V2V EDTECH LLP

Online Coaching at an Affordable Price.

OUR SERVICES:

- Diploma in All Branches, All Subjects
- Degree in All Branches, All Subjects
- BSCIT / CS
- Professional Courses

+91 93260 50669

v2vedtech.com

V2V EdTech LLP

v2vedtech

Winter – 19 EXAMINATION

Subject Name: Software Testing **Model Answer** **Subject Code: 22518**

Important Instructions to examiners:

1) The answers should be examined by key words and not as word-to-word as given in the model answer scheme.
2) The model answer and the answer written by candidate may vary but the examiner may try to assess the understanding level of the candidate.
3) The language errors such as grammatical, spelling errors should not be given more Importance (Not applicable for subject English and Communication Skills.
4) While assessing figures, examiner may give credit for principal components indicated in the figure. The figures drawn by candidate and model answer may vary. The examiner may give credit for any equivalent figure drawn.
5) Credits may be given step wise for numerical problems. In some cases, the assumed constant values may vary and there may be some difference in the candidate's answers and model answer.
6) In case of some questions credit may be given by judgement on part of examiner of relevant answer based on candidate's understanding.
7) For programming language papers, credit may be given to any other program based on equivalent concept.

Q. No.	Sub Q. N.	Answer	Marking Scheme
1.		**Attempt any Five of the following:**	**10 M**
	a	**Define static and dynamic testing.**	**2M**
	Ans	**Static testing**: In static testing code is not executed. Rather it manually checks the code, requirement documents, and design documents to find errors. Main objective of this testing is to improve the quality of software products by finding errors in early stages of the development cycle. **Dynamic testing:** The dynamic testing is done by executing program. Main objective of this testing is to confirm that the software product works in conformance with the business requirements.	1 M for each definition
	b	**State any two examples of integration testing.**	**2M**
	Ans	1. Verifying the interface link between the login page and the home page i.e. when a user enters the credentials and logs it should be directed to the homepage 2. Check the interface link between the Login and Mailbox module 3. Check the interface link between the Mailbox and Delete Mails Module. 4. Verifying the interface link between the home page and the profile page i.e. profile page should open up.	Any two similar example:2M

	c	**Enlist any two activities involved in test planning.**	**2M**
	Ans	**1. Scope Management:** Deciding what features to be tested and not to be tested. **2. Deciding Test approach /strategy:** Which type of testing shall be done like configuration, integration, localization etc. **3. Setting up criteria for testing:** There must be clear entry and exit criteria for different phases of testing. The test strategies for the various features and combinations determined how these features and combinations would be tested. **4. Identifying responsibilities, staffing and training needs.**	Any two activities 2M
	d	**Enlist objectives of software testing.**	**2M**
	Ans	**Objectives of software testing are as follows:** **1.** Finding defects which may get created by the programmer while developing the software. **2.** Gaining confidence in and providing information about the level of quality. **3.** To prevent defects. **4.** To make sure that the end result meets the business and user requirements. **5.** To ensure that it satisfies the BRS that is Business Requirement Specification and SRS that is System Requirement Specifications. **6.** To gain the confidence of the customers by providing them a quality product.	Any two Objectives 2M
	e	**Define Defect.**	**2M**
	Ans	It refers to the several troubles with the software product, with its external behavior or its internal features. **OR** A defect is an error in coding that causes a program to fail or to produce incorrect /unexpected results.	Correct Definition 2M
	f	**State any four advantages of using tools.**	**2M**
	Ans	**Save Time /Speed:** Due to advanced computing facilities, automation test tools prevail in speed of processing the tests. Automation saves time as software can execute test cases faster than human. **Reduces the tester's involvement in executing tests:** It relieves the testers to do some other work. **Repeatability/Consistency:** The same tests can be re-run in exactly the same manner eliminating the risk of human errors such as testers forgetting their exact actions, intentionally omitting steps from the test scripts, missing out steps from the test script, all of which can	Any 4 advantages : ½ M for each

		result in either defects not being identified or the reporting of invalid bugs (which can again, be time consuming for both developers and testers to reproduce) **Simulated Testing:** Automated tools can create many concurrent virtual users/data and effectively test the project in the test environment before releasing the product. **Test case design:** Automated tools can be used to design test cases also through automation, better coverage can be guaranteed than if done manually. **Reusable:** The automated tests can be reused on different versions of the software, even if the interface changes. **Avoids human mistakes:** Manually executing the test cases may incorporate errors. But this can be avoided in automation testing**.** **Internal Testing:** Testing may require testing for memory leakage or checking the coverage of testing. Automation can done this easily. **Cost Reduction:** If testing time increases cost of the software also increases. Due to testing tools time and therefore cost is reduced**.**	
	g	**Define Bug, Error, Fault, and Failure.**	**2M**
	Ans	**Bug:** A bug can be defined as the initiation of error or a problem due to which fault, failure, incident or an anomaly occurs. **Error:** A human action that produces an incorrect result. **Fault:** An incorrect step, process, or data definition in a computer program. **Failure:** A failure is said to occur whenever the external behavior of a system does not conform to that prescribed in the system specification. A software fault becomes a software failure only when it is activated.	½ M for each definition
2.		**Attempt any Three of the following:**	**12M**
	a	**Define Boundary value analysis with suitable example.**	**4M**
	Ans	Most of the defects in software products hover around conditions and boundaries. By conditions, we mean situations wherein, based on the values of various variables, certain actions would have to be taken. By boundaries, we mean ―limits of values of the various variables. • This is one of the software testing technique in which the test cases are designed to include values at the boundary. • If the input data is used within the boundary value limits, then it is said to be Positive Testing. If the input data is	Explanation:2M and 2 M for Example

		picked outside the boundary value limits, then it is said to be Negative Testing. • Boundary value analysis is another black box test design technique and it is used to find the errors at boundaries of input domain rather than finding those errors in the center of input. • Each boundary has a valid boundary value and an invalid boundary value. Test cases are designed based on the both valid and invalid boundary values. Typically, we choose one test case from each boundary. • Boundary value analysis is a black box testing and is also applies to white box testing. Internal data structures like arrays, stacks and queues need to be checked for boundary or limit conditions. When there are linked lists used as internal structures, the behavior of the list at the beginning and end has to be tested thoroughly. • Boundary value analysis help identify the test cases that are most likely to uncover defects. **Example 1:** A system can accept the numbers from 1 to 10 numeric values. All other numbers are invalid values. Under this technique, boundary values 0, 1,2,9,10,11 can be tested. **Example 2**: The exam has a pass boundary at 40 percent, merit at 75 percent and Distinction at 85 percent. The Valid Boundary values for this scenario will be as follows: • 49, 50 - for pass • 74, 75 - for merit • 84, 85 - for distinction Boundary values are validated against both the valid boundaries and invalid boundaries. The Invalid Boundary Cases for the above example can be given as follows: • 0 - for lower limit boundary value • 101 - for upper limit boundary value	
	b	**Differentiate between drivers and stub (any four points).**	**4M**
	Ans		1 M for each valid point

Stubs	Drivers
Stubs are dummy modules that always used to simulate the low level modules.	Drivers are dummy modules that always used to simulate the high level modules.
Stubs are the called programs.	Drivers are the calling programs.
Stubs are used when sub programs are under construction.	Drivers are only used when main programs are under construction.
Stubs are used in top down approach.	Drivers are used in bottom up integration.

	c	**State the contents of 'Test Summary Reports' used in test reporting.**	**4M**
	Ans	Test reporting is a means of achieving communication through the testing cycle. There are 3 types of test reporting. **1. Test incident report:** **2. Test cycle report:** **3. Test summary report:** **Test summary Report**: The final step in a test cycle is to recommend the suitability of a product for release. A report that summarizes the result of a test cycle is the test summary report. There are two types of test summary report: 1. Phase wise test summary, which is produced at the end of every phase. 2. Final test summary report, which has all the details of testing done by all phases. A Summary report should present 1. Test Summary Report Identifier 2 Description: Identify the test items being reported in this report with test id 3 Variances: Mention any deviation from test plans, test procedures, if any. 4 Summary of results: All the results are mentioned here with the resolved incidents and their solutions. 5 Comprehensive assessment and recommendation for release should include: Fit for release assessment and recommendation of release.	Explanation4 M

	d	**State any eight limitations of manual testing.**	**4M**
	Ans	1. Manual testing is slow and costly. 2. It is very labor intensive; it takes a long time to complete tests. 3. Manual tests don't scale well. As the complexity of the software increases the complexity of the testing problem grows exponentially. This leads to an increase in total time devoted to testing as well as total cost of testing. 4. Manual testing is not consistent or repeatable. Variations in how the tests are performed as inevitable, for various reasons. One tester may approach and perform a certain test differently from another, resulting in different results on the same test, because the tests are not being performed identically. 5. Lack of training is the common problem. 6. GUI objects size difference and color combinations are not easy to find in manual testing. 7. Not suitable for large scale projects and time bound projects. 8. Batch testing is not possible, for each and every test execution Human user interaction is mandatory. 9. Comparing large amount of data is impractical. 10. Processing change requests during software maintenance takes more time.	Any 8 points 1/2 M for each point
3.		**Attempt any Three of the following:**	**12M**
	a	**Describe the use of decision table in black box testing with the help of suitable example.**	**4M**
	Ans	I.Decision table testing is black box test design technique to determine the test scenarios for complex business logic. ii. Decision tables provide a systematic way of stating complex business rules, which is useful for developers as well as for testers. iii. Decision tables can be used in test design whether or not they are used in specifications, as they help testers explore the effects of combinations of different inputs and other software states that must correctly implement business rules. iv. It helps the developers to do a better job can also lead to better relationships with them. v. Testing combinations can be a challenge, as the number of combinations can often be huge. vi. Testing all combinations may be impractical if not impossible. vii. We have to be satisfied with testing just a small subset of combinations but making the choice of which combinations to test and which to leave out is also important.	Use of decision table in black box testing with example 4M

Viii.If you do not have a systematic way of selecting combinations, an arbitrary subset will be used and this may well result in an ineffective test effort.

Importance of Decision Table: Essentially it is a structured exercise to formulate requirements when dealing with complex business rules. Decision tables are used to model complicated logic. They can make it easy to see that all possible combinations of conditions have been considered and when conditions are missed, it is easy to see.

Example :

Conditions	**TC1**	**TC2**	**TC3**	**TC4**
Request login	**0**	**1**	**1**	**1**
Valid username entered	**X**	**0**	**1**	**1**
Valid password entered	**X**	**X**	**0**	**1**
Actions				
Offer recover credentials	**0**	**1**	**1**	**0**
Activate entry box username	**0**	**1**	**1**	**0**
Activate entry box Password	**0**	**0**	**1**	**0**
Enter privilege area	**0**	**0**	**0**	**1**

Where 0→ False
1→ True
X→ No action (Don't care)

	b	**Describe standards included in Test management.**	**4M**
	Ans	**Internal standards are:** 1. Naming and storage conventions for test artifacts. 2. Document standards 3. Test coding standards 4. Test reporting standards. **1. Naming and storage conventions for test artifacts:** Every test artifact (test specification, test case, test results and so on) have to be named appropriately and meaningfully. It enables a) Easy identification of the product functionality. b) Reverse mapping to identify the functionality corresponding to a given set of tests. E.g. modules shall be M01, M02. Files types can be .sh, .SQL.	Standards included in Test management4M

		2. Documentation standards: a) Appropriate header level comments at the beginning of a file that outlines the functions to be served by the test. b) Sufficient inline comments, spread throughout the file c) Up-to-Date change history information, reading all the changes made to the test file. **3. Test coding standards**: a) Enforce right type of initialization b) Stipulate ways of naming variables. c) Encourage reusability of test artifacts d) Provide standard interfaces to external entities like operating system, hardware and so on. 4. **Test reporting standard**: All the stakeholders must get a consistent and timely view of the progress of tests. It provides guidelines on the level of details that should be present in the test report, their standard formats and contents. **5.External Standards:** These are the standards made by an entity external to an organization. These standards are standards that a product should comply with, are externally visible and are usually stipulated by external parties. The three types of external standards are: • Customer standard: refer to something defined by the customer as per his/her business requirement for the given product. • National Standard: refer to something defined by the regulatory entities of the country where the supplier / customer resides. • International Standard: are defined at international level and these are applicable to all customers across the globe.	
	c	**Enlist different techniques for finding defects and describe any one technique with an example.**	**4M**
	Ans	Different techniques for finding defects are as given below: **a) Quick Attacks:** **i. Strengths** • The quick-attacks technique allows you to perform a cursory analysis of a system in a very compressed timeframe. • Even without a specification, you know a little bit about the software, so the time spent is also time invested in developing expertise.	List of any relevant techniques 1M, explanation of 1 technique with example 3M

- The skill is relatively easy to learn, and once you've attained some mastery your quick-attack session will probably produce a few bugs.
- Finally, quick attacks are quick.
- They can help you to make a rapid assessment. You may not know the requirements, but if your attacks yielded a lot of bugs, the programmers probably aren't thinking about exceptional conditions, and it's also likely that they made mistakes in the main functionality.
- If your attacks don't yield any defects, you may have some confidence in the general, happy-path functionality.

ii. Weaknesses

- Quick attacks are often criticized for finding "bugs that don't matter"— especially for internal applications.
- While easy mastery of this skill is strength, it creates the risk that quick attacks are "all there is" to testing; thus, anyone who takes a two day course can do the work.

b) Equivalence and Boundary Conditions

i. Strengths

- Boundaries and equivalence classes give us a technique to reduce an infinite test set into something manageable.
- They also provide a mechanism for us to show that the requirements are "covered".

ii. Weaknesses

- The "classes" in the table in Figure 1 are correct only in the mind of the person who chose them.
- We have no idea whether other, "hidden" classes exist—for example, if a numeric number that represents time is compared to another time as a set of characters, or a "string," it will work just fine for most numbers.

c) Common Failure Modes

i. Strengths

- The heart of this method is to figure out what failures are common for the platform, the project, or the team; then try that test again on this build.
- If your team is new, or you haven't previously tracked bugs, you can still write down defects that "feel" recurring as they occur—and start checking for them.

ii. Weaknesses

- In addition to losing its potency over time, this technique also entirely fails to find "black swans"—defects that exist outside the team's recent experience.
- The more your team stretches itself (using a new database, new programming language, new team members, etc.), the

riskier the project will be—and, at the same time, the less valuable this technique will be.

d) State-Transition Diagrams

In this technique the state transition diagram is prepared with respect to the applied inputs and produced output. It clearly shows how the state transition of software takes place from one to another and hence can be useful to find the defects.

One of the example is as shown in the diagram below:

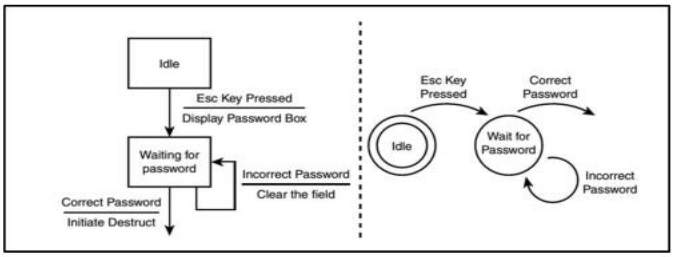


Figure 4: State Transition Map

i. Strengths

- Mapping out the application provides a list of immediate, powerful test ideas.
- Model can be improved by collaborating with the whole team to find "hidden" states—transitions that might be known only by the original programmer or specification author.
- Once you have the map, you can have other people draw their own diagrams, and then compare theirs to yours.
- The differences in those maps can indicate gaps in the requirements, defects in the software, or at least different expectations among team members.

ii. Weaknesses

- The map you draw doesn't actually reflect how the software will operate; in other words, "the map is not the territory."
- Drawing a diagram won't find these differences, and it might even give the team the illusion of certainty.
- Like just about every other technique on this list, a state-transition diagram can be helpful, but it's not sufficient by itself to test an entire application.

e) Use Cases and Soap Opera Tests

Use cases and scenarios focus on software in its role to enable a human being to do something.

i. Strengths

- Use cases and scenarios tend to resonate with business customers, and if done as part of the requirement process, they sort of magically generate test cases from the requirements.

- They make sense and can provide a straightforward set of confirmatory tests. Soap opera tests offer more power, and they can combine many test types into one execution.

ii. Weaknesses

- Soap opera tests have the opposite problem; they're so complex that if something goes wrong, it may take a fair bit of troubleshooting to find exactly where the error came from!

f) Code-Based Coverage Models

Imagine that you have a black-box recorder that writes down every single line of code as it executes.

i. Strengths

- Programmers love code coverage. It allows them to attach a number— an actual, hard, real number, such as 75%—to the performance of their unit tests, and they can challenge themselves to improve the score.
- Meanwhile, looking at the code that isn't covered also can yield opportunities for improvement and bugs!

ii. Weaknesses

- Customer-level coverage tools are expensive, programmer-level tools that tend to assume the team is doing automated unit testing and has a continuous-integration server and a fair bit of discipline.
- After installing the tool, most people tend to focus on statement coverage—the least powerful of the measures.
- Even decision coverage doesn't deal with situations where the decision contains defects, or when there are other, hidden equivalence classes; say, in the third-party library that isn't measured in the same way as your compiled source code is.
- Having code-coverage numbers can be helpful, but using them as a form of process control can actually encourage wrong behaviors. In my experience, it's often best to leave these measures to the programmers, to measure optionally for personal improvement (and to find dead spots), not as a proxy for actual quality.

g) Regression and High-Volume Test Techniques

People spend a lot of money on regression testing, taking the old test ideas described above and rerunning them over and over.

This is generally done with either expensive users or very expensive programmers spending a lot of time writing and later maintaining those automated tests.

i. Strengths

- For the right kind of problem, say an IT shop processing files through a database, this kind of technique can be extremely powerful.
- Likewise, if the software deliverable is a report written in SQL, you can hand the problem to other people in plain English, have them write their own SQL statements, and compare the results.
- Unlike state-transition diagrams, this method shines at finding the hidden state in devices. For a pacemaker or a missile-launch device, finding those issues can be pretty important.

ii. Weaknesses

- Building a record/playback/capture rig for a GUI can be extremely expensive, and it might be difficult to tell whether the application hasn't broken, but has changed in a minor way.
- For the most part, these techniques seem to have found a function in IT/database work, at large companies like Microsoft and AT&T, which can have programming testers doing this work in addition to traditional testing, or finding large errors such as crashes without having to understand the details of the business logic.
- While some software projects seem ready-made for this approach, others aren't.
- You could waste a fair bit of money and time trying to figure out where your project falls.

OR

Different techniques for finding defects are:

1. Static technique
2. Dynamic technique
3. Operational technique

1. Static Techniques: Static techniques of quality control define checking the software product and related artifacts without executing them. It is also termed desk checking/verification /white box testing. It may include reviews, walkthroughs, inspection, and audits here; the work product is reviewed by the reviewer with the help of a checklist, standards, any other artifact, knowledge and experience, in order to locate the defect with respect to the established criteria. Static technique is so named because it involves no execution of code, product, documentation, etc. This technique helps in establishing conformance to requirements view.

2. Dynamic Testing: Dynamic testing is a validation technique which includes dummy or actual execution of work products to

		evaluate it with expected behavior. It includes black box testing methodology such as system testing and unit testing. The testing methods evaluate the product with respect to requirements defined; designs created and mark it as pass or fail. **3.Operational techniques:** Operational techniques typically include auditing work products and projects to understand whether the processes defined for development /testing are being followed correctly or not, and also whether they are effective or not. It also includes revisiting the defects before and after fixing and analysis. Operational technique may include smoke testing and sanity testing of a work product.	
	d	**Enlist factors considered for selecting a testing tool for test automation.**	**4M**
	Ans	**The following factors are important during tool selection:** i. **Assessment of the organization's maturity** (e.g. readiness for change); ii. Identification of the areas within the organization where **tool support will help to improve testing processes;** iii. **Evaluation of tools against clear requirements and objective criteria;** iv. **Proof-of-concept to see whether the product works as desired** and meets the requirements and objectives defined for it; v. **Evaluation of the vendor (training, support and other commercial aspects**) or open-source network of support; vi. **Identifying and planning internal implementation** (including coaching and mentoring for those new to the use of the tool). **OR** **The industry experts have suggested following four major criteria for selection of testing tools.** 1) Meeting requirements. 2) Technology expectations. 3) Training / skills. 4) Management aspects. **1) Meeting Requirements:** a) There are many tools available in the market today but rarely do they meet all the requirements of given product or a given organization. Evaluating different tools for different requirements involves lot of effort, money and time. Huge delay is involved in selecting and implanting test tools. b) Test tools may not provide backward or forward compatibility with the product-under-test (PUT). c) Test tools may not go through the same amount of evaluation for new requirements. For example: some tools had Y2K-problem.	Any relevant factors minimum 4M

		d) A number of test tools cannot distinguish between a product failure and a test failure. This increases analysis time and manual testing. The test tools may not provide the required amount of trouble-shooting/debug/error messages to help in analysis. For example, in case of GUI testing, the test tools may determine the results based on messages and screen coordinates at run-time. Hence, if the screen elements of the product are changed, it requires the test suite to be changed. The test tool must have some intelligence to proactively find out the changes that happened in the product and accordingly analyze the results. **2) Technology Expectations:** a) In general, test tools may not allow test developers to extend / modify the functionality of the framework. So, it involves going back to the tool vendor with additional cost and effort. Very few tools available in market provide source code for extending functionality or fixing some problems. Extensibility and customization are important expectations of a test tool. b) A good number of test tools require their libraries to be linked with product binaries. When these libraries are linked with the source code of the product, it is called as the "instrumented code". This causes portion of testing be repeated after those libraries are removed, as the results of certain types of testing will be different and better when those libraries are removed. For example, the instrumented code has a major impact on the performance testing since the test tools introduce an additional code and there could be a delay in executing the additional code. c) Finally, test tools are not 100% cross-platform. They are supported only on some O.S. platforms and the scripts generated from these tools may not be compatible on other platforms. Moreover, many of the test tools are capable of testing only the product, not the impact of the product/test tool to the system or network. When there is an impact analysis of the product on the network or system, the first suspect is the test tool and it is uninstalled when such analysis starts. **3) Training Skills:** Test tools require plenty of training, but very few vendors provide the training to the required level. Organization-level training is needed to deploy the test tools, as the users of the test suite are not only the test team but also the development team and other areas like SCM (Software Configuration Management). Test tools expect the users to learn new language/scripts and may not use standard languages/scripts. This increases skill requirements for automation and increases the need for a learning curve inside the organization. **4) Management Aspects:**	

		A test tool increases the system requirement and requires the hardware and software to be upgraded. This increases the cost of the already-expensive test tool. When selecting the test tool, it is important to note the system requirements and the cost involved in upgrading the software and hardware needs to be included with the cost of the tool. Migrating from one test tool to another may be difficult and requires a lot of effort. Not only is this difficult, as the test suite that is written cannot be used with other test tools but also because of the cost involved. As the tools are expensive and unless the management feels that the returns on investment (ROI) are justified, changing tools are generally not permitted. Deploying a test tool requires as much effort as deploying a product in a company. However, due to project pressures, test tools effort at deploying gets diluted, not spent. Thus, later it becomes one of the reasons for delay or for automation not meeting expectations. The support available on the tool is another important point to be considered while selecting and deploying the test tool.	
4.		**Attempt any THREE of the following.**	**12M**
	a	**Differentiate between alpha and beta testing. (four points)**	**4M**
	Ans		4 differences 4M, 1M each. Any other relevant differences shall be given Marks.

Alpha Testing	Beta Testing
Alpha testing performed by Testers who are usually internal employees of the organization.	Beta testing is performed by Clients or End Users who are not employees of the organization.
Alpha Testing performed at developer's site.	Beta testing is performed at a client location or end user of the product.
Reliability and Security Testing are not performed in-depth Alpha Testing.	Reliability, Security, Robustness is checked during Beta Testing.
Alpha testing involves both the white box and black box techniques.	Beta Testing typically uses Black Box Testing.
Alpha testing requires a lab environment or testing environment.	Beta testing doesn't require any lab environment or testing environment. The software is made available to the public and is said to be real time environment.
Long execution cycle may be required for Alpha testing.	Only a few weeks of execution are required for Beta testing

			Critical issues or fixes can be addressed by developers immediately in Alpha testing.	Most of the issues or feedback is collected from Beta testing will be implemented in future versions of the product.
			Alpha testing is to ensure the quality of the product before moving to Beta testing	Beta testing also concentrates on the quality of the product, but gathers users input on the product and ensures that the product is ready for real time users.
	b	**Describe test infrastructure management.**		**4M**
	Ans	**Test infrastructure management** Testing requires a robust infrastructure to be planned upfront. This infrastructure is made up of three essential elements. **1. A test case database (TCDB):** A test case database captures all the relevant information about the test cases in an organization. Some of the entities and the attributes are given in following table		Test infrastructure management description :4M

Sr. No.	Test Case	Purpose	Attributes
1	Test case	Records all static information about tests.	1)Test case Id 2) Test case name (File name) 3) Test case owner 4) Associated files for test case.
2	Test case product cross reference	Provide mapping between the tests and the corresponding product features, enables identification of test cases for given feature.	Test case Id Module Id
3	Test case run history	Gives the history of when the test case was run and what was result , provided inputs on selection of test for regression runs	1) Test case Id 2) Run date 3) Time taken 4) Run status(Success/ Failure)
4	Test casedefect crossreference	Gives details of test cases introduced to test certain specific defects detected in the product, provides inputs on the selection of test for regression runs.	1) Test case Id 2) Defect reference

A test case database captures all the relevant information about the test cases in an organization.
Some of the entities and attributes in each of the entities in such a TCDB are:

- Test case
- Test case-product cross reference
- Test case run history
- Test case- defect cross reference

2. Defect repository

It captures all the relevant information of defect repository for a product. The information that a defect repository includes

- Defect details
- Defect test detail
- Fix details
- Communication

2. Defect repository

It captures all the relevant information of defect repository for a product. The
information that a defect repository includes

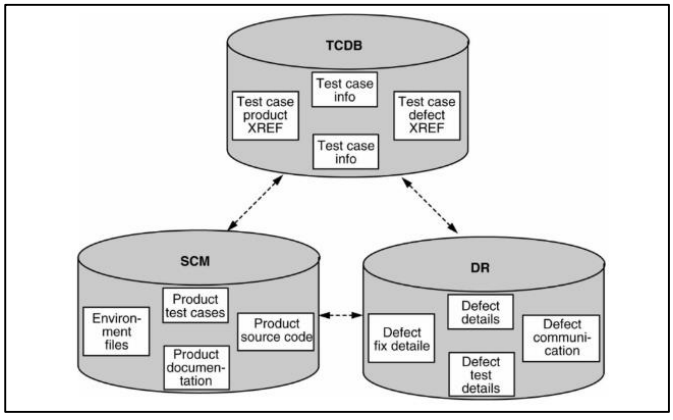
- Defect details
- Defect test detail
- Fix details
- Communication

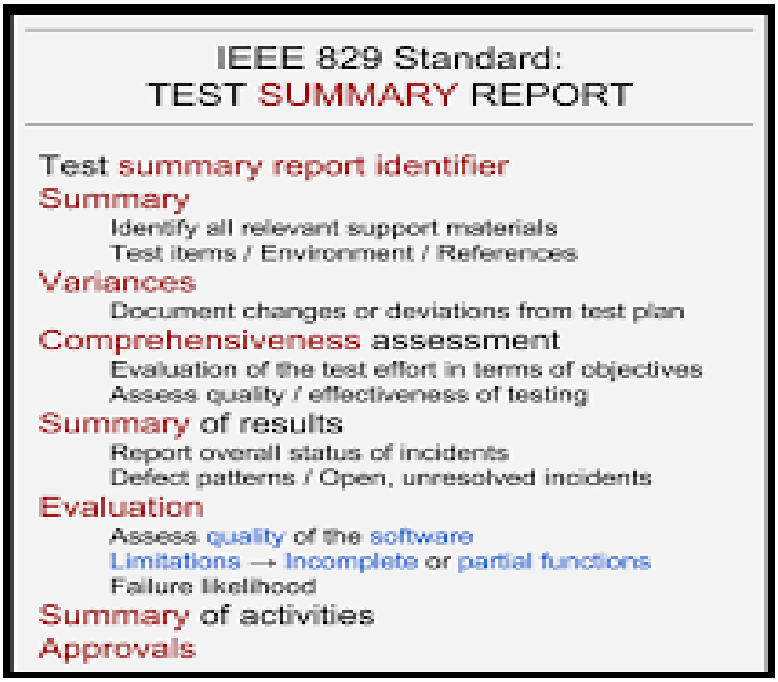
3. Configuration Management (CM) repository and **tool**

Software Configuration Management is defined as a process to systematically manage, organize, and control the changes in the documents, codes, and other entities during the Software Development Life Cycle.

It keeps track of change control and version control of all the files/entities that make up a software product. Change control ensures that

- Changes to test files are made in a controlled fashion and only with proper approvals
- Change are made by one test engineer are not accidently lost or overwritten by other changes
- Each change produces distinct version of the file that is re-creatable at any point of time
- Everyone gets access to only the most recent version of the test files.



	c	**Describe the process of preparing summary report in test planning.**	**4M**
	Ans	**Preparing test summary report** At the completion of a test cycle, a test summary report is produced. This report gives insights to the senior management about the fitness of the product for release. There are two types of reports that are required: 1. The Incident Report 2. Test Cycle Report 3. Test Summary Report A summary report should present the following things: 1. A summary of the activities carried out during the test cycle; 2. Variance of the activities carried out from the activities planned; 3. Summary of results should include tests that failed and severity of impact of defect; 4.Comprehensive assessment and recommendation for release should include "Fit for release" assessment and Recommendation of release  	Process of preparing summary report in test planning 4M , any other relevant answer shall be given Marks.
	d	**Describe object oriented metrics in testing.**	**4M**
	Ans	**Object oriented metrics in testing:** OBJECT-ORIENTED METRICS AND MEASURES As object-oriented approach emerged to support major applications, the effectiveness of applying traditional software metrics to	Any 4 object oriented metrics in testing 4M; Relevant answer shall be given Marks.

		object-oriented systems was challenged. The object-oriented design approach gives opportunity to classify metrics naturally. The classification captures object-oriented software features and properties hierarchically. It begins with the high-level characteristics of an object-oriented system and moves down to the low-level characteristics. **Source code size metrics**: Traditional metrics which are applied to object oriented software give insight into an overall system size and allow comparing systems and evaluating productivity. They can also be used as a refactoring effectiveness indicator. **Lines of Code (LOC) metric** is most common software project measure. The metric becomes a baseline to measure the degree of work performed on a project and it is used to create time and cost estimates. **Effective Lines of Code Metric (eLOC)** is a measure of all lines that are not comments, blanks or standalone braces or parenthesis. This metric more closely represents the quantity of work performed. **Comment Line and Comment Percent** (or Comment to Code Ratio) is a degree of commenting within the source code. It measures the care taken by programmers to make the source code and algorithms understandable. Poorly commented code makes the maintenance activities an extremely expensive. Recommended minimum is 20%. **Blank Line and White Space Percent Metric** is the number of blank lines within source code. It indicates the readability of product. And File Count Metric counts the files processed and generates metrics based on the file extension. It provides the distribution of the source code types, source code types and distribution of the specifications to the implementations. **Procedural metrics**: Cyclomatic Complexity is a popular procedural (called also function) software metric equal to the number of decisions that can be taken in a procedure A decision is defined as an occurrence of keywords such as: "while", "for", “for each”, “continue”, "if", "case", "go to", “try” and “catch” within the function. Cyclomatic Complexity is the sum of these constructs. That metric helps to identify software need of inspection or redesign, and also to allocate resources for evaluation and test. **Class metrics**: Class metrics describe structure of a class and relationship between classes. The volume of a class is a basic size measure connected with the amount of information inside it. The class volume can be measured by Number of Variables and by Number of Methods. Also Average LOC per Class and per Method metrics can provide insight into the average module size in the system.	

		Method metrics are used to estimate effort for testing early. Those metrics can be measured by Number of Parameters per Method, Weighted Methods per Class, Maximum Nesting Level, and Method Rank. Number of Parameter per Method counts parameters of a method and also references. **Afferent Coupling and Efferent Coupling** at method level are another object coupling metrics. Afferent Coupling for a particular method is the number of methods that depends directly on it and the Efferent Coupling for a particular method is the number of methods it directly depends on. Afferent Coupling is an indicator for the responsibility. The higher this value is the higher is the element's responsibility. Efferent Coupling means that a element depends on several other implementation details and it makes it instable. Therefore it is good practice to keep the Efferent Coupling for all artefacts at a minimum. **Inheritance metrics** :The inheritance relationships characteristic between classes and their parents indicate to a designer where changes would improve the development. The metrics connected to classes inheritance should take into account both the depth and breadth of the relationships. The Height of Inheritance Tree metric is counted as the maximum number of nodes from the class node to the root of the inheritance hierarchy. The deeper within the hierarchy, the more methods the class can inherit, increasing its complexity.	
	e	**State the testing approaches that are considered during client server testing.**	**4M**
	Ans	Testing approaches of client server system: • **Component Testing**: One need to define the approach and test plan for testing client and server individually. When server is tested there is need of a client simulator, whereas testing client a server simulator, and to test network both simulators are used at a time. • **Integration testing**: After successful testing of server, client and network, they are brought together to form system testing. • **Performance testing:** System performance is tested when number of clients is communicating with server at a time. Volume testing and stress testing may be used for testing, to test under maximum load as well as normal load expected. Various interactions may be used for stress testing. • **Concurrency Testing:** It is very important testing for client-server architecture. It may be possible that multiple users may be accessing same record at a time, and concurrency	Testing approaches of client server testing 4 approaches 4 marks;1 M each

testing is required to understand the behavior of a system in this situation.

- **Disaster Recovery /Business continuity testing:** When the client server are communicating with each other , there exit a possibility of breaking of the communication due to various reasons or failure of either client or server or link connecting them. The requirement specifications must describe the possible expectations in case of any failure.

- **Testing for extended periods**: In case of client server applications generally server is never shutdown unless there is some agreed Service Level Agreement (SLA) where server may be shut down for maintenance. It may be expected that server is running 24X7 for extended period. One needs to conduct testing over an extended period to understand if service level of network and server deteriorates over time due to some reasons like memory leakage.

Compatibility Testing: Client server may be put in different environments when the users are using them in production. Servers may be in different hardware, software, or operating system environment than the recommended. Other testing such as security testing and compliance testing may be involved if needed, as per testing and type of system.

5.		**Attempt any Three of the following:**	**12M**
	a	**Design test cases for railway reservation system.**	**4M**
	Ans	**Test cases for railway reservation system:**	Any 6 valid test cases :6 M, 1 M each Any other relevant test Cases shall be considered

Test case ID	Test case objective	Input data	Expected result	Actual result	Status
TC1	Login field	Any valid login name (abcxyz)	It should accept the login name	It accepted the login name	Pass
TC2	Password field	Valid password	It should accept the valid password	It accepted the valid password; successful	Pass

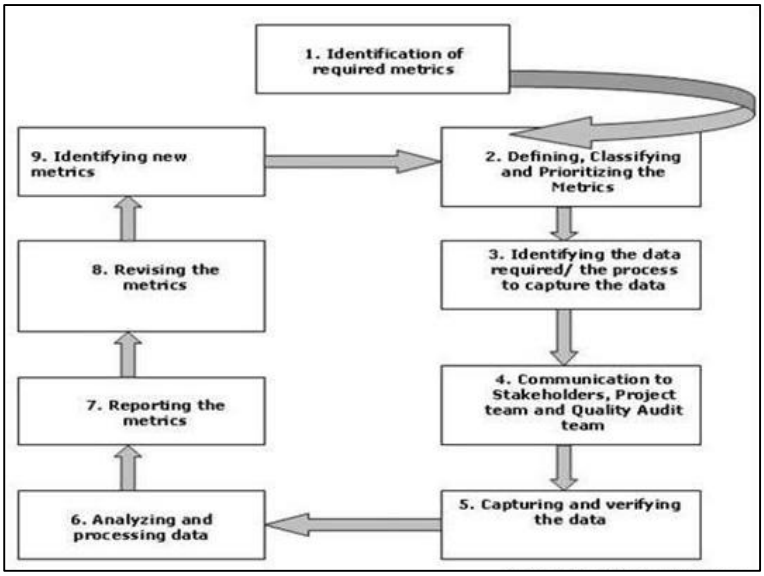
							login message		
			TC3	Password field	Invalid password	It should not accept the valid password	Message displayed as invalid login or wrong password.	Pass	
			TC4	Date of journey	Date format not before the current date	It should accept date	Accepted the date	Pass	
			TC5	Date of return journey	Date format, date greater than the date of journey	It should accept the date	Accepted the date	Pass	
			TC6	Boarding station	Valid boarding station	It should accept	Accepted the boarding station	Pass	
			TC7	Train number	Valid train number	It should accept the valid train number	Train number accepted	Pass	
	b	**With respect to GUI testing write the test cases for Amazon login form.**							**4M**

Ans

Test case ID	Test case objective	Input data	Expected result	Actual result	Status
TC1	Check cursor position at email or mobile number field	Click on email or mobile number field	Cursor should be placed on the field	Placed the cursor on the field	Pass
TC2	Check cursor position at password field	Click on password field	Cursor should be placed on the password field	Placed the cursor on the password field	Pass
TC3	Check the continue button	Click on continue button	It should redirect to password page	It redirected to the password page.	Pass
TC4	Readability of font	Try to read the contents on login page	Contents should be readable	Contents are readable	Pass
TC5	Testing of	Check the spelling of login	Login spelling should	Spelling of Login	Pass

Any 6 valid test cases :6M, 1M each Any other relevant test Cases shall be considered

				spelling of login		be correct	is correct		
			TC6	Testing of hyperlink	Hover the mouse on hyperlink	It should change the cursor and should redirect to respective page on click	Cursor changed and redirects to other page.	Pass	
	c	**Elaborate the term metrics and measurement and write the need of software measurement.**							**4M**
	Ans	**Metrics and measurement :** A Metric is a measurement of the degree that any attribute belongs to a system, product or process. For example the number of errors per person hours would be a metric. Thus, software measurement gives rise to software metrics. A measurement is an indication of the size, quantity, amount or dimension of a particular attribute of a product or process. For example the number of errors in a system is a measurement. A Metric is a quantitative measure of the degree to which a system, system component, or process possesses a given attribute. Metrics can be defined as "STANDARDS OF MEASUREMENT". Software Metrics are used to measure the quality of the project. Simply, Metric is a unit used for describing an attribute. Metric is a scale for measurement. **Need of Software measurement:** 1. Establish the quality of the current product or process. 2. To predict future qualities of the product or process. 3. To improve the quality of a product or process. 4. To determine the state of the project in relation to budget and schedule.								



6.		**Attempt any Three of the following:**	**12M**
	a	**Design test cases for hostel admission form of your institute.**	**4M**
	Ans		6 test cases of test cases for hostel admission form of institute : 6 M; 1M each; any other valid test cases shall be considered

Test case ID	**Test case objective**	**Input data**	**Expected result**	**Actual result**	**Status**
TC1	Student name field	Any valid alphabetical characters (John)	It should accept the name	Students name is accepted	Pass
TC2	Date of birth field	Date format before the current date	It should accept the date less than the current date	It accepted the valid date	Pass

			TC3	Gender field	Radio button should be selected. F or M	It should select the proper radio button	Proper radio button is selected	Pass	
			TC4	Date of admission	Date format not before the current date	It should accept date	Accepted the date	Pass	
			TC5	Age field	Any numerical data greater than or equal to 16	It should accept the number greater than or equal to 16	Accepted the age	Pass	
			TC6	Address field	Valid alpha numeric characters	It should accept the address	Accepted the address	Pass	
			TC7	Pin code	Valid 6 digits numeric format	It should accept the valid pin code	Pin code accepted	Pass	
	b	**Design a test plan along with the test cases for edit function in notepad.**							**4M**
	Ans								Any 3 valid test cases 3 M ; 1M each for edit function in notepad test plan 3 M

Test case ID	Test case objective	Input data	Expected result	Actual result	Status
TC1	Test the select all option	Click on select all	All the text should be selected	All the text is selected	Pass
TC2	Cut option	Select the text and click on cut	Selected text should be cut	Selected text is cut	Pass
TC3	Paste option	Click on paste	Contents should be pasted	Contents are pasted	Pass
TC4	Delete option	Select text and click on delete	Contents should be deleted	Contents are deleted	Pass

Test plan :

Test Plan Identifier

TP_10

Introduction: The purpose of this document is to create an application test plan for EDIT option of Notepad. The purpose of testing this program is to check the correct operation of its functionality, ease of use.

Test Items: Working with the document (select, cut, copy etc.)

Features to be tested

- Select all text
- Cut some text
- Paste the text

- Delete the text
- Copy the text
- Finding and replacing text

Features to be tested

- Working with Help
- Time and date option

Approach

- On the test object:
  - functional
  - non-functional
- According to the requirements
  - positive
  - negative
- By degree of preparedness - intuitive testing (ad hoc)

Item Pass/Fail Criteria: All test cases with high priority are closed with the result - pass. The test coverage is checked and sufficient, where the criterion of sufficiency is not less than 99% of the coverage of requirements by tests. The test report was compiled and approved by the team lead and customer.

Suspension Criteria and Resumption Requirements

Criterion for interrupting testing:

- The appearance and entering into the bug-tracking system of blocking bugs. Criterion for continuation of testing:
- Closing the blocking bug in the bug tracking system.

Test Deliverables: Test plan, test cases, test report.

Test Tasks

- Writing a test plan
- Writing test cases
- Development of criteria for the success of testing
- Conducting the testing and evaluation of the results
- Creating test reports

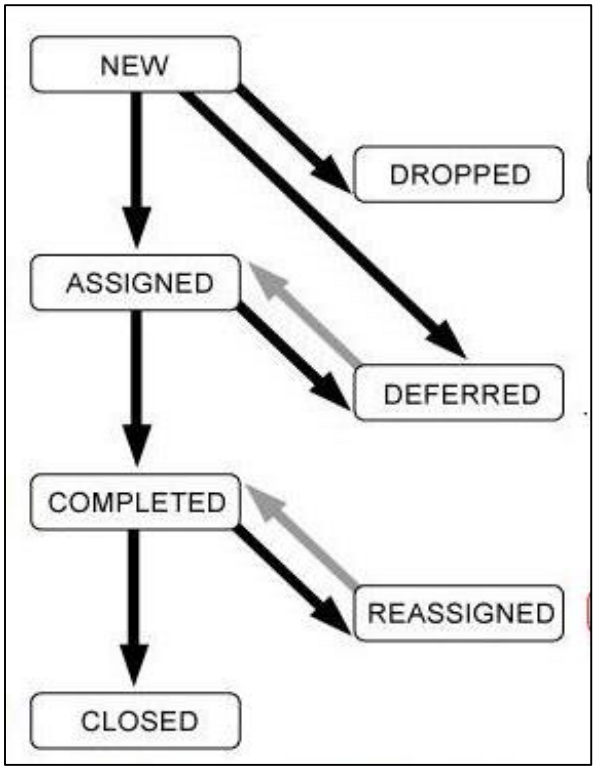
Environmental Needs

Notepad
Computer
Windows os

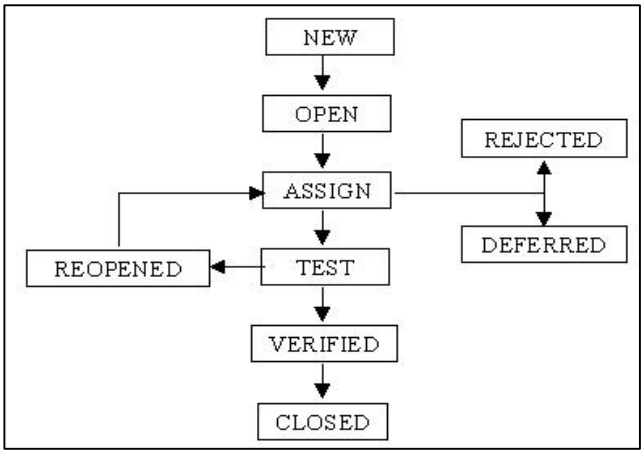
Responsibilities

Sr. no	Functionality and Responsibilities	Responsible
1	select all text	Test engineer 1
2	cut the text	Test engineer 1
3	paste the text	Test engineer 1

			3 copy the text — Test engineer 1 **5** find the text — Test engineer 2 **6** replacing text — Test engineer 2 **7** delete the selected text — Test engineer 2 **Staffing and Training Needs** To perform the tasks, you need to have the following knowledge and skills: • knowledge and practical application of the notepad; • knowledge and ability to apply in practice the basic techniques of test design • Knowledge of various types of testing including functional and non-functional. **Schedule** The deadline for completion of all works and delivery of the project is 06/12/2019 by 5.00pm **Risks and Contingencies** Possible risks during testing: • Insufficient human resources for testing the application in deadlines. • Changing the requirements for the product **Approvals** Team Lead Test engineer 1 Test engineer 2 Test engineer 3 Test engineer 4	
	c	**Draw a diagram for defect life cycle and write example for defect template.**		
	Ans	**Defect life cycle**		Defect life cycle diagram : 3 M; defect template : 3 M


NEW
DROPPED
ASSIGNED
DEFERRED
COMPLETED
REASSIGNED
CLOSED

OR


NEW
OPEN
REJECTED
ASSIGN
DEFERRED
REOPENED
TEST
VERIFIED
CLOSED

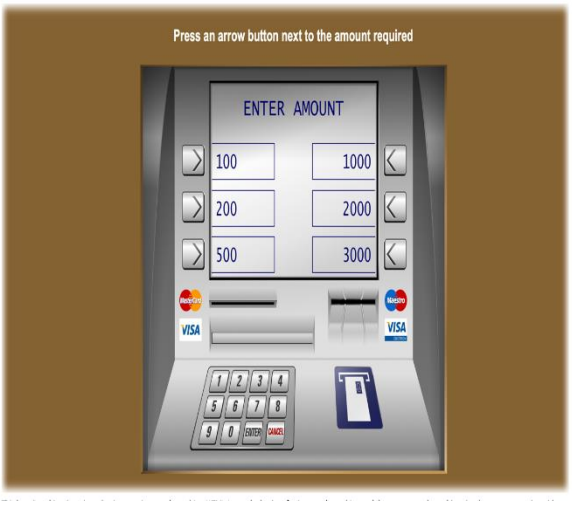
Defect template:

ID	Unique identifier given to the defect. (Usually Automated)
Project	Project name.
Product	Product name.
Release Version	Release version of the product. (e.g. 1.2.3)
Module	Specific module of the product where the defect was detected.
Detected Build Version	Build version of the product where the defect was detected (e.g. 1.2.3.5)
Summary	Summary of the defect. Keep this clear and concise.
Description	Detailed description of the defect. Describe as much as possible but without Repeating anything or using complex words. Keep it simple but comprehensive.
Steps to Replicate	Step by step description of the way to reproduce the defect. Number the steps.
Actual Result	The actual result you received when you followed the steps.
Expected Results	The expected results.
Attachments	Attach any additional information like screenshots and logs.
Remarks	Any additional comments on the defect.
Defect Severity	Severity of the Defect.

Example of Defect Template: (Varies defect wise):

ID	**R1**
Project	**Cash Simulator Cash (ATM)**
Product	http://www.motc.gov.qa/en/ditoolkit/migrant-workers/cash-machine-simulator-atm

Release Version	v1.0
Module	Home Page> Our Programs > Digital Inclusion tools
Detected Build Version	V1.1
Summary	Limited denomination options in cash withdrawal function, restricting cash withdrawal only till 3000.
Description	No option of withdrawing of amount excess of 3000.
Steps to Replicate	1) Open the website 2) Select our programs 3) Proceed to Digital Inclusion tools and select cash machine simulator (ATM) 4) Select language and skip to simulator 5) Enter the card 6) Select the account type 7) Go to Other functions and select cash withdrawal
Expected Results	It should add more options in denominations in withdrawal function or it should take amount input from the user.
Actual Results	It is displaying limited options of denominations in cash withdrawal option.

Attachments	Cash Machine Simulator (ATM) 
Remarks	Causes inconvenience to the user in terms of limited cash withdrawal options.
Defect Severity	High
Defect Priority	High
Reported By	Test Engineer1
Assigned To	XYZ
Status	Assigned