A

Scoring Indicators

COURSE NAME: SOFTWARE TESTING

COURSE CODE: TED (21) – 6131C **QID:** 2102240138

Q. No.		Scoring Indicators	Split Score	Sub Total	Total Score
PART A					9
I	1	Black box testing and white box testing		1	
I	2	New, open, Assign, Deferred, rejected, test, verified, reopened, closed *(Write any four)*		1	
I	3	Valid equivalence classes. Invalid equivalence classes		1	
I	4	21 *(4n+1)*		1	
I	5	Inspections, Walkthroughs, Technical Reviews *(Write any two)*		1	
I	6	Author		1	
I	7	Unit testing (Unit validation testing)		1	
I	8	navigate() or to() *(Write any one)*		1	
I	9	Loadrunner or Jmeter *(or any other load testing tool)*		1	
PART B					24
II	1	Failure is the inability of a system or component to perform a required function according to its specification. Fault is a condition that in actual courses a system to produce failure Whenever a development team member makes a mistake in any phase of SDLC, errors are produced.		3	
II	2	**Requirements and specifications bugs** The first type of bug in SDLC is in the requirement gathering and specification phase. It has been observed that most of the bugs appear in this phase only. **Design bugs** Design bugs maybe the bugs from the previous phase and in addition which are introduced in the present phase. **Coding bugs** Coding bugs can occur due to unclear data, unclear routines, typographical errors, documentation bugs, etc. **Interface and integration bugs** Integration bugs results from inconsistencies or incompatibilities between modules discussed in the form of interface bugs. **System Bugs** There may be bugs while testing the system as a whole based on various parameters like performance, stress, compatibility, usability, etc. **Testing bugs** Are failure to notice/report a problem, failure to use the most promising test case, failure to make it clear how to reproduce a bug, failure to verify fixes, failure to provide summary reports, etc.		3	
II	3	Decision table is a useful method to represent the information in a tabular method. Decision tables obtain their power from logical expressions. Each operand or variable in a logical expression takes on the value, TRUE or FALSE. *FORMATION OF DECISION TABLE* A decision table is formed with the following components Decision table structure **Condition stub** It is a list of input conditions for which the complex combination is made.		3	

		Action stub It is a list of resulting actions which will be performed if a combination of input condition is satisfied. **Condition entry** It is a specific entry in the table corresponding to input conditions mentioned in the condition stub. When we enter TRUE or FALSE for all input conditions for a particular combination, then it is called a Rule. Thus, a rule defines which combination of conditions produces the resulting action. **Action entry** It is the entry in the table for the resulting action to be performed when one rule (which is a combination of input condition) is satisfied. 'X' denotes the action entry in the table.			
II	4	Equivalence partitioning is a method for deriving test cases wherein classes of input conditions called equivalence classes are identified such that each member of the class causes the same kind of processing and output to occur. Instead of testing every input, only one test case from each partitioned class can be executed. This test case will have a representative value of a class which is equivalent to a test case containing any other value in the same class. If one test case in an equivalence class detects a bug, all other test cases in that class have the same probability of finding bugs. Therefore instead of taking every value in one domain, only one test case is chosen from one class. In this way, testing covers the whole input domain, thereby reduce, the total number of test cases. It is an attempt to get a good hit rate to find maximum errors with the smallest number of test cases. **Goals:** **Completeness** Without executing all the test cases, we strive to touch the completeness of testing domain. **Non-redundancy** When the test cases are executed having inputs from the same class, then there is redundancy in executing the test cases. Time and resources are wasted in executing these redundant test cases, as they explore the same type of bug. To use equivalence partitioning, one needs to perform two steps: 1. Identify equivalence classes 2. Design test cases		3	
II	5	Structural testing considers the program code, and test cases are designed based on the logic of the program such that every element of the logic is covered. Therefore the intention in white-box testing is to cover the whole logic. **Statement Coverage** The first kind of logic coverage can be identified in the form of statements. It is assumed that if all the statements of the module are executed once, every bug will be notified. **Decision or Branch Coverage** Branch coverage states that each decision takes on all possible outcomes (True or False) at least once. In other words, each branch direction must be traversed at least once. **Condition Coverage** Condition coverage states that each condition in a decision takes on all possible outcomes at least once. **Decision/condition Coverage** Condition coverage in a decision does not mean that the decision has been covered.		3	
II	6	Alpha is the test period during which the product is complete and usable in a test environment, but not necessarily bug-free. It is the final chance to get verification from the customers that the tradeoffs made in the final development stage are coherent. Therefore, alpha testing is typically done for two reasons:		3	

(i) to give confidence that the software is in a suitable state to be seen by the customers (but not necessarily released).

(ii) to find bugs that may only be found under operational conditions. Any other major defects or performance issues should be discovered in this stage.

Since alpha testing is performed at the development site, testers and users together perform this testing. Therefore, the testing is in a controlled manner such that if any problem comes up, it can be managed by the testing team. Once the alpha, phase is complete, development enters the beta phase. Beta is the test period during which the product should be complete and usable in a production environment. The purpose of the beta ship and test period is to test the company's ability to deliver and support the product (and not to test product itself). Beta also serves as a chance to get a final 'vote of confidence from a few customers to help validate our own belief that the product is now ready for volume shipment to all customers.

II

7

Test case Number	Test Description	Steps	Steps to execute the test cases	expected output	actual output
TC_01	Login to the application with correct user id and correct password	1	Launch the application	Login page should be displayed	
		2	Enter correct user id and correct password and try to login	System should allow the user to login to the application	
TC_02	Login to the application with incorrect user id and correct password	1	Launch the application	Login page should be displayed	
		2	Enter incorrect user id and correct password and try to login	System **should not** allow the user to login to the application	
TC_03	Login to the application with correct user id and correct password	1	Launch the application	Login page should be displayed	
		2	Enter correct user id and correct password and try to login	System **should not** allow the user to login to the application	

3

II	8	**Stubs** The module under testing may also call some other module which is not ready at the time of testing. Therefore, these modules need to be simulated for testing. In most cases, dummy modules instead of actual modules, which are e not ready, are prepared for these subordinate modules. These dummy modules are called *stubs* **Drivers** Suppose a module is to be tested, wherein some inputs are to be received from another module. However, this module which passes inputs to the module to be tested is not ready and under development. In such a situation, we need to simulate the inputs required in the module to be tested. For this purpose, a main program is prepared, wherein the required inputs are either hard-coded or entered by the user and passed on to the module under test. This module where the required inputs for the module under test are simulated for the purpose of module or unit testing is known as a driver module. The driver module may print or interpret the results produced by the module under testing.		3	
II	9	Mercury Interactive's WinRunner It is a tool used for performing functional /regression testing. It automatically creates the test scripts by recording the user interactions on GUI of the software. These scripts can be run repeatedly whenever needed without any manual intervention. The test scripts can also be modified if required because there is support of Test Script language (TSL) with a 'C' like syntax. There is also provision for bringing the application to a known state if any problem has occurred during automated testing. WinRunner executes the statements by default with an interleaving of one second. But if some activities take more time to complete, then it synchronizes the next test case automatically by waiting for the current operations to be completed. Segue Software's SilkTest This tool is also for functional/regression testing. it supports 4Test as a scripting language which is an object-oriented scripting language. SilkTest has a provision for customized in-built recovery system which helps in continuing the automated testing even if there is some failure in between. IBM Rational SQA Robot It is another powerful tool for functional/regression testing. Synchronization of test cases with a default delay of 20 seconds is also available. Mercury Interactive's LoadRunner This tool is used for performance and load testing of a system. Generally, the tool is helpful for client/server applications of various parameters with their actual load like response time, the number of users, etc. The major benefit of using this tool is that it creates virtual users on a single machine and tests the system on various parameters. Thus, performance and load testing is done with minimum infrastructure. *(Write any 2)*		3	
II	10	**Reduction of testing effort** In verification and validation strategies, numerous test case design methods have been studied. Test cases for a complete software may be hundreds of thousands or more in number. Executing all of them manually takes a lot of testing effort and time. Thus, execution of test suits through software tools greatly reduces the amount of time required. **Reduces the testers' involvement in executing tests** Sometimes executing the test cases takes a long time. Automating this process of executing the test suit will relieve the testers to do some other work, thereby increasing the parallelism in testing efforts. **Facilitates regression testing**		3	

		As we know, regression testing is the most time-consuming process. If we automate the process of regression testing, then testing effort as well as the time taken will reduce as compared to manual testing. **Avoids human mistakes** Manually executing the test cases may incorporate errors in the process or sometimes, we may be biased towards limited test cases while checking the software. Testing tools will not cause these problems which are introduced due to manual testing. **Reduces overall cost of the software** As we have seen that if testing time increases, cost of the software also increases. But due to testing tools, time and therefore cost can be reduced to a greater level as testing tools ease the burden of the test case production and execution. **Simulated testing** Load performance testing is an example of testing where the real-life situation needs to be simulated in the test environment. Sometimes, it may not be possible to create the load of a number of concurrent users or large amount of data in a project. Automated tools, on the other hand, can create millions of concurrent virtual users/data and effectively test the project in the test environment before releasing the product. **Internal testing** Testing may require testing for memory leakage or checking the coverage of testing. Automation tools can help in these tasks quickly and accurately, whereas doing this manually would be cumbersome, inaccurate, and time-consuming. **Test enablers** While development is not complete, some modules for testing are not ready. At that time, stubs or drivers are needed to prepare data, simulate environment, make calls, and then verify results. Automation reduces the effort required in this case and becomes essential. **Test case design** Automated tools can be used to design test cases also. Through automation, better coverage can be guaranteed, than if done manually. *(Explain any 3)*			
		PART C			42
III		1. Short Term or Immediate Goals These goals may be set in the individual phases of SDLC. 1.1Bug Discovery The immediate goal of software testing is to find errors at any stage of software development. More bugs discovered at an early stage, better will be the success rate of software testing. 1.2Bug Prevention From the behavior and interpretation of bugs discovered, the software development team gets to learn how to code safely such that the bugs discovered should not be repeated in later stages or projects. Thus we can minimize the bugs. 2. Long Term Goals. These goals affect the product quality in the long run. 2.1Quality Thorough testing ensures high quality of the software products. Understanding and performing the testing process is to enhance the quality of the software product. Quality depends on various factors, such as correctness, integrity, efficiency, etc. Reliability is the major factor to achieve quality.		7	

2.2Customer satisfaction

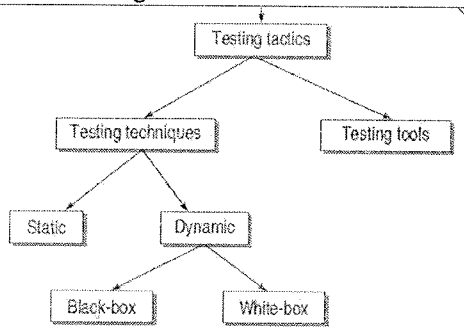
From the user's perspective, the prime concern of testing is customer satisfaction only. A complete testing process achieves reliability, reliability enhances the quality and quality in turn increases the customer satisfaction.

2.3Risk management

Risk is the probability that Undesirable event will occur in a system. The purpose of software testing as a control is to provide information to management so that they can better react to risks situations.

It is the testers' responsibility to evaluate business risks and make the same a basis for testing choices. So risk management becomes the long term goal for software testing.

IV



Static Testing

It is a technique for assessing the structural characteristics of source code, design specifications or any notational representation that conforms to well defi ned syntactic rules

Dynamic Testing

All the methods that execute the code to test a software are known as dynamic testing techniques. In this technique, the code is run on a number of inputs provided by the user and the corresponding results are checked. This type of testing is further divided into two parts:

(a) black-box testing and

(b) white-box testing.

Testing Tools

Testing tools provide the option to automate the selected testing technique with the help of tools. A tool is a resource for performing a test process.

7

V

Software Testing Life Cycle(STLC)

The testing process divided in to a well-defined sequence of steps is termed as software testing life cycle(STLC). The major contribution of STLC is to involve the testers at early stages of development. This has a significant benefit in the project schedule and cost. The STLC also helps the management in measuring specific milestones.

STLC consists of the following phases.

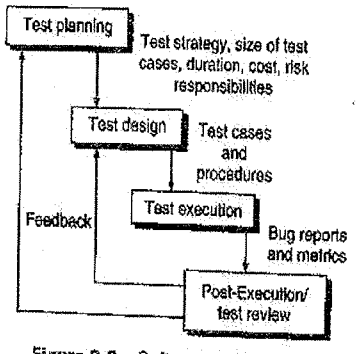


Figure 2.8 Software testing life cycle

Test Planning

7

		The goal of test planning is to take into account the important issues of testing strategy, viz. resources, schedules, responsibilities, risks, and priorities, as a roadmap. Test planning issues are in tune with the overall project planning. **Test Design** One of the major activities in testing is the design of test cases. It is a well-planned process. The test design is an important phase after test planning. It includes the following critical activities. **Determining the test objectives and their prioritization** The test objectives reflect the fundamental elements that need to be tested to satisfy an objective. **Preparing list of items to be tested** The objectives thus obtained are now converted into lists of items that are to be tested under an objective. **Mapping items to test cases** A matrix can be created for this purpose, identifying which test case will be covered by which item **Creating test cases and test data** The test cases mention the objective under which a test case is being designed, the inputs required, and the expected outputs. While giving input specifications, test data must also be chosen and specified with care, as this may lead to incorrect execution of test cases. **Test Execution** In this phase, all test cases are executed including verification and validation. Test results are documented in the test incident reports, test logs, testing status, and test summary reports **Post-Execution/Test Review** After successful test execution, bugs will be reported to the concerned developers. This phase is to analyze bug-related issues and get feed-back so that maximum number of bugs can be removed.			
VI		**VALIDATION ACTIVITIES** The validation activities are divided into Validation Test Plan and Validation Test Execution which are described as follows: **Validation Test Plan** On the basis of the understanding of SDLC phase and related documents, testers must prepare the related test plans which are used at the time of validation testing. Under test plans, they must prepare a sequence of test cases for validation testing. The following test plans have been recognized which the testers have already prepared with the incremental progress of SDLC phases: **Acceptance test plan** This plan is prepared in the requirement phase according to the acceptance criteria prepared from the user feedback. This plan is used at the time of Acceptance Testing. **System test plan** This plan is prepared to verify the objectives specified in the SRS. Here, test cases are designed keeping in view how a complete integrated system will work or behave in different conditions. The plan is used at the time of System Testing. **Function test plan** This plan is prepared in the HLD phase. In this plan, test cases are designed such that all the interfaces and every type of functionality can be tested. The plan is used at the time of Function Testing. **Integration test plan** This plan is prepared to validate the integration of all the modules such that all their interdependencies are checked. It also validates whether the integration is in conformance to the whole system design. This plan is used at the time of Integration Testing. **Unit test plan** This plan is prepared in the LLD phase. It consists of a test plan of every module in the system separately. Unit test plan of every unit or module is designed such that every functionality related to individual unit can be tested. This plan is used at the time of Unit Testing. **Validation Test Execution**		7	

		Validation test execution can be divided in the following testing activities: **Unit validation testing** Unit testing is a process of testing the individual components of a system. A unit or module must be validated before integrating it with other modules. Unit validation is the first validation activity after the coding of one module is over. **Integration testing** It is the process of combining and testing multiple components or modules together. The individual tested modules, when combined with other modules, are not tested for their interfaces. **Function testing** Function testing is to explore the bugs related to discrepancies between the actual system behavior and its functional specifications.			
VII		BOUNDARY VALUE CHECKING (BVC) In this method, the test cases are designed by holding one variable of its extreme value and other variables at their nominal values in the input domain. The variable at its extreme value can be selected at: (a) Minimum value (Min) (b) Value just above the minimum value (c) Maximum value (d) Value just below the maximum value (Max) Let us take the example of two variables, A and B. If we consider all the above combinations with nominal values, then following test cases (see Fig. 4.3) can be designed: 1. A_{nom}, B_{min} 2. A_{nom}, B_{min+} 3. A_{nom}, B_{max} 4. A_{nom}, B_{max+} 5. A_{min}, B_{nom} 6. A_{min+}, B_{nom} 7. A_{max}, B_{nom} 8. A_{max-}, B_{nom} 9. A_{nom}, B_{nom} It can be generalized that for *n* variables in a module, *4n + 1* test cases can be designed with boundary value checking method. ROBUSTNESS TESTING METHOD The idea of BVC can be extended such that boundary values are exceeded as • A value just greater than the Maximum value (Max+) • A value just less than Minimum value (Min-) When test cases are designed considering the above points in addition to BVC, it is called ***robustness testing.*** Let us take the previous example again. Add the following test cases to the list of 9 test cases designed in BVC 10. A_{max+}, B_{nom} 11. A_{min-}, B_{nom} 12. A_{nom}, B_{max+} 13. A_{nom}, B_{min-} It can be generalized that for *n* input variables in a module. ***6n + 1*** test cases can be designed with robustness testing.		7	
VIII		**Test cases using BVC** Since there are three variables, *A*, *B*, and *C*, the total number of test cases will be $4n + 1 = 13$. The set of boundary values is shown below: Min value = 1 Min+ value = 2 Max value = 50 Max- value = 49 Nominal value = 25–30 Using these values, test cases can be designed		7	

Test Case ID	A	B	C	Expected Output
1	1	25	27	C is largest
2	2	25	28	C is largest
3	49	25	25	B and C are largest
4	50	25	29	A is largest
5	25	1	30	C is largest
6	25	2	26	C is largest
7	25	49	27	B is largest
8	25	50	28	B is largest
9	25	28	1	B is largest
10	25	27	2	B is largest
11	25	26	49	C is largest
12	25	26	50	C is largest
13	25	25	25	Three are equal

IX

There are three different techniques for regression testing. They are discussed below.

Regression test selection technique This technique attempts to reduce the time required to retest a modified program by selecting some subset of the existing test suite.

Testcase prioritization technique Regression test prioritization attempts to reorder a regression test suite so that those tests with the highest priority, according to some established criteria, are executed earlier in the regression testing process rather than those with lower priority. There are two types of prioritization:

(a) General Test Case Prioritization For a given program P and test suite T, we prioritize the test cases in T that will be useful over a succession of subsequent modified versions of P, without any knowledge of the modified version.

(b) Version-Specific That Case Prioritization We prioritize the test cases in T, when P is modified to P', with the knowledge of the changes made in P.

Test Suite reduction technique It reduces testing costs by permanently eliminating redundant test cases from test suites in terms of codes or functionalities exercised.

7

X

Design hierarchy of a software can be seen in a tree-like structure. In this tree-like structure, incremental integration can be done either from top to bottom or bottom to top. Based on this strategy, incremental integration testing is divided into two categories.

Top-down Integration Testing

The strategy in top-down integration is to look at the design hierarchy from top to bottom. Start with the high-level modules and move downward through the design hierarchy.

Top-Down Integration Procedure

The procedure for top-down integration process is discussed in the following steps:

1. Start with the top or initial module in the software. Substitute the stubs for all the subordinate modules of top module. Test the top module.
2. After testing the top module, stubs are replaced one at a time with the actual modules for integration.
3. Perform testing on this recent integrated environment.
4. Regression testing may be conducted to ensure that new errors have not appeared.

7

		5. Repeat steps 2-4 for the whole design hierarchy. Listed below are the drawbacks of top-down integration testing. 1. Stubs must be prepared as required for testing one module. 2. Stubs are often more complicated than they first appear. 3. Before the I/O functions are added, the representation of test cases in stubs can be difficult. **Bottom-up Integration Testing** The bottom-up strategy begins with the terminal or modules at the lowest level in the software structure. After testing these modules, they are integrated and tested moving from bottom to top level. Since the processing require, modules subordinate to a given level is always available, stubs are not required in this strategy. Bottom-up integration can be considered as the opposite of top-down approach. Unlike top-down strategy, this strategy does not require the architectural design of the system to be complete. Thus, bottom-up integration to be performed at an early stage in the developmental process. It may be used where the system reuses and modifies components from other systems. The steps in bottom-up integration are as follows: 1. Start with the lowest level modules in the design hierarchy. These are the modules from which no other module is being called. 2. Look for the super-ordinate module which calls the module selected in step 1. Design the driver module for this super-ordinate module. 3. Test the module selected in step 1 with the driver designed in step 2. 4. The next module to be tested is any module whose subordinate mod. ules (the modules it calls) have all been tested. 5. Repeat steps 2 to 5 and move up in the design hierarchy. 6. Whenever, the actual modules are available, replace stubs and drivers with the actual one and test again.			
XI		INSPECTION TEAM For the inspection process, a minimum of the following four team members are required. **Author/Owner/Producer** A programmer or designer responsible for producing, the program or document. He is also responsible for fixing defects discovered during the inspection process. **Inspector** A peer member of the team, i.e. he is not a manager or supervisor. He is not directly related to the product under inspection and may be concerned with some other product. He finds errors, omissions, and inconsistencies in programs and documents. **Moderator** A team member who manages the whole inspection process. He schedules, leads, and controls the inspection session. He is the key person with the responsibility of planning and successful execution of the inspection. **Recorder** One who records all the results of the inspection meeting.		7	
XII		*Recovery Testing* Recovery is jug like the exception handling feature of a programming Language. It is the ability of a system to restart operations after the integrity of the application has been lost. It reverts to a point where the system was functioning correctly and then, reprocesses the transactions to the point of failure.	1 + 3 X 2	7	

		Some software systems (e.g. operating system, database management systems etc.) must recover from programming errors, hardware failures, data errors, or any disaster in the system. So the purpose of this type of system testing is to show that these recovery functions do not work correctly. The main purpose of this test is to determine how good the developed software is when it faces a disaster. Disaster can be anything from unplugging the system which is running the software from power, network etc., also stopping the database, or crashing the developed software itself. Thus, *recovery testing is the activity of testing how well the software is able to recover from crashes, hardware failures, and other similar problems.* *Security Testing* Safety and security issues are gaining importance due to the proliferation of commercial applications on the Internet and the increasing concern about privacy. Security is a protection system that is needed to assure the customers that their data will be protected. **Elements of security testing** The basic security concepts that need to be covered by security testing are discussed below: **Confidentiality** **Integrity** **Authentication** *Performance Testing* Performance specifications (requirements) are documented in a performance test plan. Ideally, this is done during the requirement development phase of any system development project, prior to design effort. A system along with its functional requirements, must meet the quality requirements. One example of quality requirement is performance level. The risers may have objectives for a software system in terms of memory use, response time, throughput, and delays. Thus, performance testing is to test the runtime performance of the software on the basis of various performance wore. Performance testing becomes important for real-time embedded systems, as they demand critical performance requirements. *Load Testing* Normally, we don't test the system with its full load, i.e. with maximum values of all resources in the system. The normal system testing takes into consideration. When a system is tested with a load that causes it to allocate its resources in maximum amounts, it is called load testing. The idea is to create an environment more demanding than the application wood experience under normal workloads. *Stress Testing* Stress testing is also a type of load testing, but the difference is that the system is put under loads beyond the limits to that the system breaks. Thus, stress testing tries to break the system under test by overwhelming its resources in order to find the circumstances under which it will crash. The areas that may be stressed in a system are: ■ Input transactions ■ Disk space ■ Output ■ Communications ■ Interaction with users. *(Any Three is enough)*			
XIII		Explain the commands 1. to() Command		7	

		Loads a new web page in the current browser window. It accepts a string parameter that specifies the URL of the web page to be loaded. 2. back() Command Moves back one step in the browser's history stack. It does not accept any parameters and does not return anything. 3. forward() Command Moves forward one step in the browser's history stack. It does not accept any parameters and does not return anything. 4. refresh() Command Reloads the current web page in the browser window. It does not accept any parameters and does not return anything.			
XIV		***Basic unit for testing*** There is nearly universal agreement that class is the natural unit for test case design. Other units for testing are aggregations of classes: class clusters, and application systems. The intended use of a class implies different test requirements, e.g., application-specific vs general-purpose classes, abstract classes, and parameterized (template) classes. Testing a class instance (an object) can verify a class in isolation. However, when verified classes are used to create objects in an application system, the entire system must be tested as a whole before it can be considered verified. ***Implications of inheritance*** In case of inheritance, the inherited features require retesting because these features are now in, new context of usage. Therefore, while planning to test inheritance, include all inherited features. Moreover, planning should also consider the multiple contexts of usage in case of multiple inheritances. ***Polymorphism*** In case of polymorphism, each possible binding of a polymorphic component requires a separate test to validate all of them. However, it may be hard to find all such bindings. This again increases the chance of errors and poses an obstacle in reaching the coverage goals. ***White-box testing*** There is also an issue regarding the testing techniques for OO software. The issue is that the conventional white-box techniques cannot be adapted for OO software. In OO software, class is the major component to be tested, but conventional white-box testing techniques cannot be applied on testing a class. These techniques may be applicable to individual methods but not suitable for OO classes. ***Black-box testing*** The black-box testing techniques may be suitable to OO software for conventional software, but sometimes re-testing of inherited features require examination of class structure. This problem also limits the blackbox methods for OO software. ***Integration strategies*** Integration testing is another issue which requires attention and there is need to devise newer methods for it. However, there is no obvious hierarchy of methods or classes. Therefore, conventional methods of incremental integration testing cannot be adopted.		7	