BTECH DEGREE EXAMINATION , JAN 2023

Seventh Semester

Information Technology

Elective : SOFTWARE TESTING

(2013-14 Regulations)

PART-A

1. List the levels of TMM.

Level 1: Initial.
Level 2: Phase Definition.
Level 3: Integration.
Level 4: Management and Measurement.
Level 5: Optimization/Defect prevention and quality control.

2. Write short notes Test,Test Set, and Test Suite.

A test is a group of related test cases, or a group of related test cases and test procedures.

A test set T is said to be mutation adequate for program P provided that for every inequivalent mutant Pi of P there is an element t in T such that Pi(t) is not equal to P(t).

A test suite is an environment that contains all the hardware and software needed to test a software component or a software system.

3. List the various iterations of Loop testing.

Loop testing strategies focus on detecting common defects associated with these structures. For example, in a simple loop that can have a range of zero to n iterations, test cases should be developed so that there are:

- zero iterations of the loop, i.e., the loop is skipped in its entirely;
- one iteration of the loop;
- two iterations of the loop;
- k iterations of the loop where k _ n;
- n _ 1 iterations of the loop;
- n _ 1 iterations of the loop (if possible).

4. What are the factors affecting less than 100% degree of coverage?

The planned degree of coverage may be less than 100% possibly due to the following:

a. The nature of the unit

- Some statements/branches may not be reachable.

- The unit may be simple, and not mission, or safety, critical, and so complete coverage is thought to be unnecessary.

b.The lack of resources

- The time set aside for testing is not adequate to achieve 100% coverage.
- There are not enough trained testers to achieve complete coverage for all of the units.
- There is a lack of tools to support complete coverage.

c.Other project-related issues such as timing, scheduling, and marketing Constraints

5. Give example for security testing.

- ✓ Password checking
- ✓ Legal and illegal entry with password
- ✓ Password Expiration
- ✓ Encryption
- ✓ Browsing
- ✓ Trap doors
- ✓ Viruses.

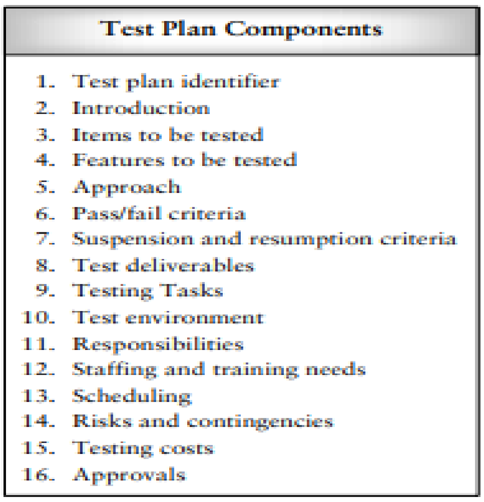
6. Write the necessity of alpha, beta and acceptance tests.

Alpha, beta, and acceptance tests allow users to evaluate the software in terms of their expectations and goals. When software is being developed for a specific client, acceptance tests are carried out after system testing. The acceptance tests must be planned carefully with input from the client/users.

7. When and where is milestones used?

Milestones are tangible events that are expected to occur at a certain time in the project's lifetime. Managers use them to determine project status. These milestones usually appear in the project plan as well as the test plan.

8. What are the components of test planner?



Test Plan Components
1. Test plan identifier
2. Introduction
3. Items to be tested
4. Features to be tested
5. Approach
6. Pass/fail criteria
7. Suspension and resumption criteria
8. Test deliverables
9. Testing Tasks
10. Test environment
11. Responsibilities
12. Staffing and training needs
13. Scheduling
14. Risks and contingencies
15. Testing costs
16. Approvals

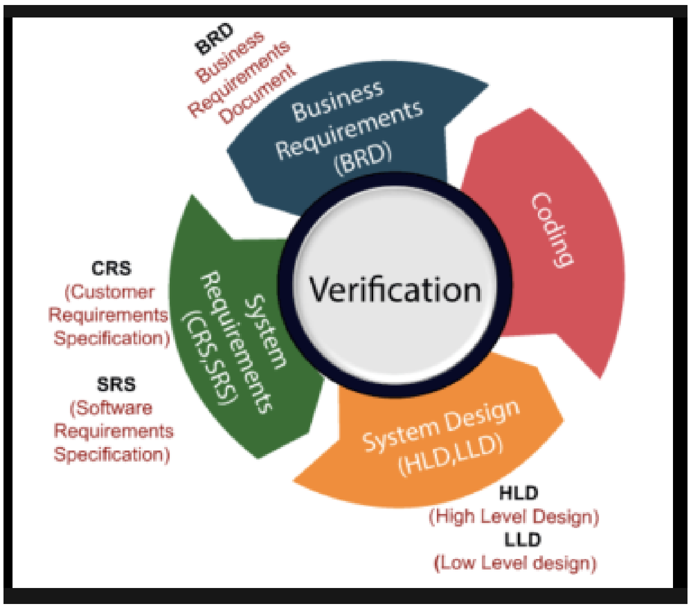
9. List the challenges in automation.

- ✓ Software Test automation makes use of specialized tools to control the execution of tests and compares the actual results against the expected result. Usually regression tests, which are repetitive actions, are automated.
- ✓ Testing Tools not only help us to perform regression tests but also helps us to automate data set up generation, product installation, GUI interaction, defect logging, etc.

10. How verification can be performed using test automation tools?

It is the static practice of studying and verifying the specific requirements of a particular stage in development.

There are 2 main types,functional and non-functional.


BRD
Business Requirements Document
Business Requirements (BRD)
Coding
Verification
CRS
(Customer Requirements Specification)
SRS
(Software Requirements Specification)
System Requirements (CRS,SRS)
System Design (HLD,LLD)
HLD
(High Level Design)
LLD
(Low Level design)

PART-B

11. What do you mean by design defects? what are the different design defects? Explain.

Design defects occur when system components, interactions between system components, interactions between the components and outside software/hardware, or users are incorrectly designed. This covers defects in the design of algorithms, control, logic, data elements, module interface descriptions, and external software/hardware/user interface descriptions.

When describing these defects we assume that the detailed design description for the software modules is at the pseudo code level with processing steps, data structures, input/output parameters, and major control structures defined. If module design is not described in such detail then many of the defects types described here may be moved into the coding defects class.

1 . Algorithmic and Processing Defects

These occur when the processing steps in the algorithm as described by the pseudo code are incorrect. For example, the pseudo code may contain a calculation that is incorrectly specified, or the processing steps in the algorithm written in the pseudo code language may not be in the correct order. In the latter case a step may be missing or a step may be duplicated.

Another example of a defect in this subclass is the omission of error condition checks such as division by zero. In the case of algorithm reuse, a designer may have selected an inappropriate algorithm for this problem (it may not work for all cases).

2 . Control, Logic, and Sequence Defects

Control defects occur when logic flow in the pseudo code is not correct. For example, branching to soon, branching to late, or use of an incorrect branching condition. Other examples in this subclass are unreachable pseudo code elements, improper nesting, improper procedure or function calls. Logic defects usually relate to incorrect use of logic operators, such as less than (_), greater than (_), etc. These may be used incorrectly in a Boolean expression controlling a branching instruction.

3 . Data Defects

These are associated with incorrect design of data structures. For example, a record may be lacking a field, an incorrect type is assigned to a variable or a field in a record, an array may not have the proper number of elements assigned, or storage space may be allocated incorrectly. Software reviews and use of a data dictionary work well to reveal these types of defects.

4 . Module Interface Description Defects

These are defects derived from, for example, using incorrect, and/or inconsistent parameter types, an incorrect number of parameters, or an incorrect ordering of parameters.

5 . Functional Description Defects

The defects in this category include incorrect, missing, and/or unclear design elements. For example, the design may not properly describe the correct functionality of a module. These defects are best detected during a design review.

6 . External Interface Description Defects

These are derived from incorrect design descriptions for interfaces with COTS components, external software systems, databases, and hardware devices (e.g., I/O devices). Other examples are user interface description defects where there are missing or improper commands, improper sequences of commands, lack of proper messages, and/or lack of feedback messages for the user.

12.(a) Describe about the components of software development process

Process, in the software engineering domain, is the set of methods, practices, standards, documents, activities, policies, and procedures that software engineers use to develop and maintain a software system and its associated artifacts, such as project and test plans, design documents, code, and manuals as shown in Figure 1.

It also was clear that adding individual practices to an existing software development process in an ad hoc way was not satisfactory. The software development process, like most engineering artifacts, must be engineered. That is, it must be designed, implemented, evaluated, and maintained. As in other engineering disciplines, a software development process must evolve in a consistent and predictable manner, and the best technical and managerial practices must be integrated in a systematic way. Models such as the Capability Maturity Model (CMM)* and SPICE were developed to address process issues

These models allow an organization to evaluate its current software process and to capture an understanding of its state. Strong support for incremental process improvement is provided by the models, consistent with historical process evolution and the application of quality principles. The models have received much attention from industry, and resources have been invested in process improvement efforts with many successes recorded

All the software process improvement models that have had wide acceptance in industry are high-level models, in the sense that they focus on the software process as a whole and do not offer adequate support to evaluate and improve specific software development sub processes such as design and testing. Most software engineers would agree that testing is a vital component of a quality software process, and is one of the most challenging and costly activities carried out during software development and maintenance. In spite of its vital role in the production of quality software, existing process evaluation and improvement models such as the CMM, Bootstrap, and ISO-9000 have not adequately addressed testing process issues.

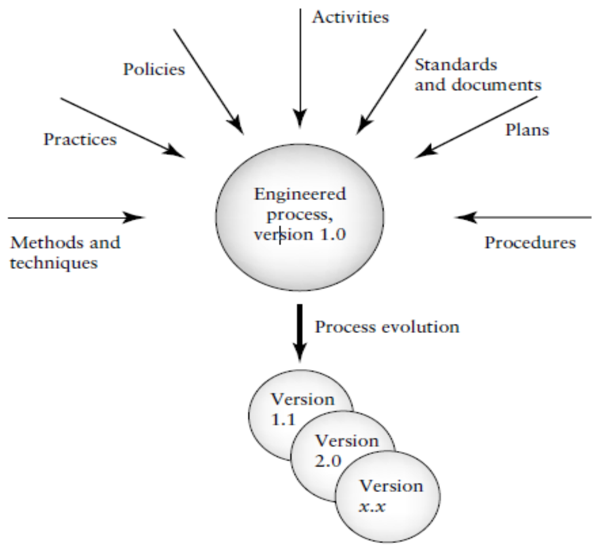


Figure 1. Components of an engineered process

The software development process has been described as a series of phases, procedures, and steps that result in the production of a software product. Embedded within the software development process are several other processes including testing. Some of these are shown in Figure2.Testing itself is related to two other processes called verification and validation as shown in Figure 2.

Validation is the process of evaluating a software system or component during, or at the end of, the development cycle in order to determine whether it satisfies specified requirements

Validation is usually associated with traditional execution-based testing, that is, exercising the code with test cases.

Verification is the process of evaluating a software system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase

Verification is usually associated with activities such as inspections and reviews of software deliverables. Testing itself has been defined in several ways. Two definitions are shown below.

Testing is generally described as a group of procedures carried out to evaluate some aspect of a piece of software.

Testing can be described as a process used for revealing defects in software, and for establishing that the software has attained a specified degree of quality with respect to selected attributes

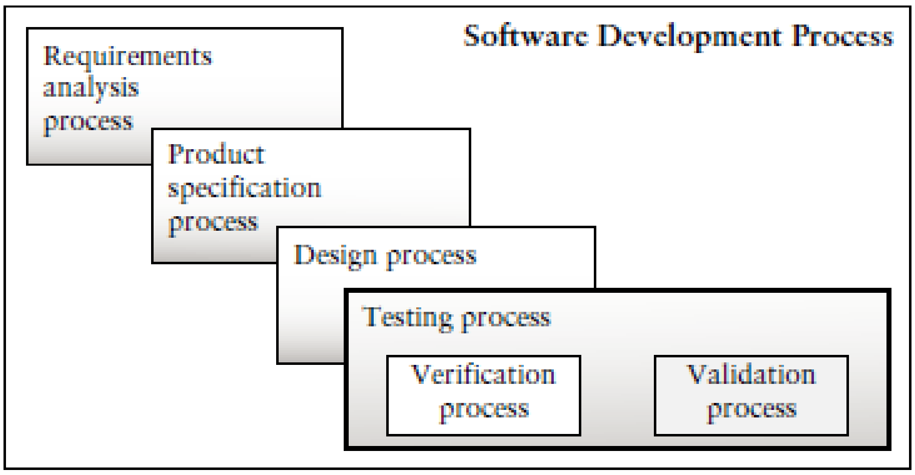

Software Development Process
Requirements analysis process
Product specification process
Design process
Testing process
Verification process
Validation process

Figure 2 Components of Software Development Process

(b) Describe about Tester Support for developing a Defect Repository.

As software engineers and test specialists we should follow the examples of engineers in other disciplines who have realized the usefulness of defect data. A requirement for repository development should be a part of testing and/or debugging policy statements. You begin with development of a defect classification scheme and then initiate the collection defect data from organizational projects. Forms and templates will need to be designed to collect the data. It need to be cautious about recording each defect after testing, and also recording the frequency of occurrence for each of the defect types. Defect monitoring should continue for each on-going project.

The distribution of defects will change as you make changes in your processes. The defect data is useful for test planning, a TMM level 2 maturity goal. It helps you to select applicable testing techniques, design (and reuse) the test cases you need, and allocate the amount of resources you will need to devote to detecting and removing these defects. This in turn will allow you to estimate testing schedules and costs.

The defect data can support debugging activities as well. In fact, as Figure 3 shows, a defect repository can help to support achievement and continuous implementation of several TMM maturity goals including controlling and monitoring of test, software quality evaluation and control, test measurement, and test process improvement.

1. **Functional Description Defects**

The overall description of what the product does, and how it should behave (inputs/outputs), is incorrect, ambiguous, and/or incomplete.

2. **Feature Defects**

Features may be described as distinguishing characteristics of a software component or system.

Features refer to functional aspects of the software that map to functional requirements as described by the users and clients. Features also map to quality requirements such as performance and reliability. Feature defects are due to feature descriptions that are missing, incorrect, incomplete, or superfluous.

3. **Feature Interaction Defects**

These are due to an incorrect description of how the features should interact. For example, suppose one feature of a software system supports adding a new customer to a customer database. This feature interacts with another feature that categorizes the new customer. The classification feature impacts on where the storage algorithm places the new customer in the database, and also affects another feature that periodically supports sending advertising information to customers in a specific category. When testing we certainly want to focus on the interactions between these features.

4. **Interface Description Defects**

These are defects that occur in the description of how the target software is to interface with external software, hardware, and users.

For detecting many functional description defects, black box testing techniques, which are based on functional specifications of the software, offer the best approach.

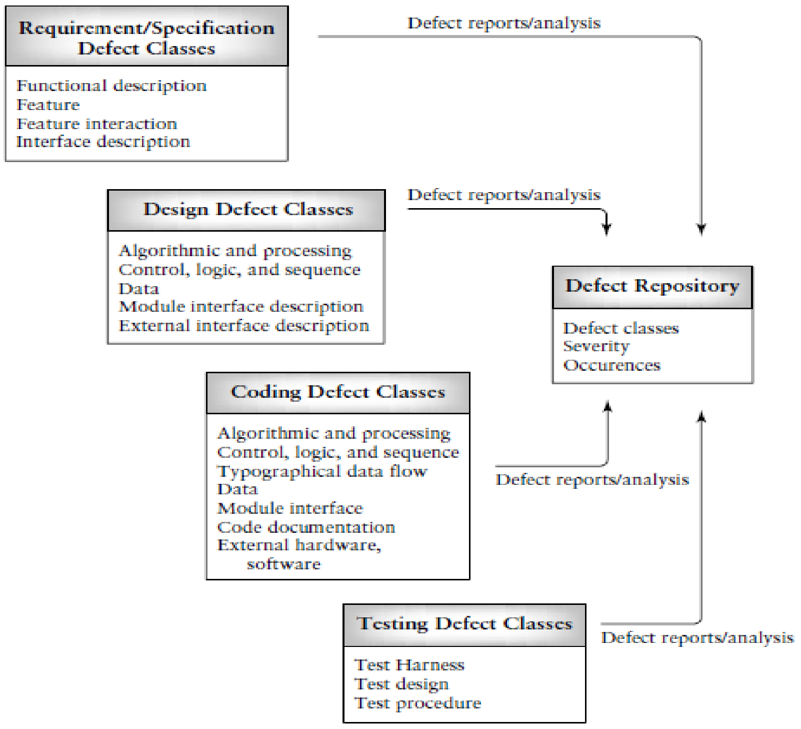


Figure 3. Defects and Defect Repository

13.Demonstrate the various black box test cases using equivalence class Partitioning and boundary values analysis to test a module for ATM system.

Equivalence class partitioning gives the tester a useful tool with which to develop black box based-test cases for the software-under-test. The method requires that a tester has access to a specification of input/output behavior for the target software. The test cases developed based on equivalence class partitioning can be strengthened by use of another technique called boundary value analysis. With experience, testers soon realize that many defects occur directly on, and above and below, the edges of equivalence classes. Test cases that consider these boundaries on both the input and output spaces as shown in Figure 4 are often valuable in revealing defects. Whereas equivalence class partitioning directs the tester to select test cases from any element of an equivalence class, boundary value analysis requires that the tester select elements close to the edges, so that both the upper and lower edges of an equivalence class are covered by test cases. As in the case of equivalence class partitioning, the ability to develop high quality test cases with the use of boundary values requires experience. The rules-of-thumb described below are useful for getting started with boundary value analysis.

1. If an input condition for the software-under-test is specified as a range of values, develop valid test cases for the ends of the range, and invalid test cases for possibilities just above and below the ends of the range.

For example, if a specification states that an input value for a module must lie in the range between _1.0 and _1.0, valid tests that include values for ends of the range, as well as invalid test cases for values just above and below the ends, should be included. This would result in input values of _1.0, _1.1, and 1.0, 1.1.

2.If an input condition for the software-under-test is specified as a number of values, develop valid test cases for the minimum and maximum numbers as well as invalid test cases that include one lesser and one greater than the maximum and minimum.

For example, for the real-estate module mentioned previously that specified a house can have one to four owners, tests that include 0,1 owners and 4,5 owners would be developed.

The following is an example of applying boundary value analysis to output equivalence classes. Suppose a table of 1 to 100 values is to be produced by a module. The tester should select input data to generate an output table of size 0,1, and 100 values, and if possible 101 values.

3. If the input or output of the software-under-test is an ordered set, such as a table or a linear list, develop tests that focus on the first and last elements of the set. It is important for the tester to keep in mind that equivalence class partitioning and boundary value analysis apply to testing both inputs and outputs of the software-under-test, and, most importantly, conditions are not combined for equivalence class partitioning or boundary value analysis. Each condition is considered separately, and test cases are developed to insure coverage of all the individual conditions. An example follows. Suppose we are testing a module that allows a user to enter new widget identifiers into a widget data base. We will focus only on selecting equivalence classes and boundary values for the inputs.

The input specification for the module states that a widget identifier should consist of 3–15 alpha numeric characters of which the first two must be letters. We have three separate conditions that apply to the input: (i) it must consist of alphanumeric characters, (ii) the range for the total number of characters is between 3 and 15, and, (iii) the first two characters must be letters. Our approach to designing the test cases is as follows. First, we will identify input equivalence classes and give them each an identifier. Then we will augment these with the results from boundary value analysis. Tables will be used to organize and record our findings.

We will label the equivalence classes with an identifier ECxxx, where xxx is an integer whose value is one or greater. Each class will also be categorized as valid or invalid for the input domain.

First, we consider condition 1, the requirement for alphanumeric characters.

This is a “must be” condition. We derive two equivalence classes.

EC1. Part name is alphanumeric, valid.

EC2. Part name is not alphanumeric, invalid.

Then we treat condition 2, the range of allowed characters 3–15.
EC3. The widget identifier has between 3 and 15 characters, valid.
EC4. The widget identifier has less than 3 characters, invalid.
EC5. The widget identifier has greater than 15 characters, invalid.
Finally, we treat the "must be" case for the first two characters.
EC6. The first 2 characters are letters, valid.
EC7. The first 2 characters are not letters, invalid.

Note that each condition was considered separately. Conditions are not combined to select equivalence classes. The tester may find later on that a specific test case covers more than one equivalence class. The equivalence classes selected may be recorded in the form of a table as shown in Table. By inspecting such a table, the tester can confirm that all the conditions and associated valid and invalid equivalence classes have been considered. Boundary value analysis is now used to refine the results of equivalence class partitioning.

1	**EC1**	**EC2**
2	**EC3**	**EC4, EC5**
3	**EC6**	**EC7**

The boundaries to focus on are those in the allowed length for the widget identifier. An experienced tester knows that the module could have defects related to handling widget identifiers that are of length equal to, and directly adjacent to, the lower boundary of 3 and the upper boundary of 15. A simple set of abbreviations can be used to represent the bounds groups.
For example:
BLB—a value just below the lower bound
LB—the value on the lower boundary
ALB—a value just above the lower boundary
BUB—a value just below the upper bound
UB—the value on the upper bound
AUB—a value just above the upper bound
For our example module the values for the bound's groups are:
BLB—2 **BUB—**14
LB—3 **UB—**15
ALB—4 **AUB—**16

Note that in this discussion of boundary value analysis, values just above the lower bound (ALB) and just below the upper bound (BUB) were selected. These are both valid cases and may be omitted if the tester does not believe they are necessary.
The next step in the test case design process is to select a set of actual input values that covers all the equivalence classes and the boundaries. Once again, a table can be used to organize the results. Table. shows the inputs for the sample module. Note that the table has the module name, identifier, a date of creation for the test input data, and the author of the test cases.

Table only describes the tests for the module in terms of inputs derived from equivalence classes and boundaries. These include test inputs as shown in Table, along with test conditions and expected outputs. Test logs are used to record the actual outputs and conditions when execution is complete. Actual outputs are compared to expected outputs to determine whether the module has passed or failed the test. Note that by inspecting the completed table the tester can determine whether all the equivalence classes and boundaries have been covered by actual input test cases.

For this example, the tester has selected a total of nine test cases. The reader should also note then when selecting inputs based on equivalence classes, a representative value at the midpoint of the bounds of each relevant class should be included as a typical case. In this example, a test case was selected with 9 characters, the average of the range values of 3 and 15 (test case identifier 9). The set of test cases presented here is not unique: other sets are possible that will also cover all the equivalence classes and bounds.

Based on equivalence class partitioning and boundary value analysis these test cases should have a high possibility of revealing defects in the module as opposed to selecting test inputs at random from the input domain. In the latter case there is no way of estimating how productive the input choices would be. This approach is also a better alternative to exhaustive testing where many combinations of characters, both valid and invalid cases, would have to be used. Even for this simple module exhaustive testing would not be feasible.

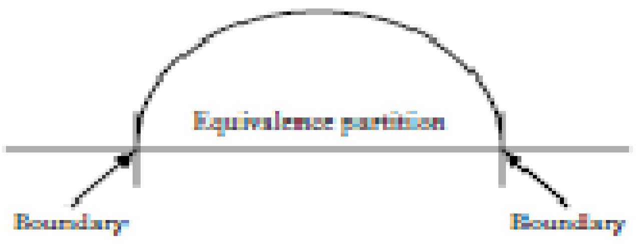


Figure 4. Equivalence classes boundaries

Automated Teller Machine

Black Box Testing

This specification describes testing methods which can be used to ensure that the system fulfils the requirements indicated in the project brief.

The following procedures are described:

1. Card Authorization
2. Cash Withdrawal (with or without receipt)
3. Display Balance (on screen or printout)
4. Order Statement or Chequebook

1. Card Authorization

Before any transactions take place the customer must insert their card into the system and enter their Personal Identification Number (PIN). If the number entered matches the number on the card the customer is allowed to continue with their transaction. If the customer fails to enter the correct PIN their card will be confiscated and a message is sent to the bank's main computer. The customer is allowed three attempts to enter their PIN.

OBJECTIVE: To test the PIN verification procedure

TEST 1: Enter the correct PIN
EXPECTED OUTCOME: Successful PIN validation

TEST 2: Enter 1 incorrect PIN and then the correct PIN
EXPECTED OUTCOME: Successful PIN validation

TEST 2: Enter 2 incorrect PINs and then the correct PIN
EXPECTED OUTCOME: Successful PIN validation

TEST 3: Enter 3 incorrect PINs
EXPECTED OUTCOME: Unsuccessful PIN validation - retain card, notify bank

2. Cash Withdrawal

The system should allow customers to withdraw money from their current accounts. Before allowing the withdrawal, the system should check the customer's balance to ensure that funds are available to cover it. If not, the system should offer any funds that are available (if any) or otherwise refuse the withdrawal. Each transaction should be recorded by the bank's main computer and the customer's accounts updated. A receipt for the transaction may be requested.

OBJECTIVE: To test the cash withdrawal procedure

TEST 1: Request available funds without receipt
EXPECTED OUTCOME: Requested cash dispensed, accounts updated

TEST 2: Request available funds with receipt
EXPECTED OUTCOME: Requested cash & receipt dispensed, accounts updated

TEST 3: Request more than available funds, accept available
EXPECTED OUTCOME: Available cash dispensed, accounts updated

TEST 4: Request more than available funds, refuse available
EXPECTED OUTCOME: No cash dispensed

TEST 5: Request funds with none available
EXPECTED OUTCOME: No cash dispensed

3. Display Balance

The customer should be able to check the balance in their account. This must be retrieved from the bank's main computer. A printout of the balance and available funds may be requested.

OBJECTIVE: To test the balance checking procedure

TEST 1: Request balance on screen
EXPECTED OUTCOME: The balance is retrieved and displayed on the screen

TEST 2: Request printout of balance
EXPECTED OUTCOME: The balance is retrieved and a printout is dispensed

4. Order Statement or Chequebook

The customer should be able to order a statement or a chequebook. The request is sent to the bank's main computer.

OBJECTIVE: To test the statement/chequebook ordering procedure

TEST 1: Request a statement
EXPECTED OUTCOME: The request is sent to bank's main computer

TEST 2: Request a chequebook
EXPECTED OUTCOME: The request is sent to bank's main computer

14. What inference can you make from random testing, requirement based testing and domain testing?Explain.

Random Testing

Each software module or system has an input domain from which test input data is selected. If a tester randomly selects inputs from the domain, this is called random testing.
For example, if the valid input domain for a module is all positive integers between 1 and 100, the tester using this approach would randomly, or unsystematically, select values from within that domain; for example, the values 55, 24, 3 might be chosen. Given this approach, some of the issues that remain open are the following:
• Are the three values adequate to show that the module meets its specification when the tests are run? Should additional or fewer values be used to make the most effective use of resources?
Are there any input values, other than those selected, more likely to reveal defects? For example, should positive integers at the beginning or end of the domain be specifically selected as inputs?
• Should any values outside the valid domain be used as test inputs?
For example, should test data include floating point values, negative values, or integer values greater than 100?
More structured approaches to black box test design address these issues.
Use of random test inputs may save some of the time and effort that more thoughtful test input selection methods require.

This type of testing can be very useful especially at the system level. Usually the tester specifies a range for the random value generator, or the test inputs are generated according to a statistical distribution associated with a pattern of usage.

Requirements-based testing

Requirements-based testing is a testing approach in which test cases, conditions and data are derived from requirements. It includes functional tests and also non-functional attributes such as performance, reliability or usability.

Stages in Requirements based Testing:

- **Defining Test Completion Criteria -** Testing is completed only when all the functional and non-functional testing is complete.
- **Design Test Cases -** A Test case has five parameters namely the initial state or precondition, data setup, the inputs, expected outcomes and actual outcomes.
- **Execute Tests** - Execute the test cases against the system under test and document the results.
- **Verify Test Results -** Verify if the expected and actual results match each other.
- **Verify Test Coverage -** Verify if the tests cover both functional and non-functional aspects of the requirement.
- **Track and Manage Defects -** Any defects detected during the testing process goes through the defect life cycle and are tracked to resolution. Defect Statistics are maintained which will give us the overall status of the project.

Requirements Testing process:

- Testing must be carried out in a timely manner.
- Testing process should add value to the software life cycle, hence it needs to be effective.
- Testing the system exhaustively is impossible hence the testing process needs to be efficient as well.
- Testing must provide the overall status of the project, hence it should be manageable.

Domain Testing

Domain Testing is a Software Testing process in which the application is tested by giving a minimum number of inputs and evaluating its appropriate outputs. The primary goal of Domain testing is to check whether the software application accepts inputs within the acceptable range and delivers required output.

It is a Functional Testing technique in which the output of a system is tested with a minimal number of inputs to ensure that the system does not accept invalid and out of range input values. It is one of the most important White Box Testing methods. It also verifies that the system should not accept inputs, conditions and indices outside the specified or valid range.

Domain testing differs for each specific domain so you need to have domain specific knowledge in order to test a software system.

15.Compare and Contrast Alpha and Beta Testing with Suitable Example.

- ✓ Alpha Testing is a type of software testing performed to identify bugs before releasing the product to real users or to the public. Alpha Testing is one of the user acceptance testing.
- ✓ Beta Testing is performed by real users of the software application in a real environment. Beta testing is one of the type of User Acceptance Testing.

The difference between Alpha and Beta Testing is as follow:

Alpha Testing	Beta Testing
Alpha testing involves both the white box and black box testing.	Beta testing commonly uses black box testing.
Alpha testing is performed by testers who are usually internal employees of the organization.	Beta testing is performed by clients who are not part of the organization.
Alpha testing is performed at developer's site.	Beta testing is performed at end-user of the product.
Reliability and security testing are not checked in alpha testing.	Reliability, security and robustness are checked during beta testing.
Alpha testing ensures the quality of the product before forwarding to beta testing.	Beta testing also concentrates on the quality of the product but collects users input on the product and ensures that the product is ready for real time users.
Alpha testing requires a testing environment or a lab.	Beta testing doesn't require a testing environment or lab.
Alpha testing may require long execution cycle.	Beta testing requires only a few weeks of execution.
Developers can immediately address the critical issues or fixes in alpha testing.	Most of the issues or feedback collected from beta testing will be implemented in future versions of the product.

Alpha Testing can be defined as a form of acceptance testing carried out to identify various types of issues or bugs before publishing the build or executable of the software public or

market. This test type focuses on real users through black box and white box testing techniques. The focus remains on the task which a general user might want or experience.
Alpha testing any product is done when product development is on the verge of completion. Slight changes in design can be made after conducting the alpha test. This testing methodology is performed in lab surroundings by the developers. Here developers see things in the software from users' points and try to detect the problems. These testers are internal company or organization's employees or maybe a part of the testing team. Alpha testing is done early at the end of software development before beta testing.

Beta Testing can be defined as the second stage of testing any product before release, where a sample of the released product with minimum features and characteristics is being given to the intended audience for trying out or temporarily using the product. Unlike an alpha test, the beta test is being carried out by real users in the real environment. This allows the targeted customers to dive into the product's design, working, interface, functionality, etc.

16.List the types of Testing performed while developing a website. Explain the process of testing a Website.

To perform comprehensive web application testing, it is necessary to do different types of tests that fulfill the requirements mentioned in the web app testing checklist above. These different types of tests are described below.

1. Functional Testing

Functional Testing aims at testing the overall functional capabilities of the web app. In Functional Testing, all the features and operational behavior of a web app are tested to ensure the compatibility of features and specifications. It doesn’t have anything to test the internal mechanism and output specific to any input.

The functional testing intends to verify if your product has all the functional specifications as desired or mentioned in the development documentation. It ensures that the functionalities and requirements work from the end user’s perspective. It tests every minute possibility & input by a user that developers didn’t expect. The testing method should be completed with automated test cases to provide an effective positive user experience.
You can perform Functionality tests by following the below steps:

- Check the list of all the functional requirements
- Get a clear understanding of all the requirements through it
- Based on all the requirements, identify test input
- Based on the chosen test input values, compute the outcomes
- Execute all the test case

2. Usability Testing

Usability testing checks the user-friendliness of the elements in the web app. It tests the flow of the web app and how the user can navigate through it seamlessly. Usability testing relies on standard practices that ensure the ease of use and effectiveness of a web app. It roots out all the errors, like typos, punctuation, and weak flow. Together, these elements can make a poor impression of a website and reduce its capacity. Usability tests the application in the form that is accessed by the end-users. Hence, studying the user personas and their requirements is essential for usability tests.
You can perform Usability tests by following the below steps:

- Create a prototype or product ready to test
- Find the right users of testing
- Conduct a usability test
- Document all the tests
- Make a detailed analysis and repeat the process

3. Interface Testing

Interface Testing checks how the website communicates at different interfaces such as Application, Web Server, and the Database. It is different from Integration testing, where the different internal components of the application are put together and tested. In Interface testing, how the application communicates with the web server or the database is checked. During Interface testing, various interfaces are tested as follows:

- **Application:** Test requests are sent correctly to the Database, and the client-side output is displayed correctly. In this scenario, any error will be caught by the application. The error will not go to the end-user but to the administrator.
- **Web Server:** The web server is responsible for handling all application requests
- **Database Server:** It ensures that all the sent queries give desired outcomes to the Database.
Any web application has two major interfaces where the tests need to be performed:

- Between a web server and an application server
- Between the application server and the database server
Interface testing performed between any of the above interface is used to:

- Check whether servers are working perfectly in different scenarios
- Check if all faults are managed properly or any error pops out
- Examine all the results and check the link to a web server

4. Database Testing

Database testing is important to check how the web application stores data and performs various functions that include:

- Executing queries during testing if any error comes up
- Keep a regular check on the response time of queries
- Retrieved test data must be reflected accurately in the web application
- Any change when creating, updating, or deleting data in the database must not have data integrity

You can perform Database tests by following the below steps:

- Set up the test environment to test the software application.
- Execute the test by run a test cases under the test environment.
- Check the results and validate the output with the expected results.

5. Performance Testing

In Performance Testing, the test process relates to a system's speed, stability, and scalability under different scenarios. It checks whether the system can deliver the same efficiency with various outputs from different users. Performance Testing ensures:

- The architecture can work with heavier load conditions.
- Fast response time with every new version of the software.
- Application can handle the maximum load and deliver optimal performance.

You can perform Performance tests by following the below steps:

- Identify the Test Environment and Tools
- Define Acceptable Performance Criteria
- Plan and Design Tests
- Prepare Test Environment and Tools
- Run the Performance Tests
- Resolve and Retest

6. Compatibility Testing

The same website renders differently in different browsers and their versions, which is why it is essential to perform cross browser compatibility tests. Cross Browser Testing allows you to test if your web application functions seamlessly and consistently across multiple browsers on different desktop and mobile devices. Every web element like buttons, images, text fields, lists, and more changes with the change in OS, which is why it is important to test cross-platform compatibility by testing on real devices as opposed to emulators or simulators.

7. Security Testing

The process involves testing, analyzing, and reporting every security aspect of your app. Security administrators do it to test the security level of the app through both manual and automated security testing methods. Security testing performs different functions such as:

- It roots out unauthorized access to secure pages and puts an end to downloading authorized pages without approval.
- It kills automatic sessions after prolonged user inactivity
- It redirects website to encrypted SSL pages
  You can perform Performance tests by following the below steps:

- Make a threat List and prepare test plan
- Create a traceability Matrix for every error and Vulnerability
- Choose the testing tools as per the requirement
- Prepare a test case document
- Execute Security Test Cases

17.(a) List and explain the skills needed by a test specialist.

By supporting a test group an organization acquires leadership relate to testing and quality issues. For example, there will be staff with the necessary skills and motivation to be responsible for:

•Maintenance and application of test policies;

•Development and application of test-related standards;

•Participating in requirements, design, and code reviews;

•Test planning;

•Test design;• Test execution;

•Test measurement;

• Test monitoring (tasks, schedules, and costs);

•Defect tracking, and maintaining the defect repository;

•Acquisition of test tools and equipment;

•Identifying and applying new testing techniques, tools, and methodologies; mentoring and training of new test personnel;

•Test reporting.

The staff members of such a group are called test specialists or test engineers.

Skills needed by a test specialist

Given, many managerial and personal skills are necessary for success in the area of work. On the personal and managerial level specialist must have:

•Organizational, and planning skills;

• The ability to keep track of, and pay attention to, details;

• The determination to discover and solve problems;

•The ability to work withothersand be ableto resolve conflicts;

•The ability to mentor and train others;

•The ability to work with usersand clients;

• Strong written and oral communication skills;

• The ability to work in a variety of environments;

• The ability to think creatively

The first three skills are necessary because testing is detail and problem oriented. In addition, testing involves policy making, knowledge of different types of application are as, planning, and the ability to organize and monitor information, tasks, and people. Testing also requires interactions with many other engineering professionals such as project managers, developers, analysts, process personal, and software quality assurance staff. Test professionals often interact with clients to prepare certain types of tests, for example acceptance tests. Testers also have to prepare test- related documents and make presentations. Training and mentoring of new hires to the testing group is also apart of the tester‘s job. In addition, test specialists must be creative, imaginative, and experiment oriented.

They need to be able to visualize the many ways that a software item should be tested, and make hypotheses about the different types of defects that could occur and the different ways the software could fail. On the technical level testers need to have:

• An education that includes an understanding of general software engineering principles, practices, and methodologies;

• Strong coding skills and an understanding of code structure and behavior;

• A good understanding of testing principles and practices;

• A good understanding of basic testing strategies, methods, and techniques;

• The ability and experience to plan, design, and execute test cases and test procedures on multiple levels(unit, integration, etc.);

• knowledge of process issues;

• Knowledge of how networks, databases, and operating systems are organized and how they work;

• Knowledge of configuration management;

• knowledge of test-related documents and the role each documents plays in the testing process;

• The ability to define, collect, and analyze test-related measurements;

• The ability, training, and motivation to work with testing tools and equipment;

• knowledge of quality issues.

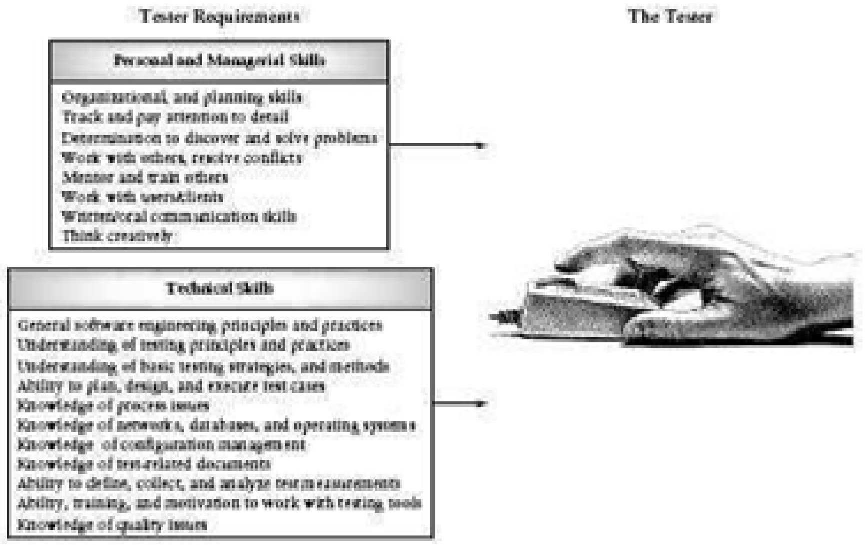

Tester Requirements
The Tester
Personal and Managerial Skills
Organizational, and planning skills
Track and pay attention to detail
Determination to discover and solve problems
Work with others, resolve conflicts
Mentor and train others
Work with users/clients
Written/oral communication skills
Think creatively
Technical Skills
General software engineering principles and practices
Understanding of testing principles and practices
Understanding of basic testing strategies, and methods
Ability to plan, design, and execute test cases
Knowledge of process issues
Knowledge of networks, databases, and operating systems
Knowledge of configuration management
Knowledge of test-related documents
Ability to define, collect, and analyze test measurements
Ability, training, and motivation to work with testing tools
Knowledge of quality issues

FIG. 8.1
Test specialist skills.

In order to carry out testing tasks testers need to have knowledge of how requirements, specifications, and designs are developed and how different methodologies can be applied. They should understand how errors and defects are introduced into the software artifacts even at early stages of the lifecycle. Testers should have strong programming backgrounds to help them visualize how code works, how it behaves, and the possible defects it could contain. They also need coding experience to support the development of the test harnesses which often involve a considerable coding effort in themselves.

Testers must have knowledge of both white and black box techniques and methods and the ability to use them to design test cases. Organizations need to realize that this knowledge is a necessary prerequisite for tool use and test automation. Testers need to understand the need for multilevel tests and approaches used for testing a teach level.

(b) Name the reports of test results and the contents available in each test reports.

The test plan and its attachments are test-related documents that are prepared prior to test execution. There are additional documents related to testing that are prepared during and after execution of the tests. The IEEE Standard for Software Test Documentation describes the following documents.

Test Log

The test log should be prepared by the person executing the tests. It is a diary of the events that take place during the test. It supports the concept of a test as a repeatable experiment. In the experimental equivalence classes world of engineers and scientists detailed logs are kept when carrying out experimental work. Software engineers and testing specialists must follow this example to allow others to duplicate their work.

The test log is in valuable for use in defect repair. It gives the developer a snapshot of the events associated with a failure. The test log, in combination with the test incident report which should be generated in case of anomalous behavior, gives valuable clues to the developer whose task it is to locate the source of the problem. The combination of documents helps to prevent incorrect decisions based on incomplete or erroneous test results that often lead to repeated, but in effective, test-patch-test cycles.

Retest that follows defect repair is also supported by the test log. In addition, the test log is valuable for (i) regression testing that takes place in the development of future releases of a software product, and (ii) circumstances where code from are use library is to be reused. In all these cases it is important that the exact conditions of a test run are clearly documented so that it can be repeated with accuracy.

Test Log Identifier

Each test log should have a unique identifier.

Description

In the description section the tester should identify the items being tested, their version/revision number, and their associated Test Item/Transmittal Report. The environment in which the test is conducted should be described including hardware and operating system details.

Activity and Event Entries

The tester should provide dates and names of test log authors for each event and activity. This section should also contain:

1. Execution description: Provide a test procedure identifier and also the names and functions of personnel involved in the test.

2. Procedure results: For each execution, record the results and the location of the output. Also report pass/fail status.

3. Environmental information: Provide any environmental conditions specific to this test.

4. Anomalous events : Any events occurring before / after an unexpected event should be recorded. If a tester is unable to start or complete test procedure, details relating to these happenings should be recorded (e.g., a power failure or operating system crash).

5. Incident report identifiers : Record the identifiers of incident reports generated while the test is being executed.

Test Incident Report

The tester should record in a test incident report (sometimes called a problem report)any event that occurs during the execution of the tests that is unexpected, unexplainable, and that requires a follow-up investigation. The IEEE Standard for Software Test Documentation recommends the following sections in the report:

1. Test Incident Report identifier: to uniquely identify this report.

2. Summary: to identify the test items involved, the test procedures, test cases, and test log associated with this report.

3. Incident description: this should describe time and date, testers, observers, environment, inputs, expected outputs, actual outputs, anomalies, procedure step, environment, and attempts to repeat. Any other information useful for the developers who will repair the code should be included.

4. Impact: what impact will this incident have on the testing effort, the test plans, the test procedures, and the test cases? A severity rating should be inserted here.

Test Summary Report

This report is prepared when testing is complete. It is a summary of the results of the testing efforts. It also becomes a part of the project‘s historical database and provides a basis for lessons learned as applied to future projects. When a project post mortem is conducted, the Test Summary Report can help managers, testers, developers, and SQA staff to evaluate the effectiveness of the testing efforts. The IEEE test documentation standard describes the following sections for the Test Summary Report:

1. Test Summary Report identifier: to uniquely identify this report.

2. Variances : these are descriptions of any variances of the test items from their original design. Deviations and reasons for the deviation from the test plan, test procedures, and test designs are discussed.

3. Comprehensiveness assessment: the document author discusses the comprehensiveness of the test effort as compared to test objectives and test completeness criteria as described in the test plan. Any features or combination of features that were not as fully tested as was planned should be identified.

4. Summary of results: the document author summarizes the testing results. All resolved incidents and their solutions should be described. Unresolved incidents should be recorded.

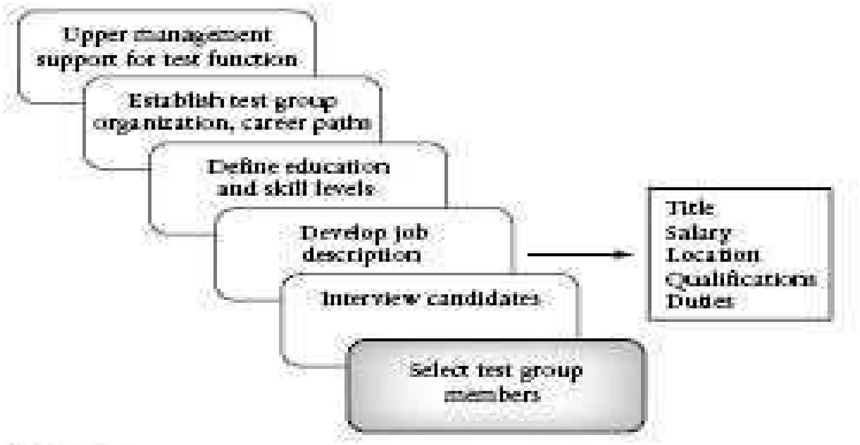
5. Evaluation: in this section the author evaluates each test item based on test results. Did it pass/fail the tests? If it failed, what was the level of severity of the failure?

6. Summary of activities: all testing activities and events are summarized.
Resource consumption, actual task durations, and hardware and software tool usage should be recorded.
7. Approvals: the names of all persons who are needed to approve this document are listed with space for signatures and dates.

18.Analyze the various steps in forming the test group .How will you build a testing group discuss with an example.

It was mentioned that organizing, staffing, and directing were major activities required to manage a project and a process. These apply to managing the testing process as well. Staffing activities include filling positions, as simulating new personnel, education and training, and staff evaluation. Directing includes providing leadership, building teams, facilitating communication, motivating personnel, resolving conflicts, and delegating authority. Organizing includes selecting organizational structures, creating positions, defining responsibilities, and delegating authority. Hiring staff for the testing group, organizing the testing staff members into teams, motivating the team members, and integrating the team into the overall organizational structure are organizing, staffing, and directing activities your organization will need to perform to build a managed testing process.

To initiate the process, upper management must support the decision to establish a test group and commit resources to the group. Decisions must be made on how the testing group will be organized, what career paths are available, and how the group fits in to the organizational structure. When hiring staff to fill test specialist positions, management should have a clear idea of the educational and skill levels required for each testing position and develop formal job descriptions to fill the test groups lots. When the job description has been approved and distributed, the interviewing process takes place. Interviews should be structured and of a problem-solving nature. The interviewer should prepare an extensive list of questions to determine the interviewee‘s technical background as well as his or her personal skills and motivation. Zawacki has developed a general guide for selecting technical staff members that can be used by test managers. When the team has been selected and is up and working on projects, the team manager is responsible for keeping the test team positions filled. He must continually evaluate team member performance.



Upper management support for test function
Establish test group organization, career paths
Define education and skill levels
Develop job description
Interview candidates
Select test group members
Title
Salary
Location
Qualifications
Duties

FIG. 8.2
Steps in forming a test group.

STRUCTURE OF TEST GROUP

It is important for a software organization to have an independent testing group. The group should have a formalized position in the organizational hierarchy. A reporting structure should be established and resources allocated to the group. Will eventually need to upgrade their testing function to the best case scenario which is a permanent centralized group of dedicated testers with the skills .This group is solely responsible for testing work. The group members are assigned to projects throughout the organization where they do their testing work. When the project is completed they return to the test organization for reassignment. They report to a test manager or test director , not a project manager. In such an organization testers are viewed as assets. They have defined career paths to follow which contributes to long-term job satisfaction. Since they can be assigned to a project at its initiation, they can give testing support throughout the software lifecycle. Because of the permanent nature of the test organization there is at test infrastructure that endures. There is a test knowledge base of test processes, test procedures, test tools, and test histories (lessons learned).Dedicated staff is responsible for maintaining a test case and test harness library.

A test organization is expensive, it is a strategic commitment. Given the complex nature of the software being built, and its impact on society, organizations must realize that a test organization is necessary and that it has many benefits. By investing in a test organization a company has access to a group of specialists who have the responsibilities and motivation to:

• Maintain testing policy statements;

• Plan the testing efforts;

• Monitor and track testing efforts so that they are on time and within budget;

• Measure process and product attributes;

• Provide management with independent product and process quality information;

• Design and execute tests with no duplication of effort;

• Automate testing;

• Participate in reviews to insure quality; are meet.

The duties of the team members may vary in individual organizations. The following gives a brief description of the duties for each tester that are common to most organizations.

The Test Manager

In most organizations with a testing function, the test manager(or test director) is the central person concerned with all aspects of testing and quality issues. The test manager is usually responsible for test policy making, customer interaction, test planning, test documentation, controlling and monitoring of tests, training, test tool acquisition, participation in inspections and walkthroughs, reviewing test work, the test repository, and staffing issues such as hiring, firing, and evaluation of the test team members. He or she is also with upper management, project management, and the quality assurance and marketing staff.

The Test Lead

The test lead assists the test manager and works with a team of test engineers on individual projects. He or she may be responsible for duties such as test planning, staff supervision, and status reporting. The test lead also participates in test design, test execution and reporting, technical reviews, customer interaction, and tool training.

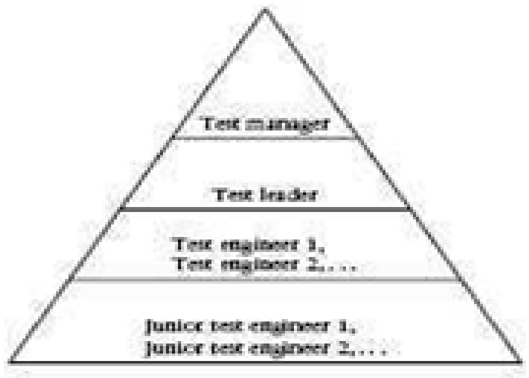


FIG. 8.3

The test team hierarchy.

The Test Engineer

The test engineers design, develop, and execute tests, develop test harnesses, and set up test laboratories and environments. They also give input to test planning and support maintenance of the test and defect repositories.

The Junior Test Engineer

The junior test engineers are usually new hires. They gain experience by participating in test design, test execution, and test harness development. They may also be asked to review user manuals and user help facilities defect and maintain the test and defect repositories.

19. (a) Outline the challenges in automation.

Software Test automation makes use of specialized tools to control the execution of tests and compares the actual results against the expected result. Usually regression tests, which are repetitive actions, are automated.

Testing Tools not only help us to perform regression tests but also helps us to automate data set up generation, product installation, GUI interaction, defect logging, etc.

Criteria for Tool Selection:

For automating any application, the following parameters should be considered.

- Data driven capabilities
- Debugging and logging capabilities
- Platform independence
- Extensibility & Customizability
- E-mail Notifications
- Version control friendly
- Support unattended test runs

Types of Frameworks:

Typically, there are 4 test automation frameworks that are adopted while automating the applications.

- Data Driven Automation Framework
- Keyword Driven Automation Framework
- Modular Automation Framework

- Hybrid Automation Framework

Popular Tools that are used for Functional automation:

Product	Vendor	URL
Quick Test Professional	HP	www.hp.com/go/qtp
Rational Robot	IBM	http://www-03.ibm.com/software/products/us/en/robot/
Coded UI	Microsoft	http://msdn.microsoft.com/en-us/library/dd286726.aspx
Selenium	Open Source	http://docs.seleniumhq.org/
Auto IT	Open Source	http://www.autoitscript.com/site/

Popular Tools that are used for Non-Functional automation:

Product	Vendor	URL
Load Runner	HP	www.hp.com/go/LoadRunner
Jmeter	Apache	jmeter.apache.org/
Burp Suite	PortSwigger	http://portswigger.net/burp/
Acunetix	Acunetix	http://www.acunetix.com/

Challenges in Automated Testing

1. Test Case Prioritization

2. Communication and Collaboration

3. Finding the right Testing Framework or Tool

4. Taking Real User Conditions into Account

5. Finding the right skills

Challenges in Test Automation Life Cycle is the subset of Software Testing Life Cycle. Through automation, we can expedite the process of validating software and it reduces time and effort to deliver quality product. Of course, there are challenges exists from all the angles while automating application under test (AUT).

(b) What is the need for metrics in testing ? Analyze about Productivity metrics.

In software testing, **Metric** is a quantitative measure of the degree to which a **system, system component, or process**possesses a given attribute.

In other words, metrics helps estimating the progress, quality and health of a software testing effort. The ideal example to understand metrics would be a weekly mileage of a car compared to its ideal mileage recommended by the manufacturer.

Sr#	Steps to test metrics	Example
1	Identify the key software testing processes to be measured	• Testing progress tracking process
2	Use the data as baseline to define the metrics	• The number of test cases planned to be executed per day
3	Determination of the information to be tracked, frequency of tracking and the person responsible	• The actual test execution per day will be captured by the test manager at the end of the day
4	Effective calculation, management and interpretation of the defined metrics	• The actual test cases executed per day

5	Identify the areas of improvement depending on the interpretation of defined metrics	• The test case execution falls below the goal set, we need to investigate the reason and suggest the improvement measures

Types of Metrics

Process Metrics
Product Metrics
Project Metrics

- **Process Metrics:** It can be used to improve the process efficiency of the SDLC (Software Development Life Cycle)
- **Product Metrics:** It deals with the quality of the software product
- **Project Metrics:** It can be used to measure the efficiency of a project team or any tools being used by the team members

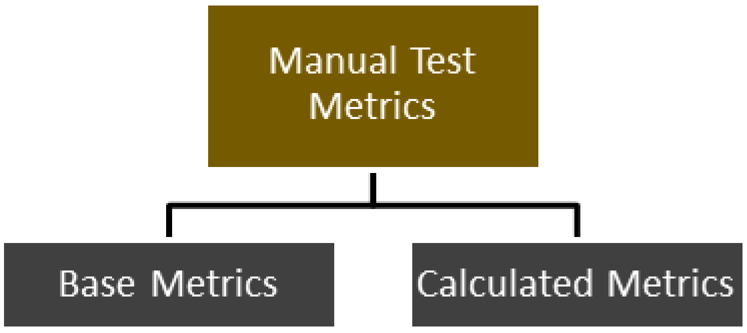
Identification of correct testing metrics is very important. Few things need to be considered before identifying the test metrics

- Fix the target audience for the metric preparation
- Define the goal for metrics
- Introduce all the relevant metrics based on project needs
- Analyse the cost benefits aspect of each metrics and the project lifestyle phase in which it results into the maximum output

Manual Test Metrics

Manual test metrics is classified into two classes

- **Base Metrics**
- **Calculated Metrics**



Base metrics is the raw data collected by Test Analyst during the test case development and execution (**# of test cases executed, # of test cases**). While, calculated metrics is derived from the data gathered in base metrics. Calculated metrics is usually tracked by the test manager for test reporting purpose (**% Complete, % Test Coverage**).

Process and Project Metrics

- Metrics should be collected so that process and product indicators can be ascertained
- Process metrics used to provide indictors that lead to long term process improvement
- Project metrics enable project manager to
  - Assess status of ongoing project
  - Track potential risks
  - Uncover problem are before they go critical
  - Adjust work flow or tasks
  - Evaluate the project team's ability to control quality of software wrok products

Process Metrics

- Private process metrics (e.g. defect rates by individual or module) are only known to by the individual or team concerned.
- Public process metrics enable organizations to make strategic changes to improve the software process.
- Metrics should not be used to evaluate the performance of individuals.
- Statistical software process improvement helps and organization to discover where they are strong and where are week.

Statistical Process Control

1. Errors are categorized by their origin
2. Record cost to correct each error and defect
3. Count number of errors and defects in each category
4. Overall cost of errors and defects computed for each category
5. Identify category with greatest cost to organization
6. Develop plans to eliminate the most costly class of errors and defects or at least reduce their frequency

Project Metrics

- A software team can use software project metrics to adapt project workflow and technical activities.
- Project metrics are used to avoid development schedule delays, to mitigate potential risks, and to assess product quality on an on-going basis.
- Every project should measure its inputs (resources), outputs (deliverables), and results (effectiveness of deliverables).

Size-Oriented Metrics

- Derived by normalizing (dividing) any direct measure (e.g. defects or human effort) associated with the product or project by LOC.
- Size oriented metrics are widely used but their validity and applicability is widely debated.

Function-Oriented Metrics

- Function points are computed from direct measures of the information domain of a business software application and assessment of its complexity.
- Once computed function points are used like LOC to normalize measures for software productivity, quality, and other attributes.
- The relationship of LOC and function points depends on the language used to implement the software.

Object-Oriented Metrics

- Number of scenario scripts (NSS)
- Number of key classes (NKC)
- Number of support classes (e.g. UI classes, database access classes, computations classes, etc.)
- Average number of support classes per key class
- Number of subsystems (NSUB)

Software Quality Metrics

- Factors assessing software quality come from three distinct points of view (product operation, product revision, product modification).
- Software quality factors requiring measures include

Metric is defined as “Standard of measurement by which efficiency, progress, performance, productivity, quality of a deliverable, process, project or product can be assessed”. Metrics help in building predictability, improving organization’s decision making ability, and lay out what is working and what is not working within the organization and help guide the management focus in the right directions.
Project management metrics enable Project managers to:

- Assess status of ongoing project in terms of schedule, cost and profitability.
- Foresee any potential risks.
- Nail down the problems much before they become severe.
- Keep a check on project profitability.

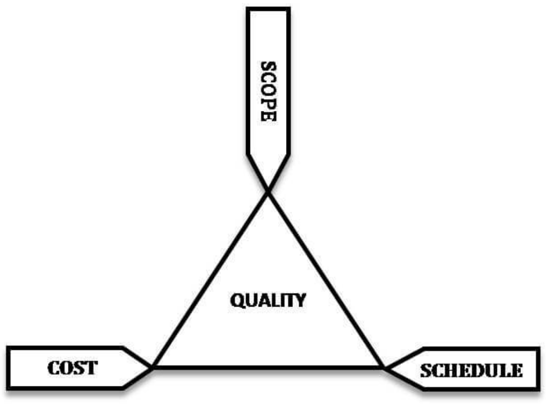
- Assess productivity of team.
- Assess quality of work products to be delivered.

There can be different project management metrics defined based on complexity and nature of project. However, following five performance metric groups cover all the important aspects of a project to measure during execution.

20. Explain the different types of Test defect metrics under progress metrics based on what they measure and what area they focus on?

Software Testing Metrics are the quantitative measures used to estimate the progress, quality, productivity and health of the software testing process. The goal of software testing metrics is to improve the efficiency and effectiveness in the software testing process and to help make better decisions for further testing process by providing reliable data about the testing process.

A Metric defines in quantitative terms the degree to which a system, system component, or process possesses a given attribute. The ideal example to understand metrics would be a weekly mileage of a car compared to its ideal mileage recommended by the manufacturer.



Defect metrics – A variety of metrics on the number and nature of defects found by annual testers, including:

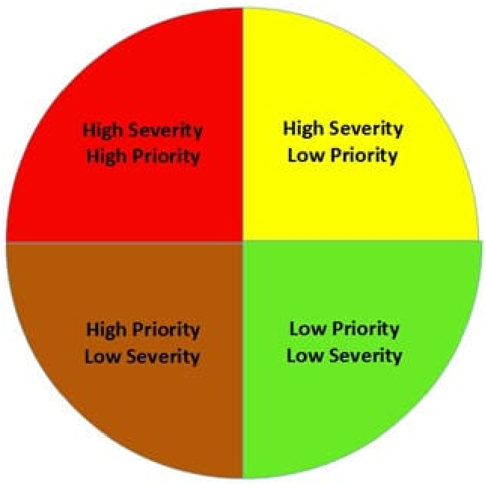
- **Defects by priority**
- **Defects by severity**
- **Defect slippage ratio** – The percentage of defects that manual testers did not manage to identify before the software was shipped.

Priority and Severity have some classifications amongst them that aid in determining how the defect must be handled. A lot of different organizations have different defect logging tools, so the levels might vary.
Let’s take a look at the different levels for both Priority and Severity.

- High Priority, High Severity
- High Priority, Low Severity

- High Severity, Low Priority
- Low Severity, Low Priority

The following figure depicts the classification of the categories in a single snippet.



#1) High Severity and High Priority

Any Critical/major business case failure automatically gets promoted to this category.

Any defects due to which the testing cannot continue at any cost or causes a severe system failure to fall into this category.

For Example, clicking on a particular button doesn't load the feature itself. Or performing a particular function brings down the server consistently and causes data loss. The red lines in the above figure indicate these kinds of defects.

For Example,
The system crashes after you made the payment or when you are not able to add the items to the Cart, this defect is marked as High Severity and High Priority defect.

Another example would be ATM vending currency feature wherein after entering the correct username and the password, the machine does not dispense money but deducts the transferred from your account.

#2) High Priority and Low Severity
Any minor severity defects that could directly impact the user experience automatically gets promoted to this category.

Defects that have to be fixed but do not affect the application come under this category.

For Example, the feature is expected to display a particular error to the user with respect to its return code. In this case, functionally the code will throw an error, but the message will need to be more relevant to the return code generated. The blue lines in the figure indicate these kinds of defects.

For Example,
The logo of the company in the front-page is wrong, it is considered to be **High**

Priority and Low Severity defect**.**

__Example 1)__ In the Online shopping website when the Front Page logo is spelled wrong.
__Example 2)__ In the bank logo, instead of ICICI, it is written as ICCCI.
In terms of functionality, it is not affecting anything so we can mark as Low Severity, but it has an impact on user experience. This kind of defect needs to be fixed on high priority even though they have very less impact on the application side.

#3) High Severity and Low Priority

Any defect that is functionally not meeting the requirements or have any functional implications on the system but sidelined to back seat by the stakeholders when it comes to business criticality automatically gets promoted to this category.

Defects that have to be fixed but not immediately. This can specifically occur during ad-hoc testing. It means that the functionality is affected to a large extent, but is observed only when certain uncommon input parameters are used.

__For Example,__ a particular functionality can be used only on a later version of the firmware, so in order to verify this – the tester actually downgrades his system and performs the test and observes a serious functionality issue that is valid. In such a case the defects will be classified in this category denoted by pink lines, as normally end users will be expected to have a higher version of the firmware.

__For Example,__

In a social networking site, if a beta version of a new feature is released with not many active users using that facility as of today. Any defect found on this feature can be classified as a low priority as the feature takes back seat due to business classification as not important.

Though this feature is having a functional defect, as it is not impacting the end customers directly, a business stakeholder can classify the defect under low priority though it has a severe functional impact on the application.

This is a high severity fault but can be prioritized to low priority as it can be fixed with the next release as a change request. Business stakeholders also prioritize this feature as a rarely used feature and do not impact any other features that have a direct impact on user experience. This kind of defect can be classified under the **High Severity but Low Priority** category.

#4) Low Severity and Low Priority

Any spelling mistakes /font casing/ misalignment in the paragraph of the 3rd or 4th page of the application and not in the main or front page/ title.
These defects are classified in the green lines as shown in the figure and occur when there is no functionality impact, but still not meeting the standards to a small degree. Generally cosmetic errors or say dimensions of a cell in a table on UI are classified here.
__For Example,__
If the privacy policy of the website has a spelling mistake, this defect is set as **Low Severity and Low Priority.**