**SEPM UNIT 4**

Introduction to testing, Verification, Validation – topics not covered in these notes

**1 TEST STRATEGY AND PLANNING:**

* Software testing is a vast field in itself
* So, it can be considered as a separate project
* There are many techniques available to execute software test projects. It depends on the kind of test project.
* Most test projects must have a test plan and a test strategy before the project can be ready for execution.
* Due to time constraints, testing cycles are cut shorted. This leads to a half-tested product that is pushed out the door.

**1.1 TEST PLANNING**

Test planning is an essential practice for any organization that wishes to develop a test process that is repeatable and manageable.

Test planning should begin early in the software life cycle, although for many organizations whose test processes are immature this practice is not yet in place

**TEST PLANNER - STEPS**

Test plans for software projects are very complex and detailed documents.

The planner usually includes the following essential high-level items.

**1. Overall test objectives**. As testers, why are we testing, what is to be achieved by the tests,

and what are the risks associated with testing this product?

**2. What to test** (scope of the tests). What items, features, procedures, functions, objects,

clusters, and subsystems will be tested?

**3. Who will test**. Who are the personnel responsible for the tests?

**4. How to test.** What strategies, methods, hardware, software tools, and techniques are going

to be applied? What test documents and deliverable should be produced?

**5. When to test**. What are the schedules for tests? What items need to be available?

**6.** **When to stop testing**. It is not economically feasible or practical toplan to test until all

defects have been revealed. This is a goal that testers can never be sure they have reached.

**TEST PLAN - HIERARCHY**

* Test plans can be organized in several ways depending on organizational policy.
* There is often a hierarchy of plans that includes several levels of quality assurance

and test plans.

* The complexity of the hierarchy depends on the type, size, risk-proneness, and the

mission/safety criticality of software system being developed.

* Depending on organizational policy, another level of the hierarchy could contain a

separate test plan for **unit, integration, system, and acceptance tests**.

* The level-based plans give a more detailed view of testing appropriate to that level.
* The persons responsible for developing test plans depend on the type of plan under

development.

* Usually staff from one or more groups cooperates in test plan development.
* At the top of the plan hierarchy there may be a software quality assurance plan.
* This plan gives an overview of all verification and validation activities for the project, as well as details related to other quality issues such as audits, standards, configuration control, and supplier control.
* Then in the plan hierarchy there may be a master test plan that includes an overall description of all execution-based testing for the software system.
* A master verification plan for reviews inspections/walkthroughs would also fit in at this level.
* The master test plan itself may be a component of the overall project plan or exist as a separate document.



**1.2 TEST STRATEGIES**

* Identifying the mistakes by the end-user is costly. Fixing these defects at this stage is costly.
* Test strategies should include things like test prioritization, automation strategy, risk analysis, etc.
* Test planning should include a work breakdown structure, requirement review, resource

allocation, effort estimation, tools selection, setting up communication channels, etc.

**1.2.1 TEST PRIORITIZATION**

* All parts of the software product will not be used by end users with the same intensity. Some parts of the product are used by end users extensively, while other parts are seldom used.
* So the extensively used parts of the product should not have any defects at all and thus they

need to be tested thoroughly.

* High priority on tests which are to be done for these critical parts of the software product and put a low priority on uncritical parts.
* The high priority areas are first tested.
* Once testing is thoroughly done for these parts, then you should start testing low priority areas.

**1.2.2 RISK MANAGEMENT**

* Test manager should plan for all known risks
* If proper risk management not done, costs could escalate, quality could go down
* Risks that have a severe effect:
  + Unrealistic schedule
  + Resource unavailability
  + Skill unavailability
  + Frequent requirement changes
* If requirement changes, whole test plan must be changed
* Some test engineers calculate less effort than actually required
* Unrealistic schedules used to win over customer
* Testing cycle hence gets delayed
* Test manager stuck in such situations
* Professionals may leave the project at short notice
* Hence good alternative resource planning required
* Buffer in schedule must be kept so that schedules don’t get affected
* Test manager must ensure that schedule is realistic

**1.2.3 EFFORT ESTIMATION**

* Effort estimate should include information such as project size, productivity, and test strategy.
* The wideband Delphi technique uses brainstorming sessions to arrive at effort estimate figures after discussing the project details with the project team.
* This is a good technique because the people who will be assigned the project work will know their own productivity levels and can figure out the size of their assigned project tasks from their own experience.
* In the Experience-based technique, instead of group sessions, the test manager meets each team member and asks his estimate for the project work he has been assigned.
* Effort estimation is an area where no test manager can have a good grasp, at the initial stages of the project. As the project unfolds, things become clearer.

**1.2.3.1 TEST POINT ANALYSIS**

* Test Point Analysis (TPA) is a software testing estimate approach, which is specifically designed to estimate the black box testing efforts.
* Unlike Functional point Analysis (FPA), which is used to project the White box testing efforts, TPA is used to estimate the black box testing efforts, especially in performing the system and the acceptance test of the software application.
* The test point analysis technique consists of three inputs to estimate the black box testing efforts: **project size, Test strategy, and productivity.**
* Test points, in turn, are calculated from function points.
* The number of function points is calculated from the number of functions and function complexity.
* TAP calculates the test efforts estimation in test points for highly important functions, according to user and for whole system.
* As the test points of functions are measured, the tests can test important functionalities first then be predicting the risk in testing.
* This technique also consider the quality characteristics, such as functionality, security, usability, efficiency, etc. with proper weighting process for calculating TAP
* **Static testing**: It's a verification activity which examines the whole structure of the application without executing it.
* **Dynamic testing:** It's an actual testing activity performed over the application to evaluate and assess its different features, functionalities and quality aspects.
* It may consist of both explicit and implicit type of testing.

**1.2.3.2 TEST POINT ANALYSIS – STEPS**

1. a) Dynamic and static test points are calculated. Dynamic test points are number of test which are based on dynamic measurable quality. Characteristic of functions in system. To calculate dynamic test point:

* Function points (FPs) assigned to function.
* Function dependent factor (FDC) such as interfacing, function importance, etc.
* Quality characteristics (QC)

b) Static test points are number of test points which are based on quality characteristics of system, it is calculated based on –

* Function points (FPs) assigned to system.
* Quality requirements or test strategy for static quality requirements (QC) like Flexibility, Testability, security, continuity, traceability.

2. Dynamic test point is added to static test point. To get total test points (TTP) for system.

3. Total Test Points is used for calculation of primary test hours (PTH) PTH is effort estimation for primary test hour’s activity such as preparation for primary test hour’s activity such as preparation, specification and execution.

4. PTH is calculated based on environmental factors and productivity factors.

5. Secondary testing activities include management activities like controlling the testing activities.

6. TTH is calculated by adding some allowance to secondary activities and PTH.

7. Thus TTH is final effort estimation for testing activities.



**2. TEST PROJECT MONITORING AND CONTROL:**

* Project monitoring (or tracking) refers to the activities and tasks managers engage in to periodically check the status of each project.
* Reports are prepared that compare the actual work done to the work that was planned.
* Monitoring requires a set of tools, forms, techniques, and measures.
* A precondition for monitoring a project is the existence of a project plan

**2.1 PROJECT CONTROLLING**

* Project controlling consists of developing and applying a set of corrective actions to get a project on track when monitoring shows a deviation from what was planned.
* If monitoring results show deviations from the plan have occurred, controlling mechanisms must be put into place to direct the project back on its proper track.
* Controlling a project is an important activity which is done to ensure that the project goals will be achieved occurring to the plan
  + 1. **MAJOR TASKS – PROJECT CONTROLLING**

1. **Develop standards of performance -** These set the stage for defining goals that will be achieved when project tasks are correctly accomplished
2. **Plan each project -** The plan must contain measurable goals, milestones, deliverables, and well-defined budgets and schedules that take into consideration project types, conditions, and constraints**.**
3. **Establish a monitoring and reporting system -** In the monitoring and reporting system description the organization must describe the measures to be used, how/when they will be collected, what questions they will answer, who will receive the measurement reports, and how these will be used to control the project. Each project plan must describe the monitoring and reporting mechanisms
4. **Measure and Analyse Results -** Measurements for monitoring and controlling must be collected, organized, and analyzed. They are then used to compare the actual achievements with standards, goals, and plans**.**
5. **Initiate corrective actions for projects that are off-track -** These actions may require changes in the project requirements and the project plan.
6. **Reward and Discipline -** Reward those staff who have shown themselves to be good performers, and discipline, retrain, relocate those that have consistently performed poorly.
7. **Document the monitoring and controlling system -** All the methods, forms, measures, and tools that are used in the monitoring and controlling process must be documented in organization standards and be described in policy statements.
8. **Utilize a configuration management system -** A configuration management system is needed to manage versions, releases, and revisions of documents, code, plans, and reports.

**2.2 TEST CASE DESIGN**

* A good test case design technique is crucial to improving the quality of the software testing process. This helps to improve the overall quality and effectiveness of the released software.
* The main purpose of test case design techniques is to test the functionalities and features of the software with the help of effective test cases.
* The test case design techniques are broadly classified into three major categories.

1. Specification-Based techniques
2. Structure-Based techniques
3. Experience-Based techniques

**2.2.1 Specification-Based or Black-Box techniques**

This technique leverages the external description of the software such as technical specifications, design, and client’s requirements to design test cases. The technique enables testers to develop test cases that provide full test coverage. The Specification-based or black box test case design techniques are divided further into 5 categories. These categories are as follows:

1. Boundary Value Analysis (BVA)
2. Equivalence Partitioning (EP)
3. Decision Table Testing
4. 4 State Transition Diagrams
5. Use Case Testing
6. **Boundary Value Analysis (BVA)**

* This technique is applied to explore errors at the boundary of the input domain. BVA catches any input errors that might interrupt with the proper functioning of the program.

1. **Equivalence Partitioning (EP)**

* In Equivalence Partitioning, the test input data is partitioned into a number of classes having an equivalent number of data.
* The test cases are then designed for each class or partition. This helps to reduce the number of test cases.

1. **Decision Testing Coverage**

* This technique is also known as branch coverage is a testing method in which each one of the possible branches from each decision point is executed at least once to ensure all reachable code is executed.
* This helps to validate all the branches in the code. This helps to ensure that no branch leads to unexpected behavior of the application.

1. **State Transition Diagrams**

* In this technique, the software under test is perceived as a system having a finite number of states of different types.
* The transition from one state to another is guided by a set of rules.
* The rules define the response to different inputs.
* This technique can be implemented on the systems which have certain workflows within them.

1. **Use Case Testing**

* A use case is a description of a particular use of the software by a user. In this technique, the test cases are designed to execute different business scenarios and end-user functionalities.
* Use case testing helps to identify test cases that cover the entire system.

**2.2.2 STRUCTURE BASED / WHITE-BOX TECHNIQUE**

* The structure-based or white-box technique design test cases based on the internal structure of the software.
* This technique exhaustively tests the developed code.
* Developers who have complete information of the software code, its internal structure, and design help to design the test cases.
* This technique is further divided into five categories.
  1. Statement Testing & Coverage
  2. Decision Testing Coverage
  3. Condition Testing
  4. Multiple Condition Testing
  5. All Path Testing

1. **Statement Testing & Coverage**

* This technique involves execution of all the executable statements in the source code at least once.
* The percentage of the executable statements is calculated as per the given requirement.
* This is the least preferred metric for checking test coverage.

1. **Decision Testing Coverage**

* This technique is also known as branch coverage is a testing method in which each one of the possible branches from each decision point is executed at least once to ensure all reachable code is executed.
* This helps to validate all the branches in the code. This helps to ensure that no branch leads to unexpected behavior of the application.

1. **Condition Testing**

* Condition testing also is known as Predicate coverage testing, each Boolean expression is predicted as TRUE or FALSE.
* All the testing outcomes are at least tested once.
* This type of testing involves 100% coverage of the code.
* The test cases are designed as such that the condition outcomes are easily executed.

1. **Multiple Condition Testing**

* The purpose of Multiple condition testing is to test the different combination of conditions to get 100% coverage.
* To ensure complete coverage, two or more test scripts are required which requires more efforts.

1. **All Path Testing**

* In this technique, the source code of a program is leveraged to find every executable path.
* This helps to determine all the faults within a particular code.

**2.2.3 EXPERIENCE BASED TESTING**

These techniques are highly dependent on tester’s experience to understand the most important areas of the software. The types of experience-based testing are

1. **Error Guessing**

* In this technique, the testers anticipate errors based on their experience, availability of data and their knowledge of product failure.
* Error guessing is dependent on the skills, intuition, and experience of the testers.

1. **Exploratory Testing**

* This technique is used to test the application without any formal documentation.
* There is minimum time available for testing and maximum for test execution.
* In exploratory testing, the test design and test execution are performed concurrently.

**MASTER TEST PLAN**

* The master test plan is a document that describes in detail how the testing is being planned and how it will be managed across different test levels.
* It gives a bird’s eye view of the key decisions taken, the strategies to be implemented and the testing effort involved in the project.
* The master test plan for software testing provides the following details:
  + 1. List of tests to be performed
    2. Testing levels to be covered
    3. Relationship among different test levels and related coding activity
    4. Test Implementation Strategy
    5. Explain testing effort which is a component of the project
    6. Master test plan should align with test policy and test strategy. It should list any exceptions or deviations and their possible impact
* The exact structure of the master test plan and its content depends on the following factors:
  + Type of organization
  + Documentation standards followed by the organization
  + Level of project formality

**2.3 TEST CASE MANAGEMENT**

Test case management involves managing different versions of test cases, keeping track of changes in them, keeping a separate repository of test cases based on type of tests, as well as creating and managing automation scripts.

**Understanding the Idea of a Test Case**

* Requirements tell you what the software needs to do.
* Test scenarios define a means for confirming the requirement.
* Test cases describe how, the component pieces of confirming that requirement.
* Test scripts (whether automated or executed manually) tell you exactly how to execute that component piece.
* Typically, you'll have a number of test cases per test scenario, covering various permutations of inputs and behaviors. You then have one or possibly more scripts per test case.

**ANATOMY – TEST CASE**

1. **An ID:** A unique way of identifying the test case.
2. **Title or brief description:** A quick means of understanding the software activity in question.
3. **Related requirement and/or test:** What broader scenario and requirement does this test case roll up to?
4. **Remarks/Notes:** Free form comments about the test case.
5. **Script:** Exact steps for executing the test.
6. **Pass/Fail Status:** Is the test case currently passing or failing?
7. **History and Audit Trail:** You should be able to see its pass/fail history as well as general changes and who has run/modified the test case.

**2.4 TEST BED**

* Test bed preparation involves installing the application on a machine that is accessible to all test teams.
* Care is taken to ensure that this machine is free of any interference from unauthorized access.
* Test data is populated in the application.
* Care should also be taken to ensure that the test bed resembles the production environment as closely as possible, including all software and hardware configurations.

**2.5 TEST CASE EXECUTION**

Test case execution involves executing prepared test cases manually or using automation tools to execute them.

* For regression tests, automated test execution is a preferred method.
* After each test case is executed, it may pass or fail.
* If it fails then defects have to be logged.
* Exit criteria for test case execution cycle are generally defined in advance.
* Generally, when a certain level of quality of the application is reached, then test execution stops.

**2.6 DEFECT TRACKING**

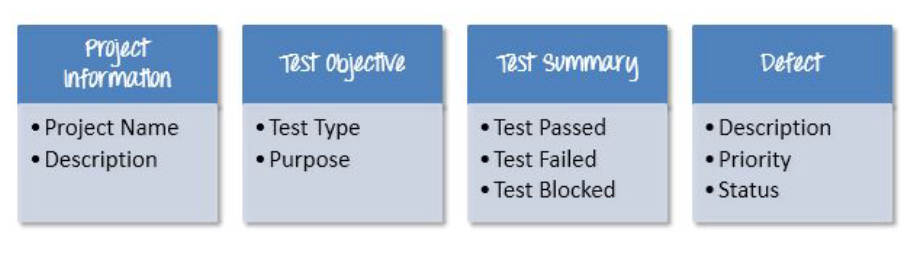
* Defect tracking is one of the most important activities in a test project
* During defect tracking it is ensured that defects are logged and get fixed.
* All defects and their fixing are tracked carefully
* Defect count per hour per day is a common way of measuring performance of a test team



* If the testing is done for an in-house software product, traditionally, it used to not be a performance evaluation measurement.
* A good defect tracking application should be deployed on a central server that is accessible to all test and development teams.
* Each defect should be logged in such a way that it could be understood by both development and testing teams.
* Generally, the defects should be reproducible, but in many instances, this is difficult

**3 TEST REPORTING**

* During the execution of a test project, many initial and final reports are made.
* Test Report is a document which contains a summary of test activities and final test results
* It contains an assessment of how well the testing is performed
* Based on the test report, the stakeholders can evaluate the quality of the tested product
* Make a decision on the software release





**Project Information**

All information of the project such as the project name, product name, and version should be described in the test report.

**Test Objective**

Test Report should include the objective of each round of testing, such as Unit Test, Performance Test, System Test.. etc

**Test Summary**

This section includes the summary of testing activity in general. Information detailed here includes

* The number of test cases executed
* The numbers of test cases pass
* The numbers of test cases fail
* Pass percentage
* Fail percentage
* Comments

This information should be displayed visually by using color indicator, graph, and highlighted table.

**Defect**

One of the most important information in Test Report is defect. The report should contain following information

* Total number of bugs
* Status of bugs (open, closed, responding)
* Number of bugs open, resolved, closed
* Breakdown by severity and priority
* Like test summary, you can include some simple metrics like Defect density, % of fixed defects.

**OTHER REPORTING DOCUMENTS**

Test reports include

* test planning reports
* test strategy reports,
* requirement document
* review comments
* number of test cases created
* automation scripts created
* test execution cycle reports
* defect tracking reports, etc.

Some other reports include traceability matrix reports, defect density, test execution rate, test creation rate, test automation and script writing rate, etc.

**4 TEST ARTIFACTS**

An integral part of software testing, test artifacts are the various by-products generated during the process of software testing, which are then shared with the clients, team managers, team lead, and other team members and stakeholders associated with the project.

They include test plan document, test strategy document, test cases, test cycle logs, defect list, verification and validation reports, and product certification.

**Management Artifacts**

* Customers are concerned not only with project cost and schedule, but they are also concerned with critical defects, which the test team has either detected or not.
* So the management artifacts (metrics) include project cost compliance, project schedule compliance, and quality (number of critical defects caught versus number of critical defects which went into production).
* Management artifacts include traceability matrix, defect density rate, resource loading etc.
* **Requirement traceability matrix**, as the name indicates is a matrix that contains tables which show many to many relationships between the requirement and the test cases.
* The matrix shows which requirements are implemented by which one test case or test cases and which test case/ cases are mapped to which specific requirement or requirements
* **Defect Density** - Defect Density is the number of defects confirmed in software/module during a specific period of operation or development divided by the size of the software/module.
* It enables one to decide if a piece of software is ready to be released. Defect density is counted per thousand lines of code also known as KLOC

**5 SOFTWARE TESTING IN ITERATIVE MODEL**

In an iterative model, each iteration is a short cycle. So the amount of testing in each iteration is also small. Thus, unlike in waterfall model, software testing has a lesser role in the iterative development life cycle.

Generally, software defects tend to increase with the size of software products. Since in iteration mode the software product is small, there will be fewer defects in the product. Although in reality, as the software product grows in size over many iterations, the number of defects per line of software code is bound to increase. In iterative development, regression testing is also a big issue. In each iteration, there will be a large number of regression test cases to run. As the product size increases with iterations, the set of regression test cases also increases. It becomes a liability after a while. Manually running all those regression tests takes a lot of time, which becomes a hindrance for the release schedule. In such cases, the best option is to go for automation of these regression test cases. Automated test cases take much less time (sometimes if the manual running of test cases was taking 5 days, after automation it took only 5 h) to run.