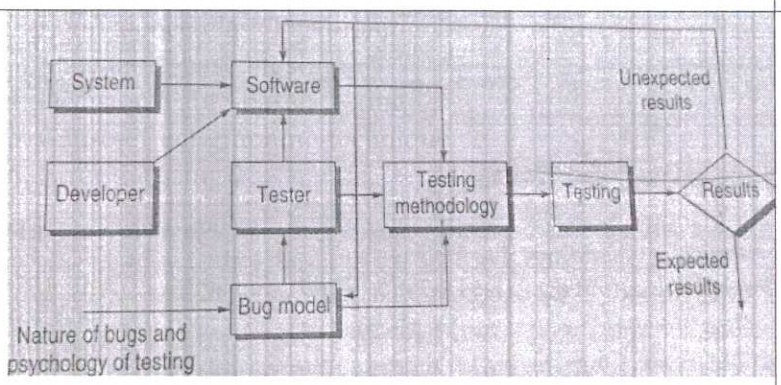
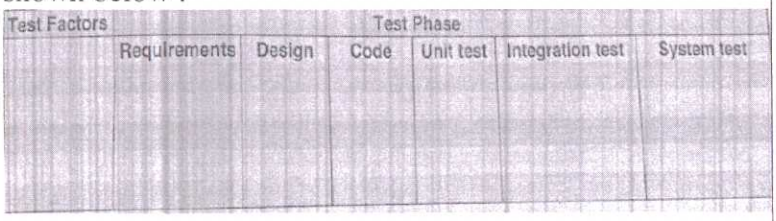
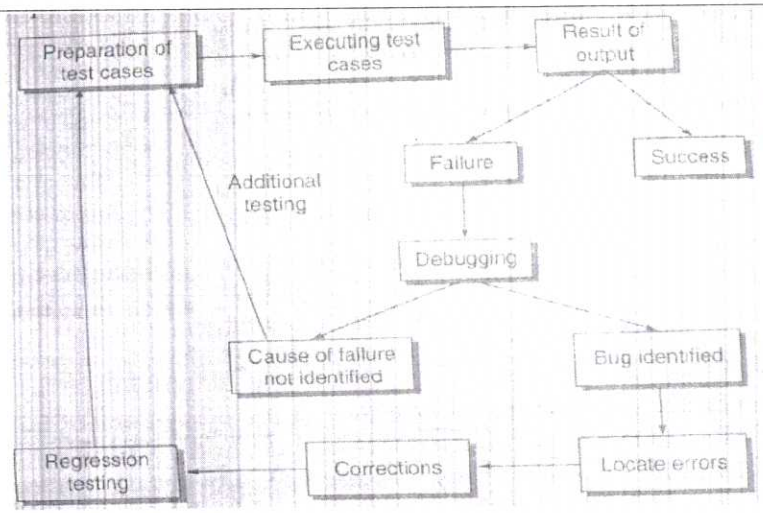
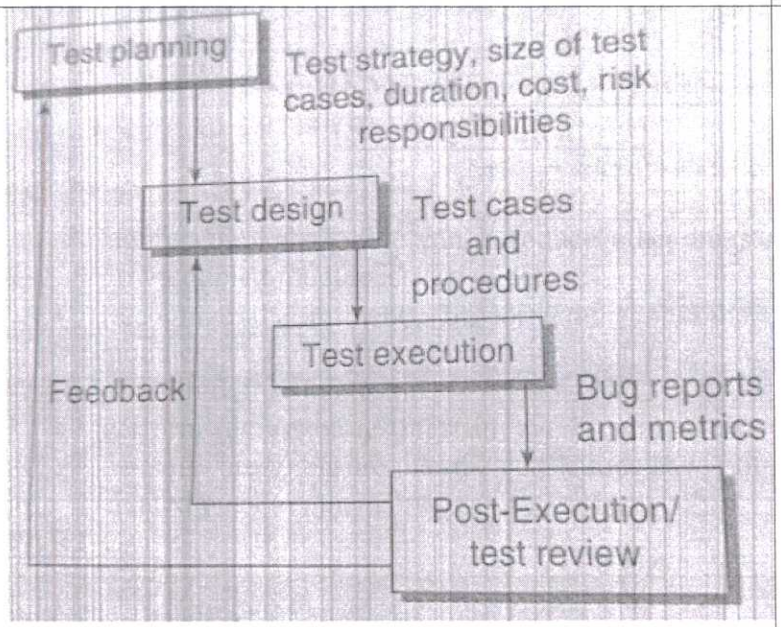
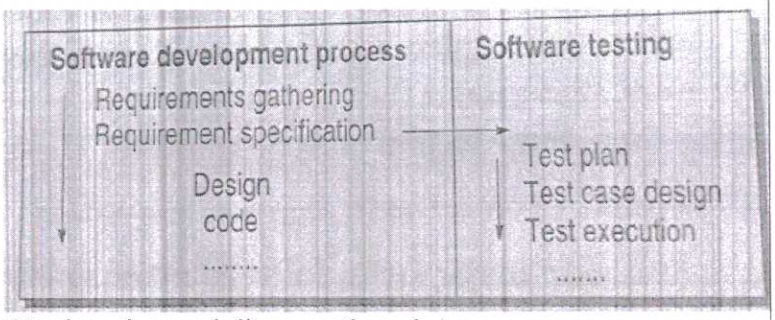
SCHEME OF VALUATION
(Scoring Indicators)

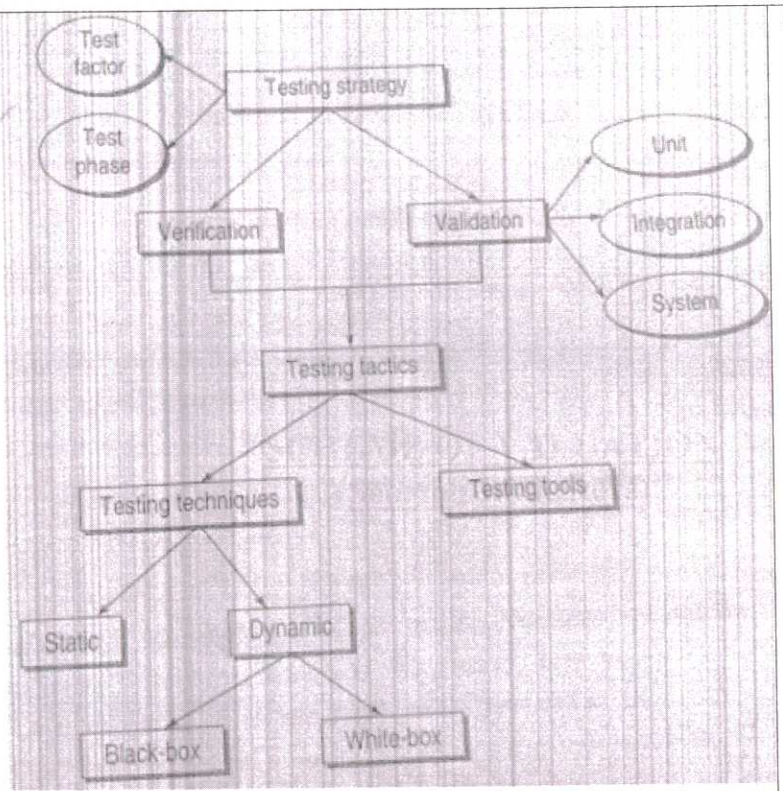
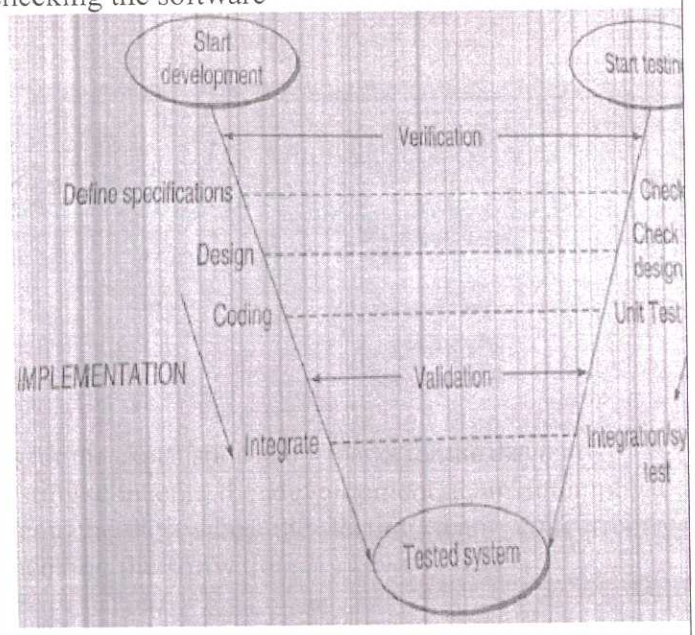
Revision:2015		Course Code:6136		
Course Title: SOFTWARE TESTING				
Qst. No	**Scoring Indicator**	**Split up Score**	**Sub Total**	**Total**
	PART A			
I(i)	Reliability, Quality, Customer Satisfaction, Risk management	4*0.5=2	2	2
I(ii)	• Static testing do not involve actual execution • Dynamic testing executes the code on some test data	2	2	2
I(iii)	1. Boundary Value Analysis 2. Equivalence class Partitioning 3. State table-based testing 4. Decision Table-based testing 5. Cause-effect graphing technique **6.** Error guessing (List any 4)	4*0.5=2	2	2
I(iv)	C Unit Test System (CUT or CuTest), Cgreen, EMMA, FindBugs, Cfix (List any 2)	2	2	2
I(v)	Eclipse debugger, Firefox JavaScript debugger, GNU debugger, LLDB, Microsoft Visual Studio Debugger, Radare2, Valgrind, Watcom debugger, WinDbg (List any 2)	2	2	2
	PART B			
II(i)	 (4 marks for the explanation and 2 marks for the diagram)	6	6	6
II(ii)	**Test Strategy Matrix** A test strategy matrix can be used while developing	6	6	6

	testing strategy. This matrix is prepared using test factors and test phases as shown below :  			
II(iii)	• White-box testing is the initial stage of testing of modules. • Black-box testing cannot be executed until the code is produced and checked using white-box testing techniques. So white-box testing is essential. • There are categories of bugs that can be revealed by white-box testing and not by black-box testing. So it is necessary. • Design phase errors will be reflected in code. So white-box testing must be done for code Verification • White-box testing explores all logical paths which are commonly as well as rarely executing. • White-box testing helps to detect typographical errors which are not covered by black-box testing.	6	6	6
II(iv)	It is a numeric measurement that shows the logical complexity of a Program. It can be calculated by considering the control flow graph. Complexity number actually represents the number of independent paths in the control structure of the program 1. Cyclomatic Complexity Number **V(G) = e – n + 1** **e** = number of edges and **n** = number of nodes 2. **V(G) = d + 1**	6	6	6

	d is the number of decision nodes for, **k** – way decision, **d = k – 1** 3. **V(G) = e – n + 2p**, where **p** is the number of graphs (procedures) **e & n** are edges and nodes of the whole graph 4. **V(G) = d + p** (2 marks for the definition and 4 marks for the methods)			
II(v)	1. Reduction of testing effort 2. Reduces the tester's involvement in executing tests 3. Facilitates regression testing 4. Avoids human mistakes 5. Reduces overall cost of the software 6. Simulated testing 7. Internal testing 8. Test enablers 9. Test case design (Answer any 6)	6	6	6
II(vi)	 (Explanation 4 marks and diagram 2 marks)	6	6	6
II(vii)	• Evaluate coupling of the logic and data structure where corrections are to be made. Correction of	6	6	6

	highly coupled module can introduce other bugs. Low-coupled module is easy to debug. • Recognize the influence of corrections on other modules and then plan regression test cases to perform regression testing • Perform regression testing with every correction in the software to ensure that the corrections have not created bugs in other parts.			
	PART C			
III(a)	 (6 marks for the explanation and 3 for diagram)	9	9	9
III(b)	Software testing is a process which runs parallel to SDLC  (Explanation and diagram 6 marks)	6	6	6

SCHEME OF VALUATION
(Scoring Indicators)

IV(a)	 (6 marks for the explanation and 3 marks for the diagram)	9	9	9
IV(b)	In V-testing concept, the development team attempts to implement the software, the testing team concurrently starts checking the software 	6	6	6
V(a)	• Mutation testing is the process of mutating some	3	9	9

segment of code (putting some error in the code) and then testing this mutated code with some test data.

- During testing, many versions of the program will be created, by introducing one fault to each.
- Test data is used to execute these fault versions of the program
- Faulty programs are called *Mutants* of the original program and a mutant is said to be *killed* when a test case causes it to fail. When this happens, the mutant is considered *dead* and no longer needs to remain in the testing process
- The main objective of testing is to select efficient test data which have error detection power.
- The criterion for this test data is to differentiate the original program from the mutant.

3

Primary Mutants

When the mutants are single modifications of the initial program using some operators, they are called primary mutants

Example:

```
If ( a> b)
x = x + y;
 else
 x = y
printf ( "%d",x );
```

We can consider the following primary mutants for the above example :

M1 : x = x – y; M2 : x = x / y;
M3 : x = x + 1; M4 : printf("%d", y);

Secondary Mutants 3

When multiple levels of mutation are applied on the initial program, those mutants are called secondary mutants.

Example:

```
if ( a < b )
    c = a;
```

Following are some of its secondary mutants.

M1 : if (a <= b-1)
c = a;
M2 : if (a + 1 <= b)

	c = a;
	M3 : if (a == b)
	c = a + 1;			

V(b)	**Stubs:** - The modules under testing may also call some other module which is not ready at the time of testing. So these modules need to be simulated. These simulated dummy modules prepared for testing are called stubs **Drivers**: - In unit testing, a module to be tested needs input from another module which is not ready, and then a main program is prepared to provide the required inputs to the module under testing. This simulated main programs are known as driver modules	6	6	6
VI(a)	**Integration testing** Integration is the activity of combining the modules together according to the design of software • **Decomposition-based integration** This type of integration is based on the decomposition of design into functional components or modules. • **Call Graph-based integration** • **Path-based integration**	9	9	9
VI(b)	• In progressive testing, the testing process progresses from verification to validation towards the release of product. All testing techniques black-box testing, white-box testing, static testing, validation testing etc. are progressive in nature. • A system under test (SUT) is said to regress if : 1. A modified component fails 2. A new component causes failure in the unchanged components by generating side effects. • Therefore , all systems will have three versions : 1. ***Baseline version***: The version of a system that has passed a test suite. 2. ***Delta Version*** : A changed version that has not passed a regression test 3. ***Delta build*** : An executable configuration of the SUT that contains all the delta and baseline	6	6	6

	components • Most test cases begin as progressive test cases and eventually become regression test cases • Regression testing can be defined as the software maintenance task performed on a modified program to instill confidence that changes are correct and have not adversely affected the unchanged portions of the program.			
VII(a)	1. Match the tool to its appropriate use 2. Select the tool to its appropriate SDLC phase 3. Select the tool to the skill of the tester 4. Select a tool which is affordable 5. Determine how many tools are required for testing the system 6. Select the tool after examining the schedule of testing	9	9	9
VII(b)	1. Basic unit of testing 2. Implications of inheritance 3. Polymorphism 4. White-box testing 5. Black-box testing 6. Integration strategies	6	6	6
VIII(a)	1. Static and Dynamic testing tools 2. Testing Activity Tools a. Tools for review and inspections b. Tools for test planning c. Tools for test design and development d. Test execution and evaluation tools i. Capture/playback tools ii. Coverage analysis tool iii. Memory testing tools iv. Test management tools v. Network testing tools vi. Performance testing tools	9	9	9
VIII(b)	1. Diversity and Complexity 2. Dynamic Environment 3. Very short development time 4. Continuous evolution 5. Compatibility and Interoperability	6	6	6
IX(a)	1. Debugging with memory dump 2. Debugging with watch points 3. Backtracking	3*3=9	9	9

SCHEME OF VALUATION
(Scoring Indicators)

IX(b)	It is for dealing with problems with an OS kernel on its own or for interactions between OS-dependent applications and the OS referred before fixing the error. Example gdb,KD	6	6	6
X(a)	1. Kernel debugger: 2. Basic machine-level debugger 3. In-circuit emulator 4. Interpretive programming environment debugger:	9	9	9
X(b)	 (Explanation and diagram 6 marks)	6	6	6

1