Unit 5: Software Testing

1. Describe Software Verification and Validation:

Verification and Validation is the process of investigating that a software system satisfies specifications and standards and it fulfils the required purpose.

Barry Boehm described verification and validation as the following:

Verification: Are we building the product right?

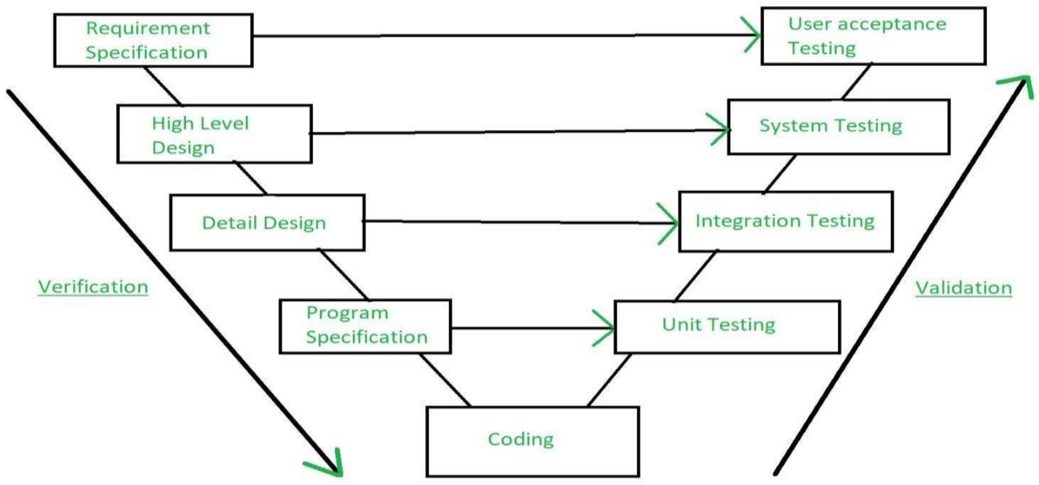
Validation: Are we building the right product?

Verification:
Verification is the process of checking that a software achieves its goal without any bugs. It is the process to ensure whether the product that is developed is right or not. It verifies whether the developed product fulfills the requirements that we have.
Verification is **Static Testing**.
Activities involved in verification:

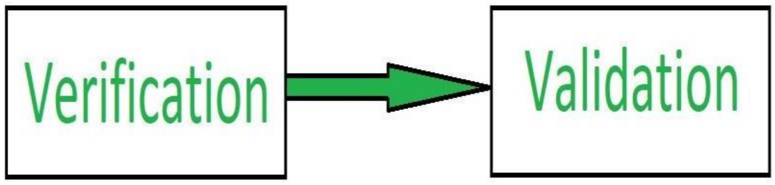
1. Inspections
2. Reviews
3. Walkthroughs
4. Desk-checking

Validation:
Validation is the process of checking whether the software product is up to the mark or in other words product has high level requirements. It is the process of checking the validation of product i.e. it checks what we are developing is the right product. It is validation of actual and expected product.
Validation is the **Dynamic Testing**.
Activities involved in validation:

1. Black box testing
2. White box testing
3. Unit testing
4. Integration testing



Note: Verification is followed by Validation.



Verification is the process of checking that a software achieves its goal without any bugs. It is the process to ensure whether the product that is developed is right or not. It verifies whether the developed product fulfills the requirements that we have. Verification is static testing.
Verification means **Are we building the product right?**
Validation is the process of checking whether the software product is up to the mark or in other words product has high level requirements. It is the process of checking the validation of product i.e. it checks what we are developing is the right product. it is validation of actual and expected product. Validation is the dynamic testing.
Validation means **Are we building the right product?**

The difference between Verification and Validation is as follow:

Black Box Testing	**White Box Testing**
It is a way of software testing in which	It is a way of testing the software in

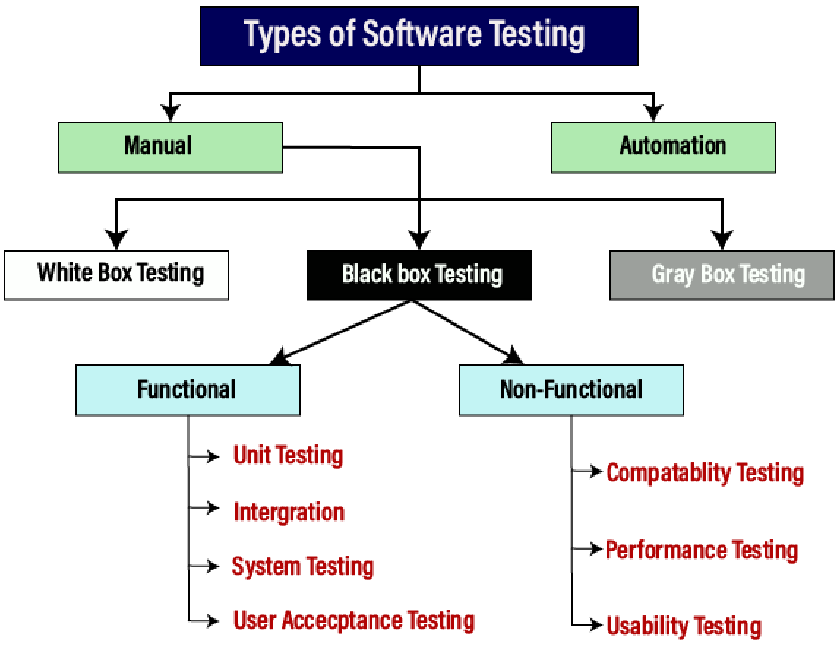
the **internal structure or the program or the code** is hidden and nothing is known about it.	which **the tester has knowledge about the internal structure or the code or the program of the software**.
Implementation of code is not needed for black box testing.	**Code implementation is necessary** for white box testing.
It is mostly done by **software testers.**	It is mostly done by **software developers.**
No knowledge of implementation is needed.	**Knowledge of implementation** is required.
It can be referred to as **outer or external software testing**.	It is **the inner or the internal software testing.**
It is **a functional test** of the software.	It is **a structural test** of the software.
This testing can be initiated based on the **requirement specifications document.**	This type of testing of software is started **after a detail design document**.
No knowledge of programming is required.	It is **mandatory** to have knowledge of programming.

It is the **behavior testing** of the software.	It is the **logic testing** of the software.
It is applicable to **the higher levels** of testing of software.	It is generally applicable to **the lower levels** of software testing.
It is also called **closed testing.**	It is also called as **clear box testing**.
It is **least time consuming**.	It is **most time consuming.**
It is **not suitable or preferred for algorithm** testing.	It is **suitable** for algorithm testing.
Can be done by trial and error ways and methods.	Data domains along with inner or internal boundaries can be better tested.
Example: Search something on Google by using keywords	**Example:** By input to check and verify loops
Black-box test design techniques- • Decision table testing • All-pairs testing • Equivalence partitioning • Error guessing	**White-box test design techniques-** • Control flow testing • Data flow testing • Branch testing

Types of Black Box Testing: • Functional Testing • Non-functional testing • Regression Testing	**Types of White Box Testing:** • Path Testing • Loop Testing • Condition testing
It **is less exhaustive** as compared to white box testing.	It is comparatively **more exhaustive** than black box testing.

2. Different types of Software Testing

Testing is the process of executing a program to find errors. To make our software perform well it should be error-free. If testing is done successfully it will remove all the errors from the software.



Principles of Testing:-

(i) All the tests should meet the customer requirements.
(ii) To make our software testing should be performed by a third party.
(iii) Exhaustive testing is not possible. As we need the optimal amount of testing based on the risk assessment of the application.
(iv) All the tests to be conducted should be planned before implementing it
(v) It follows the Pareto rule(80/20 rule) which states that 80% of errors come from 20% of program components.
(vi) Start testing with small parts and extend it to large parts.

Types of Testing:-

1. Unit Testing

It focuses on the smallest unit of software design. In this, we test an individual unit or group of interrelated units. It is often done by the programmer by using sample input and observing its corresponding outputs.

Example:

a) In a program we are checking if the loop, method, or function is working fine

b) Misunderstood or incorrect, arithmetic precedence.

c) Incorrect initialization

2. Integration Testing

The objective is to take unit-tested components and build a program structure that has been dictated by design. Integration testing is testing in which a group of components is combined to produce output.

Integration testing is of four types: (i) Top-down (ii) Bottom-up (iii) Sandwich (iv) Big-Bang
Example:

(a) Black Box testing:- It is used for validation.

In this, we ignore internal working mechanisms and focus on **what is the output?**.

(b) White box testing:- It is used for verification. In this, we focus on internal mechanisms i.e. **how the output is achieved?**

3. Regression Testing

Every time a new module is added leads to changes in the program. This type of testing makes sure that the whole component works properly even after adding components to the complete program.

Example

In school, record suppose we have module staff, students and finance combining these modules and checking if on integration of these modules works fine in regression testing

4. Smoke Testing

This test is done to make sure that the software under testing is ready or stable for further testing . It is called a smoke test as the testing of an initial pass is done to check if it did not catch the fire or smoke in the initial switch on.
Example:

If the project has 2 modules so before going to the module make sure that module 1 works properly

5. Acceptance Testing

Acceptance testing is done by the customers to check whether the delivered products perform the desired tasks or not, as stated in requirements.

- ### Alpha Testing

This is a type of validation testing. It is a type of *acceptance testing* which is done before the product is released to customers. It is typically done by QA people.
Example:
When software testing is performed internally within the organization

- ### Beta Testing

The beta test is conducted at one or more customer sites by the end-user of the software. This version is released for a limited number of users for testing in a real-time environment
Example:

When software testing is performed for the limited number of people

7. System Testing

This software is tested such that it works fine for the different operating systems. It is covered under the black box testing technique. In this, we just

focus on the required input and output without focusing on internal working. In this, we have security testing, recovery testing, stress testing, and performance testing
Example:

This includes functional as well as non-functional testing

8. Stress Testing
In this, we give unfavourable conditions to the system and check how they perform in those conditions.
Example:

(a) Test cases that require maximum memory or other resources are executed

(b) Test cases that may cause thrashing in a virtual operating system

(c) Test cases that may cause excessive disk requirement

9. Performance Testing
It is designed to test the run-time performance of software within the context of an integrated system. It is used to test the speed and effectiveness of the program. It is also called load testing. In it we check, what is the performance of the system in the given load.
Example: Checking several processor cycles.

10. Object-Oriented Testing
This testing is a combination of various testing techniques that help to verify and validate object-oriented software. This testing is done in the following manner:

- Testing of Requirements,
- Design and Analysis of Testing,
- Testing of Code,
- Integration testing,
- System testing,
- User Testing.

3. Differences between Black Box Testing vs White Box Testing

1. Black Box Testing is a software testing method in which the internal structure/design/implementation of the item being tested is not known to the tester. Only the external design and structure are tested.

2. White Box Testing is a software testing method in which the internal structure/design/implementation of the item being tested is known to the tester. Implementation and impact of the code are tested.

Differences between Black Box Testing vs White Box Testing:

S. No.	Black Box Testing	White Box Testing
1.	It is a way of software testing in which the **internal structure or the program or the code** is hidden and nothing is known about it.	It is a way of testing the software in which **the tester has knowledge about the internal structure or the code or the program of the software**.
2.	**Implementation of code is not needed** for black box testing.	**Code implementation is necessary** for white box testing.
3.	It is mostly done by **software testers**.	It is mostly done by **software developers.**
4.	**No knowledge of implementation** is needed.	**Knowledge of implementation** is required.
5.	It can be referred to as **outer or external software testing**.	It is **the inner or the internal software testing.**

S. No.	Black Box Testing	White Box Testing
6.	It is **a functional test** of the software.	It is **a structural test** of the software.
7.	This testing can be initiated based on the **requirement specifications document.**	This type of testing of software is started **after a detail design document**.
8.	**No knowledge of programming** is required.	It is **mandatory** to have knowledge of programming.
9.	It is the **behavior testing** of the software.	It is the **logic testing** of the software.
10.	It is applicable to **the higher levels** of testing of software.	It is generally applicable to **the lower levels** of software testing.
11.	It is also called **closed testing.**	It is also called as **clear box testing**.
12.	It is **least time consuming**.	It is **most time consuming.**
13.	It is **not suitable or preferred for algorithm** testing.	It is **suitable** for algorithm testing.
14.	Can be done by trial and error	Data domains along with inner or

S. No.	Black Box Testing	White Box Testing
	ways and methods.	internal boundaries can be better tested.
15.	**Example:** Search something on Google by using keywords	**Example:** By input to check and verify loops
16.	**Black-box test design techniques-** • Decision table testing • All-pairs testing • Equivalence partitioning • Error guessing	**White-box test design techniques-** • Control flow testing • Data flow testing • Branch testing
17.	**Types of Black Box Testing:** • Functional Testing • Non-functional testing • Regression Testing	**Types of White Box Testing:** • Path Testing • Loop Testing • Condition testing
18.	It **is less exhaustive** as compared to white box testing.	It is comparatively **more exhaustive** than black box testing.

4. Describe different requirement validation checks

Requirements validation is the process of checking that requirements defined for development, define the system that the customer really wants. To check issues related to requirements, we perform requirements validation. We usually use requirements validation to check error at the initial phase of development as the error may increase excessive rework when detected later in the development process.

In the requirements validation process, we perform a different type of test to check the Software Requirements Specification (SRS) :

- Completeness checks
- Consistency checks
- Validity checks
- Realism checks
- Ambiguity checks
- Verifiability

The output of requirements validation is the list of problems and agreed on actions of detected problems. The lists of problems indicate the problem detected during the process of requirement validation. The list of agreed action states the corrective action that should be taken to fix the detected problem.

There are several techniques which are used either individually or in conjunction with other techniques to check to check entire or part of the system:

a. Test case generation:
Requirement mentioned in SRS document should be testable, the conducted tests reveal the error present in the requirement. It is generally believed that if the test is difficult or impossible to design than, this usually means that requirement will be difficult to implement and it should be reconsidered.

b. Prototyping:
In this validation techniques the prototype of the system is presented before the end-user or customer, they experiment with the presented model and check if it meets their need. This type of model is generally used to collect feedback about the requirement of the user.

c. Requirements Reviews:
In this approach, the SRS is carefully reviewed by a group of people including people from both the contractor organisations and the client side, the reviewer systematically analyses the document to check error and ambiguity.

d. Automated Consistency Analysis:
This approach is used for automatic detection of an error, such as non_determinism, missing cases, a type error, and circular definitions, in requirements specifications.

First, the requirement is structured in formal notation then CASE tool is used to check in-consistency of the system, The report of all inconsistencies is identified and corrective actions are taken.

e. **Walk-through:**

A walkthrough does not have a formally defined procedure and does not require a differentiated role assignment.

- Checking early whether the idea is feasible or not.
- Obtaining the opinions and suggestion of other people.
- Checking the approval of others and reaching an agreement.

Principles of Requirements Validation:

Considering the following six principles of requirements validation increases the quality of the validation results:

- **Principle 1:** Involvement of the correct stakeholders
- **Principle 2:** Separating the identification and the correction of errors
- **Principle 3:** Validation from different views
- **Principle 4:** Adequate change of documentation type
- **Principle 5:** Construction of development artifacts
- **Principle 6:** Repeated validation.

Validation inputs and outputs

Requirements document

Organisational knowledge

Organisationalstandards

Validation inputs

➢ Requirements document

Should be a complete version of the document, not an unfinished draft. Formatted and organized according to organizational standards

➢ Organizational knowledge

Knowledge, often implicit, of the organization which may be used to judge the realism of the requirements

- Organizational standards

  Local standards e.g. for the organization of the requirements document

Validation outputs

- Problem list

  List of discovered problems in the requirements document

- Agreed actions

  List of agreed actions in response to requirements problems. Some problems may have several corrective actions; some problems may have no associated actions

5. Describe clean room software development process with its advantages

Software Engineering-The Cleanroom Approach

The philosophy of the "cleanroom" in hardware fabrication technologies is really quite simple: **It is cost-effective and time-effective** to establish a fabrication approach that precludes the introduction of product defects. Rather than fabricating a product and then working to remove defects, the cleanroom approach demands the discipline required to eliminate defects in specification and design and then fabricate in a **"clean"** manner.

The cleanroom philosophy was first proposed for software engineering by Mills, Dyer, and Linger during the 1980s. Although early experiences with this disciplined approach to software work showed significant promise, it has not gained widespread usage.

Henderson suggests three possible reasons:

1. A belief that the cleanroom methodology is too theoretical, too mathematical, and too radical for use in real software development.

2. It advocates no unit testing by developers but instead replaces it with correctness verification and statistical quality control—concepts that represent a major departure from the way most software is developed today.

3. The maturity of the software development industry. The use of cleanroom processes requires rigorous application of defined processes in all life cycle phases. Since most of the industry is still operating at the adhoc level (as defined by the Software Engineering Institute Capability Maturity Model), the industry has not been ready to apply those techniques.

Dyer alludes to the differences of the cleanroom approach when he defines the process:

Cleanroom represents the first practical attempt at putting the software development process under statistical quality control with a well-defined strategy for continuous process improvement. To reach this goal, a cleanroom unique life cycle was defined which focused on mathematics- based software engineering for correct software designs and on statistics-based software testing for certification of software reliability.

1. It makes explicit use of statistical quality control.
2. It verifies design specification using a mathematically based proof of correctness.
3. It relies heavily on statistical use testing to uncover high-impact errors.

In cleanroom software engineering, unit testing and debugging are replaced by correctness verification and statistically based testing. These activities, coupled with the record keeping necessary for continuous improvement, make the cleanroom approach unique.

Clean room software engineering is a software development approach to producing quality software. It is different from classical software engineering as in classical software engineering QA (Quality Assurance) is the last phase of development that occurs at the completion of all development stages while there is a chance of less reliable and fewer quality products full of bugs, and errors and upset client, etc. But in clean room software engineering, an efficient and good quality software product is delivered to the client as QA (Quality Assurance) is performed each and every phase of software development.

The cleanroom software engineering follows a quality approach to software development which follows a set of principles and practices for gathering requirements, designing, coding, testing, managing, etc. which not only improves the quality of the product but also increases productivity and reduces development cost. From the beginning of the system development to the completion of system development it emphasizes removing the dependency on the costly processes and preventing defects during development rather than removing the defects.

The clean room approach was developed by Dr. Harlan Mills of IBM's Federal Systems Division, and it was released in the year 1981 but got popularity after 1987 when IBM and other organizations started using it.

Processes of Cleanroom development :
Clean room software development approaches consist of four key processes i.e.

1. **Management –**
   It is persistent throughout the whole project lifetime which consists of project mission, schedule, resources, risk analysis, training, configuration management, etc.
2. **Specification –**
   It is considered the first process of each increment which consists of requirement analysis, function specification, usage specification, increment planning, etc.
3. **Development –**
   It is considered the second process of each increment which consists of software reengineering, correctness verification, incremental design, etc.
4. **Certification –**
   It is considered the final process of each increment which consists of usage

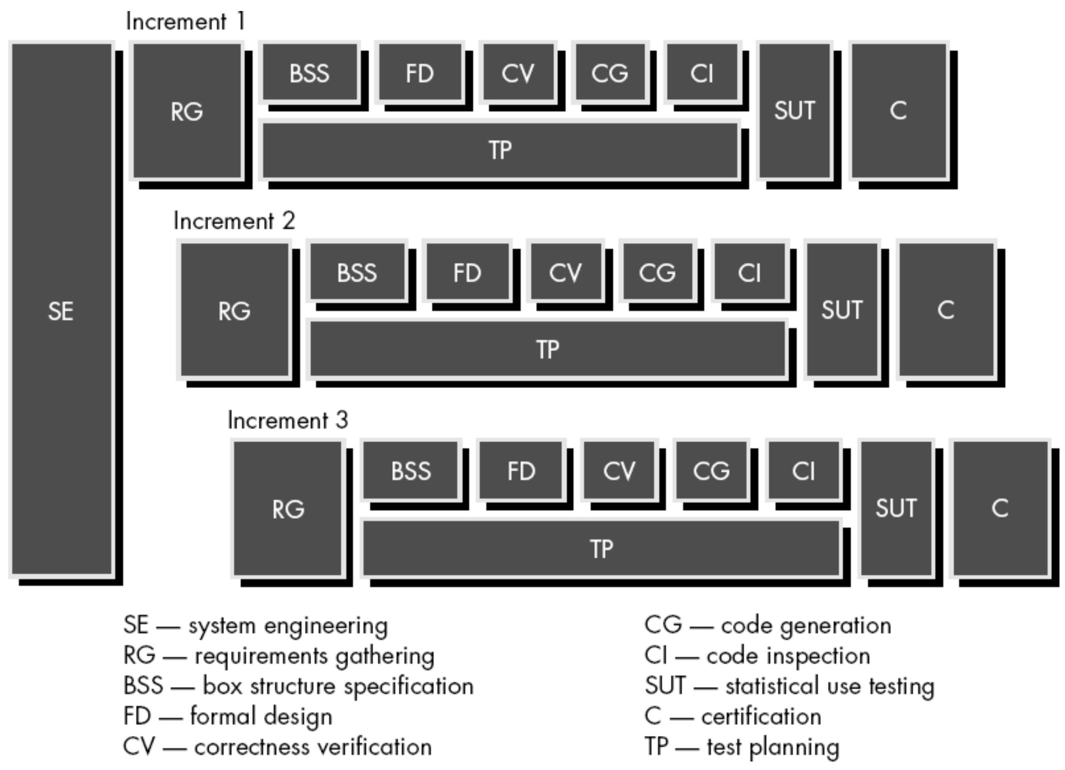
modeling and test planning, statistical training and certification process, etc.

While separate teams are allocated for different processes to ensure the development of the highest quality software product.

The Cleanroom Strategy

The cleanroom approach makes use of a specialized version of **the incremental software model** . A “pipeline of software increments” is developed by small independent software engineering teams. As each increment is certified, it is integrated in the whole. Hence, functionality of the system grows with time.

The sequence of cleanroom tasks for each increment is illustrated in figure.



Overall system or product requirements are developed using the system engineering methods . Once functionality has been assigned to the software element of the system, the pipeline of cleanroom increments is initiated.

The following tasks occur:

- **Increment planning**

  A project plan that adopts the incremental strategy is developed. The functionality of each increment, its projected size, and a cleanroom development schedule are created. Special care must be taken to ensure that certified increments will be integrated in a timely manner.

- **Requirements gathering**
  Using techniques , a more-detailed description of customer-level requirements (for each increment) is developed.

- **Box structure specification.**
  A specification method that makes use of box structures is used to describe the functional specification. Conforming to the operational analysis principles, box structures "isolate and separate the creative definition of behavior, data, and procedures at each level of refinement."

It generally uses three types of boxes i.e.

1. **Black box –**
   It identifies the behavior of the system.
2. **State box –**
   It identifies state data or operations.
3. **Clear box –**
   It identifies the transition function used by the state box.

- **Formal design.**
  Using the box structure approach, cleanroom design is a natural and seamless extension of specification. Although it is possible to make a

clear distinction between the two activities, specifications (called black boxes) are iteratively refined (within an increment) to become analogous to architectural and component-level designs (called stateboxes and clear boxes, respectively).

- **Correctness verification.**

  The cleanroom team conducts a series of rigorous correctness verification activities on the design and then the code. Verification begins with the highest-level box structure (specification) and moves toward design detail and code. The first level of correctness verification occurs by applying a set of "correctness questions" . If these do not demonstrate that the specification is correct, more formal (mathematical) methods for verification are used.

- **Code generation, inspection, and verification.**

  The box structure specifications, represented in a specialized language, are translated into the appropriate programming language. Standard walkthrough or inspection techniques are then used to ensure semantic conformance of the code and box structures and syntactic correctness of the code. Then correctness verification is conducted for the source code.

- **Statistical test planning.**

  The projected usage of the software is analyzed and a suite of test cases that exercise a "probability distribution" of usage are planned and designed. **Referring to figure,** this cleanroom activity is conducted in parallel with specification, verification, and code generation.

- **Statistical use testing.**

  Recalling that exhaustive testing of computer software is impossible, it is always necessary to design a finite number of test cases. Statistical use techniques execute a series of tests derived from a statistical sample of all possible program executions by all users from a targeted population.

- **Certification.**

  Once verification, inspection, and usage testing have been completed (and all errors are corrected), the increment is certified as ready for integration.

Benefits of Clean Room Software engineering :

- Delivers high-quality products.
- Increases productivity.
- Reduces development cost.
- Errors are found early.
- Reduces the overall project time.
- Saves resources.

Clean room software engineering ensures good quality software with certified reliability and for that only it has been incorporated into many new software practices. Still, according to the IT industry experts, it is not very adaptable as it is very theoretical and includes too mathematical to use in the real world. But they consider it as a future technology for the IT industries.

6. Write a note on different types of software maintenance

Software Maintenance is the process of modifying a software product after it has been delivered to the customer. The main purpose of software maintenance is to modify and update software applications after delivery to correct faults and to improve performance.

Software maintenance is the process of changing, modifying, and updating software to keep up with customer needs. Software maintenance is done after the product has launched for several reasons including improving the software overall, correcting issues or bugs, to boost performance, and more.

Software maintenance is also an important part of the Software Development Life Cycle(SDLC). To update the software application and do all modifications in software application so as to improve performance is the main focus of software maintenance. Software is a model that runs on the basis of real world. so, whenever any change requires in the software that means the need of real world changes wherever possible.

Need for Maintenance –
Software Maintenance must be performed in order to:

- Correct faults.
- Improve the design.
- Implement enhancements.
- Interface with other systems.
- Accommodate programs so that different hardware, software, system features, and telecommunications facilities can be used.
- Migrate legacy software.
- Retire software.
- Requirement of user changes.
- Run the code fast

why modifications are required, some of them are briefly mentioned below:

- **Market Conditions** - Policies, which changes over the time, such as taxation and newly introduced constraints like, how to maintain bookkeeping, may trigger need for modification.
- **Client Requirements** - Over the time, customer may ask for new features or functions in the software.
- **Host Modifications** - If any of the hardware and/or platform (such as operating system) of the target host changes, software changes are needed to keep adaptability.
- **Organization Changes** - If there is any business level change at client end, such as reduction of organization strength, acquiring another company, organization venturing into new business, need to modify in the original software may arise.

Why is software maintenance important?

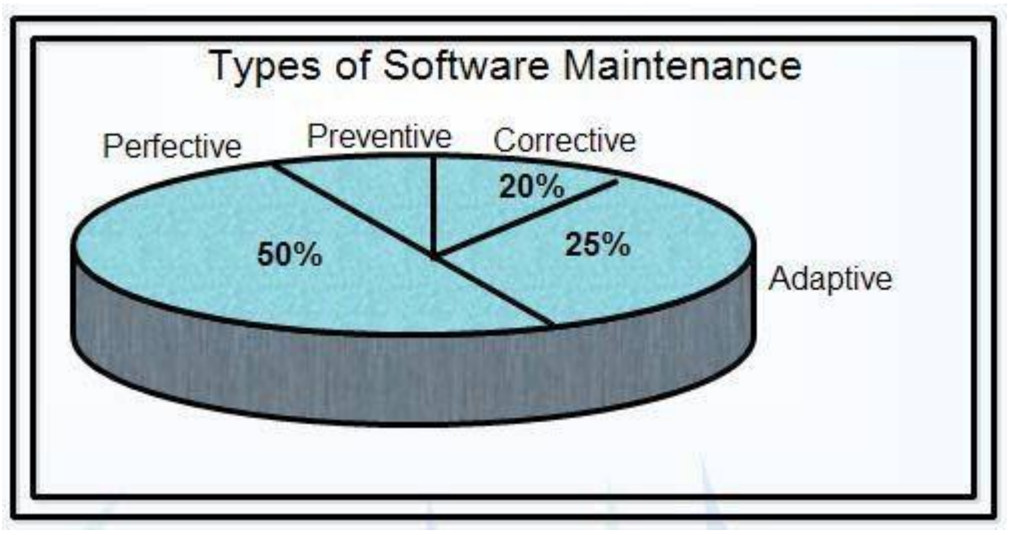
- To fix the bugs and errors in the software system.
- To improve the functionality of the software to make your product more compatible with the latest marketing and business environments.
- To remove out-dated functions from your software that is inhibiting your product efficiency.
- To improve your software performance.

What are the 4 types of software maintenance?

The four different types of software maintenance are each performed for different reasons and purposes. A given piece of software may have to undergo one, two, or all types of maintenance throughout its lifespan.

The four types are:

1. Corrective Software Maintenance
2. Preventative Software Maintenance
3. Perfective Software Maintenance
4. Adaptive Software Maintenance



1. Corrective Software Maintenance

Defect in the software arises due to errors and faults in design, logic, and code of the software. Corrective maintenance action (commonly referred to as "bug fixing") addresses these errors and faults in your software system.

Corrective changes in the software are required when:

- Software is not working the way it is expected due to some acute issues, such as faulty logic flow, incorrect implementation, invalid or incomplete tests, etc.
- The issues in the software are affecting users after you have released the software.

Most commonly, bug reports are created by users and sent as feedback to the company that designed the software. Then the company’s developers and testers review the code and make corrective changes to the software accordingly. Its purpose is to enhance and improve the software.

If you detect and resolve the flaws in the software before users discover it, then the maintenance action is preventive or adaptive. However, if you fix the problem after getting bug reports from the user’s side, then it is corrective maintenance action.

If you are spending the majority of time dealing with corrective maintenance task then pay attention to the following:-

- Adopt a robust testing practice.
- Develop high-quality code.
- Focus on the correct implementation of the design specification.
- Enhance your ability to anticipate the problems.

(Note: definition-

Corrective maintenance of a software product may be essential either to rectify some bugs observed while the system is in use, or to enhance the performance of the system.)

2. Preventative Software Maintenance

Preventive maintenance is a software change you make to prevent the occurrence of errors in the future. It increases the software maintainability by reducing its complexity. Preventive maintenance task include:

1. **Updating the documentation**: Updating the document according to the current state of the system.
2. **Optimizing the code**: Modifying the code for faster execution of programs or making efficient use of storage space.

3. **Reconstructing the code**: Transforming the structure of the program by reducing the source code, making it easily understandable.

Preventative software maintenance is looking into the future so that your software can keep working as desired for as long as possible.

This includes making necessary changes, upgrades, adaptations and more. Preventative software maintenance may address small issues which at the given time may lack significance but may turn into larger problems in the future. These are called latent faults which need to be detected and corrected to make sure that they won't turn into effective faults.

3. Perfective Software Maintenance

Perfective software maintenance is performed when you update the software system to improve its value, according to user demands. This includes:

- Speed optimization
- Improvement in user interfaces
- Improvements in software usability
- Enhancement of software functionality
- Improvement in software performance

Some key points about Perfective software maintenance:

- It involves making enhancements in software functionality by implementing new or changed user requirements (even when the changes are not considered a defect, error or fault).
- It is often, but not always, initiated by customer feedback.
- It accounts for 50% of all the maintenance activities.

Examples of perfective maintenance include modifying an ERP (Enterprise Resource Planning) program to include a new payment settlement feature in a software system.

As with any product on the market, once the software is released to the public, new issues and ideas come to the surface. Users may see the need for new features or requirements that they would like to see in the software to make it the best tool available for their needs. This is when perfective software maintenance comes into play.

Perfective software maintenance aims to adjust software by adding new features as necessary and removing features that are irrelevant or not effective in the given software. This process keeps software relevant as the market, and user needs, change.

4. Adaptive Software Maintenance

Adaptive software maintenance has to do with the changing technologies as well as policies and rules regarding your software. These include operating system changes, cloud storage, hardware, etc. When these changes are performed, your software must adapt in order to properly meet new requirements and continue to run well.

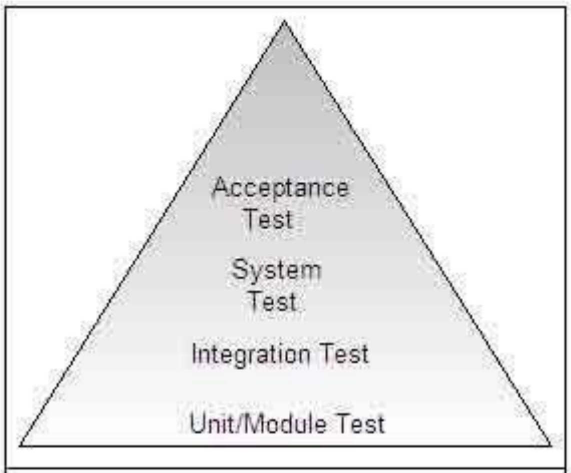
Adaptive maintenance aims at updating and modifying the software when:

- The platform in which your software operates is changing (due to technology, laws, policies, rules, operating system,etc.)
- Your customers need the product to interface with new hardware or software.
- You have predicted defects in the software that will affect your customers in the future.

Note : definition:

Adaptive maintenance is the implementation of changes in a part of the system, which has been affected by a change that occurred in some other part of the system. Adaptive maintenance consists of adapting software to changes in the environment such as the hardware or the operating system. The term environment in this context refers to the conditions and the influences which act (from outside) on the system. For example, business rules, work patterns, and government policies have a significant impact on the software system.)

7. Explain different levels of testing



Levels of Software Testing

The software is tested at different levels. Initially, the individual units are tested arid once they are tested; they are integrated and checked for interfaces established between them. After this, the entire software is tested to ensure that the output produced is according to user requirements. There are four levels of software testing, namely, unit testing, integration testing, system testing, and acceptance testing.

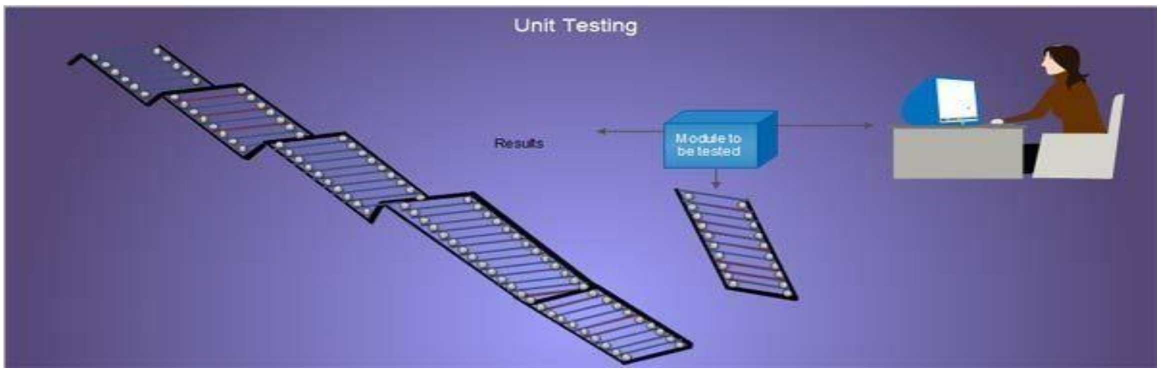
The software is tested at different levels. Initially, the individual units are tested arid once they are tested, they are integrated and checked for interfaces established between them. After this, the entire software is tested to ensure that the output produced is according to user requirements. There are four levels of software testing, namely, unit testing, integration testing, system testing, and acceptance testing.

Levels of Software Testing

- Unit Testing
- Integration Testing
- System Testing
- Acceptance Testing

i) Unit Testing

Unit testing is performed to test the individual units of software. Since the software comprises various units/modules, detecting errors in these units is simple and consumes less time, as they are small in size. However, it is possible that the outputs produced by one unit become input for another unit. Hence, if incorrect output produced by one unit is provided as input to the second unit then it also produces wrong output. If this process is not corrected, the entire software may produce unexpected outputs. To avoid this, all the units in the software are tested independently using unit testing.



Unit testing is not just performed once during the software development, but repeated whenever the software is modified or used in a new environment. Some other points noted about unit testing are listed below.

- Each unit is tested separately regardless of other units of software.
- The developers themselves perform this testing.
- The methods of white box testing are used in this testing.

Unit testing is used to verify the code produced during software coding and is responsible for assessing the correctness of a particular unit of source code.

In addition, unit testing performs the following functions.

- It tests all control paths to uncover maximum errors that occur during the execution of conditions present in the unit being tested.
- It ensures that all statements in the unit have been executed at least once.
- It tests data structures (like stacks, queues) that represent relationships among individual data elements.
- It checks the range of inputs given to units. This is because every input range has a maximum and minimum value and the input given should be within the range of these values.
- It ensures that the data entered in variables is of the same data type as defined in the unit.
- It checks all arithmetic calculations present in the unit with all possible combinations of input values.

Unit testing is performed by conducting a number of unit tests where each unit test checks an individual component that is either new or modified. A unit test is also referred to as a module test as it examines the individual units of code that constitute the program and eventually the software. In a conventional structured programming language such as C, the basic unit is a function or subroutine while in object-oriented language such as C++, the basic unit is a class.

Various tests that are performed as a part of unit tests are listed below.

a) Module interface: This is tested to check whether information flows in a proper manner in and out of the 'unit' being tested. Note that test of data-flow (across a module interface) is required before any other test is initiated.
b) Local data structure: This is tested to check whether temporarily stored data maintains its integrity while an algorithm is being executed.
c) Boundary conditions: These are tested to check whether the module provides the desired functionality within the specified boundaries.
d) Independent paths: These are tested to check whether all statements in a module are executed at least once. Note that in this testing, the entire control structure should be exercised.
e) Error-handling paths: After successful completion of various tests, error handling paths are tested.

Various unit test cases are generated to perform unit testing. Test cases are designed to uncover errors that occur due to erroneous computations, incorrect comparisons, and improper control flow. A proper unit test case ensures that unit testing is performed efficiently. To develop test cases, the following points should always be considered.

a) Expected functionality: A test case is created for testing all functionalities present in the unit being tested. For example, an SQL query that creates Table_A and alters Table_B is used. Then, a test case is developed to make sure that Table_A is created and Table_B is altered.
b) Input values: Test cases are developed to check various aspects of inputs, which are discussed here.
c) Every input value: A test case is developed to check every input value, which is accepted by the unit being tested. For example, if a program is developed to print a table of five, then a test case is developed which verifies that only five is entered as input.
d) Validation of input: Before executing software, it is important to verify whether all inputs are valid. For this purpose, a test case is developed

which verifies the validation of all inputs. For example, if a numeric field accepts only positive values, then a test case is developed to verify that the numeric field is accepting only positive values.

e) Boundary conditions: Generally, software fails at the boundaries of input domain (maximum and minimum value of the input domain). Thus, a test case is developed, which is capable of detecting errors that caused the software to fail at the boundaries of input domain. For example, errors may occur while processing the last element of an array. In this case, a test case is developed to check if an error occurs while processing the last element of the array.

f) Limitation of data types: Variable that holds data types has certain limitations. For example, if a variable with data type long is executed then a test case is developed to ensure that the input entered for the variable is within the acceptable limit of long data type.

g) Output values: A test case is designed to determine whether the desired output is produced by the unit. For example, when two numbers, '2' and '3' are entered as input in a program that multiplies two numbers, a test case is developed to verify that the program produces the correct output value, that is, '6'.

h) Path coverage: There can be many conditions specified in a unit. For executing all these conditions, many paths have to be traversed. For example, when a unit consists of tested 'if statements and all of them are to be executed, then a test case is developed to check whether all these paths are traversed.

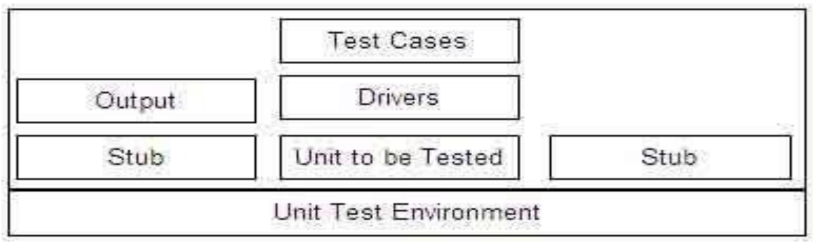
i) Assumptions: For a unit to execute properly, certain assumptions are made. Test cases are developed by considering these assumptions. For example, a test case is developed to determine whether the unit generates errors in case the assumptions are not met.

j) Abnormal terminations: A test case is developed to check the behavior of a unit in case of abnormal termination. For example, when a power cut results in termination of a program due to shutting down of the computer, a test case is developed to check the behavior of a unit as a result of abnormal termination of the program.

k) Error messages: Error messages that appear when the software is executed should be short, precise, self-explanatory, and free from grammatical mistakes. For example, if 'print' command is given when a printer is not installed, error message that appears should be 'Printer not installed' instead of 'Problem has occurred as no printer is installed and hence unable to print'. In this case, a test case is developed to check whether the error message displayed is according to the condition occurring in the software.

Unit tests can be designed before coding begins or after the code is developed. Review of this design information guides the creation of test cases, which are used to detect errors in various units. Since a component is not an independent program, the drivers and/ or stubs are used to test the units independently. Driver is a module that takes input from test case, passes this input to the unit to be tested and prints the output produced. Stub is a module that works as a unit referenced by the unit being tested. It uses the interface of the subordinate unit, does minimum data manipulation, and returns control back to the unit being tested.

Unit Test Environment

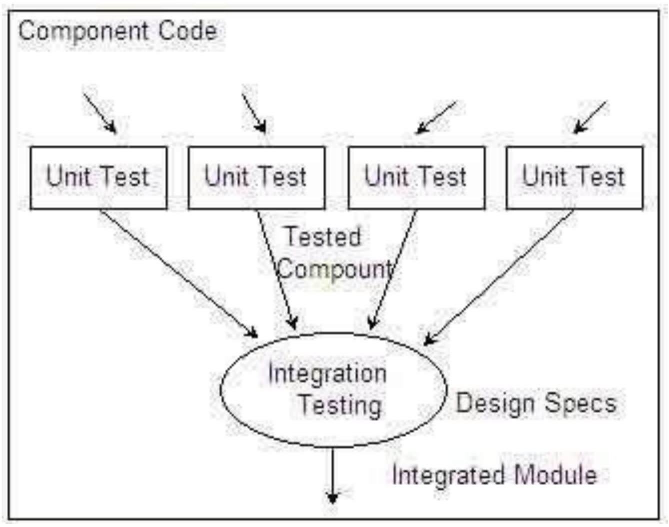


Note: Drivers and stubs are not delivered with the final software product. Thus, they represent an overhead.

ii) Integration Testing

Once unit testing is complete, integration testing begins. In integration testing, the units validated during unit testing are combined to form a subsystem. The integration testing is aimed at ensuring that all the modules work properly as per the user requirements when they are put together (that is, integrated).

The objective of integration testing is to take all the tested individual modules, integrate them, test them again, and develop the software, which is according to design specifications.



- It ensures that all modules work together properly and transfer accurate data across their interfaces.
- It is performed with an intention to uncover errors that lie in the interfaces among the integrated components.
- It tests those components that are new or have been modified or affected due to a change.

The big bang approach and incremental integration approach are used to integrate modules of a program. In the big bang approach, initially all modules are integrated and then the entire program is tested. However, when the entire program is tested, it is possible that a set of errors is detected. It is difficult to correct these errors since it is difficult to isolate the exact cause of the errors when the program is very large. In addition, when one set of errors is corrected, new sets of errors arise and this process continues indefinitely.

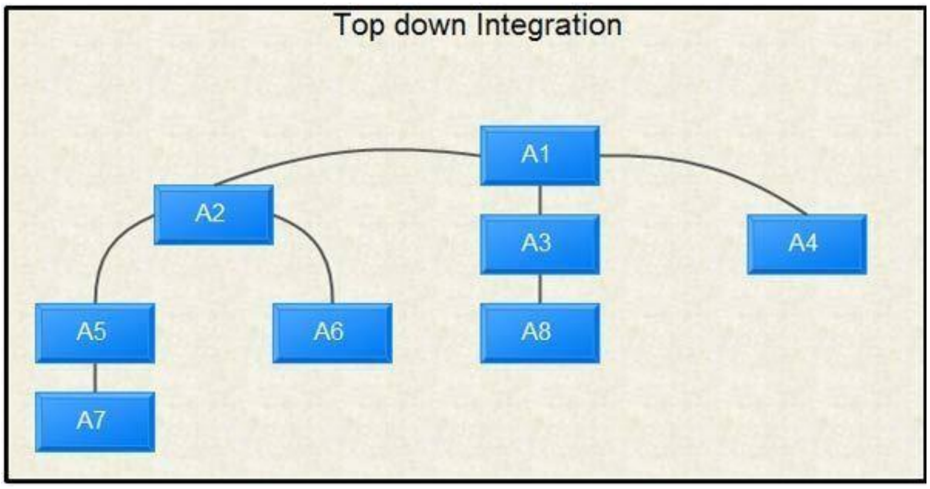
To overcome this problem, incremental integration is followed. The incremental integration approach tests program in small increments. It is easier to detect errors in this approach because only a small segment of software code is tested at a given instance of time. Moreover, interfaces can be tested completely if this approach is used. Various kinds of approaches are used for performing incremental integration testing, namely, top-down integration testing, bottom-up integration testing, regression testing, and smoke testing.

In this testing, the software is developed and tested by integrating the individual modules, moving downwards in the control hierarchy. In top-down integration testing, initially only one module known as the main control module is tested. After this, all the modules called by it are combined with it and tested. This process continues till all the modules in the software are integrated and tested.

It is also possible that a module being tested calls some of its subordinate modules. To simulate the activity of these subordinate modules, a stub is written. Stub replaces modules that are subordinate to the module being tested. Once the control is passed to the stub, it manipulates the data as least as possible, verifies the entry, and passes the control back to the module under test. To perform top-down integration testing, the following steps are used.

- The main control module is used as a test driver and all the modules that are directly subordinate to the main control module are replaced with stubs.
- The subordinate stubs are then replaced with actual modules, one stub at a time. The way of replacing stubs with modules depends on the approach (depth first or breadth first) used for integration.
- As each new module is integrated, tests are conducted.
- After each set of tests is complete, its time to replace another stub with actual module.
- In order to ensure no new errors have been introduced, regression testing may be performed.

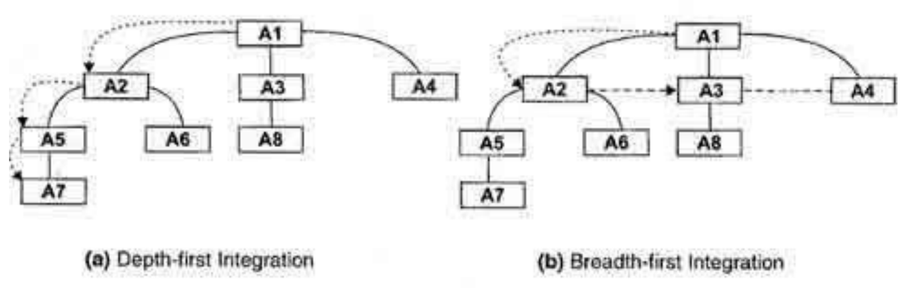
Top-down Integration



Top-down integration testing uses either depth-first integration or breadth-first integration for integrating the modules. In depth-first integration, the modules are integrated starting from left and then move down in the control hierarchy. Initially, modules A1, A2, A5 and A7 are integrated. Then, module A6 integrates with module A2. After this, control moves to the modules present at the centre of control hierarchy, that is, module A3integrates with module Al and then module A8 integrates with module A3. Finally, the control moves towards right, integrating module A4 with module A1.Top-down Integration

In breadth-first integration, initially all modules at the first level are integrated moving downwards, integrating all modules at the next lower levels. Initially modules A2, A3, and A4 are integrated with module Al and then it moves down integrating modules A5 and A6with module A2and module A8 with module A3.

Finally, module A7 is integrated with module A5.

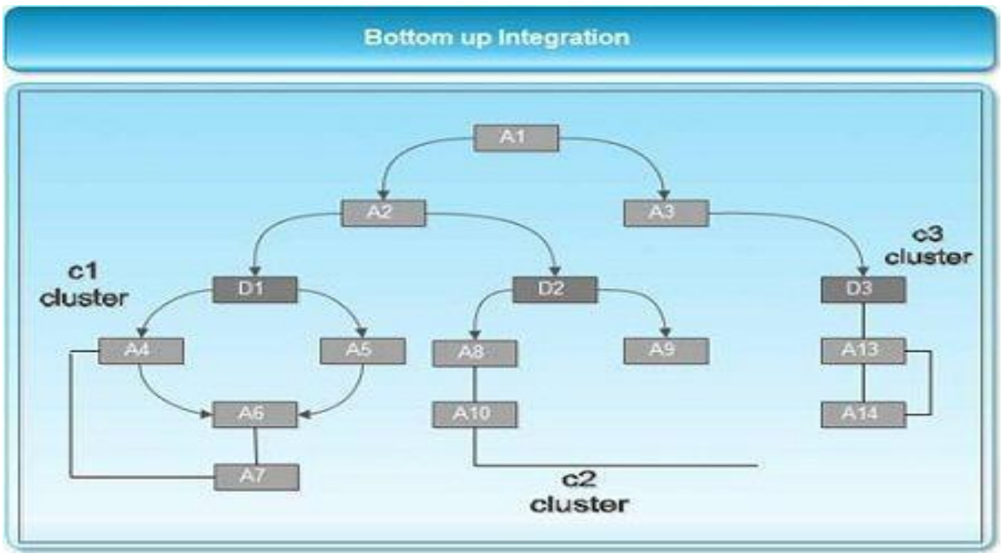


Top-down Integration

Advantages and Disadvantages of Top-down Integration

Advantages	**Disadvantages**
▪ Behavior of modules at high level is verified early. ▪ None or only one driver is required. ▪ Modules can be added one at a time with each step. ▪ Supports both breadth-first method and depth-first method. ▪ Modules are tested in isolation from other modules.	▪ Delays the verification of behavior of modules present at lower levels. ▪ Large numbers of stubs are required in case the lowest level of software contains many functions. ▪ Since stubs replace modules present at lower levels in the control hierarchy, no data flows upward in program structure. To avoid this, tester has to delay many tests until stubs are replaced with actual modules or has to integrate software from the bottom of the control hierarchy moving upward. ▪ Module cannot be tested in isolation from other modules because it has to invoke other modules.

In this testing, individual modules are integrated starting from the bottom and then moving upwards in the hierarchy. That is, bottom-up integration testing combines and tests the modules present at the lower levels proceeding towards the modules present at higher levels of the control hierarchy.



Some of the low-level modules present in software are integrated to form clusters or builds (collection of modules). A test driver that coordinates the test case input and output is written and the clusters are tested. After the clusters have been tested, the test drivers are removed and the clusters are integrated, moving upwards in the control hierarchy.

The low-level modules A4, A5, A6, andA7 are combined to form cluster C1. Similarly, modulesA8, A9, A10, A11, andA12 are combined to form cluster C2. Finally, modules A13and A14are combined to form cluster C3. After clusters are formed, drivers are developed to test these clusters. Drivers Dl, D2, and D3 test clusters C1, C2, and C3 respectively. Once these clusters are tested, drivers are removed and clusters are integrated with the modules. Cluster el and cluster C2 are integrated with module A2. Similarly, cluster C3 is integrated with module A3. Finally, both the modules A2and A3are integrated with module Al.

Advantages and Disadvantages of Bottom-up Integration

Advantages	Disadvantages
▪ Behavior of modules at lower levels is verified earlier. ▪ No stubs are required. ▪ Uncovers errors that occur at the lower levels in software. ▪ Modules are tested in isolation from other modules.	▪ Delays in verification of modules at higher levels. ▪ Large numbers of drivers are required to test clusters.

Software undergoes changes every time a new module is integrated with the existing subsystem. Changes can occur in the control logic or input/output media, and so on. It is possible that new data-flow paths are established as a result of these changes, which may cause problems in the functioning of some parts of the software that was previously working perfectly. In addition, it is also possible that new errors may surface during the process of correcting existing errors. To avoid these problems, regression testing is used.

Regression testing 're-tests' the software or part of it to ensure that the components, features, and functions, which were previously working properly, do not fail as a result of the error correction process and integration of modules. It is regarded as an important activity as it helps in ensuring that changes (due to error correction or any other reason) do not result in additional errors or unexpected outputs from other system components.

To understand the need of regression testing, suppose an existing module has been modified or a new function is added to the software. These changes may result in errors in other modules of the software that were previously working properly. To illustrate this, consider the part of the code written below that is working properly.

```
x: = b + 1;

proc (z);
```

```
b: = x + 2; x: = 3;
```

Now suppose that in an attempt to optimize the code, it is transformed into the following.

```
proc (z);

b: = b + 3;

x: = 3;
```

This may result in an error if procedure proc accesses variable x. Thus, testing should be organized with the purpose of verifying possible degradations of correctness or other qualities due to later modifications. During regression testing, a subset of already defined test cases is re-executed on the changed software so that errors can be detected. Test cases for regression testing consist of three different types of tests, which are listed below.

- Tests that check all functions of the software.
- Tests that check the functions that can be affected due to changes.
- Tests that check the modified software modules.

Advantages and Disadvantages of Regression Testing

Advantages	**Disadvantages**
• Ensures that the unchanged parts of software work properly. • Ensures that all errors that have occurred in software due to modifications are corrected andare not affecting the working of software.	• Time consuming activity. • Considered to be expensive.

Smoke testing is defined as an approach of integration testing in which a subset of test cases designed to check the main functionality of software are used to test whether the vital functions of the software work correctly. This testing is best suitable for testing time-critical software as it permits the testers to evaluate the software frequently.

Smoke testing is performed when the software is finder development. As the modules of the software are developed, they are integrated to form a 'cluster'. After the cluster is formed, certain tests are designed to detect errors that prevent the cluster to perform its function. Next, the cluster is integrated with other clusters thereby leading to the development of the entire software, which is smoke tested frequently.

A smoke test should possess the following characteristics.

- It should run quickly.
- It should try to cover a large part of the software and if possible the entire software.
- It should be easy for testers to perform smoke testing on the software.
- It should be able to detect all errors present in the cluster being tested.
- It should try to find showstopper errors.

Generally, smoke testing is conducted every time a new cluster is developed and integrated with the existing cluster. Smoke testing takes minimum time to detect errors that occur due to integration of clusters. This reduces the risk associated with the occurrence of problems such as introduction of new errors in the software. A cluster cannot be sent for further testing unless smoke testing is performed on it. Thus, smoke testing determines whether the cluster is suitable to be sent for further testing.

Other benefits associated with smoke testing are listed below.

- Minimizes the risks, which are caused due to integration of different modules: Since smoke testing is performed frequently on the software, it allowsthe testers to uncover errors as early as possible, thereby reducing the chanceof causing severe impact on the schedule when there is delay in uncoveringerrors.

- Improves quality of the final software: Since smoke testing detects both functional and architectural errors as early as possible, they are corrected early, thereby resulting in a high-quality software.
- Simplifies detection and correction of errors: As smoke testing is performed almost every time a new code is added, it becomes clear that the probable cause of errors is the new code.
- Assesses progress easily: Since smoke testing is performed frequently, it keeps track of the continuous integration of modules, that is, the progress of software development. This boosts the morale of software developers.

Integration Test Documentation

To understand the overall procedure of software integration, a document known as test specification is prepared. This document provides information in the form of a test plan, test procedure, and actual test results.

The test specification document comprises the following sections.

- Scope of testing: Outlines the specific design, functional, and performance characteristics of the software that need to be tested. In addition, it describes the completion criteria for each test phase and keeps track of the constraints that occur in the schedule.
- Test plan: Describes the testing strategy to be used for integrating the software. Testing is classified into two parts, namely, phases and builds. Phases describe distinct tasks that involve various subtasks. On the other hand, builds are groups of modules that correspond to each phase. Some of the common test phases that require integration testing include user interaction, data manipulation and analysis, display outputs, database management, and so on. Every test phase consists of a functional category within the software. Generally, these phases can be related to a specific domain within the architecture of the software. The criteria commonly considered for all test phases include interface integrity, functional validity, information content, and performance.

In addition to test phases and builds, a test plan should also include the following.

- ➢ A schedule for integration, which specifies the 0tart and end date for each phase.
- ➢ A description of overhead software that focuses on the characteristics for which extra effort may be required.
- ➢ A description of the environment and resources required for testing.

- ✓ Test procedure 'n': Describes the order of integration and the corresponding unit tests for modules. Order of integration provides information about the purpose and the modules that are to be tested. Unit tests are performed for the developed modules along with the description of tests for these modules. In addition, test procedure describes the development of overhead software, expected results during integration testing, and description of test case data. The test environment and tools or techniques used for testing are also mentioned in a test procedure.
- ✓ Actual test results: Provides information about actual test results and problems that are recorded in the test report. With the help of this information, it is easy to carry out software maintenance.
- ✓ References: Describes the list of references that are used for preparing user documentation. Generally, references include books and websites.
- ✓ Appendices: Provides information about the test specification document. Appendices serve as a supplementary material that is provided at the end of the document.

System Testing

Software is integrated with other elements such as hardware, people, and database to form a computer-based system. This system is then checked for errors using system testing. IEEE defines system testing as 'a testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirement.'

In system testing, the system is tested against non-functional requirements such as accuracy, reliability, and speed. The main purpose is to validate and verify the functional design specifications and to check how integrated modules work together. The system testing also evaluates the system's interfaces to other applications and utilities as well as the operating environment.

During system testing, associations between objects (like fields), control and infrastructure, and the compatibility of the earlier released software versions with new versions are tested. System testing also tests some properties of the developed software, which are essential for users.

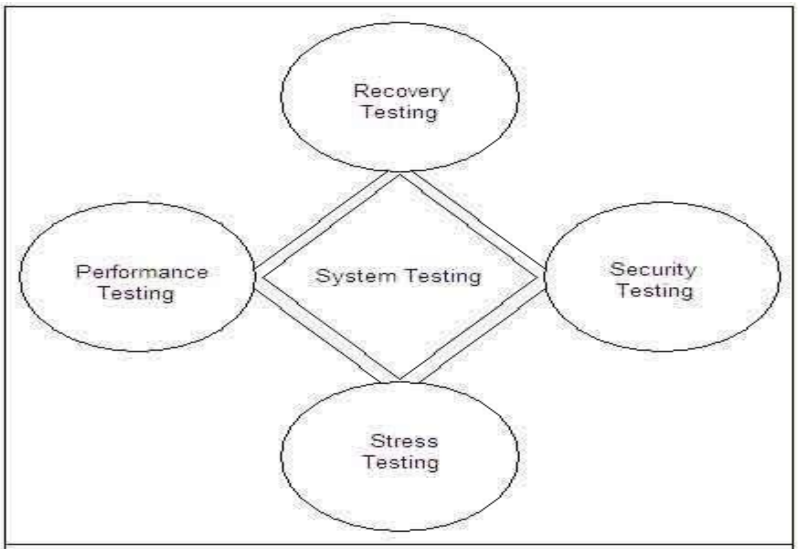
- ✓ Usable: Verifies that the developed software is easy to use and is understandable
- ✓ Secure: Verifies that access to important or sensitive data is restricted even for those individuals who have authority to use the software
- ✓ Compatible: Verifies that the developed software works correctly in conjunction with existing data, software and procedures
- ✓ Documented: Verifies that manuals that give information about the developed software are complete, accurate and understandable
- ✓ Recoverable: Verifies that there are adequate methods for recovery in case of failure.

System testing requires a series of tests to be conducted because software is only a component of computer-based system and finally it is to be integrated with other components such as information, people, and hardware. The test plan plays an important role in system testing as it describes the set of test cases to be executed, the order of performing different tests, and the required documentation for each test. During any test, if a defect or error is found, all the system tests that have already been executed must be re-executed after the repair has been made. This is required to ensure that the changes made during error correction do not lead to other problems.

While performing system testing, conformance tests and reviews can also be conducted to check the conformance of the application (in terms of inter-operability, compliance, and portability) with corporate or industry standards.

System testing is considered to be complete when the outputs produced by the software and the outputs expected by the user are either in line or the difference between the two is within permissible range specified by the user. Various kinds of testing performed as a part of system testing are recovery testing, security testing, stress testing, and performance testing.

Types of System Testing



Types of System Testing

Recovery testing is a type of system testing in which the system is forced to fail in different ways to check whether the software recovers from the failures without any data loss. The events that lead to failure include system crashes, hardware failures, unexpected loss of communication, and other catastrophic problems.

To recover from any type of failure, a system should be fault-tolerant. A fault tolerant system can be defined as a system, which continues to perform the intended functions even when errors are present in it. In case the system is not fault-tolerant, it needs to be corrected within a specified time limit after failure has occurred so that the software performs its functions in a desired manner.

Test cases generated for recovery testing not only show the presence of errors in a system, but also provide information about the data lost due to problems such as power failure and improper shutting down of computer system. Recovery testing also ensures that appropriate methods are used to restore the lost data. Other advantages of recovery testing are listed below.

- ✓ It checks whether the backup data is saved properly.
- ✓ It ensures that the backup data is stored in a secure location.
- ✓ It ensures that proper detail of recovery procedures is maintained.

Systems with sensitive information are generally the target of improper or illegal use. Therefore, protection mechanisms are required to restrict unauthorized access to the system. To avoid any improper usage, security testing is performed which identifies and removes the flaws from software (if any) that can be exploited by the intruders and thus, result in security violations. To find such kind of flaws, the tester like an intruder tries to penetrate the system by performing tasks such as cracking the password, attacking the system with custom software, intentionally producing errors in the system, etc.

The security testing focuses on the following areas of security.

- ✓ Application security: To check whether the user can access only those data and functions for which the system developer or user of system has given permission. This security is referred to as authorization.
- ✓ System security: To check whether only the users, who have permission to access the system, are accessing it. This security is referred to as authentication.

Generally, the disgruntled/dishonest employees or other individuals outside the organization make an attempt to gain unauthorized access to the system. If such people succeed in gaining access to the system, there is a possibility that a large amount of important data can be lost resulting in huge loss to the organization or individuals.

Security testing verifies that the system accomplishes all the security requirements and validates the effectiveness of these security measures. Other advantages associated with security testing are listed below.

- ✓ It determines whether proper techniques are used to identify security risks.
- ✓ It verifies that appropriate protection techniques are followed to secure the system.
- ✓ It ensures that the system is able to protect its data and maintain its functionality.
- ✓ It conducts tests to ensure that the implemented security measures are working properly.

Stress Testing

Stress testing is designed to determine the behavior of the software under abnormal situations. In this testing, the test cases are designed to execute the system in such a way that abnormal conditions arise. Some examples of test cases that may be designed for stress testing are listed below.

- ✓ Test cases that generate interrupts at a much higher rate than the average rate
- ✓ Test cases that demand excessive use of memory as well as other resources
- ✓ Test cases that cause 'thrashing' by causing excessive disk accessing.

IEEE defines stress testing as 'testing conducted to evaluate a system or component at or beyond the limits of its specified requirements.'

For example, if a software system is developed to execute 100 statements at a time, then stress testing may generate 110 statements to be executed. This load may increase until the software fails. Thus, stress testing specifies the way in which a system reacts when it is made to operate beyond its performance and capacity limits.

Some other advantages associated with stress testing are listed below.

- ✓ It indicates the expected behavior of a system when it reaches the extreme level of its capacity.
- ✓ It executes a system till it fails. This enables the testers to determine the difference between the expected operating conditions and the failure conditions.
- ✓ It determines the part of a system that leads to errors.
- ✓ It determines the amount of load that causes a system to fail.
- ✓ It evaluates a system at or beyond its specified limits of performance.

Performance Testing

Performance testing is designed to determine the performance of the software (especially real-time and embedded systems) at the run-time in the context of the entire computer-based system. It takes various performance factors like load, volume, and response time of the system into consideration and ensures that they are in accordance with the specifications. It also determines and informs the software developer about the current performance of the software under various parameters (like condition to complete the software within a specified time limit).

Often performance tests and stress tests are used together and require both software and hardware instrumentation of the system. By instrumenting a system, the tester can reveal conditions that may result in performance degradation or even failure of a system. While the performance tests are designed to assess the throughput, memory usage, response time, execution time, and device utilization of a system, the stress tests are designed to assess its robustness and error handling capabilities. Performance testing is used to test several factors that play an important role in improving the overall performance of the system.

Some of these factors are listed below.

- Speed: Refers to the extent how quickly a system is able to respond to its users. Performance testing verifies whether the response is quick enough.
- Scalability: Refers to the extent to which the system is able to handle the load given to it. Performance testing verifies whether the system is able to handle the load expected by users.
- Stability: Refers to the extent how long the system is able to prevent itself from failure. Performance testing verifies whether the system remains stable under expected and unexpected loads.

The outputs produced during performance testing are provided to the system developer. Based on these outputs, the developer makes changes to the system in order to remove the errors. This testing also checks the system characteristics such as its reliability.

Other advantages associated with performance testing are listed below.

- It assess whether a component or system complies with specified performance requirements.
- It compares different systems to determine which system performs better.

Acceptance Testing

Acceptance testing is performed to ensure that the functional, behavioral, and performance requirements of the software are met. IEEE defines acceptance testing as a 'formal testing with respect to user needs, requirements, and business processes conducted to determine whether or not a system satisfies the acceptance criteria and to enable the user, customers or other authorized entity to determine whether or not to accept the system.'

During acceptance testing, the software is tested and evaluated by a group of users either at the developer's site or user's site. This enables the users to test the software themselves and analyze whether it is meeting their requirements.

To perform acceptance testing, a predetermined set of data is given to the software as input. It is important to know the expected output before performing acceptance testing so that outputs produced by the software as a result of testing can be compared with them. Based on the results of tests, users decide whether to accept or reject the software. That is, if both outputs (expected and produced) match, the software is considered to be correct and is accepted; otherwise, it is rejected.

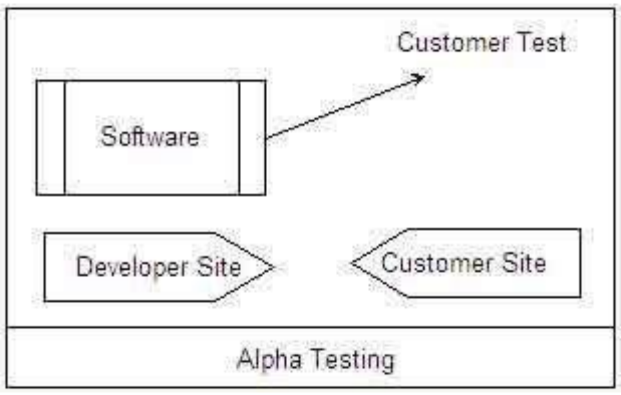
Advantages and Disadvantages of Acceptance Testing

Advantages	Disadvantages
▪ Gives user an opportunity toensure that software meets user requirements, before actually accepting it from the developer. ▪ Enables both users and software developers to identify and resolve problems in software. ▪ Determines the readiness (state of being ready to operate) of software to perform operations. ▪ Decreases the possibility ofsoftware failure to a large extent.	▪ The users may provide feedback without having proper knowledge of the software. ▪ Since users are not professional testers, they may not be able to either discover all software failures or accurately describe some failures

Since the software is intended for large number of users, it is not possible to perform acceptance testing with all the users. Therefore, organizations engaged in software development use alpha and beta testing as a process to detect errors by allowing a limited number of users to test the software.

Alpha testing

Alpha testing is considered as a form of internal acceptance testing in which the users test the software at the developer's site. In other words, this testing assesses the performance of the software in the environment in which it is developed. On completion of alpha testing, users report the errors to software developers so that they can correct them.

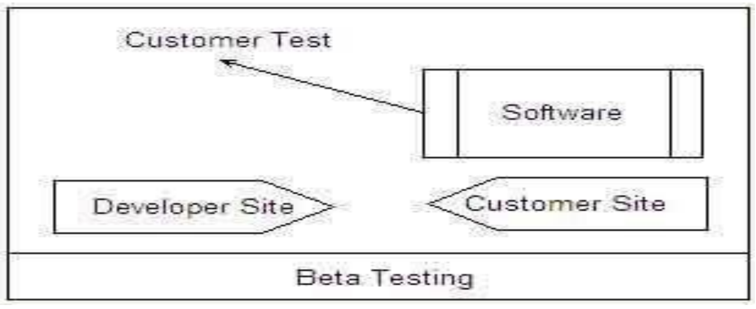


Some advantages of alpha testing are listed below.

- It identifies all the errors present in the software.
- It checks whether all the functions mentioned m the requirements are implemented properly in the software.

Beta Testing

Beta testing assesses the performance of the software at user's site. This testing is 'live' testing and is conducted in an environment, which is not controlled by the developer. That is, this testing is performed without any interference from the developer. Beta testing is performed to know whether the developed software satisfies user requirements and fits within the business processes.



Beta testing is considered as external acceptance testing which aims to get feedback from the potential customers. For this, the system and the limited public tests (known as beta versions) are made available to the groups of people or the open public (forgetting more feedback). These people test the software to detect any faults or bugs that may not have been detected by the developers and report their feedback. After acquiring the feedback, the system is modified and released either for sale or for further beta testing.

Some advantages of beta testing are listed below.

- It evaluates the entire documentation of the software. For example, it examines the detailed description of software code, which forms a part of documentation of the software.
- It checks whether the software is operating successfully in user environment.

8. Explain COCOMO model in details.

Boehm proposed COCOMO (Constructive Cost Estimation Model) in 1981.COCOMO is one of the most generally used software estimation models in the world. COCOMO predicts the efforts and schedule of a software product based on the size of the software.

The necessary steps in this model are:

1. Get an initial estimate of the development effort from evaluation of thousands of delivered lines of source code (KDLOC).
2. Determine a set of 15 multiplying factors from various attributes of the project.
3. Calculate the effort estimate by multiplying the initial estimate with all the multiplying factors i.e., multiply the values in step1 and step2.

The initial estimate (also called nominal estimate) is determined by an equation of the form used in the static single variable models, using KDLOC as the measure of the size. To determine the initial effort E_i in person-months the equation used is of the type is shown below

$$E_i=a*(KDLOC)b$$

The value of the constant a and b are depends on the project type.

In COCOMO, projects are categorized into three types:

1. Organic
2. Semidetached
3. Embedded

1.Organic: A development project can be treated of the organic type, if the project deals with developing a well-understood application program, the size of the development team is reasonably small, and the team members are experienced in developing similar methods of projects. **Examples of this type of projects are simple business systems, simple inventory management systems, and data processing systems.**

2. Semidetached: A development project can be treated with semidetached type if the development consists of a mixture of experienced and inexperienced staff. Team members may have finite experience in related systems but may be unfamiliar with some aspects of the order being developed. **Example of Semidetached system includes developing a new operating system (OS), a Database Management System (DBMS), and complex inventory management system.**

3. Embedded: A development project is treated to be of an embedded type, if the software being developed is strongly coupled to complex hardware, or if the stringent regulations on the operational method exist. **For Example:** ATM, Air Traffic control.

For three product categories, Bohem provides a different set of expression to predict effort (in a unit of person month)and development time from the size of estimation in KLOC(Kilo Line of code) efforts estimation takes into account the productivity loss due to holidays, weekly off, coffee breaks, etc.

According to Boehm, software cost estimation should be done through three stages:

1. **Basic Model**
2. **Intermediate Model**
3. **Detailed Model**

1. Basic COCOMO Model: The basic COCOMO model provide an accurate size of the project parameters. The following expressions give the basic COCOMO estimation model:

$Effort=a_1*(KLOC)\ a_2$ PM
$Tdev=b_1*(efforts)b_2$ Months

Where,

- **KLOC** is the estimated size of the software product indicate in Kilo Lines of Code,
- a_1,a_2,b_1,b_2 are constants for each group of software products,
- **Tdev** is the estimated time to develop the software, expressed in months,
- **Effort** is the total effort required to develop the software product, expressed in **person months (PMs)**.

Estimation of development effort

For the three classes of software products, the formulas for estimating the effort based on the code size are shown below:

Organic: Effort = 2.4(KLOC) 1.05 PM

Semi-detached: Effort = 3.0(KLOC) 1.12 PM

Embedded: Effort = 3.6(KLOC) 1.20 PM

Estimation of development time

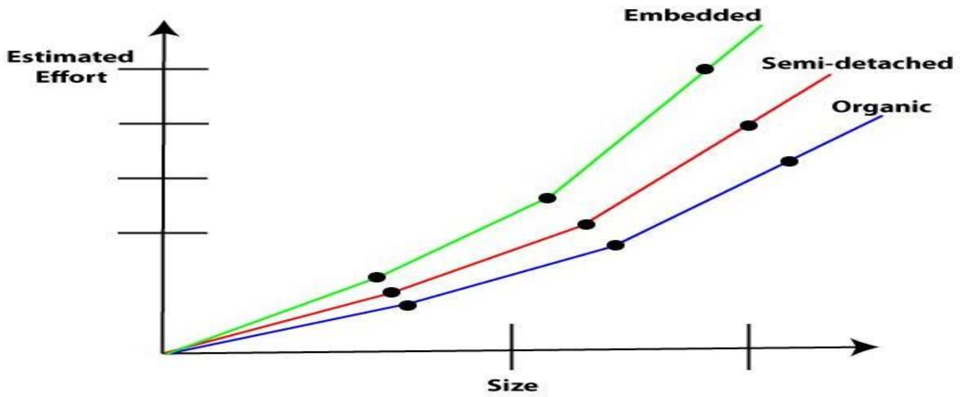
For the three classes of software products, the formulas for estimating the development time based on the effort are given below:

Organic: Tdev = 2.5(Effort) 0.38 Months

Semi-detached: Tdev = 2.5(Effort) 0.35 Months

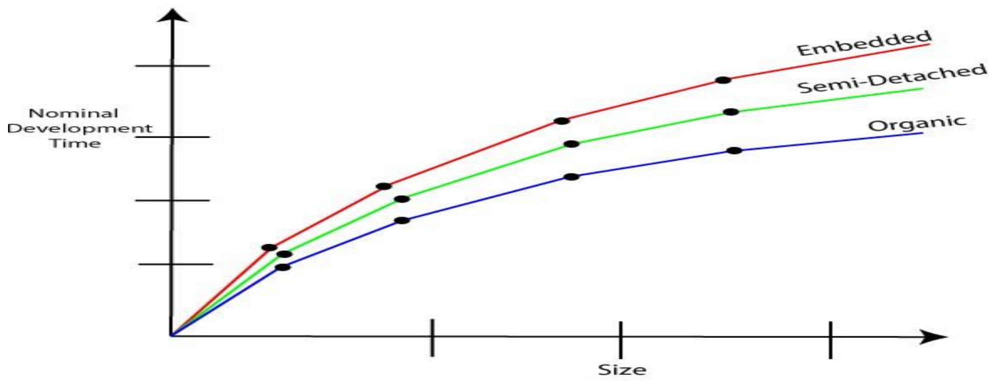
Embedded: Tdev = 2.5(Effort) 0.32 Months

Some insight into the basic COCOMO model can be obtained by plotting the estimated characteristics for different software sizes. Fig shows a plot of estimated effort versus product size. From fig, we can observe that the effort is somewhat superliner in the size of the software product. Thus, the effort required to develop a product increases very rapidly with project size.



Effort versus product size

The development time versus the product size in KLOC is plotted in fig. From fig it can be observed that the development time is a sub linear function of the size of the product, i.e. when the size of the product increases by two times, the time to develop the product does not double but rises moderately. This can be explained by the fact that for larger products, a larger number of activities which can be carried out concurrently can be identified. The parallel activities can be carried out simultaneously by the engineers. This reduces the time to complete the project. Further, from fig, it can be observed that the development time is roughly the same for all three categories of products. For example, a 60 KLOC program can be developed in approximately 18 months, regardless of whether it is of organic, semidetached, or embedded type.



Development time versus size

From the effort estimation, the project cost can be obtained by multiplying the required effort by the manpower cost per month. But, implicit in this project cost computation is the assumption that the entire project cost is incurred on account of the manpower cost alone. In addition to manpower cost, a project would incur costs due to hardware and software required for the project and the company overheads for administration, office space, etc.

It is important to note that the effort and the duration estimations obtained using the COCOMO model are called a nominal effort estimate and nominal duration estimate. The term nominal implies that if anyone tries to complete the project in a time shorter than the estimated duration, then the cost will increase drastically. But, if anyone completes the project over a longer period of time than the estimated, then there is almost no decrease in the estimated cost value.

Example1: Suppose a project was estimated to be 400 KLOC. Calculate the effort and development time for each of the three model i.e., organic, semi-detached & embedded.

Solution: The basic COCOMO equation takes the form:

Effort=a_1*(KLOC) a_2 PM
Tdev=b_1*(efforts)b_2 Months
Estimated Size of project= 400 KLOC

(i)Organic Mode

E = 2.4 * (400)1.05 = 1295.31 PM
D = 2.5 * (1295.31)0.38=38.07 PM

(ii)Semidetached Mode

E = 3.0 * (400)1.12=2462.79 PM
D = 2.5 * (2462.79)0.35=38.45 PM

(iii) Embedded Mode

E = 3.6 * (400)1.20 = 4772.81 PM
D = 2.5 * (4772.8)0.32 = 38 PM

Example2: A project size of 200 KLOC is to be developed. Software development team has average experience on similar type of projects. The project schedule is not very tight. Calculate the Effort, development time, average staff size, and productivity of the project.

Solution: The semidetached mode is the most appropriate mode, keeping in view the size, schedule and experience of development time.

Hence E=3.0(200)1.12=1133.12PM
D=2.5(1133.12)0.35=29.3PM

P = 176 LOC/PM

2. Intermediate Model: The basic Cocomo model considers that the effort is only a function of the number of lines of code and some constants calculated according to the various software systems. The intermediate COCOMO model recognizes these facts and refines the initial estimates obtained through the basic COCOMO model by using a set of 15 cost drivers based on various attributes of software engineering.

Classification of Cost Drivers and their attributes:

(i) Product attributes -

- Required software reliability extent
- Size of the application database
- The complexity of the product

Hardware attributes -

- Run-time performance constraints
- Memory constraints
- The volatility of the virtual machine environment
- Required turnabout time

Personnel attributes -

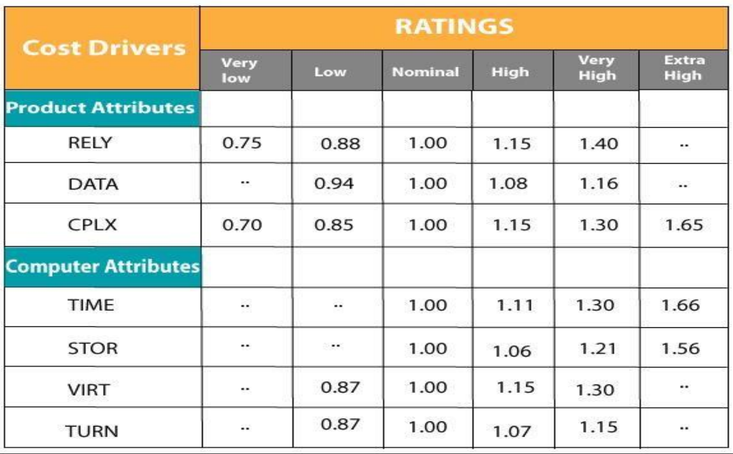
- Analyst capability

- Software engineering capability
- Applications experience
- Virtual machine experience
- Programming language experience

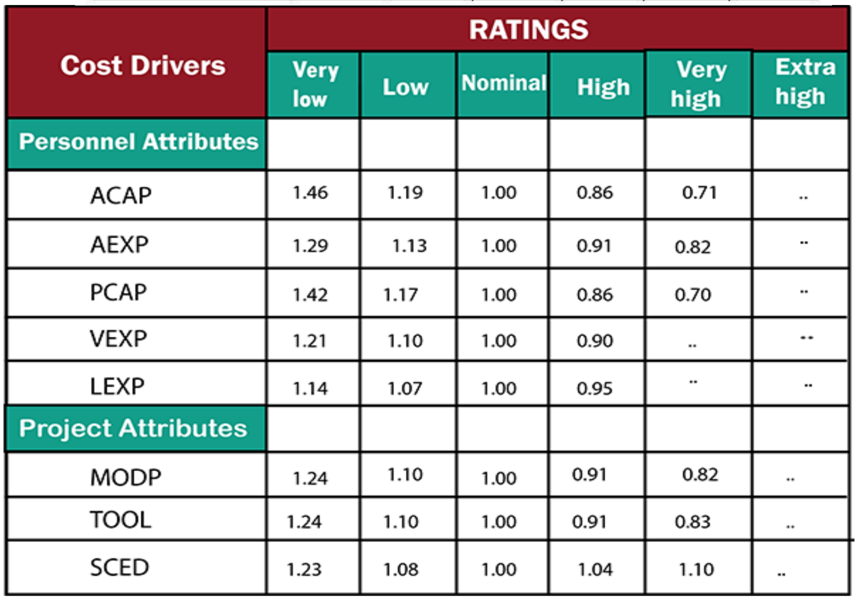
Project attributes -

- Use of software tools
- Application of software engineering methods
- Required development schedule

The cost drivers are divided into four categories:



Cost Drivers	RATINGS					
	Very low	Low	Nominal	High	Very High	Extra High
Product Attributes						
RELY	0.75	0.88	1.00	1.15	1.40	..
DATA	..	0.94	1.00	1.08	1.16	..
CPLX	0.70	0.85	1.00	1.15	1.30	1.65
Computer Attributes						
TIME	..	..	1.00	1.11	1.30	1.66
STOR	..	..	1.00	1.06	1.21	1.56
VIRT	..	0.87	1.00	1.15	1.30	..
TURN	..	0.87	1.00	1.07	1.15	..



Cost Drivers	RATINGS					
	Very low	Low	Nominal	High	Very high	Extra high
Personnel Attributes						
ACAP	1.46	1.19	1.00	0.86	0.71	..
AEXP	1.29	1.13	1.00	0.91	0.82	..
PCAP	1.42	1.17	1.00	0.86	0.70	..
VEXP	1.21	1.10	1.00	0.90	..	..
LEXP	1.14	1.07	1.00	0.95	..	..
Project Attributes						
MODP	1.24	1.10	1.00	0.91	0.82	..
TOOL	1.24	1.10	1.00	0.91	0.83	..
SCED	1.23	1.08	1.00	1.04	1.10	..

Intermediate COCOMO equation:

$E=a_i\ (KLOC)\ b_i*EAF$
$D=c_i\ (E)d_i$

Coefficients for intermediate COCOMO

Project	a_i	b_i	c_i	d_i
Organic	2.4	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

3. Detailed COCOMO Model: Detailed COCOMO incorporates all qualities of the standard version with an assessment of the cost driver? effect on each method of the software engineering process. The detailed model uses various effort multipliers for each cost driver property. In detailed cocomo, the whole software is differentiated into multiple modules, and then we apply COCOMO in various modules to estimate effort and then sum the effort.

The Six phases of detailed COCOMO are:

1. Planning and requirements
2. System structure
3. Complete structure
4. Module code and test
5. Integration and test
6. Cost Constructive model

The effort is determined as a function of program estimate, and a set of cost drivers are given according to every phase of the software lifecycle.

9. Explain the contents of test plan

A test plan is a detailed document which describes software testing areas and activities. It outlines the test strategy, objectives, test schedule, required resources (human resources, software, and hardware), test estimation and test deliverables.

The test plan is a base of every software's testing. It is the most crucial activity which ensures availability of all the lists of planned activities in an appropriate sequence.

The test plan is a template for conducting software testing activities as adefined process that is fully monitored and controlled by the testing manager. The test plan is prepared by the Test Lead (60%), Test Manager(20%), and by the test engineer(20%).

Types of Test Plan

There are three types of the test plan

- Master Test Plan
- Phase Test Plan
- Testing Type Specific Test Plans

Master Test Plan

Master Test Plan is a type of test plan that has multiple levels of testing. It includes a complete test strategy.

Phase Test Plan

A phase test plan is a type of test plan that addresses any one phase of the testing strategy. For example, a list of tools, a list of test cases, etc.

Specific Test Plans

Specific test plan designed for major types of testing like security testing, load testing, performance testing, etc. In other words, a specific test plan designed for non-functional testing.

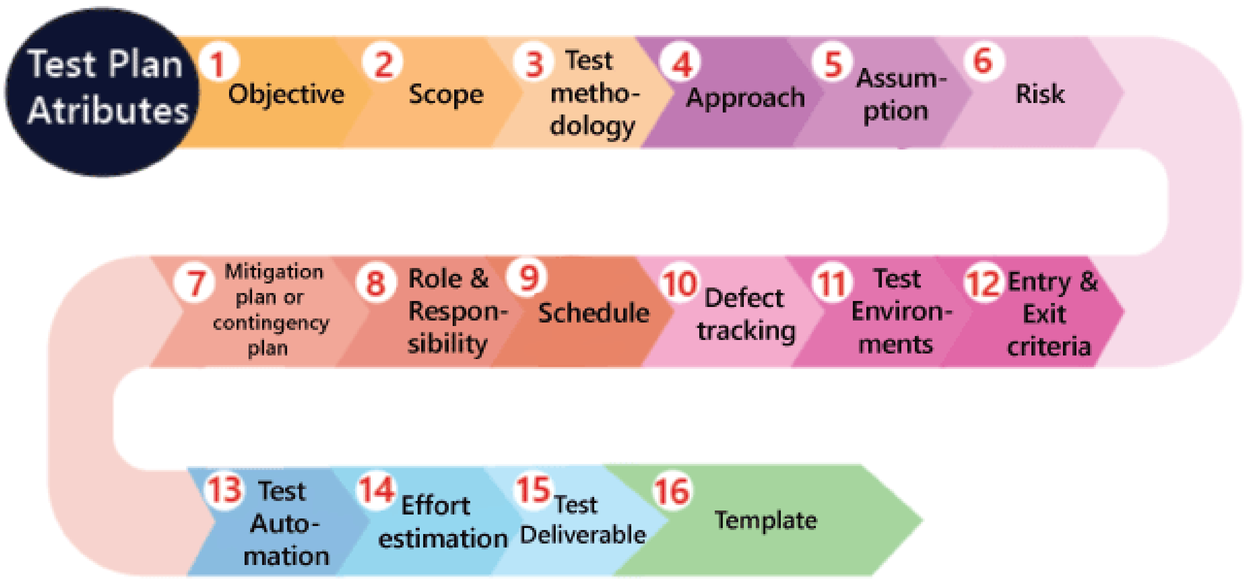
How to write a Test Plan

seven steps to prepare a test plan.

1. First, analyze product structure and architecture.
2. Now design the test strategy.
3. Define all the test objectives.
4. Define the testing area.
5. Define all the useable resources.
6. Schedule all activities in an appropriate manner.
7. Determine all the Test Deliverables.

Test plan components or attributes

The test plan consists of various parts, which help us to derive the entire testing activity.



a. **Objectives:** It consists of information about modules, features, test data etc., which indicate the aim of the application means the application behavior, goal, etc.
b. **Scope:** It contains information that needs to be tested with respective of an application. The Scope can be further divided into two parts:

- In scope

- Out scope

In scope: These are the modules that need to be tested rigorously (in-detail).

Out scope: These are the modules, which need not be tested rigorously.

For example, Suppose we have a Gmail application to test, where **features to be tested** such as **Compose mail, Sent Items, Inbox, Drafts** and the **features which not be tested** such as **Help**, and so on which means that in the planning stage, we will decide that which functionality has to be checked or not basedon the time limit given in the product.

c. Test methodology:

It contains information about performing a different kind of testing like Functional testing, Integration testing, and System testing, etc. on the application. In this, we will decide what type of testing; we will perform on the various features based on the application requirement. And here, we should also define that what kind of testing we will use in the testing methodologies so that everyone, like the management, the development team, and the testing team can understand easily because the testing terms are not standard.

For example, for standalone application such as **Adobe Photoshop**, we will perform the following types of testing:

Smoke testing→ Functional testing → Integration testing →System testing →Adhoc testing → Compatibility testing → Regression testing→ Globalization testing → Accessibility testing → Usability testing → Reliability testing → Recovery testing → Installation or Uninstallation testing

Smoke testing	Functional testing	Integration testing
System testing	Adhoc testing	Compatibility testing
Regression testing	Globalization testing	Accessibility testing
Usability testing	Performance testing	

d. Approach

This attribute is used to describe the flow of the application while performing testing and for the future reference.

- **By writing the high-level scenarios**
- **By writing the flow graph**

By writing the high-level scenarios

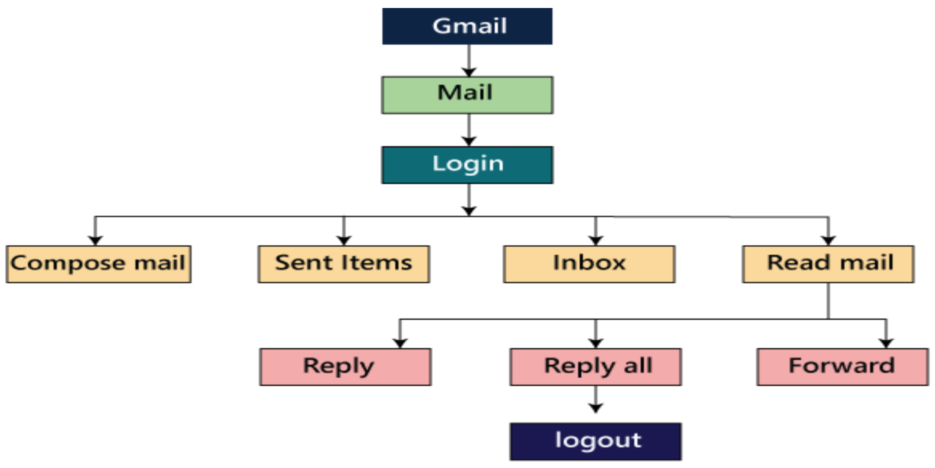
For example, suppose we are testing the **Gmail** application:

- Login to Gmail- sends an email and check whether it is in the Sent Items page
- Login to …….

We are writing this to describe the approaches which have to be taken for testing the product and only for the critical features where we will write the high-level scenarios. Here, we will not be focusing on covering all the scenarios because it can be decided by the particular test engineer that which features have to be tested or not.

By writing the flow graph

The flow graph is written because writing the high-level scenarios are bit time taking process, as we can see in the below image:



We are creating flow graphs to make the following benefits such as:

- Coverage is easy
- Merging is easy

The approach can be classified into two parts which are as following:

- Top to bottom approach
- Bottom to top approach

e. Assumption

It contains information about a problem or issue which maybe occurred during the testing process and when we are writing the test plans, the assured assumptions would be made like resources and technologies, etc.

f. Risk

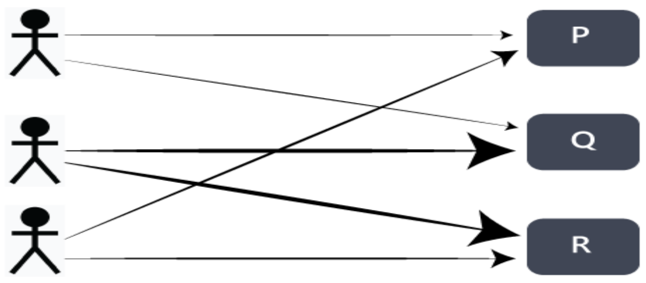
These are the challenges which we need to face to test the application in the current release and if the assumptions will fail then the risks are involved.

For example, the effect for an application, release date becomes postponed.

g. Mitigation Plan or Contingency Plan

It is a back-up plan which is prepared to overcome the risks or issues.

Let us see one example for assumption, risk, and the contingency plan together because they are co-related to each other.



In any product, the **assumption** we will make is that the all 3 test engineers will be there until the completion of the product and each of them is assigned different modules such as P, Q, and R. In this particular scenario, the **risk** could be that if the test engineer left the project in the middle of it.

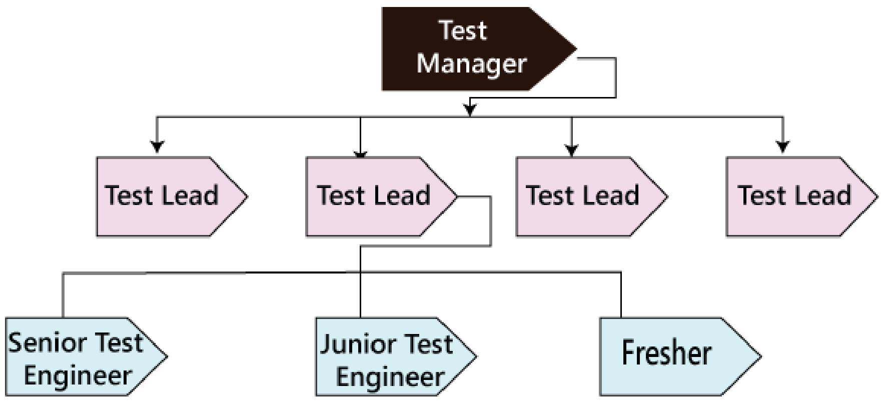
Therefore, the **contingency plan** will be assigned a primary and subordinate owner to each feature. So if the one test engineer will leave, the subordinate owner takes over that specific feature and also helps the new test engineer, so he/she can understand their assigned modules.

The assumptions, risk, and mitigation or contingency plan are always precise on the product itself. The various types of risks are as follows:

- Customer perspective
- Resource approach
- Technical opinion

h. Role & Responsibility

It defines the complete task which needs to be performed by the entire testing team. When a large project comes, then the **Test Manager** is a person who writes the test plan. If there are 3-4 small projects, then the test manager will assign each project to each Test Lead. And then, the test lead writes the test plan for the project, which he/she is assigned.



Let see one example where we will understand the roles and responsibility of the Test manager, test lead, and the test engineers.

Role: Test Manager

Name: Mahith

Responsibility:

- Prepare(write and review) the test plan
- Conduct the meeting with the development team
- Conduct the meeting with the testing team
- Conduct the meeting with the customer
- Conduct one monthly stand up meeting
- Sign off release note
- Handling Escalations and issues

Role: Test Lead

Name: SAI

Responsibility:

- Prepare(write and review) the test plan
- Conduct daily stand up meeting
- Review and approve the test case
- Prepare the RTM and Reports
- Assign modules
- Handling schedule

Role: Test Engineer 1, Test Engineer 2 and Test Engineer 3

Name: Louis, Jessica, Donna

Assign modules: M1, M2, and M3

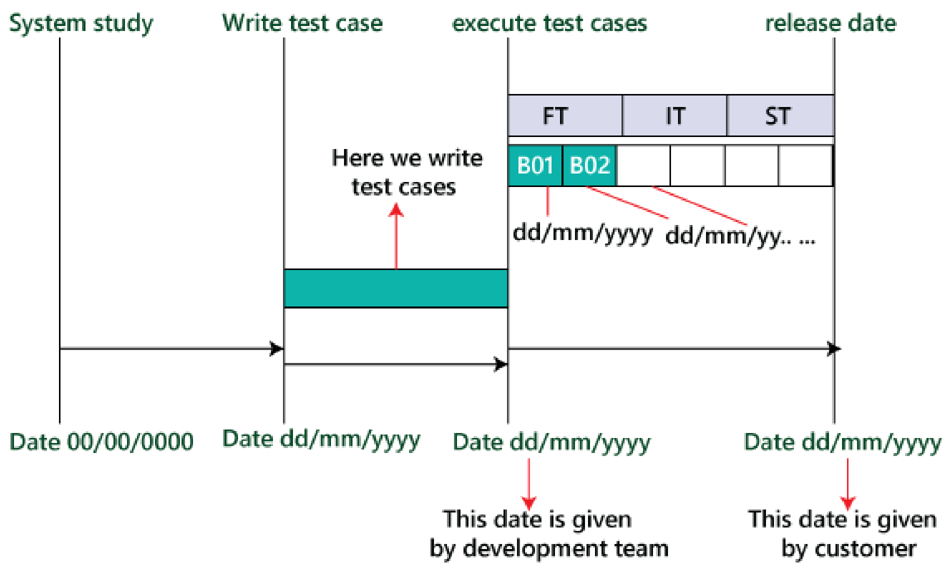
Responsibility:

- Write, Review, and Execute the test documents which consists of test case and test scenarios

- Read, review, understand and analysis the requirement
- Write the flow of the application
- Execute the test case
- RTM for respective modules
- Defect tracking
- Prepare the test execution report and communicate it to the Test Lead.

i. Schedule

It is used to explain the timing to work, which needs to be done or this attribute covers when exactly each testing activity should start and end? And the exact data is also mentioned for every testing activity for the particular date.



Therefore as we can see in the below image that for the particular activity, there will be a starting date and ending date; for each testing to a specific build, there will be the specified date.

For example

- Writing test cases
- Execution process

j. Defect tracking

It is generally done with the help of tools because we cannot track the status of each bug manually. And we also comment about how we communicate the bugs which are identified during the testing process and send it back to the development team and how the development team will reply. Here we also mention the priority of the bugs such as high, medium, and low.

Following are various aspects of the defect tracking:

- **Techniques to track the bug**
- **Bug tracking tools**
  We can comment on the name of the tool, which we will use for tracking the bugs. Some of the most commonly used bug tracking tools are Jira, Bugzilla, Mantis, and Trac, etc.
- **Severity**
  The severity could be as following:
  a. Blocker or showstopper
  For example, there will be a defect in the module; we cannot go further to test other modules because if the bug is blocked, we can proceed to check other modules.
  b. Critical
  In this situation, the defects will affect the business.
  c. Major
  d. Minor
  These defects are those, which affect the look and feel of the application.
- **Priority**
  High-P1
  …..
  Medium-P2
  …..
  Low-P3
  …..
  …..
  P4

Therefore, based on the priority of bugs liike high, medium, and low, we will categorize it as P1, P2, P3, and P4.

k. Test Environments

These are the environments where we will test the application, and here we have two types of environments, which are of **software** and **hardware** configuration.

The **software configuration** means the details about different **Operating Systems** such as **Windows, Linux, UNIX, and Mac** and various **Browsers** like **Google Chrome, Firefox, Opera, Internet Explorer**, and so on.

And the **hardware configuration** means the information about different sizes of **RAM, ROM, and the Processors**.

For example

- The **Software** includes the following:

Server

Operating system	Linux
Webserver	Apache Tomcat
Application server	Websphere
Database server	Oracle or MS-SQL Server

Client

Operating System	Window XP, Vista 7
Browsers	Mozilla Firefox, Google Chrome, Internet Explorer, Internet Explorer 7, and Internet Explorer 8

- The **Hardware** includes the following:

Server: Sun StarCat 1500

This particular server can be used by the testing team to test their application.

Client:

Processor	Intal2GHz
RAM	2GB
….	….

- **Process to install the software**

  …..

The development team will provide the configuration of how to install the software. If the development team will not yet provide the process, then we will write it as Task-Based Development (TBD) in the test plan.

I. Entry and Exit criteria

It is a necessary condition, which needs to be satisfied before starting and stopping the testing process.

Entry Criteria

The entry criteria contain the following conditions:

- White box testing should be finished.
- Understand and analyze the requirement and prepare the test documents or when the test documents are ready.
- Test data should be ready.

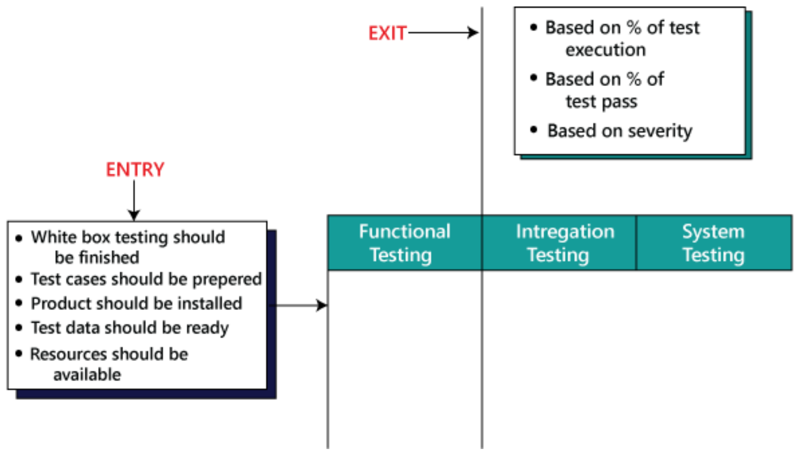
- Build or the application must be prepared
- Modules or features need to be assigned to the different test engineers.
- The necessary resource must be ready.

Exit Criteria

The exit criteria contain the following conditions:

- When all the test cases are executed.
- Most of the test cases must be **passed**.
- Depends on severity of the bugs which means that there must not be any blocker or major bug, whereas some minor bugs exist.

Before we start performing functional testing, all the above **Entry Criteria** should be followed. After we performed functional testing and before we will do integration testing, then the **Exit criteria of** the functional testing should be followed because the % of exit criteria are decided by the meeting with both development and test manager because their collaboration can achieve the percentage. But if the exit criteria of functional testing are not followed, then we cannot proceed further to integration testing.

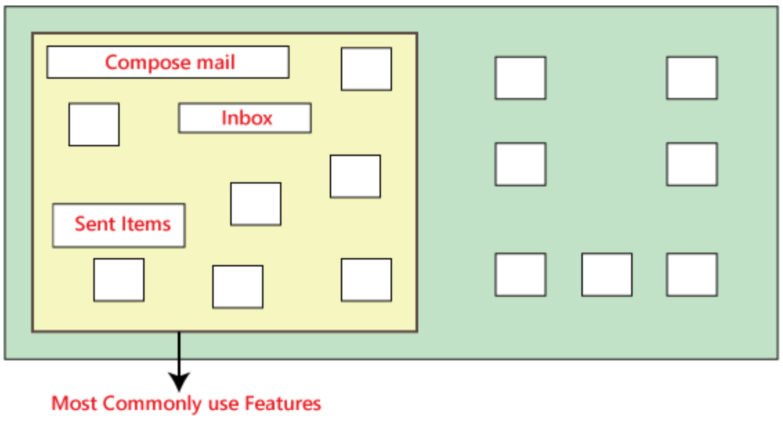

EXIT
• Based on % of test execution
• Based on % of test pass
• Based on severity
ENTRY
• White box testing should be finished
• Test cases should be prepered
• Product should be installed
• Test data should be ready
• Resources should be available
Functional Testing
Intregation Testing
System Testing

Here **based on the severity** of the bug's means that the testing team would have decided that to proceed further for the next phases.

m. Test Automation

- Which feature has to be automated and not to be automated?
- Which test automation tool we are going to use on which automation framework?

- We automate the test case only after the first release.
- Here the question arises that on what basis **we** will **decide which features have to be tested?**



n. Effort estimation

we will plan the effort need to be applied by every team member.

o. Test Deliverable

These are the documents which are the output from the testing team, which we handed over to the customer along with the product. It includes the following:

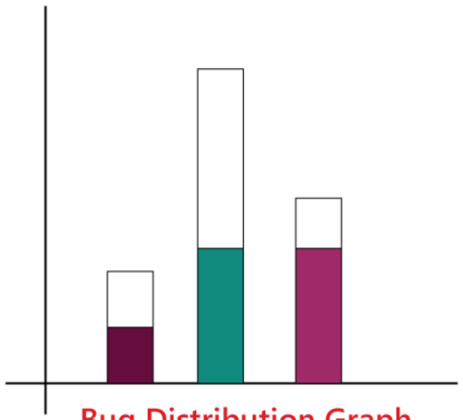
- **Test plan**
- **Test Cases**
- **Test Scripts**
- **RTM(Requirement Traceability Matrix)**
- **Defect Report**
- **Test Execution Report**
- **Graphs and metrics**
- **Release Notes**

p. Graphs and Metrics

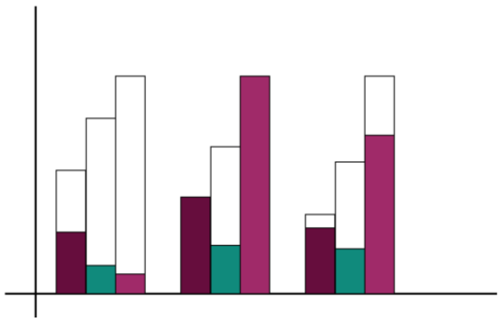
Graph

we have five different graphs that show the various aspects of the testing process.

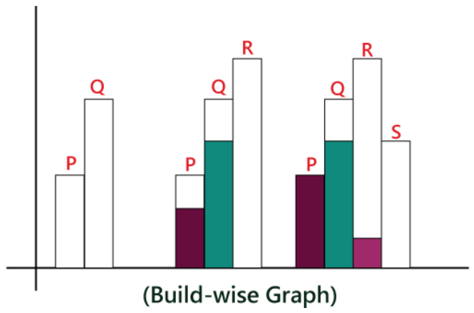
Graph1: In this, we will show how many defects have been identified and how many defects have been fixed in every module.



Graph 2: Figure one shows how many critical, major, and minor defects have been identified for every module and how many have been fixed for their respective modules.

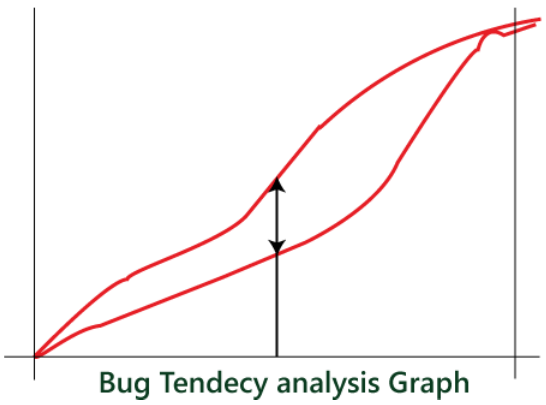


Graph3: In this particular graph, we represent the **build wise graph**, which means that in every builds how many defects have been identified and fixed for every module. Based on the module, we have determined the bugs. We will add **R** to show the number of defects in P and Q, and we also add **S** to show the defects in P, Q, and R.



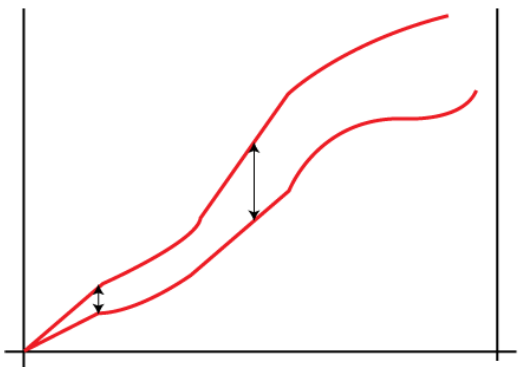
(Build-wise Graph)

Graph4: The test lead will design the **Bug Trend analysis** graph which is created every month and send it to the Management as well. And it is just like prediction which is done at the end of the product. And here, we can also **rate the bug fixes** as we can observe that **arc** has an **upward tendency** in the below image.

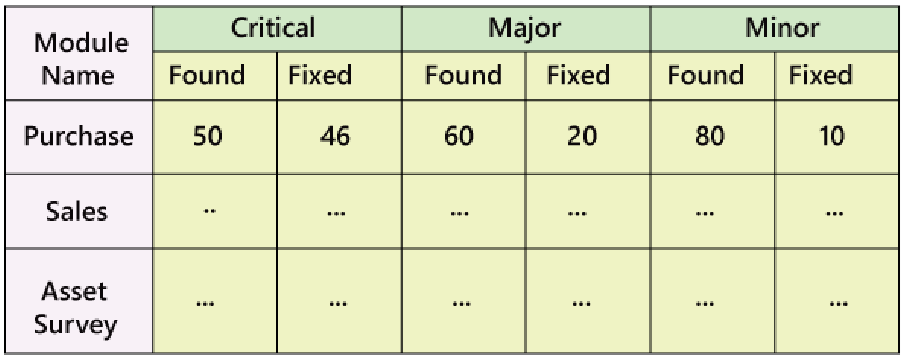


Bug Tendecy analysis Graph

Graph5: The **Test Manager** has designed this type of graph. This graph is intended to understand the gap in the assessment of bugs and the actual bugs which have been occurred, and this graph also helps to improve the evaluation of bugs in the future.



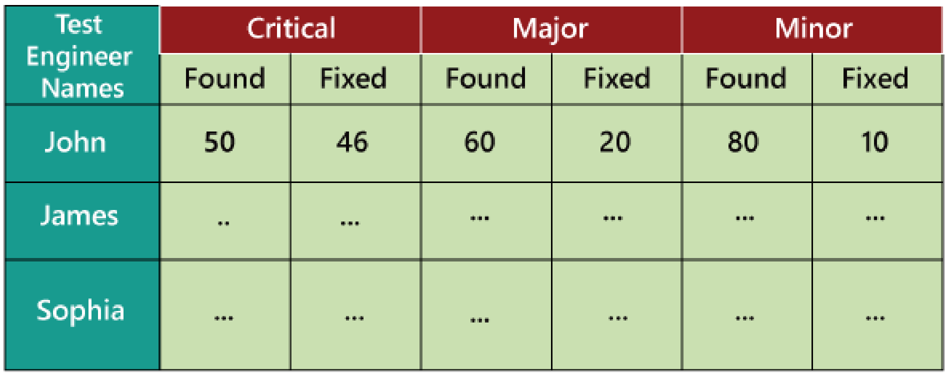
Metrics



Module Name	Critical		Major		Minor	
	Found	Fixed	Found	Fixed	Found	Fixed
Purchase	50	46	60	20	80	10
Sales	..	...	...	...	...	...
Asset Survey	...	...	...	...	...	...

As above, we create the bug distribution graph, which is in the figure 1, and with the help of above mention data, we will design the metrics as well.

For example

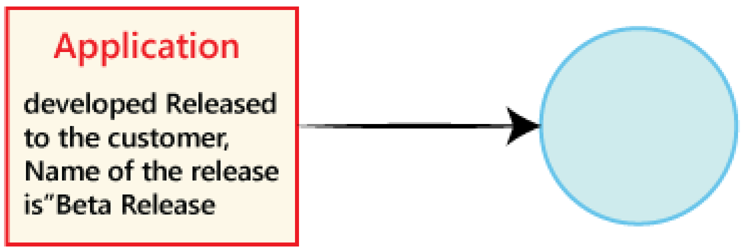


Test Engineer Names	Critical		Major		Minor	
	Found	Fixed	Found	Fixed	Found	Fixed
John	50	46	60	20	80	10
James	..	...	...	...	...	...
Sophia	...	...	...	...	...	...

In the above figure, we retain the records of all the test engineers in a particular project and how many defects have been identified and fixed. We can also use this data for future analysis. When a new requirement comes, we can decide whom to provide the challenging feature for testing based on the number of defects they have found earlier according to the above metrics. Andwe will be in a better situation to know who can handle the problematic features very well and find maximum numbers of defects.

Release Note: It is a document that is prepared during the release of theproduct and signed by the Test Manager.

In the below image, we can see that the final product is developed and deployed to the customer, and the latest release name is **Beta**.

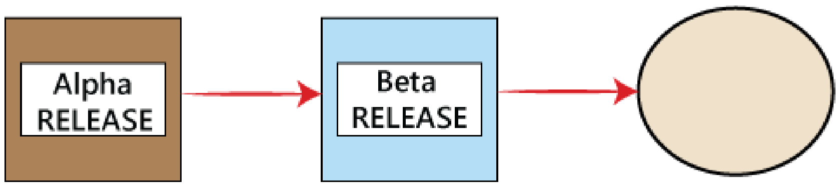

Application
developed Released to the customer, Name of the release is"Beta Release

The **Release note** consists of the following:

- User manual.
- List of pending and open defects.
- List of added, modified, and deleted features.
- List of the platform (Operating System, Hardware, Browsers) on which the product is tested.
- Platform in which the product is not tested.
- List of bugs fixed in the current release, and the list of fixed bugs in the previous release.
- Installation process
- Versions of the software

For Example

Suppose that **Beta** is the second release of the application after the first release **Alpha** is released. Some of the defect identified in the first released and that has been fixed in the later released. And here, we will also point out the list of newly added, modified, and deleted features from alpha release to the beta release.



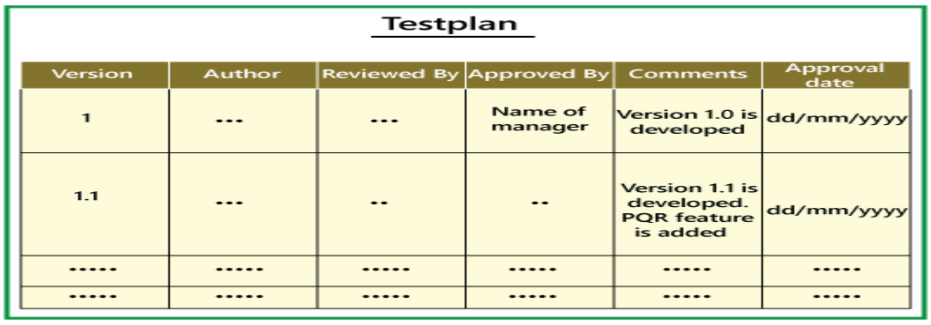
q. Template

This part contains all the templates for the documents that will be used in the product, and all the test engineers will use only these templates in the project to maintain the consistency of the product. Here, we have different types of the template which are used during the entire testing process such as:

- Test case template
- Test case review template
- RTM Template
- Bug Report Template
- Test execution Report

Let us see one sample of test plan document

Page-1



Testplan

Version	Author	Reviewed By	Approved By	Comments	Approval date
1	---	---	Name of manager	Version 1.0 is developed	dd/mm/yyyy
1.1	---	--	--	Version 1.1 is developed. PQR feature is added	dd/mm/yyyy
-----	-----	-----	-----	-----	-----
-----	-----	-----	-----	-----	-----

Page3-page18

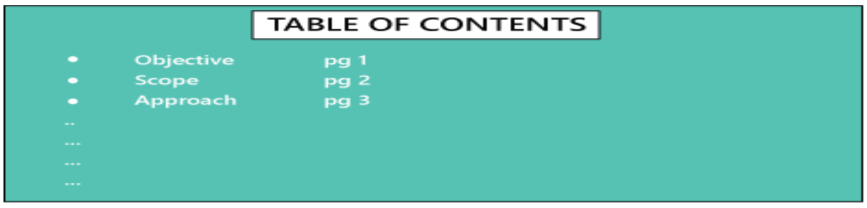
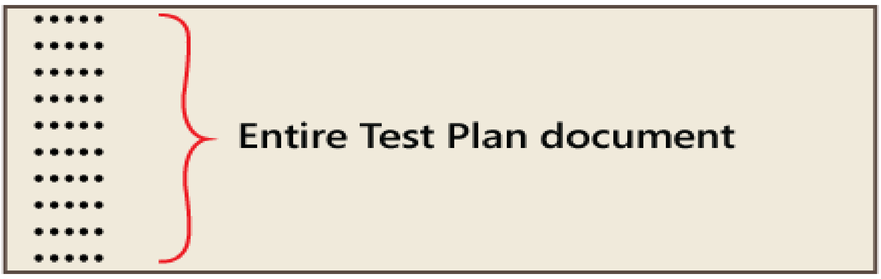
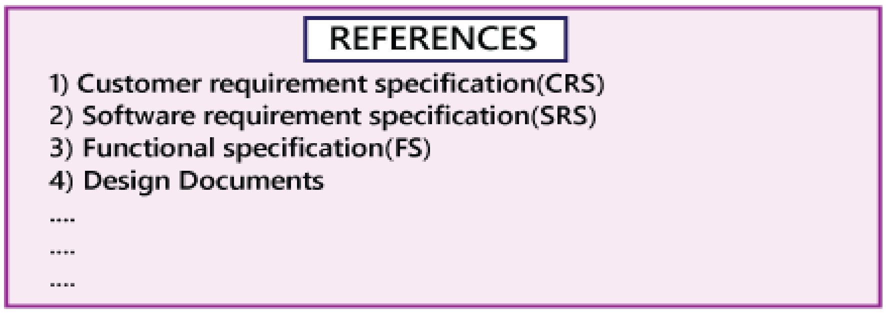


TABLE OF CONTENTS

•	Objective	pg 1
•	Scope	pg 2
•	Approach	pg 3
..		
...		
...		
...		



Page-20


REFERENCES

1) Customer requirement specification(CRS)
2) Software requirement specification(SRS)
3) Functional specification(FS)
4) Design Documents

....
....
....

In-Page 1, we primarily fill only the **Versions, Author, Comments, and Reviewed By** fields, and after the manager approves it, we will mention the details in the **Approved By and Approval Date** fields.

Mostly the test plan is approved by the Test Manager, and the test engineers only reviews it. And when the new features come, we will modify the test plan and do the necessary modification in **Version** field, and then it will be sent again for further review, update, and approval of the manager. The test plan must be updated whenever any changes have occurred. On page 20,the **References** specify the details about all the documents which we are going to use to write the test plan document.

Note:

Who writes the test plan?

- Test Lead→60%
- Test Manager→20%
- Test Engineer→20%

Therefore, as we can see from above that in 60% of the product, the test plan is written by the Test Lead.

Who reviews the Test Plan?

- Test Lead
- Test Manager
- Test engineer

- Customer
- Development team

The Test Engineer review the Test plan for their module perspective and the test manager review the Test plan based on the customer opinion.

Who approve the test Plan?

- Customer
- Test Manager

Who writes the test case?

- Test Lead
- Test Engineer

Who review the test case?

- Test Engineer
- Test Lead
- Customer
- Development Team

Who approves the Test cases?

- Test Manager
- Test Lead
- Customer

Test Plan Guidelines

- Collapse your test plan.
- Avoid overlapping and redundancy.
- If you think that you do not need a section that is already mentioned above, then delete that section and proceed ahead.

- Be specific. For example, when you specify a software system as the part of the test environment, then mention the software version instead of only name.
- Avoid lengthy paragraphs.
- Use lists and tables wherever possible.
- Update plan when needed.
- Do not use an outdated and unused document.

Importance of Test Plan

- The test plan gives direction to our thinking. This is like a rule book, which must be followed.
- The test plan helps in determining the necessary efforts to validate the quality of the software application under the test.
- The test plan helps those people to understand the test details that are related to the outside like developers, business managers, customers, etc.
- Important aspects like test schedule, test strategy, test scope etc are documented in the test plan so that the management team can review them and reuse them for other similar projects.

In general, testing commences with a test plan and terminates with acceptance testing. A test plan is a general document for the entire project that defines the scope, approach to be taken, and the schedule of testing as well as identifies the test items for the entire testing process and the person responsible for the different activities of testing.

The test planning can be done well before the actual testing commences and can be done in parallel with the coding and design phases. The inputs for forming the test plan are: (1) project plan (2) requirements document and (3) system design document. The project plan is needed to make sure that the test plan is consistent with the overall plan for the project and the testing the test plan is consistent with the overall plan for the project and the testing schedule matches that of the project plan.

The requirements document and the design document are the basic documents used for selecting the test units and deciding the approaches to be used during testing.

A test plan should contain the following:

- Test unit specification
- Features to be tested
- Approach for testing
- Test deliverable
- Schedule
- Personnel allocation.

One of the most important activities of the test plan is to identify the test units. A test unit is a set of one or more modules, together with associate data, that are from a single computer program and that are the objects of testing. A test unit can occur at any level and can contain from a single module to the entire system.

Thus, a test unit may be a module, a few modules, or a complete system. The levels are specified in the test plan by identifying the test units for the project. Different units are usually specified for unit integration, and system testing. The identification of test units may be a module, a few modules or a complete system. The levels are specified in the test plan by identifying the test units for the project.

Different units are usually specified for unit, integration and system testing. The identification of test units establishes the different levels of testing that will be performed in the project. The basic idea behind forming test units is to make sure that testing is being performed incrementally, with each increment including only a few aspects that need to be tested. A unit should be such that it can be easily tested.

In other words, it should be possible to form meaningful test cases and execute the unit without much effort with these test cases. Features to be tested include all software features and combinations of features that should be tested. A software feature is a software characteristic specified or implied by the requirements or design documents. These may include functionality, performance, design constraints, and attributes.

The approach for testing specifies the overall approach to be followed in the current project. The technique that will be used to judge the testing effort should also be specified. This is sometimes called the testing criterion. Testing deliverable should be specified in the test plan before the actual testing begins. Deliverables could be a list of test cases that were used, detailed result of testing, test summary report, test log, and data about the code coverage. In general, a test case specification report, test summary report, and a test log should always be specified as deliverables.

10. Explain the contents of test plan template

Test Plan Template

- 1 Introduction
  - 1.1 Scope
    - 1.1.1 In Scope
    - 1.1.2 Out of Scope
  - 1.2 Quality Objective
  - 1.3 Roles and Responsibilities
- 2 Test Methodology
  - 2.1 Overview
  - 2.2 Test Levels
  - 2.3 Bug Triage
  - 2.4 Suspension Criteria and Resumption Requirements
  - 2.5 Test Completeness
- 3 Test Deliverables
- 4 Resource & Environment Needs

1) Introduction

Brief introduction of the test strategies, process, workflow and methodologies used for the project

1.1) Scope

1.1.1) In Scope

Scope defines the features, functional or non-functional requirements of the software that **will be** tested

1.1.2) Out of Scope

Out Of Scope defines the features, functional or non-functional requirements of the software that **will NOT be** tested

1.2) Quality Objective

Here make a mention of the overall objective that you plan to achive with your manual testing and automation testing.

Some objectives of your testing project could be

- Ensure the Application Under Test conforms to functional and non-functional requirements
- Ensure the AUT meets the quality specifications defined by the client
- Bugs/issues are identified and fixed before go live

1.3) Roles and Responsibilities

Detail description of the Roles and responsibilities of different team members like

- QA Analyst
- Test Manager
- Configuration Manager
- Developers
- Installation Team

Amongst others

2) Test Methodology

2.1) Overview

Mention the reason of adopting a particular test methodology for the project. The test methodology selected for the project could be

- Waterfall
- Iterative
- Agile
- Extreme Programming

The methodology selected depends on multiple factors.

2.2) Test Levels

Test Levels define the Types of Testing to be executed on the Application Under Test (AUT). The Testing Levels primarily depends on the scope of the project, time and budget constraints.

2.3) Bug Triage

The goal of the triage is to

- To define the type of resolution for each bug
- To prioritize bugs and determine a schedule for all “To Be Fixed Bugs’.

2.4) Suspension Criteria and Resumption Requirements

Suspension criteria define the criteria to be used to suspend all or part of the testing procedure while Resumption criteria determine when testing can resume after it has been suspended

2.5) Test Completeness

Here you define the criterias that will deem your testing complete.

For instance, a few criteria to check Test Completeness would be

- 100% test coverage
- All Manual & Automated Test cases executed
- All open bugs are fixed or will be fixed in next release

3) Test Deliverables

Here mention all the Test Artifacts that will be delivered during different phases of the testing lifecycle.

Here are the simple deliverables

- Test Plan
- Test Cases
- Requirement Traceability Matrix
- Bug Reports
- Test Strategy
- Test Metrics
- Customer Sign Off

4) Resource & Environment Needs

4.1) Testing Tools

- Requirements Tracking Tool
- Bug Tracking Tool
- Automation Tools

Required to test the project

4.2) Test Environment

It mentions the minimum **hardware** requirements that will be used to test the Application.

Following **software's** are required in addition to client-specific software.

- Windows 8 and above
- Office 2013 and above
- MS Exchange, etc.

5) Terms/Acronyms

Make a mention of any terms or acronyms used in the project

TERM/ACRONYM	DEFINITION
API	Application Program Interface
AUT	Application Under Test

------------------------------All the best---

Sample Test Plan Document Banking Web Application Example

1 Introduction

The Test Plan is designed to prescribe the scope, approach, resources, and schedule of all testing activities of the project Guru99 Bank.

The plan identify the items to be tested, the features to be tested, the types of testing to be performed, the personnel responsible for testing, the resources and schedule required to complete testing, and the risks associated with the plan.

1.1 Scope

1.1.1 In Scope

Module Name	Applicable Roles	Description
Balance Enquiry	Manager Customer	**Customer**: A customer can have multiple bank accounts. He can view balance of his accounts only **Manager**: A manager can view balance of all the customers who come under his supervision
Fund Transfer	Manager Customer	**Customer:** A customer can have transfer funds from his “own” account to any destination account. **Manager**: A manager can transfer funds from any source bank account to destination account
Mini Statement	Manager Customer	A Mini statement will show last 5 transactions of an account **Customer:** A customer can see mini-statement of only his “own” accounts **Manager:** A manager can see mini-statement of any account
Customized Statement	Manager Customer	A customized statement allows you to filter and display transactions in an account based on date, transaction value **Customer:** A customer can see Customized- statement of only his “own” accounts **Manager**: A manager can see Customized -statement of any account
Change Password	Manager Customer	**Customer:** A customer can change password of only his account. **Manager**: A manager can change password of only his

		account. He cannot change passwords of his customers
New Customer	Manager	**Manager**: A manager can add a new customer.
	Manager	**Manager:** A manager can edit details like address, email, telephone of a customer.
New Account	Manager	Currently system provides 2 types of accounts • Saving • Current A customer can have multiple saving accounts (one in his name, other in a joint name etc). He can have multiple current accounts for different companies he owns. Or he can have a multiple current and saving accounts. **Manager:** A manager can add a new account for an existing customer.
Edit Account	Manager	**Manager:** A manager can add a edit account details for an existing account
Delete Account	Manager	**Manager:** A manager can add a delete an account for a customer.
Delete Customer	Manager	A customer can be deleted only if he/she has no active current or saving accounts **Manager:** A manager can delete a customer.
Deposit	Manager	**Manager:** A manager can deposit money into any account. Usually done when cash is deposited at a bank branch.
Withdrawal	Manager	**Manager:** A manager can withdraw money from any account. Usually done when cash is withdrawn at a bank branch.

1.1.2 Out of Scope

These feature are not be tested because they are not included in the software requirement specs

- User Interfaces
- Hardware Interfaces
- Software Interfaces
- Database logical
- Communications Interfaces
- Website Security and Performance

1.2 Quality Objective

The test objectives are to **verify** the Functionality of website Guru99 Bank, the project should focus on testing the **banking operation** such as Account Management, Withdrawal, and Balance…etc. to **guarantee** all these operation can work **normally** in real business environment.

1.3 Roles and Responsibilities

The project should use **outsource** members as the tester to save the project cost.

No.	**Member**	**Tasks**
1.	Test Manager	Manage the whole project Define project directions Acquire appropriate resources
2.	Test	Identifying and describing appropriate test techniques/tools/automation architecture Verify and assess the Test Approach Execute the tests, Log results, Report the defects. Outsourced members
3.	Developer in Test	Implement the test cases, test program, test suite etc.

4.	Test Administrator	Builds up and ensures test environment and assets are managed and maintained Support Tester to use the test environment for test execution
5.	SQA members	Take in charge of quality assurance Check to confirm whether the testing process is meeting specified requirements

2 Test Methodology

2.1 Overview

2.2 Test Levels

- **Integration** Testing (Individual software modules are combined and tested as a group)
- **System** Testing: Conducted on a **complete**, **integrated** system to evaluate the system's compliance with its specified requirements
- **API testing:** Test all the APIs create for the software under tested

2.3 Bug Triage

2.4 Suspension Criteria and Resumption Requirements

If the team members report that there are **40%** of test cases **failed**, suspend testing until the development team fixes all the failed cases.

2.5 Test Completeness

- Specifies the criteria that denote a **successful** completion of a test phase
- **Run** rate is mandatory to be **100%** unless a clear reason is given.
- **Pass** rate is **80%,** achieving the pass rate is **mandatory**

2.6 Project task and estimation and schedule

Task	Members	Estimate effort
Create the test specification	Test Designer	170 man-hour
Perform Test Execution	Tester, Test Administrator	80 man-hour

Test Report	Tester	10 man-hour
Test Delivery		20 man-hour
Total		280 man-hour

Schedule to complete these tasks

3 Test Deliverables

Test deliverables are provided as below

Before testing phase

- Test plans document.
- Test cases documents
- Test Design specifications.

During the testing

– Test Tool Simulators.

– Test Data

– Test Trace-ability Matrix – Error logs and execution logs.

After the testing cycles is over

- Test Results/reports
- Defect Report
- Installation/ Test procedures guidelines
- Release notes

4 Resource & Environment Needs

4.1 Testing Tools

No.	Resources	Descriptions
1.	Server	Need a Database server which install MySQL server Web server which install Apache Server

2.	Test tool	Develop a Test tool which can auto generate the test result to the predefined form and automated test execution
3.	Network	Setup a LAN Gigabit and 1 internet line with the speed at least 5 Mb/s
4.	Computer	At least 4 computer run Windows 7, Ram 2GB, CPU 3.4GHZ

4.2 Test Environment

It mentions the minimum hardware and software requirements that will be used to test the Application.

Following software's are required in addition to client-specific software.

- Windows 11 and above
- Office 2021 and above
- MS Exchange, etc.