UNIT 4

TEST MANAGEMENT

Test management is concerned with both test resource and test environment management. It is the role of test management to ensure that new or modified service products meet the business requirements for which they have been developed or enhanced. The key elements of test management are

Test organization: - It is the process of setting up and managing a suitable test organizational structure and defining explicit roles. The project framework under which the testing activities will be carried out is reviewed, high-level test phase plans are prepared, and resource schedules are considered. Test organization also involves the determination of configuration standards and defining the test environment.

Test planning: - The requirements definition and design specifications facilitate the identification of major test items and these may necessitate updating the test strategy. A detailed test plan and schedule is prepared with key test responsibilities being indicated.

Detailed test design and test specifications: - A detailed design is the process of designing a meaningful and useful structure for the tests as a whole. It specifies the details of the test approach for software functionality or feature and identifying the associated test cases.

Test monitoring and assessment: - It is the ongoing monitoring and assessment to check the integrity of development and construction. The status of configuration items should be reviewed against the phase plans and the test progress reports prepared, the ensure the verification and validation activities are correct.

Product quality assurance: - The decision to negotiate the acceptance testing program and the release and commissioning of the service product is subject to the 'product assurance' role being satisfied with the outcome of the verification and validation activities. Product assurance may oversee some of the test activity and may participate in process reviews.

TEST ORGANIZATION: -

Since testing is viewed as a process, it must have an organization such that a testing group works for better testing and high quality software. The testing group is responsible for the following activities:

- Maintenance and application of test policies „
- Development and application of testing standards „
- Participation in requirement, design, and code reviews „
- Test planning „
- Test execution „
- Test measurement „
- Test monitoring „
- Defect tracking „
- Acquisition of testing tools „
- Test reporting

The staff members of such a testing group are called test specialists or test engineers or simply testers. A tester is not a developer or an analyst. He does not debug the code or repair it. He is responsible for ensuring that testing is effective and quality issues are being addressed. The skills a tester should possess are

1. **Personal and Managerial Skills: -**
   - Testers must be able to contribute in policy-making and planning the testing activities.
   - Testers must be able to work in a team.
   - Testers must be able to organize and monitor information, tasks, and people.
   - Testers must be able to interact with other engineering professionals, software quality assurance staff, and clients.
   - Testers should be capable of training and mentoring new testers.
   - Testers should be creative, imaginative, and experiment-oriented.
   - Testers must have written and oral communication skills.

2. **Technical Skills: -**
   - Testers must be technically sound, capable of understanding software engineering principles and practices.
   - Testers must be good in programming skills.
   - Testers must have an understanding of testing basics, principles, and practices.
   - Testers must have a good understanding and practice of testing strategies and methods to develop test cases.
   - Testers must have the ability to plan, design, and execute test cases with the goal of logic coverage.
   - Testers must have technical knowledge of networks, databases, operating systems, etc. needed to work in a the project environment.
   - Testers must have the knowledge of configuration management.
   - Testers must have the knowledge of test ware and the role of each document in the testing process.
   - Testers must have known about quality issues and standards.

STRUCTURE OF TESTING GROUP: -

Testing is an important part of any software project. One or two testers are not sufficient to perform testing, especially if the project is too complex and large. Therefore, many testers are required at various levels. Figure 9.1 shows different types of testers in a hierarchy.

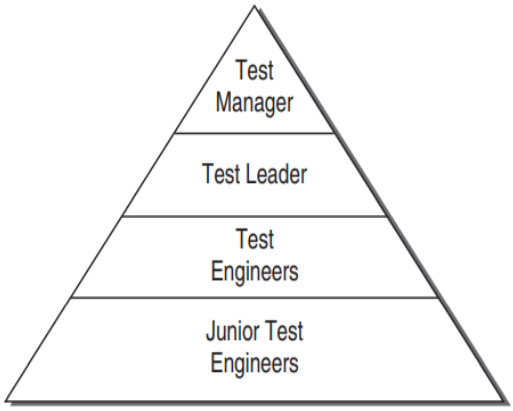

Figure 9.1 Testing Group Hierarchy

Test Manager: -

A test manager occupies the top level in the hierarchy. He has the following responsibilities:

i. He is the key person in the testing group who will interact with project management, quality assurance, and marketing staff.
ii. Takes responsibility for making test strategies with detailed master planning and schedule.
iii. Interacts with customers regarding quality issues.
iv. Acquires all the testing resources including tools.
v. Monitors the progress of testing and controls the events.

vi. Participates in all static verification meetings.
vii. Hires, fires, and evaluates the test team members.

Test Leader: -

The next tester in the hierarchy is the test leader who assists the test manager in meeting testing and quality goals. The prime responsibility of a test leader is to lead a team of test engineers who are working at the leaf-level of the hierarchy. The following are his responsibilities:

i. Planning the testing tasks given by the test manager.
ii. Assigning testing tasks to test engineers who are working under him.
iii. Supervising test engineers.
iv. Helping the test engineers in test case design, execution, and reporting.
v. Providing tool training, if required.
vi. Interacting with customers.

Test Engineers: -

Test engineers are highly experienced testers. They work under the leadership of the test leader. They are responsible for the following tasks:

i. Designing test cases.
ii. Developing test harness.
iii. Set-up test laboratories and environment.
iv. Maintain the test and defect repositories.

Junior Test Engineers: -

Junior test engineers are newly hired testers. They usually are trained about the test strategy, test process, and testing tools. They participate in test design and execution with experienced test engineers.

TEST PLANNING: -

There is a general human tendency to 'get on with the next thing', especially under pressure. This is true for the testers who are always short of time.

However, if resources are to be utilized intelligently and efficiently during the earlier testing phases and later phases, these are repaid many times over. The time spent on planning the testing activities early is never wasted and usually the total time cycle is significantly shorter.

According to the test process as discussed in STLC, testing also needs planning as is needed in SDLC. Since software projects become uncontrolled if not planned properly, the testing process is also not effective if not planned earlier. Moreover, if testing is not effective in a software project, it also affects the final software product. Therefore, for a quality software, testing activities must be planned as soon as the project planning starts.

A test plan is defined as a document that describes the scope, approach, resources, and schedule of intended testing activities. Test plan is driven with the business goals of the product. In order to meet a set of goals, the test plan identifies the following:

- Test items
- Features to be tested
- Testing tasks
- Tools selection
- Time and effort estimate
- Who will do each task
- Any risks
- Milestones

1. **TEST PLAN COMPONENTS: -**
IEEE Std 829–1983 has described the test plan components. These components (see Fig. 9.2) must appear for every test item. These components are described below.

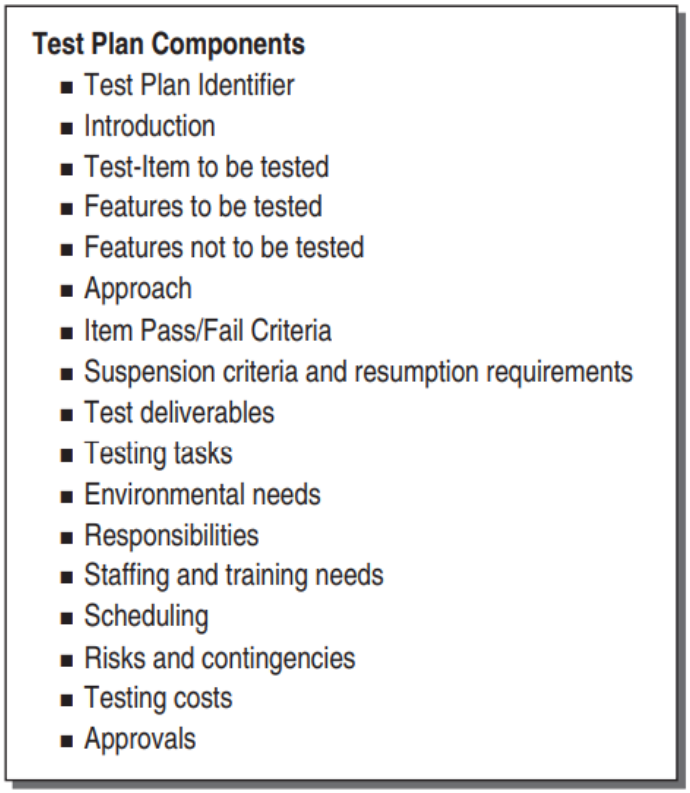

Figure 9.2 Test plan components

- **Test Plan Identifier: -**Each test plan is tagged with a unique identifier so that it is associated with a project.
- **Introduction: -** The test planner gives an overall description of the project with:
  - Summary of the items and features to be tested.
  - The requirement and the history of each item (optional).
  - High-level description of testing goals.
  - References to related documents, such as project authorization, project plan, QA plan, configuration management plan, relevant policies, and relevant standards.
- **Test-Item to be Tested: -**
  - Name, identifier, and version of test items.
  - Characteristics of their transmitting media where the items are stored, for example, disk, CD, etc.
  - References to related documents, such as requirements specification, design specification, users' guide, operations guide, installation guide.
  - References to bug reports related to test items.
  - Items which are specifically not going to be tested (optional).
- **Features to be Tested: -**
This is a list of what needs to be tested from the user's viewpoint. The features may be interpreted in terms of functional and quality requirements.
  - All software features and combinations of features are to be tested.
  - References to test-design specifications associated with each feature and combination of features.
- **Features Not to be Tested: -**
This is a list of what should 'not' be tested from both the user's viewpoint and the configuration management/version control view:
  - All the features and the significant combinations of features which will not be tested.

- Identify why the feature is not to be tested. There can be many reasons:
  - ✓ Not to be included in this release of the software.
  - ✓ Low-risk, has been used before, and was considered stable.
  - ✓ Will be released but not tested or documented as a functional part of the release of this version of the software.

- **Approach: -**
  - Overall approach to testing.
  - For each major group of features or combinations of features, specify the approach.
  - Specify major activities, techniques, and tools which are to be used to test the groups. „
  - Specify the metrics to be collected. „
  - Specify the number of configurations to be tested. „
  - Specify a minimum degree of comprehensiveness required. „
  - Identify which techniques will be used to judge comprehensiveness. „
  - Specify any additional completion criteria. „
  - Specify techniques which are to be used to trace requirements. „
  - Identify significant constraints on testing, such as test-item availability, testing-resource availability, and deadline.
- **Item Pass/Fail Criteria: -**

  This component defines a set of criteria based on which a test case is passed or failed. The failure criteria are based on the severity levels of the defect. Thus, an acceptable severity level for the failures revealed by each test case is specified and used by the tester. If the severity level is beyond an acceptable limit, the software fails.
- **Suspension Criteria and Resumption Requirements: -**

  Suspension criteria specify the criteria to be used to suspend all or a portion of the testing activities, while resumption criteria specify when the testing can resume after it has been suspended.

  For example, system integration testing in the integration environment can be suspended in the following circumstances:
  - Unavailability of external dependent systems during execution.
  - When a tester submits a 'critical' or 'major' defect, the testing team will call for a break in testing while an impact assessment is done.

  System integration testing in the integration environment may be resumed under the following circumstances:
  - When the 'critical' or 'major' defect is resolved.
  - When a fi x is successfully implemented and the testing team is notified to continue testing.
- **Test Deliverables: -**
  - Identify deliverable documents: test plan, test design specifications, test case specifications, test item transmittal reports, test logs, test incident reports, test summary reports, and test harness (stubs and drivers).
  - Identify test input and output data.
- **Testing Tasks: -**
  - Identify the tasks necessary to prepare for and perform testing.
  - Identify all the task interdependencies.
  - Identify any special skills required.

  All testing-related tasks and their interdependencies can be shown through a work breakdown structure (WBS). WBS is a hierarchical or tree-like representation of all testing tasks that need to be completed in a project.
- **Environmental Needs: -**
  - Specify necessary and desired properties of the test environment: physical characteristics of the facilities including hardware, communications and system software, the mode of usage (i.e. stand-alone), and any other software or supplies needed.

- Specify the level of security required.
- Identify any special test tools needed.
- Identify any other testing needs.
- Identify the source for all needs which are not currently available.

- **Responsibilities: -**
  - Identify the groups responsible for managing, designing, preparing, executing, checking, and resolving.
  - Identify the groups responsible for providing the test items identified in the test items section.
  - Identify the groups responsible for providing the environmental needs identified in the environmental needs section.
- **Staffing and Training Needs: -**
  - Specify staffing needs by skill level.
  - Identify training options for providing necessary skills.
- **Scheduling: -**
  - Specify test milestones.
  - Specify all item transmittal events. „
  - Estimate the time required to perform each testing task. „
  - Schedule all testing tasks and test milestones. „
  - For each testing resource, specify a period of use
- **Risks and Contingencies: -**

  Specify the following overall risks to the project with an emphasis on the testing process:
  - Lack of personnel resources when testing is to begin.
  - Lack of availability of required hardware, software, data, or tools. „
  - Late delivery of the software, hardware, or tools. „
  - Delays in training on the application and/or tools. „
  - Changes to the original requirements or designs. „
  - Complexities involved in testing the applications.

  Specify the actions to be taken for various events. An example is given below.

  Requirements definition will be complete by January 1, 20XX and, if the requirements change after that date, the following actions will be taken:

  The test schedule and the development schedule will move out an appropriate number of days. This rarely occurs, as most projects tend to have fixed delivery dates.
  - The number of tests performed will be reduced.
  - The number of acceptable defects will increase.
  - These two items may lower the overall quality of the delivered product.
  - Resources will be added to the test team.
  - The test team will work overtime.
  - The scope of the plan may be changed.
  - There may be some optimization of resources. This should be avoided, if possible, for obvious reasons.

  The management team is usually reluctant to accept scenarios such as the one mentioned above even though they have seen it happen in the past. The important thing to remember is that, if you do nothing at all, testing is cut back or omitted completely, neither of which should be an acceptable option.
- **Testing Costs: -**

  The IEEE standard has not included this component in its specification. However, it is a usual component of any test plan, as test costs are allocated in the total project plan. To estimate the costs, testers will need tools and techniques. The following is a list of costs to be included:
  - Cost of planning and designing the tests
  - Cost of acquiring the hardware and software required for the tests „
  - Cost to support the environment „

- Cost of executing the tests „
- Cost of recording and analysing the test results „
- Cost of training the testers, if any „
- Cost of maintaining the test database

- **Approvals: -**
  - Specify the names and titles of all the people who must approve the plan.
  - Provide space for signatures and dates.

2. TEST PLAN HIERARCHY: -

Test plans can be organized in several ways depending on the organizational policy. There is often a hierarchy of plans that includes several levels of quality assurance and test plans. At the top of the plan hierarchy is a master plan which 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, and configuration control. Below the master test plan, there is individual planning for each activity in verification and validation, as shown in Fig. 9.3.

The test plan at each level of testing must account for information that is specific to that level, e.g. risks and assumptions, resource requirements, schedule, testing strategy, and required skills. The test plans according to each level are written in an order such that the plan prepared first is executed last, i.e. the acceptance test plan is the first plan to be formalized but is executed last. The reason for its early preparation is that the things required for its completion are available first.

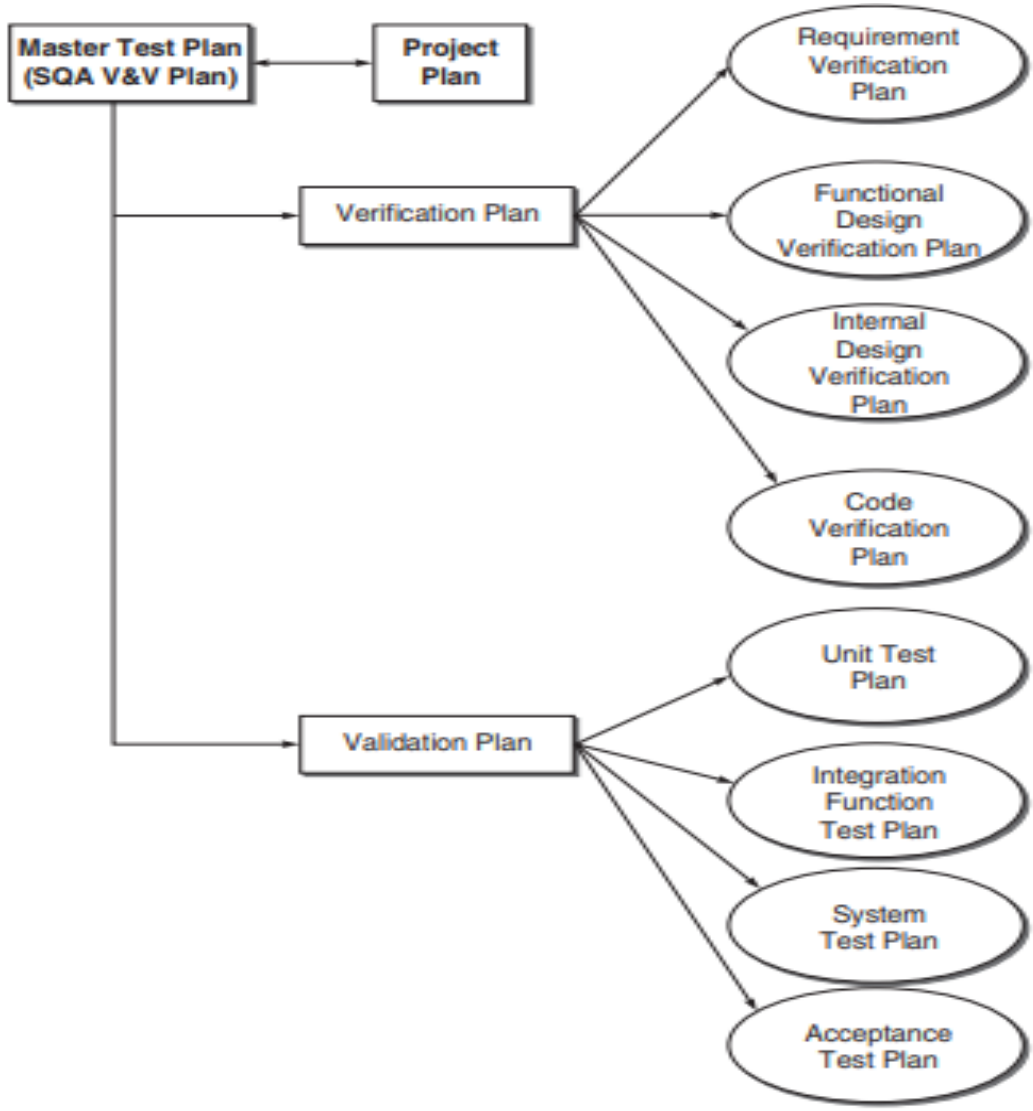

Figure 9.3 Test Plan Hierarchy

3. MASTER TEST PLAN: -

The master test plan provides the highest level description of verification and validation efforts and drives the testing at various levels. General project information is used to develop the master test plan. The following topics must be addressed before planning:

- Project identification „
- Plan goals „
- Summary of verification and validation efforts „
- Responsibilities conveyed with the plan „
- Software to be verified and validated „
- Identification of changes to organization standards

Verification and validation planning may be broken down into the following steps:

- Identify the V&V scope „
- Establish specific objectives from the general project scope „
- Analyse the project input prior to selecting V&V tools and techniques „
- Select tools and techniques „
- Develop the software verification and validation plan (SVVP).

Discussed below are the major activities of V&V planning.

Master Schedule: -

- The master schedule summarizes various V&V tasks and their relationship to the overall project.
- Describes the project life cycle and project milestones including completion dates
- Summarizes the schedule of V&V tasks and how verification and validation results provide feedback to the development process to support overall project management functions.
- Defines an orderly flow of material between project activities and V&V tasks. Use reference to PERT, CPM, and Gantt charts to define the relationship between various activities.

Resource Summary: -

This activity summarizes the resources needed to perform V&V tasks, including staffing, facilities, tools, finances, and special procedural requirements such as security, access rights, and documentation control. In this activity,

- Use graphs and tables to present resource utilization.
- Include equipment and laboratory resources required.
- Summarize the purpose and cost of hardware and software tools to be employed.
- Take all resources into account and allow for additional time and money to cope with contingencies.

Responsibilities: -

Identify the organization responsible for performing V&V tasks. There are two levels of responsibilities—general responsibilities assigned to different organizations and specific responsibilities for the V&V tasks to be performed, assigned to individuals. General responsibilities should be supplemented with specific responsibility for each task in the V&V plan.

Tools, Techniques, and Methodology: -

Identify the special software tools, techniques, and methodologies to be employed by the V&V team. The purpose of each should be defined and plans for the acquisition, training, support, and qualification of each should be described. This section may be in a narrative or graphic format. A separate tool plan may be developed for software tool acquisition, development, or modification. In this case, a separate tool plan section may be added to the plan.

4. VERIFICATION TEST PLAN: -

Verification test planning includes the following tasks:

- The item on which verification is to be performed. „
- The method to be used for verification: review, inspection, walkthrough. „
- The specific areas of the work product that will be verified. „
- The specific areas of the work product that will not be verified. „
- Risks associated. „
- Prioritizing the areas of work product to be verified. „
- Resources, schedule, facilities, tools, and responsibilities.

5. VALIDATION TEST PLAN: -

Validation test planning includes the following tasks:

- Testing techniques „
- Testing tools „
- Support software and documents „
- Configuration management „
- Risks associated, such as budget, resources, schedule, and training.

Unit Test Plan: -

Generally, unit tests are performed by the module developer informally. In this case, bugs are not recorded and do not become a part of the unit's history. It is a bad practice because these bugs' history will not be re-used, which otherwise becomes important especially in real-time systems. Therefore, to implement unit testing formally and to make it an effective testing for other projects, it is necessary to plan for unit test. To prepare for unit test, the developer must do the following tasks:

- Prepare the unit test plan in a document form.
- Design the test cases.
- Define the relationship between tests.
- Prepare the test harness, i.e. stubs and drivers, for unit testing.

Discussed below are the tasks contained in a test plan document.

- **Module overview: -** Briefly define the purpose of this module. This may require only a single phrase, i.e. calculates overtime pay amount, calculates equipment depreciation, performs date edit validation, or determines sick pay eligibility, etc.
- **Inputs to module: -** Provide a brief description of the inputs to the module under test.
- **Outputs from module: -** Provide a brief description of the outputs from the module under test.
- **Logic flow diagram: -** Provide a logic flow diagram, if additional clarity is required. For example, DFD, STD, and ERD.
- **Test approach: -** Decide the method to test the module.
- **Features to be tested: -** Identify the features to be tested for the module.
- **Features not to be tested: -** Identify the features not be tested due to some constraints. Also, assess the risks of not testing these features.
- **Test constraints: -** Indicate anticipated limitations on the test due to test conditions, such as interfaces, equipment, personnel, and databases.
- **Test harness: -** Describe the requirements for test harness that interfaces with the units to be tested.
- **Interface modules: -** Identify the modules that interface with this module indicating the nature of the interface: outputs data to, receives input data from, internal program interface, external program interface, etc. Identify the sequencing required for subsequent string tests or sub-component integration tests.

- **Test tools: -** Identify any tools employed to conduct unit testing. Specify any stubs or utility programs developed or used to invoke tests. Identify names and locations of these aids for future regression testing. If data is supplied from the unit test of a coupled module, specify module relationship.
- **Archive plan: -** Specify how and where the data is archived for use in subsequent unit tests. Define any procedures required to obtain access to data or tools used in the testing effort. The unit test plans are normally archived with the corresponding module specifications.
- **Updates: -** Define how updates to the plan will be identified. Updates may be required due to enhancements, requirement changes, etc. The same unit test plan should be re-used with revised or appended test cases identified in the update section.
- **Milestones: -** List the milestone events and dates for unit testing.
- **Budget: -** List the funds allocated to test this module. A checklist can also be maintained, as shown below:
  - Does the test plan completely test the functionality of the unit as defined in the design specification and in functional specification?
  - Are the variable (including global variables) inputs to the unit included in the features to be tested in the test plan?
  - Are the variable outputs from the unit result included in the features to be tested?
  - Is every procedure in the design exercised by the test?
  - Does the test plan include critical procedures to demonstrate that performance meets stated requirements?
  - Does the test plan include critical procedures to demonstrate that performance meets stated requirements?

Integration Test Plan: -

An integration test plan entails the development of a document, which instructs a tester what tests to perform in order to integrate and test the already existing individual code modules. The objective of the integration test plan activity is to develop a plan which specifies the necessary steps needed to integrate individual modules and test the integration. It also helps the technical team to think through the logical sequence of integration activities, so that their individual detailed development plans are well-synchronized, and integration happens in a reasonable manner.

Create a draft of the plan very early in the execution phase. By this time, you will know from the high-level design work in the initiation phase, what major pieces of hardware and/or software will have to be integrated. An early look at integration sequencing will help you sanity-check the amount of time you have put into the schedule for integration testing.

Discussed below are the tasks to be performed under integration test plan.

- **Documents to be used: -** All the documents needed should be arranged, e.g. SRS, SDD, user manual, usage scenarios containing DFDs, data dictionaries, state charts, etc.
- **Modules for integration: -** It should be planned which modules will be integrated and tested. There can be many criteria for choosing the modules for integrating. Some are discussed below.
  - **Availability: -** All the modules may not be ready at one time. But we cannot delay the integration and testing of modules which are ready. Therefore, the modules which are ready for integration must be integrated and tested first.
  - **Subsystems: -** The goal of integration testing is also to have some working subsystems and combine them into the working system as a whole. While planning for integration, the subsystems are selected based on the requirements, user needs, and availability of the modules. The subsystems may also be prioritized based on key or critical features. Sometimes, developers need to show the clients some subsystems that are working at the time of integration testing.
- **Strategy for integration: -** The strategy selection for integration testing should also be well-planned. Generally, sandwich integration testing is adopted.

- **Test harness: -** A factor to be considered while developing an integration test plan is the amount of 'dead code' or test harness that will have to be developed. The drivers and stub code that is developed is thrown away and never used again, so a minimum amount of development of this code is desired. Thus, if the program uses a lot of low-level routines such that it is not practical to write a driver for every low-level routine, then perhaps the level above should be developed first, with drivers for it.
- **List of tasks to be tested: -** The functionalities and interfaces to be tested through the integration of modules must also be planned.
- **List of tasks not to be tested: -** Plan the functionalities and interfaces not to be tested due to whatever reasons.

Function Test Plan: -

During planning, the test lead with assistance from the test team defines the scope, schedule, and deliverables for the function test cycle. The test lead delivers a test plan (document) and a test schedule (work plan)—these often undergo several revisions during the testing cycle. The following are the tasks under function test planning:

- **List the objectives of function testing: -** A list of the overall objectives for performing function testing is prepared.
- **Partitioning/Functional decomposition: -** Functional decomposition of a system (or partitioning) is the breakdown of a system into its functional components or functional areas. Another group in the organization may take responsibility for the functional decomposition (or model) of the system, but the testing organization should still review this deliverable for completeness before accepting it into the test organization. If the functional decomposition or partitions have not been defined or are deemed insufficient, then the testing organization will have to take responsibility for creating and maintaining the partitions.
- **Traceability matrix formation: -** Test cases need to be traced/mapped back to the appropriate requirement. A function coverage matrix is prepared. This matrix is a table, listing specific functions to be tested, the priority for testing each function, and test cases that contain tests for each function. Once all the aspects of a function have been tested by one or more test cases, the test design activity for that function can be considered complete. This approach gives a more accurate picture of the application when coverage analysis is done.
- **List the functions to be tested: -** A list of all the functions mentioned in the traceability matrix with their appropriate details needed for planning, is prepared.

System Test Plan: -

One of the most important parts of software development is testing. Before you implement a new software into a system, you must be sure it won't crash and that proper bug checking has been done. Testing ensures that your system runs properly and efficiently and meets all the requirements. To fully ensure that your system is free of defects, testing should follow a system test plan and must be thorough and complete.

One of the most important parts of software development is testing. Before you implement a new software into a system, you must be sure it won't crash and that proper bug checking has been done. Testing ensures that your system runs properly and efficiently and meets all the requirements. To fully ensure that your system is free of defects, testing should follow a system test plan and must be thorough and complete.

System testing is also difficult, as there are no design methodologies for test cases because requirements and objectives do not describe the functions precisely. However, requirements are general. Therefore, a system test plan first requires that requirements are formulated specifically. For this system, test cases are divided into some categories, according to which the system test plan is described. For example, system testing is described in terms of performance

testing, then the test plans are prepared according to performance checking. Similarly, test plans can be prepared for stress testing, compatibility testing, security testing, etc.
The following steps describe how a system test plan is implemented.

- **Partition the requirements: -** Partition the requirements into logical categories and for each category, develop a list of detailed requirements and plan.
- **System description: -** Provide a chart and briefly describe the inputs, outputs, and functions of the software being tested as a frame of reference for the test descriptions.
- **Features to be tested: -** The set of functionalities of the test items to be tested must be recognized, as there may be several criteria for system testing.
- **Strategy and reporting format: -** The strategy to be followed in system testing should also be planned. The type of tests to be performed, standards for documentation, mechanism for reporting, and special considerations for the type of project must also be planned, e.g. a test strategy might say that blackbox testing techniques will be used for functional testing.
- **Develop a requirement coverage matrix: -** Develop a requirement coverage matrix which is a table in which an entry describes a specific subtest, priority of the subtest, the specific test cases in which the subtest appears. This matrix specifies the functions that are to be tested, defining the objectives of the test.
- **Smoke test set: -** The system test plan also includes a group of test cases that establish that the system is stable and all major functionalities are working. This group of test cases is referred to as smoke tests or testability assessment criteria.
- **Entry/Exit criteria: -** The entry criteria will contain some exit criteria from the previous level as well as establishment of test environment, installation of tools, gathering of data, and recruitment of testers, if necessary. The exit criteria also need to be established, e.g. all test cases have been executed and no major identified bugs are open.
- **Suspension criteria: -** It identifies the conditions that suspend testing, e.g. critical bugs preventing further testing.
- **Resource requirements: -** State the resource requirements including documents, equipment, software, and personnel.
- **Test team: -** State the members who are in the test team and enlist their assignments.
- **Participating organizations: -** System testing may not be done by a single organization, as the software to be developed may be large with many modules being developed by various organizations. Thus, integrated system testing needs people from all these organizations. Identify the participating organizations and fix the date and time with them.
- **Extent and constraints: -** Since the system testing of a large software system needs many resources, some might not be available. Therefore, indicate the extent of testing, e.g. total or partial. Include the rationale for the partial one. Similarly, indicate anticipated limitations on the test due to test conditions, such as interfaces, equipment, personnel, and database.
- **Schedule estimation: -** The time schedule for accomplishing testing milestones should also be planned. But this plan should be in tune with the time allocation in the project plan for testing.

Acceptance Test Plan: -

It is important to have the acceptance criteria defined so that acceptance testng is performed against those criteria. It means that in the acceptance test plan, we must have all the acceptance criteria defined in one document. If these are not available, then prepare them and plan the acceptance test accordingly. Acceptance criteria are broadly defined for functionality requirements, performance requirements, interface quality requirements, and overall software quality requirements.

Another point in acceptance testing plan is to decide the criticality of acceptance features defined. It is necessary to define the criticality. If the system fails the high critical acceptance requirement, then it should not pass the acceptance testing.

DETAILED TEST DESIGN AND TEST SPECIFICATIONS: -

The ultimate goal of test management is to get the test cases executed. Till now, test planning has not provided the test cases to be executed. Detailed test designing for each validation activity maps the requirements or features to the actual test cases to be executed. One way to map the features to their test cases is to analyse the following:

- Requirement traceability
- Design traceability
- Code traceability

The analyses can be maintained in the form of a traceability matrix (see Table 9.1) such that every requirement or feature is mapped to a function in the functional design. This function is then mapped to a module (internal design and code) in which the function is being implemented. This in turn is linked to the test case to be executed. This matrix helps in ensuring that all requirements have been covered and tested. Priority can also be set in this table to prioritize the test cases.

Table 9.1 Traceability matrix

Requirement/Feature	Functional Design	Internal Design/Code	Test cases
R1	F1, F4,F5	abc.cpp, abc.h	T5, T8,T12,T14

1. **TEST DESIGN SPECIFICATION: -**
   A test design specification should have the following components according to IEEE recommendation:
   - **Identifier: -** A unique identifier is assigned to each test design specification with a reference to its associated test plan.
   - **Features to be tested: -** The features or requirements to be tested are listed with reference to the items mentioned in SRS/SDD.
   - **Approach refinements: -** In the test plan, an approach to overall testing was discussed. Here, further details are added to the approach. For example, special testing techniques to be used for generating test cases are explained.
   - **Test case identification: -** The test cases to be executed for a particular feature/ function are identified here, as shown in Table 9.1. Moreover, test procedures are also identified. The test cases contain input/output information and the test procedures contain the necessary steps to execute the tests. Each test design specification is associated with test cases and test procedures. A test case may be associated with more than one test design specifications.
   - **Feature pass/fail criteria: -** The criteria for determining whether the test for a feature has passed or failed, is described.

2. **TEST CASE SPECIFICATIONS: -**
   Since the test design specifications have recognized the test cases to be executed, there is a need to define the test cases with complete specifications. The test case specification document provides the actual values for input with expected outputs. One test case can be used for many design specifications and may be re-used in other situations. A test case specification should have the following components according to IEEE recommendation:
   - **Test case specification identifier: -** A unique identifier is assigned to each test case specification with a reference to its associated test plan.
   - **Purpose: -** The purpose of designing and executing the test case should be mentioned here. It refers to the functionality you want to check with this test case.
   - **Test items needed: -** List the references to related documents that describe the items and features, e.g. SRS, SDD, and user manual.
   - **Special environmental needs: -** In this component, any special requirement in the form of hardware or software is recognized. Any requirement of tool may also be specified.

- **Special procedural requirements: -** Describe any special condition or constraint to run the test case, if any.
- **Inter-case dependencies: -** There may be a situation that some test cases are dependent on each other. Therefore, previous test cases which should be run prior to the current test case must be specified.
- **Input specifications: -** This component specifies the actual inputs to be given while executing a test case. The important thing while specifying the input values is not to generalize the values, rather specific values should be provided. For example, if the input is in angle, then the angle should not be specified as a range between 0 and 360, but a specific value like 233 should be specified. If there is any relationship between two or more input values, it should also be specified.
- **Test procedure: -** The step-wise procedure for executing the test case is described here.
- **Output specifications: -** Whether a test case is successful or not is decided after comparing the output specifications with the actual outputs achieved. Therefore, the output should be mentioned complete in all respects. As in the case of input specifications, output specifications should also be provided in specific values.

Example 9.1

There is a system for railway reservation system. There are many functionalities in the system, as given below:

S. No.	Functionality	Function ID in SRS	Test cases
1	Login the system	F3.4	T1
2	View reservation status	F3.5	T2
3	View train schedule	F3.6	T3
4	Reserve seat	F3.7	T4
5	Cancel seat	F3.8	T5
6	Exit the system	F3.9	T6

Suppose we want to check the functionality corresponding to 'view reservation status'. Its test specification is given in Fig. 9.4.

- **Test case Specification Identifier**
  T2
- **Purpose**
  To check the functionality of 'View Reservation Status'
- **Test Items Needed**
  Refer function F3.5 in SRS of the system.
- **Special Environmental Requirements**
  Internet should be in working condition. Database software through which the data will be retrieved should be in running condition.
- **Special Procedural Requirements**
  The function 'Login' must be executed prior to the current test case.
- **Inter-case Dependencies**
  T1 test case must be executed prior to the current test case execution.
- **Input Specifications**
  Enter PNR number in 10 digits between 0–9 as given below:
  4102645876
  21A2345672
  234
  asdgggggggg
- **Test Procedure**
  Press 'View Reservation status' button.
  Enter PNR number and press ENTER.
- **Output Specifications**
  The reservation status against the entered PNR number is displayed as S12 or RAC13 or WL123 as applicable.

Figure 9.4 Test specifications

3. TEST PROCEDURE SPECIFICATIONS: -

A test procedure is a sequence of steps required to carry out a test case or a specific task. This can be a separate document or merged with a test case specification.

4. TEST RESULT SPECIFICATIONS: -

There is a need to record and report the testing events during or after the test execution, as shown in Fig. 9.5.

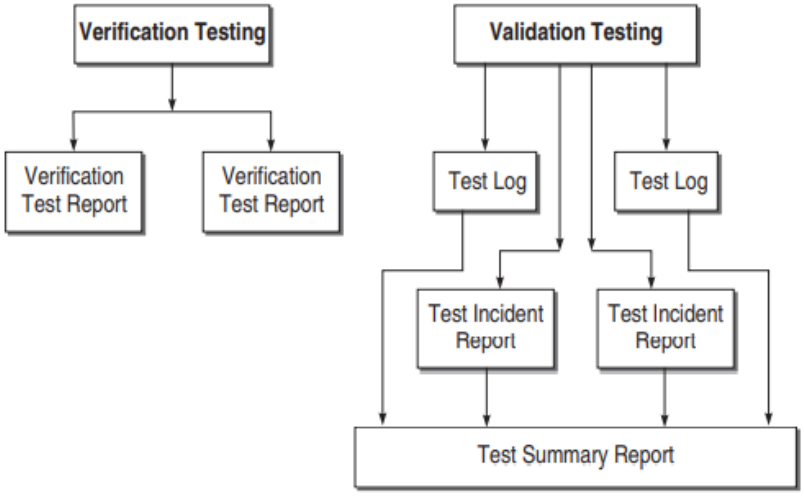

Figure 9.5 Test result specifications

- **Test Log: -**

  Test log is a record of the testing events that take place during the test. Test log is helpful for bug repair or regression testing. The developer gets valuable clues from this test log, as it provides snapshots of failures. The format for preparing a test log according to IEEE is given below:

  - **Test log identifier**
  - **Description: -** Mention the items to be tested, their version number, and the environment in which testing is being performed.
  - **Activity and event entries: -** Mention the following:
    - **i.** Date
    - **ii.** Author of the test
    - **iii.** Test case ID
    - **iv.** Name the personnel involved in testing
    - **v.** For each execution, record the results and mention pass/fail status.
    - **vi.** Report any anomalous unexpected event, before or after the execution

---

Example 9.1

There is a system for railway reservation system. There are many functionalities in the system, as given below:

S. No.	Functionality	Function ID in SRS	Test cases
1	Login the system	F3.4	T1
2	View reservation status	F3.5	T2
3	View train schedule	F3.6	T3
4	Reserve seat	F3.7	T4
5	Cancel seat	F3.8	T5
6	Exit the system	F3.9	T6

Suppose we want to check the functionality corresponding to 'view reservation status'. Its test specification is given in Fig. 9.4.

The test log corresponding to Example 9.1 is given in Fig. 9.6.

- **Test Log Identifier**
  TL2
- **Description**
  Function 3.5 in SRS v 2.1. The function tested in Online environment with Internet.
- **Activity and Event Entries**
  Mention the following:
  (i) Date: 22/03/2009
  (ii) Author of test: XYZ
  (iii) Test case ID: T2
  (iv) Name of the personnel involved in testing: ABC, XYZ
  (v) For each execution, record the results and mention pass/fail status
  The function was tested with the following inputs:

Inputs	Results	Status
4102645876	S12	Pass
21A2345672	S14	Fail
234	Enter correct 10 digit PNR number	Pass
asdgggggggg	RAC12	Fail

  (vi) Report any anomalous unexpected event before or after the execution.
  Nil

Figure 9.6 Sample test log for Example 9.1

- **Test Incident Report: -**
  This is a form of bug report. It is the report about any unexpected event during testing which needs further investigation to resolve the bug. Therefore, this report completely describes the execution of the event. It not only reports the problem that has occurred but also compares the expected output with the actual results. Listed below are the components of a test incident report:
  - **Test incident report identifier.**
  - **Summary: -** Identify the test items involved, test cases/procedures, and the test log associated with this test.
  - **Incident description: -** It describes the following:
    i. Date and time
    ii. Testing personnel names
    iii. Environment
    iv. Testing inputs
    v. Expected outputs
    vi. Actual outputs
    vii. Anomalies detected during the test
    viii. Attempts to repeat the same test
  - **Impact: -** The originator of this report will assign a severity value/rank to this incident so that the developer may know the impact of this problem and debug the critical problem first.

Example 9.1

There is a system for railway reservation system. There are many functionalities in the system, as given below:

S. No.	Functionality	Function ID in SRS	Test cases
1	Login the system	F3.4	T1
2	View reservation status	F3.5	T2
3	View train schedule	F3.6	T3
4	Reserve seat	F3.7	T4
5	Cancel seat	F3.8	T5
6	Exit the system	F3.9	T6

Suppose we want to check the functionality corresponding to 'view reservation status'. Its test specification is given in Fig. 9.4.

The test log corresponding to Example 9.1 is given in Fig. 9.7.

- **Test Incident Report Identifier**
  TI2
- **Summary**
  Function 3.5 in SRS v 2.1. Test Case T2 and Test Log TL2.
- **Incident Description**
  It describes the following:
  (i) Date and time: 23/03/2009, 2.00pm
  (ii) Testing personnel names: ABC, XYZ
  (iii) Environment: Online environment with X database
  (iv) Testing inputs
  (v) Expected outputs
  (vi) Actual outputs
  (vii) Anomalies detected during the test
  (viii) Attempts to repeat the same test

Testing Inputs	Expected outputs	Actual outputs	Anomalies detected	Attempts to repeat the same test
4102645876	S12	S12	nil	–
21A2345672	Enter correct 10 digit PNR number	S14	Alphabet is being accepted in the input.	3
234	Enter correct 10 digit PNR number	Enter correct 10 digit PNR number	nil	–
asdgggggggg	Enter correct 10 digit PNR number	RAC12	Alphabet is being accepted in the input.	5

- **Impact**
  The severity value is 1(Highest).

Figure 9.7 Sample test incident report for Example 9.1

- **Test Summary Report: -**
  It is basically an evaluation report prepared when the testing is over. It is the summary of all the tests executed for a specific test design specification. It can provide the measurement of how much testing efforts have been applied for the test. It also becomes a historical database for future projects, as it provides information about the particular type of bugs observed.

Example 9.1

There is a system for railway reservation system. There are many functionalities in the system, as given below:

S. No.	Functionality	Function ID in SRS	Test cases
1	Login the system	F3.4	T1
2	View reservation status	F3.5	T2
3	View train schedule	F3.6	T3
4	Reserve seat	F3.7	T4
5	Cancel seat	F3.8	T5
6	Exit the system	F3.9	T6

Suppose we want to check the functionality corresponding to 'view reservation status'. Its test specification is given in Fig. 9.4.

The test summary report corresponding to Example 9.1 is given in Fig. 9.8.

- **Test Summary Report Identifier**
  TS2
- **Description**
  SRS v 2.1

S. No.	Functionality	Function ID in SRS	Test cases
1	Login the system	F3.4	T1
2	View reservation status	F3.5	T2
3	View train schedule	F3.6	T3
4	Reserve seat	F3.7	T4
5	Cancel seat	F3.8	T5
6	Exit the system	F3.9	T6

- **Variances**
  Nil
- **Comprehensive Statement**
  All the test cases were tested except F3.7, F3.8, F3.9 according to the test plan.
- **Summary of Results**

S. No.	Functionality	Function ID in SRS	Test cases	Status
1	Login the system	F3.4	T1	Pass
2	View reservation status	F3.5	T2	Bug Found. Could not be resolved.
3	View train schedule	F3.6	T3	Pass

- **Evaluation**
  The functions F3.4, F3.6 have been tested successfully. Function F3.5 couldn't be tested as the bug has been found. The bug is that the PNR number entry has also accepted alphabetical entries as wrong

Figure 9.8 Test summary report

A test summary report contains the following components:

- **Test summary report identifier**
- **Description: -** Identify the test items being reported in this report with the test case/procedure ID.
- **Variances: -** Mention any deviation from the test plans, test procedures, if any
- **Comprehensive statement: -** The originator of this report compares the comprehensiveness of the testing efforts made with the test plans. It describes what has been tested according to the plan and what has not been covered.

- **Summary of results: -** All the results are mentioned here with the resolved incidents and their solutions. Unresolved incidents are also reported.
- **Evaluation: -** Mention the status of the test results. If the status is fail, then mention its impact and severity level.
- **Summary of activities: -** All testing execution activities and events are mentioned with resource consumption, actual task durations, etc.
- **Approvals: -** List the names of the persons who approve this document with their signatures and dates.

Software Metrics

Today, every technical process demands measurement. We cannot describe a product by simply saying that it should be of high quality or that it should be reliable. Today, the demand is to quantify the terms 'good or high quality', 'more reliable', 'efficient', etc. It is a world of quantification. This becomes more important in terms of software, as its development is becoming more and more complex.

With the increase in complexity, it becomes important to control the software development process to monitor its progress on factors like time, budget, and resource constraints, and measure the quality of the end-product. Thus, if we want to control all these parameters in software, we need software metrics.

DE Marco rightly said, you cannot control what you cannot measure. Quantification has become a necessity for the effective management of the software process.

Software metrics are used by the software industry to quantify the development, operation, and maintenance of software. Metrics give us information regarding the status of an attribute of the software and help us to evaluate it in an objective way. The practice of applying software metrics to a software process and to a software product is a complex task. By evaluating an attribute of the software, we can know its status. From there, we can identify and classify what its situation is; which helps us to find opportunities of improvements in the software. This also helps us to make plans for modifications that need to be implemented in the future.

Software metrics play an important role in measuring the attributes that are critical for the success of a software project. Measurement of these attributes helps to make the characteristics and relationships between the attributes clearer. This in turn supports informed decision-making. Measuring the attributes of a development process enables the management to have a better insight into the process. Thus, measurement is a tool through which the management identifies important events and trends, enabling them to make informed decisions. Moreover, measurements help in predicting outcomes and evaluating risks, which in turn decreases the probability of unanticipated surprises in different processes.

NEED OF SOFTWARE MEASUREMENT: -

Measurements are a key element for controlling software engineering processes. By controlling, it is meant that one can assess the status of the process, observe the trends to predict what is likely to happen, and take corrective action for modifying our practices. Measurements also play their part in increasing our understanding of the process by making visible relationships among process activities and entities involved. Lastly, measurements improve our processes by modifying the activities based on different measures.

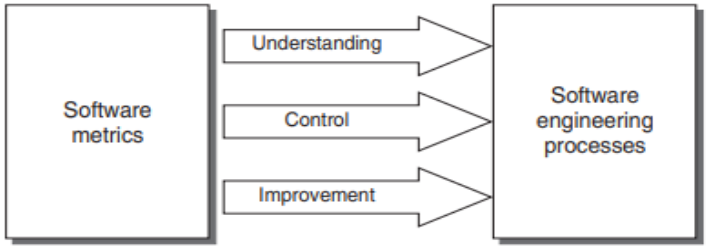

Figure 10.1 Need for software metrics

On the basis of this discussion, software measurement is needed for the following activities (see Fig. 10.1):

1. **Understanding: -** Metrics help in making the aspects of a process more visible, thereby giving a better understanding of the relationships among the activities and entities they affect.

2. **Control: -** Using baselines, goals, and an understanding of the relationships, we can predict what is likely to happen and correspondingly, make appropriate changes in the process to help meet the goals.

3. **Improvement: -** By taking corrective actions and making appropriate changes, we can improve a product. Similarly, based on the analysis of a project, a process can also be improved.

DEFINITION OF SOFTWARE METRICS: -

Software metrics can be defined as 'the continuous application of measurement-based techniques to the software development process and its products to supply meaningful and timely management information, together with the use of those techniques to improve that process and its products.'

CLASSIFICATION OF SOFTWARE METRICS: -

1. **PRODUCT VS. PROCESS METRICS: -**

   Software metrics may be broadly classified as either product metrics or process metrics. Product metrics are measures of the software product at any stage of its development, from requirements to installed system. Product metrics may measure the complexity of the software design, the size of the final program, or the number of pages of documentation produced.

   Process metrics, on the other hand, are measures of the software development process, such as the overall development time, type of methodology used, or the average level of experience of the programming staff.

2. **OBJECTIVE VS. SUBJECTIVE METRICS: -**

   Objective measures should always result in identical values for a given metric, as measured by two or more qualified observers. For subjective measures, even qualified observers may measure different values for a given metric. For example, for product metrics, the size of product measured in line of code (LOC) is an objective measure. In process metrics, the development time is an example of objective measure, while the level of a programmer's experience is likely to be a subjective measure

3. **PRIMITIVE VS. COMPUTED METRICS: -**

   Primitive metrics are those metrics that can be directly observed, such as the program size in LOC, the number of defects observed in unit testing, or the total development time for the project. Computed metrics are those that cannot be directly observed but are computed in some way from other metrics. For example, productivity metrics like LOC produced per person-month or product quality like defects per thousand lines of code.

4. **PRIVATE VS. PUBLIC METRICS: -**

   This classification is based on the use of different types to process data. Because it is natural that individual software engineers might be sensitive to the use of metrics collected on an individual basis, these data should be private to individuals and serve as an indicator for individuals only. Examples of private metrics include defect rates (by individual and by module) and errors found during development.

   Public metrics assimilate information that originally was private to individuals and teams. Project-level defect rates, effort, calendar times, and related data are collected and evaluated in an attempt to uncover indicators that can improve organizational process performance.

ENTITIES TO BE MEASURED: -

In order to measure, it is needed to identify an entity and a specific attribute of it. It is very important to define clearly what is being measured, otherwise, the measures cannot be performed or the measures obtained can have different meaning for different people.
The entities considered in software measurement are: „

- **Processes: -** Any activity related to software development. „
- **Product: -** Any artifact produced during software development. „
- **Resource: -** People, hardware, or software needed for a process.

The attributes of an entity can be internal or external.

- **Internal attributes** of any entity can be measured only based on the entity and therefore, measured directly. For example, size is an internal attribute of any software measurement.
- **External attributes** of any entity can be measured only with respect to how the entity is related with the environment and therefore, can only be measured indirectly. For example, reliability, an external attribute of a program, does not depend only on the program itself but also on the compiler, machine, and user. Productivity, an external attribute of a person, clearly depends on many factors such as the kind of process and the quality of the software delivered.

SIZE METRICS: -

The software size is an important metric to be used for various purposes. At the same time, it is difficult to measure because, unlike other physical products, software cannot be measured directly with conventional units. Various approaches used for its measurement are given below.

1. **LINE OF CODE (LOC): -**
   This metric is based on the number of lines of code present in the program. The lines of code are counted to measure the size of a program. The comments and blank lines are ignored during this measurement. The LOC metric is often presented on thousands of lines of code (KLOC). It is often used during the testing and maintenance phases, not only to specify the size of the software product, but also it is used in conjunction with other metrics to analyse other aspects of its quality and cost.
2. **TOKEN COUNT (HALSTEAD PRODUCT METRICS): -**
   The problem with LOC is that it is not consistent, because all lines of code are not at the same level. Some lines are more difficult to code than others. Another metric set has been given by Halstead. He stated that any software program could be measured by counting the number of operators and operands. From these set of operators and operands, he defined a number of formulae to calculate the vocabulary, the length, and the volume of the software program. Halstead extended this analysis to determine the effort and time. Some Halstead metrics are given below.

   Program Vocabulary: -
   It is the number of unique operators plus the number of unique operands as given below:
   $n = n1 + n2$
   Where n = program vocabulary
   n1 = number of unique operators
   n2 = number of unique operands

   Program Length: -
   It is the total usage of all the operators and operands appearing in the implementation. It is given as,
   $N = N1 + N2$
   Where N = program length
   N1 = all operators appearing in the implementation
   N2 = all operands appearing in the implementation

Program Volume: -

The volume refers to the size of the program and it is defined as the program length times the logarithmic base 2 of the program vocabulary. It is given as,

$V = N \log_2 n$

Where V = program volume

N = program length

n = program vocabulary

3. FUNCTION POINT ANALYSIS (FPA): -

It is based on the idea that the software size should be measured according to the functionalities specified by the user. Therefore, FPA is a standardized methodology for measuring various functions of software from the user's point of view.

The size of an application is measured in function points. The process of counting the functions using FPA has been standardized by the International Function Point Users Group (IFPUG). IFPUG has defined the rules and standards for calculating function points and it also promotes their use and evolution.

Process to Calculate Function Points: -

The process used to calculate the function points is given below:

a) Determine the type of project for which the function point count is to be calculated. For example, development project (a new project) or enhancement project.

b) Identify the counting scope and the application boundary.

c) Identify data functions (internal logical functions and external interface fi les) and their complexity.

d) Identify transactional functions (external inputs, external outputs, and external queries) and their complexity.

e) Determine the unadjusted function point count (UFP).

f) Determine the value adjustment factor, which is based on 14 general system characteristics (GSCs).

g) Calculate the adjusted function point count (AFP).

Function point counting has been depicted in Fig. 10.2. This figure shows all the data functions and transactional functions. The details of all these functions are described in the following sections.

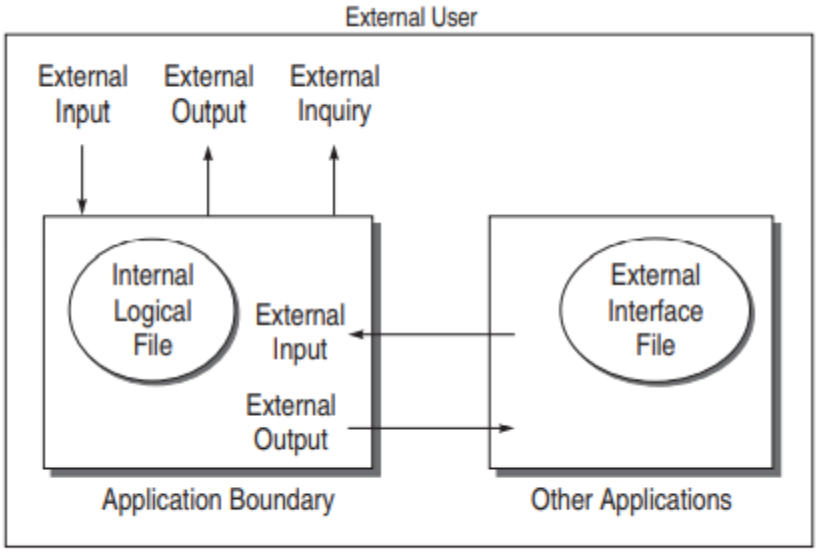

Figure 10.2 Functionality recognized in function point counting

Sizing Data Functions: -

Data functions are those functions in the project which relate to the logical data stored and available for update, reference, and retrieval. These functions are categorized in two parts:

internal logical functions and external interface files. The categories are logical groupings of logically related data, and not physical representations. These are discussed below.

- **Internal logical files (ILF): -** An internal logical fi le is a user-identifiable group of logically related data or control information maintained within the boundary of the application.
- **External interface files (EIF): -** An external interface fi le is a user-identifiable group of logically related data or control information referenced by the application but maintained within the boundary of a different application. An EIF counted for an application must be in another application.

The physical count of ILFs and EIFs, together with the relative functional complexity of each, determines the contribution of the data function types to the unadjusted function point. Each identified data function must be assigned a functional complexity based on the number of data element types (DETs) and record element types (RETs) associated with the ILF and EIF.
DETs are unique user-recognizable, non-repeatable fields or attributes. RETs are user-recognizable subgroups (optional or mandatory) of data elements contained within an ILF or EIF. Subgroups are typically represented in an entity relationship diagram as entity subtypes or attribute entities.
Functional complexity is the rating assigned to each data function. The rating is based on record types and DETs being counted, as shown in Table 10.1.

Table 10.1 Complexity matrix for ILFs and EIFs

RETs	DETs		
	1–19	20–50	>=51
1	Low	Low	Average
2–5	Low	Average	High
>5	Average	High	High

Sizing Transactional Functions: -
Information systems are usually developed to mechanize and automate manual tasks. The tasks that have been automated are identified as transactional functions which represent the functionality provided to the user for processing the data by an application. The categorization of these functions is given below.
External inputs (EIs) are incoming data or control information to alter the behavior of a system, e.g. adding, changing, deleting, etc.
External inquiries (EQs) send data outside the application through retrievals of data or control information from ILFs and/or EIFs, e.g. retrieval and display of a current description from our file.
External outputs (EOs) send data outside the application with processing logic other than or in addition to retrieval of data or control information, e.g. report of monthly sales with calculation by category.

Complexity and Contribution: -
The physical count of EIs, EOs, and EQs together with the relative functional complexity for each, determines the contribution of external inputs, outputs, and queries to the unadjusted function point count. Each identified EI, EO, and EQ must be assigned a functional complexity based on the number of DETs and FTRs (file type referenced) associated with them. Complexity matrices for EIs, EOs, and EQs are shown in Tables 10.2, 10.3, and 10.4.

Table 10.2 Complexity matrix for EIs

FTRs	DETs		
	1–4	5–15	>=16
<2	Low	Low	Average
2	Low	Average	High
>2	Average	High	High

Table 10.3 Complexity matrix for EOs

FTRs	DETs		
	1–5	6–19	>=20
<2	Low	Low	Average
2–3	Low	Average	High
>3	Average	High	High

Table 10.4 Complexity matrix for EQs

FTRs	DETs		
	1–5	6–19	>=20
<2	Low	Low	Average
2–3	Low	Average	High
>3	Average	High	High

Calculating Unadjusted Function Point (UFP): -

The steps for calculating UFP are given below:

- Count all DETs/FTRs for all five components, i.e. ILF, EIF, EI, EO, and EQ of an application and determine the level of the component as low, average, or high, based on the individual complexity matrix, as described above.
- Count the number of all five components. The sum of each count is multiplied by an appropriate weight using Table 10.5, as shown below.

Table 10.5 IFPUGs unadjusted function point table

Components	Function Levels		
	Low	Average	High
ILF	X7	X10	X15
EIF	X5	X7	X10
EI	X3	X4	X6
EO	X4	X5	X7
EQ	X3	X4	X6

- Add all the five results calculated in the previous step. This is the final UFP.

Calculating Adjusted Function Point: -

There are some characteristics regarding every project which must be considered for the effort being measured for the project. Therefore, the function point calculated above is unadjusted, as the project dependent characteristics have not been considered in the function count. Therefore, IFPUG defines a value adjustment factor (VAF) which is used as a multiplier of the unadjusted function point count in order to calculate the adjusted function point (AFP) count of an application.

Each GSC is evaluated in terms of its degree of influence (DI) on a scale of 0 to 5, as given below in Table 10.6.

Table 10.6 Degree of influence scaling

DI	Meaning
0	Not present, or no influence
1	Incidental influence
2	Moderate influence
3	Average influence
4	Significant influence throughout

Table 10.7 shows the GSCs to calculate the VAF.

Table 10.7 Components of VAF

Factor	Meaning
F1	Data communications
F2	Performance
F3	Transaction rate
F4	End user efficiency
F5	Complex processing
F6	Installation ease
F7	Multiple sites
F8	Distributed data processing
F9	Heavily used configuration
F10	Online data entry
F11	Online update
F12	Reusability
F13	Operational ease
F14	Facilitate change

The 14 GSCs shown in the table are summarized in the VAF. After applying the VAF, the UFP adjusts by 35% to determine the adjusted function point. The following steps determine the AFP.

- Evaluate the 14 GSCs on a scale of 0–5 to determine the DI for each GSC description.
- Add the DIs for all 14 GSCs to produce the total degree of influence (TDI).
- Use the TDI in the following equation to compute VAF.

$$VAF = (TDI \times 0.01) + 0.065$$

- The final adjusted function point is calculated as,

$$AFP = UFP \times VAF$$

Example 10.1

Consider a project with the following parameters: EI = 50, EO = 40, EQ = 35, ILF = 06, and ELF = 04. Assume all weighing factors are average. In addition, the system requires critical performance, average end-user efficiency, moderate distributed data processing, and critical data communication. Other GSCs are incidental. Compute the function points using FPA.

Table 10.5 IFPUGs unadjusted function point table

Components	Function Levels		
	Low	Average	High
ILF	X7	X10	X15
EIF	X5	X7	X10
EI	X3	X4	X6
EO	X4	X5	X7
EQ	X3	X4	X6

$$EI = 50,\ EO = 40,\ EQ = 35,\ ILF = 06,\ \text{and}\ ELF = 04.$$

$$UFP = 50 \times 4 + 40 \times 5 + 35 \times 4 + 6 \times 10 + 4 \times 7 = 628$$

Table 10.6 Degree of influence scaling

DI	Meaning
0	Not present, or no influence
1	Incidental influence
2	Moderate influence
3	Average influence
4	Significant influence throughout

Table 10.7 Components of VAF

Factor	Meaning
F1	Data communications
F2	Performance
F3	Transaction rate
F4	End user efficiency
F5	Complex processing
F6	Installation ease
F7	Multiple sites
F8	Distributed data processing
F9	Heavily used configuration
F10	Online data entry
F11	Online update
F12	Reusability
F13	Operational ease
F14	Facilitate change

Assume all weighing factors are average. In addition, the system requires critical performance, average end-user efficiency, moderate distributed data processing, and critical data communication. Other GSCs are incidental.

$$TDI = (4 + 3 + 2 + 4) \times 10 = 130$$

Use the TDI in the following equation to compute VAF.

$$VAF = (TDI \times 0.01) + 0.065$$

The final adjusted function point is calculated as,

$$AFP = UFP \times VAF$$

$$VAF = 130 \times 0.01 + 0.065 = 1.365$$

$$AFP = 628 \times 1.365 = 857.22$$

Solution

$$UFP = 50 \times 4 + 40 \times 5 + 35 \times 4 + 6 \times 10 + 4 \times 7 = 628$$

$$TDI = (4 + 3 + 2 + 4) \times 10 = 130$$

$$VAF = 130 \times 0.01 + 0.065 = 1.365$$

$$AFP = 628 \times 1.365 = 857.22$$

REVIEW QUESTIONS: -

1. Discuss the key components of test management
2. What are the major activities of a testing group?
3. What is meant by testing group hierarchy? Explain the role of each member in this hierarchy.
4. Suppose you are working in the testing group of a company. Identify your role in the testing group hierarchy. What are the major duties performed by you there?
5. What type of test planning will you plan for a real-time critical software? Design a test plan using test plan components.
6. How can the users/clients help in preparing the test plans?
7. What are the major activities in V&V planning?
8. Acquire SRS and SDD of any project and develop the following:
    a. Unit test plan
    b. Integration test plan
    c. System test plan
9. Explain the importance of test harness in the integration test plan.
10. What is the difference between system test plan and acceptance test plan?
11. Discuss the role of traceability matrix in designing the test cases.
12. Discuss the role of test log, test incident report, and test summary report in validation testing.
13. Take three modules of your choice in a project and prepare the following for each module:
    a. Test design specifications
    b. Test case specifications
    c. Test procedure specifications
    d. Test procedure specifications
14. List all the testing deliverables (documents) and describe the purpose of each, in the organization where you are working.
15. What is the need for software measurement?
16. Discuss the various types of software metrics.
17. What is the disadvantage of LOC metrics?
18. What is the basis of Halstead metrics to calculate the size of software?
19. What type of projects can be best counted with function point analysis?
20. Explain the process of calculating function points for a project which you are going to build.
21. Can you adopt the FPA for calculating the function points of real-time software?
22. What is the difference between UFP and AFP?
23. Consider a project with the following parameters: EI= 60, EO= 40, EQ = 45, ILF = 06, ELF = 08. Assume all weighing factors are average. In addition, the system requires significant data

communications, performance is very critical, designed code may be moderately reusable, and other GSCs are average. Compute the function points using FPA.

24. Consider a project with the following components: EI (simple) = 30, EO (average) = 20, EQ (average) = 35, ILF (complex) = 08, ELF (complex) = 05. In addition, the system requires significant end-user efficiency, moderate distributed data processing, critical data communications, and other GSCs are incidental. Compute the function points for this system using FPA.