to same path.

Link : Link is the smallest Path segment.

Eg :



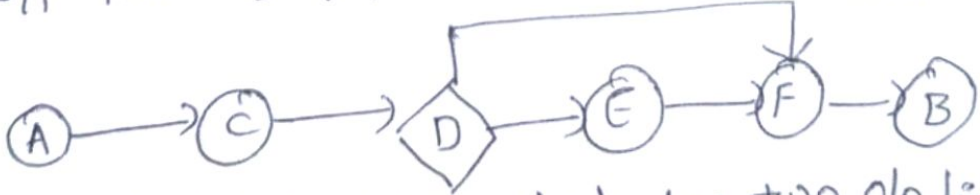
There are two different paths from an entry (A) to an exit (B) they are ACDEB & ACDB respectively.

Nodes

→ Nodes are the graphical representation of the real word entities (or) the abstract objects in any graph which resembles the objects in real world.

→ Nodes are mainly denoted by small circles.

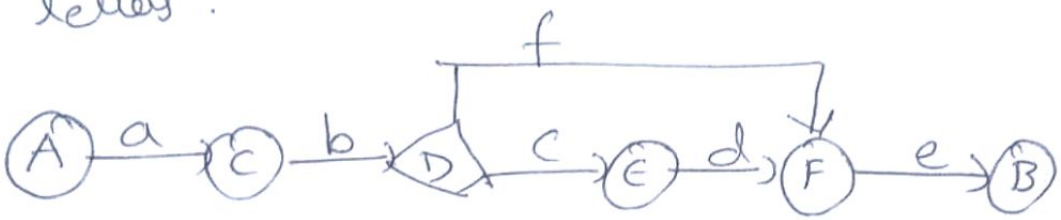
A Node which has more than one i/p link is known as a Junction, and a node which has more than one o/p link is reffered to as a decision.



→ D is the decision maker which has two o/p links, and F is a Junction which has two i/p links.

Links:

→ Every Path consists of set of processes known as links. A link is the mediator of any two nodes, links can be denoted by an 'arrow' and can be represented by the leaver case letters.



Multi-Entry / Multi-Exit Routines

→ Multi entry means, Multiple entry points and multiexit refers to multiple exit point.

→ These are certain situations in which it is appropriate to change the routine and choose an alternative way to normal control structure.

→ you may want to choose an alternative routine, when an illegitimate condition occur and will damage the systems data.

→ The other reason might to be occurrence of several fluctuations during the processing of same path.

→ Multiple choices outcomes are possible, then place an exit parameter in that routine.

Drawback:

→ The main drawback of multi-entry and multi-exit routines is that all the test cases are difficult to cover because the control flow b/n various processes can't be determined.

Fundamental path selection criteria

→ There are many paths b/n the entry and exit of a typical routine.

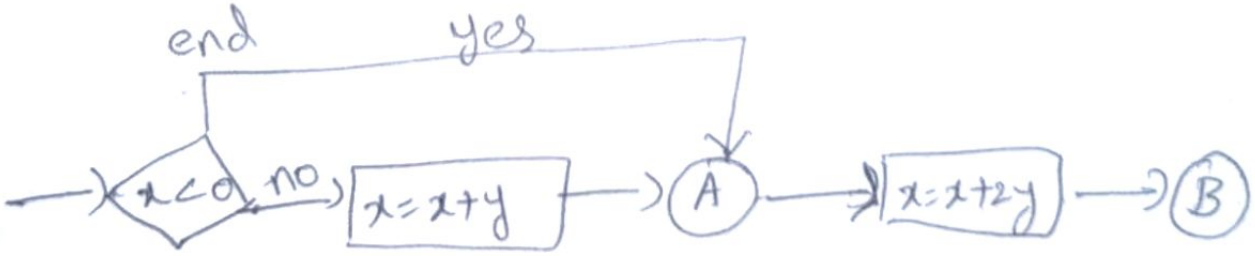
→ Every decision doubles the number of potential paths, and every loop multiplies the number of potential paths by the number of different iteration values possible for the loop.

→ Complete testing involves testing the paths in mainly three areas.

1) Testing each and every path atleast once from entry to exit.
2) Each and every statement or instruction must be executed at least once.
3) Every branch and case statement must be executed at least once in each direction.

Algorithm

```
begin
  if(x<0)GoTo A
    x := x+y
  A : x := x+2y
  B : continue
end
```



→ flow graph to describe two conditions i.e $x<0$ then $x=x+y$ else x and $x+2$ if any one of the paths is tested there would be a chance of error in the second path.

→ all the paths from decisions must be checked.

If incorrect version is considered

Modified control flowgraph

→ Hence, the correct answer is produced if x is negative and for other cases is not correct.

→ Every statement must be tested and every branch must be tested.

Path-Testing criteria

→ In testing path it is not enough to satisfy the fundamental selection. because satisfying the rules alone is not sufficient.

1) Path Testing (P_∞)
2) statement Testing (P_1)
3) Branch Testing (P_2)

1) Path Testing (P_∞)

→ All the possible paths in the program must be covered. The paths here include all the paths from entry to exit. This is the strongest criterion.

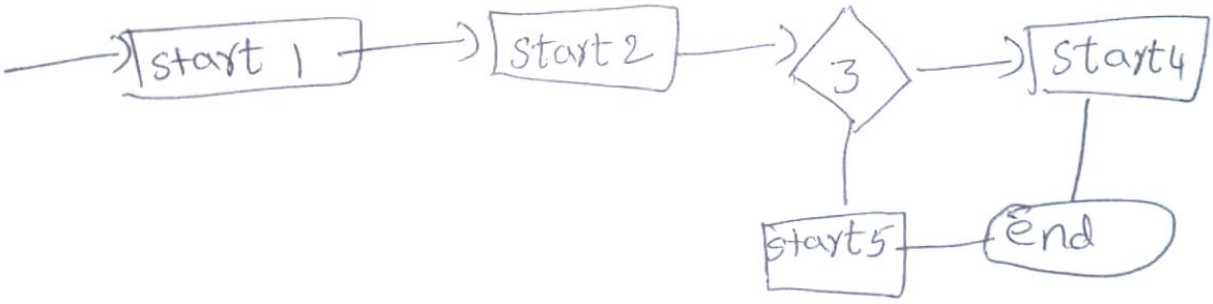
→ 100% path coverage – All the paths in the program are covered, then it is known as 100% Path coverage.

2) Statement Testing (P_1):

→ All the statement should be covered cost once. This is the weakest criteria and any new s/w should not be tested less than.

100% statement coverage: All the statements are covered then we can say it a 100% Statement coverage. It can also be called 100% node coverage.

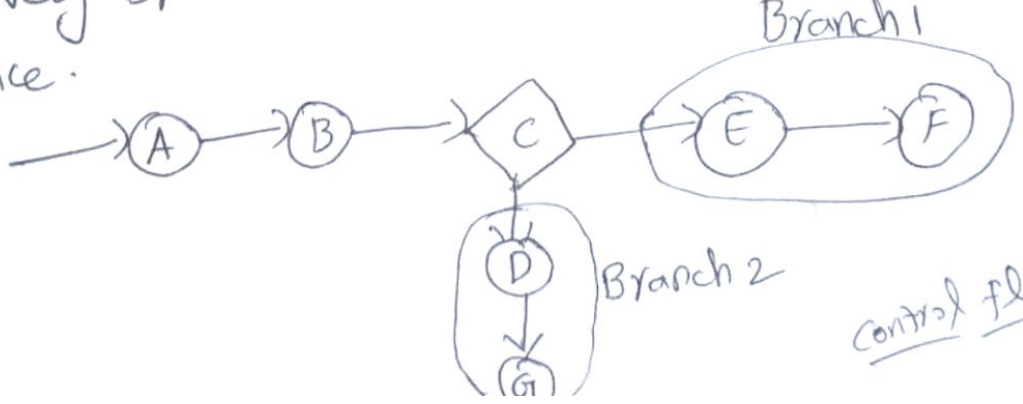
ex:



3) Branch Testing (P_2)

→ Every branch alternative should be tested atleast once.

ex:



Control flow graph

63

Which Paths

The various path selection rules defined for path testing

Rule 1: Select those paths which are very simple and functionality sensible. The path should be from entry to exit.

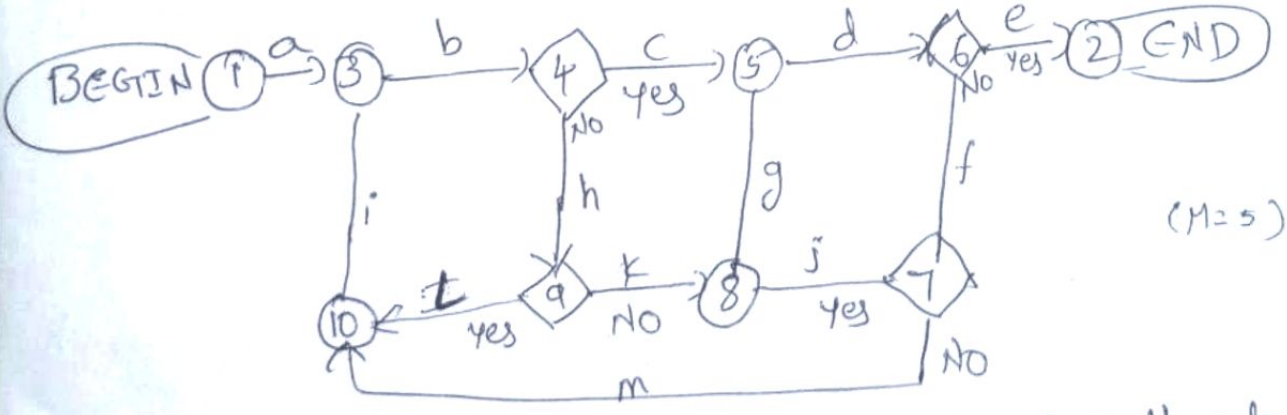
Rule 2: Select few additional paths that are different from rule 1 paths. The paths selected needs

1) free from loops
2) short
3) simple
4) sensible.

Rule 3: To provide the coverage, select those paths that has no functional meanings.

Rule 4: The paths choosen needs to provide complete coverage.

Rule 5: All the above rules must be followed only for coverage of all paths.



Example of path selection

→ The most obvious path is (1,3,4,5,6,2) if we name it by nodes or abcde. All other paths in this example lead to loops.

→ Take a simple loop first building, if possible on a previous paths, such as abhlibcde.

→ Take another loop abcdfjgde. and finally adcdfmibcde.

Practical suggestions

(i) The control flow graph on a single sheet of paper.

(ii) Make duplicate copies it.

(iii) highlightening marker for tracking path. These paths on to master sheet.

(iv) This process should be continued till all the lines on master sheet are covered.

Paths	Decisions				Process-Link											
	4	6	7	9	a	b	c	d	e	f	g	h	i	j	k	l	m
abcde	yes	yes			✓	✓	✓	✓	✓								
abhkgde	No	yes		No	✓	✓		✓	✓		✓	✓			✓		
abhlibcde	No, yes	yes		yes	✓	✓	✓	✓	✓			✓	✓			✓	
abcdfjgde	yes	No, yes	yes		✓	✓	✓	✓	✓	✓	✓			✓			
abcdfmibcde	yes	No, yes	No		✓	✓	✓	✓	✓	✓		✓					✓

65

Loops

→ The sequence of few steps number of times in a program is said to be a loop.

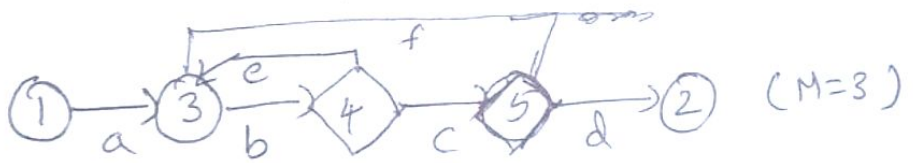
→ Loops mainly reduces the length of code and increases the reusability.

→ Three kinds of loops

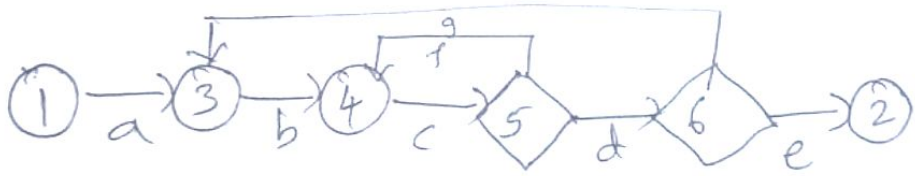
1) Nested loop
2) Concatenated loops
3) Horrible loops.

1) Nested loops:

The loop with in another loop is known as nested loop. It is a quite complex.



(M=3)



(M=3)

→ If you had five tests for one loop, a pair of nested loops would require 25 tests, and three nested loops would require 125.

Step 1: The process of testing from the most inner loop. This must be done by setting all other outer loops to their minimum values.

Step 2: Keeping the conditions of step 1 in mind the testing for inner loop must be done. ~~to the~~.

minimum value
(min+1) value
Typical value
(max-1) value
max value.

Step 3: After testing the inner most loop, come out and conduct a test on the immediate outer loop.

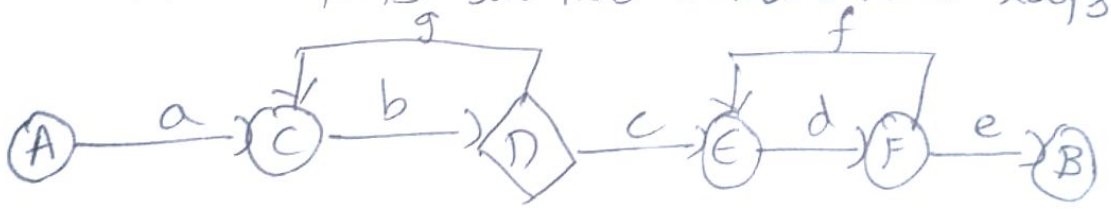
Step 4: Stop the testing once all loops have completed.

Step 5: Restart the process of testing from step 1 to step 4 if there is any other nested loop in the same path.

2) Concatenated Loops:

-> Concatenated loops are the loops which reside one beside other on the same path.

-> The loops are not on the same path they are said to be a individual loops but not concatenated loops.

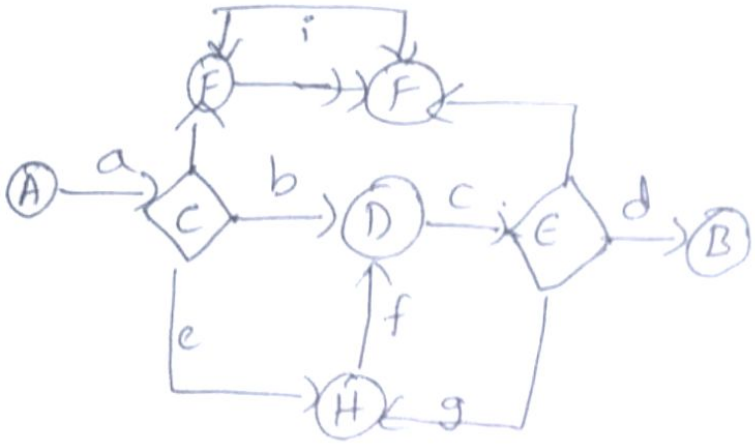


-> repetition value depends on the repetition value other loop and both i.e on same path they can be termed as nested loops.

3) Horrible loops

-> Horrible loops are the complexed of all the three loops and may involve nested loops, intersecting loops, cross connected loops allin one structure.

ex:



More on Testing Multi-entry/Multi-exit Routines

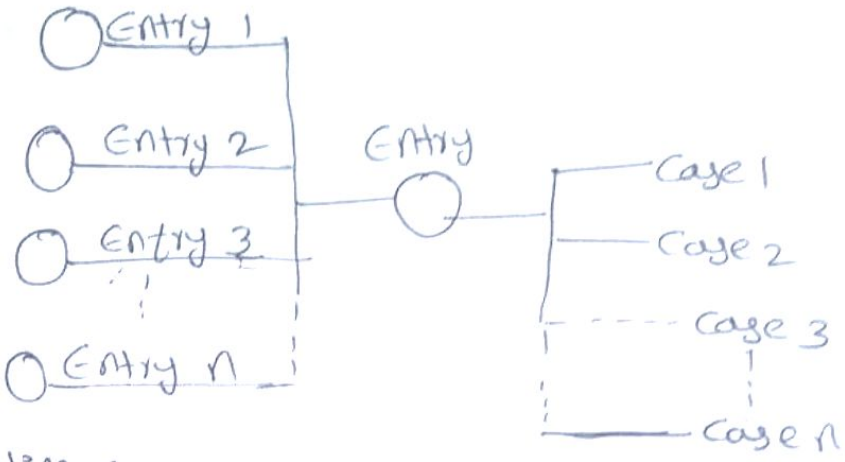
1) Weak Approach:

-> To test the program with multi-entry and multi-exit routines.

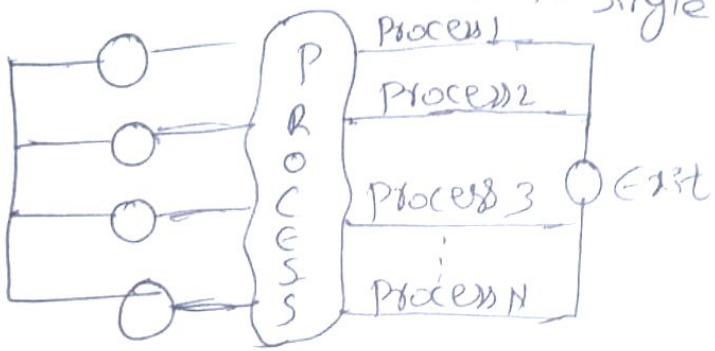
-> First build the imaginary single entry routine and imaginary exit routin with Pseudo case statements and processes respectively.

-> It's a complicated approach because, it requires you to find detailed information about all label references associated with multi-entries.

(i) Multi entry routine is converted to single entry with case statements.



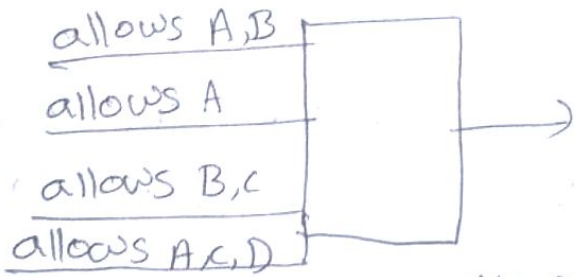
(ii) Multi-exit routine is converted to single exit with processes



Multi Exit to single exit

2) Integration Testing Issue

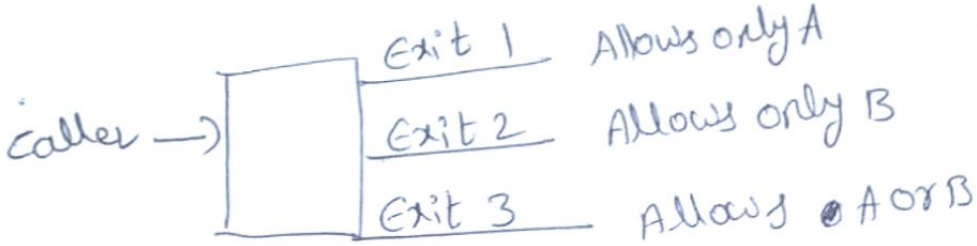
→ Weak approach does not concentrate on interfaces and called components, which are the major aspects.



Multi-entry components

→ Unit testing alone is not sufficient in this case. Integration testing focuses or interfaces and restrictions on number of inputs.

69



→ used by o/p's A & B such that exit 1 allows only o/p A, exit 2 allows only o/p B & exit 3 allows o/p's A or B.

3) Theory and Tools Issue :

→ A well formed s/w is a s/w which has single entry & single exit with a rigid structure.

4) strategy summary

→ In these routines, there is no direct connection b/n entry and exit.

→ The control flow is managed by receiving parameter values of entry/exit points.

Path testing strategies :

(i) Avoid multi-entry / multi exit routines as much as possible because it is difficult to cover all the test cases for these routines.

(ii) It is mandatory for you to use multi-entry/ multi exit routines try to achieve max control over these routines.

Effectiveness of Path Testing

(i) Effectiveness and Limitations

→ Unit testing is comparatively stronger than path testing which inturn is stronger than statement and branch Testing.

→ Path testing capture about 50% of bugs which the Unit testing captures. Unit testing can catch upto 65% of bugs of overall structure.

Apart from effectiveness, path testing also has certain limitations.

(i) Path testing has to be combined with other methods to improve the paform-ance in terms of percentage.

(ii) All the paths may not be covered if there is a bug.

(iii) Path testing doesn't deal with specification errors.

2) Working: Path testing involves alot of work i.e

(i) Development of control flow graph.

(ii) choosing a route that can over all the paths, decisions and Junctions in a flow graph.

(iii) writing test cases of loops.

Predicates, Path predicates and Achievable paths

Predicates:

→ The direction taken at a decision depends on the value of decision variables.

→ For binary decisions, decision processing ultimately results in the evaluation of logical functions whose outcome is either TRUE or FALSE.

→ The logical function evaluated at a decision is called a predicate.

→ A set of values associated with a path is known as path predicate.

Predicate Expressions

(i) Predicate Interpretation

→ A path predicate may contain local variables. They play no role in seleding i/p's that force a path to execute.

→ Predicate Interpretation can be defined as the process of "symbolically substituting operations along the path in order to express the predicate soley in terms of the i/p vector and a constant vector.

(ii) Independence and correlation of variables and predicates

→ The path predicates take on truth values (True / False) based on the values of i/p variables, either directly or indirectly.

→ If a variables value does not change as a result of processing, that variable is independent of the processing.

Path predicate Expression:

→ These expressions are the collection of expression that must be fulfilled in order to achieve the desired path. An interpreted path predicate is called path Predicate expression.

Ex: A = 18

B + 5C + 2 > 0

D - B >= 10C

Let i/p values B, C, D be 2, 1, 12 respectively.

A = 18

B + 5(1) + 2 > 0

=> 2 + 5 + 2 > 0 => 9 > 0

D - B >= 10C

=> 12 - 2 = 10(1) => 10 >= 10

All the conditions appears to be correct as per the values, so this path can be chosen.

Predicate coverage:

→ Predicate coverage is the process of testing all the truth values related to a specific path in all the possible ways.

→ If all the values are tested in all possible directions then we can say that 100% predicate coverage is achieved which needs lots of efforts.

Testing Blindness

→ Testing blindness is a pathological situation in which the desired path is achieved for the wrong reason.

→ The three kinds of predicate blindness are

1) Assignment Blindness
2) Equality Blindness
3) self Blindness.

1) Assignment Blindness

→ The buggy Predicates appears to work correctly because the specific value chosen for an assignment statements work with both the correct and incorrect predicate.

Ex:

correct	Buggy
X := 7	X := 7
...	
If Y > 0 THEN	If X+Y > 0 THEN

→ If the test case sets y := 1 the desired path is taken in either case, but there is still a bug.

2) Equality Blindness

→ Equality blindness occurs when the path selected by a prior predicate results in a value that works both for the correct & buggy predicate.

Ex:

correct	Bug
If y = 2 Then	If y = 2 Then ---
...	
If x+y > 3 Then	If x > 1 Then --

5) Self blindness:

→ The buggy predicate is a multiple of the correct predicate and as a result is indistinguishable along that path.

Ex.

Correct	Bug
X := A	X := A
IF X - 1 > 0 Then	If X + A - 2 > 0 Then

PATH SENSITIZING

→ Path predicate expressions are the collection of expressions that must be fulfilled in order to achieve the desired path.

Heuristic procedure for sensitizing paths

→ It is a workable approach. Instead of selecting the paths without considering how to sentize., attempt to choose a covering path set that is easy to sentize and pick hard to sensitize paths..

1) All the process dependent, process independent and correlated i/p varibles and first determined and classified.

2) After classifying the variables, determine and classify the predicates depending on the i/p variables into dependent, independent or correlated predicates.

3) Consider the uncorreleated and independent predicates for selection of path..

4) Now consider the correleated and independent predicates if they are not covered then start considering the dependent & uncorreleated, predicates.

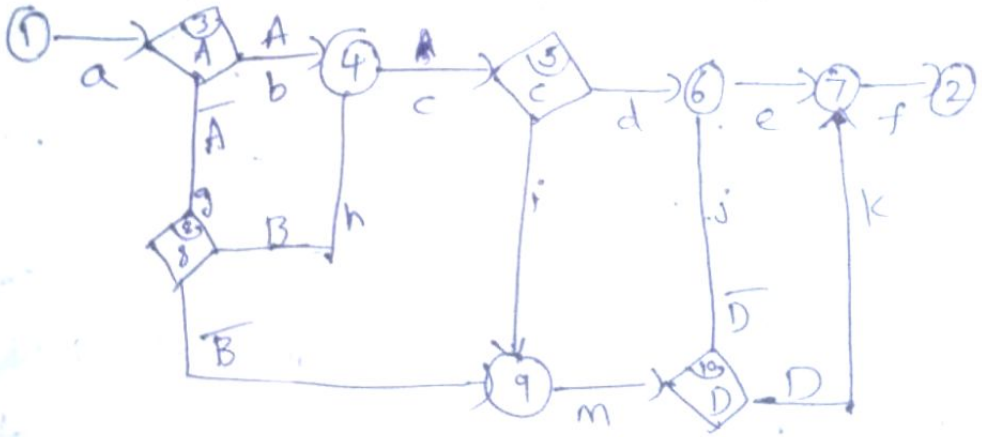
5) Display all the i/p's variables, its values. relation shing among the variables, types of links for all independent, d

Examples:

(i) simple, Independent, Uncorrelated predicates :-

Path	Predicate values
abcdef	$A\bar{C}$
aghcimkf	$\bar{A}BCD$
aglmjef	$A\bar{B}\bar{D}$

A glace at the path column all links are represented at least once. The predicate value column all predicates appearing both barred and unbarred.



(M=5)

Predicate Notation

Path	Predicate Values
abcdef	$A\bar{C}$
abcimjef	$AC\bar{D}$
abcimkf	ACD
aghcdef	$\bar{A}B\bar{C}$
aglmkf	$\bar{A}\bar{B}\bar{D}$

Path Instrumentation

→ Path instrumentation is the technique that is used to confirm whether outcome of the test achieved by the desired path or Wrong path. It is an interpretive trace program.

→ An interpretive trace program is nothing but the one which executes each & every statement in order and thereby sorting all the lables and the values of the statements covered till now.

Drawback:

The trace program is that it hay additional large information content which is of no use.

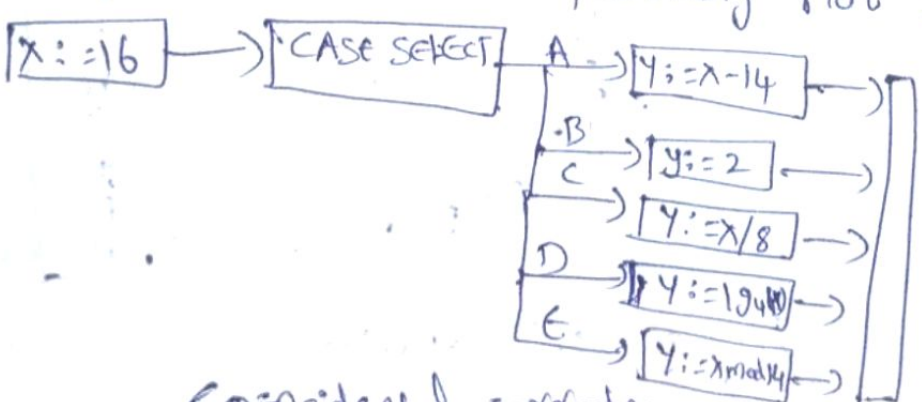
Coincidental Correctness

→ The terms related to this problem & needed to be understood.

Problem: During the process of testing, the actual outcome of the test and the predicated outcame are observed. Even if they match we cannot say that the test has been passed.

(i) Necessary conditions for test to pass or met.
(ii) Conditiony Met are probably not sufficient.

Ex:



Coincidental correctness

→ i/p value (X=16), yield the same outcome (Y=2) no matter which case we select.

→ The tests choosen this way will not tell us whether we have achieved coverage.

Link Markers

→ Link Marker is a form of path instrumentation. It is also known as traversal marker. It is more effective & simple.

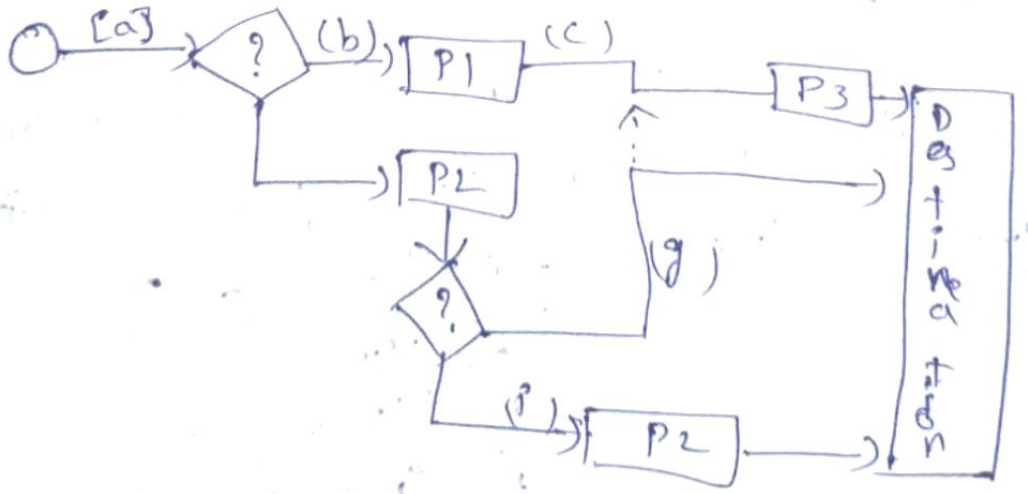
1) single marker
2) Double link marker.

1) Single Link Marker:

→ Every link is named by the lower case letters. Links are instrumented such that the link's name is recorded when a particular link is executed.

→ The concatenation of the names of all links starting from an entry to exit gives path name if there are no bugs.

Ex:

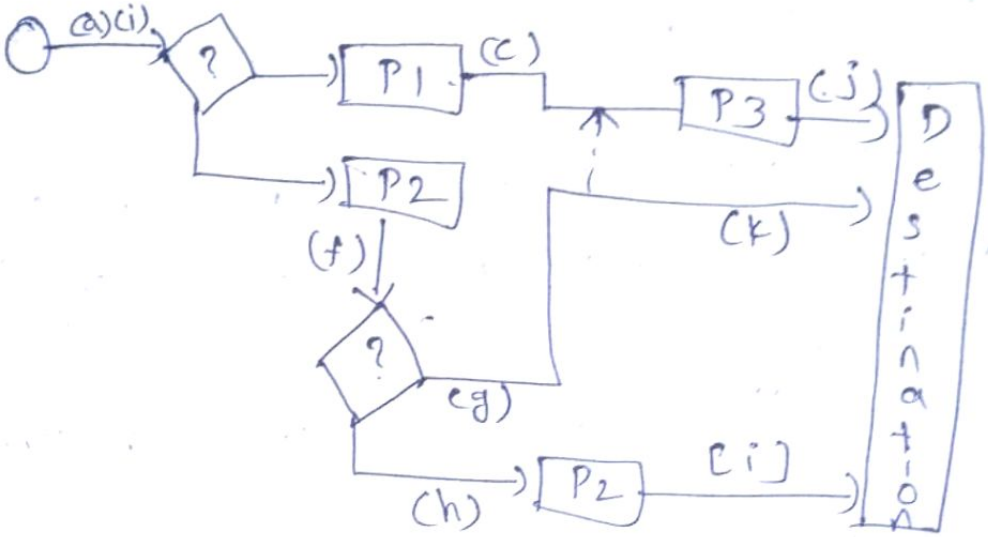


Single link Marker

2) Double link Marker :

-) to implement two markers per link, one at starting and one at end of the path.

Ex :



Double link Marker

Link counter :

-) Link counter is one of the instrumentation techniques which usually relies on the concept of counters.

-) Link counter instrumentation Method provides Comparatively less information than interpretive trace Program.

-) Its follows the same procedure as that of link marker but, make use of counters instead of using labels for each link which hay executed.

Other instrumentation Methods

-) General Methods of instrumentation other than link Marker and Link counters.

1) one of the Methods is to mark each link by a

unique prime number and multiple the link name into central register.

2) Bitmap: This Method is to use a bitmap with a single bit per link. The bit is set when the link is traversed.

3) Hash coding: This is to use the hash coding over the link names or calculate the error detecting code.

4) Symbolic Debugger:
This is to trace only sub routines & return.

Implementation and Application of Path Testing

1) Integration, coverage and paths in called components

→ The technique of path testing is employed mainly for new s/w's in unit testing.

→ The stubs are used in classical unit testing in replacing all the called components and other components so that a new component can be tested individually.

→ The process of path testing that is carried out at this phase is to analyze the control flow errors than focusing on the bugs in called and corequisite components.

80

2) New node:

When the code is modified or when it is completely new, that code should be tested using Path testing to achive C2.

→ The bug potential for the stub is lower than the called component then stubs are used:

→ The statement indicators that the components which are old and trusted will not be replaced by stubs.

3) Maintenance:

→ The Path testing is applied on modified components of new s/w but the other components such as called and corequisite components will not be modified and they are kept unchanged.

4) Rehosting:

→ Path testing with C1+C2 coverage is a powerful tool for rehosting old s/w.

→ A translator from the old to the new environment is created and tested as any piece of s/w.

→ The bugs in the rehosting process, if any will be in the translation algorithm and the translator, and the rehosting process is intended to catch those bugs.

81