GIPS INFOTECH
Global Institute of Professional Studies

GIPS INFOTECH
Global Institute of Professional Studies

Handout

Manual Software Testing

GIPS Infotech

GIPS InfoTech
4th floor, Royal Avenue, above Pradeep sweets, Shivar Chowk, Pimple Saudagar, Pune, Maharashtra 411027

©Copyright 2017, Gips Infotech, All Rights Reserved

About the Tutorial

Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not.

Testing is executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements.

This tutorial will give you a basic understanding on software testing, its types, methods, levels, and other related terminologies.

Audience

This tutorial is designed for software testing professionals who would like to understand the Testing Framework in detail along with its types, methods, and levels. This tutorial provides enough ingredients to start with the software testing process from where you can take yourself to higher levels of expertise.

Prerequisites

Before proceeding with this tutorial, you should have a basic understanding of the software development life cycle (SDLC). In addition, you should have a basic understanding of software programming using any programming language.

GIPS Infotech

Table of Contents

1. OVERVIEW 5

What is testing? 6

Why is Testing Necessary? 6

Who does Testing? 7

When to Start Testing? 7

When to Stop Testing? 7

Verification & Validation 8

SDLC and STLC (Life Cycles) 9

V Model 15

2. MYTHS 16

3. QA, QC, AND TESTING 19

Testing, Quality Assurance, and Quality Control 20

Audit and Inspection 20

Testing and Debugging ..... .21

4. TYPES OF TESTING 14

Functional Testing 23

Non Functional Testing 25

Maintenance Testing 27

Manual and Automation Testing 29

5. TESTING METHODS 31

Black-Box Testing 32

White-Box Testing 34

Grey-Box Testing 35

A Comparison of Testing Methods 36

GIPS Infotech

6. TESTING LEVELS 36

Unit Testing 37

Integration Testing 38

System Testing 43

Regression Testing 43

Acceptance Testing 44

Non-Functional Testing 45

Migration/Upgrade Testing 46

Portability Testing 46

7. TEST MANAGEMENT 47

Test Planning 48

Test Strategy 52

Testing Tools 52

Test progress monitoring and control 53

Test Closure 54

Traceability Matrix 54

8. DEFECT MANAGEMENT 59

Defect Definition 60

Defect life cycle 61

Severity and Priority 62

Defect Clustering 64

9. *GLOSSARY*.....65

Glossary of testing terminology 66

GIPS Infotech

Module-I

Software Testing an Overview

- What is testing?
- Why is testing necessary?
- Who does Testing?
- When to Start/Stop Testing?
- Verification & Validation
- Testing Life cycles
- V- Model

GIPS Infotech

Software Testing an Overview

What is testing?

Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not. In simple words, testing is executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements.

According to ANSI/IEEE 1059 standard, Testing can be defined as - *A process of analyzing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item*.

Why is Testing Necessary?

Software Testing is necessary because we all make mistakes. Some of those mistakes are unimportant, but some of them are expensive or dangerous. We need to check everything and anything we produce because things can always go wrong – humans make mistakes all the time.

Since we assume that our work may have mistaken, hence we all need to check our own work. However, some mistakes come from bad assumptions and blind spots, so we might make the same mistakes when we check our own work as we made when we did it. So, we may not notice the flaws in what we have done.
Ideally, we should get someone else to check our work because another person is more likely to spot the flaws.

There are several reasons which clearly tell us as why Software Testing is important and what are the major things that we should consider while testing of any product or application.

Software testing is very important because of the following reasons:

1. Software testing is really required to point out the defects and errors that were made during the development phases.
2. It's essential since it makes sure of the Customer's reliability and their satisfaction in the application.
3. It is very important to ensure the Quality of the product. Quality product delivered to the customers helps in gaining their confidence. (Know more about Software Quality)
4. Testing is necessary in order to provide the facilities to the customers like the delivery of high quality product or software application which requires lower maintenance cost and hence results into more accurate, consistent and reliable results.
5. Testing is required for an effective performance of software application or product.
6. It's important to ensure that the application should not result into any failures because it can be very expensive in the future or in the later stages of the development.
7. It's required to stay in the business

GIPS Infotech

Who does Testing?

It depends on the process and the associated stakeholders of the project(s). In the IT industry, large companies have a team with responsibilities to evaluate the developed software in context of the given requirements. Moreover, developers also conduct testing which is called Unit Testing. In most cases, the following professionals are involved in testing a system within their respective capacities:

- ☐ Software Tester
- ☐ Software Developer
- ☐ Project Lead/Manager
- ☐ End User

Different companies have different designations for people who test the software on the basis of their experience and knowledge such as Software Tester, Software Quality Assurance Engineer, QA Analyst, etc.

It is not possible to test the software at any time during its cycle. The next two sections state when testing should be started and when to end it during the SDLC.

When to Start Testing?

An early start to testing reduces the cost and time to rework and produce error-free software that is delivered to the client. However, in Software Development Life Cycle (SDLC), testing can be started from the Requirements Gathering phase and continued till the deployment of the software.

It also depends on the development model that is being used. For example, in the Waterfall model, formal testing is conducted in the testing phase; but in the incremental model, testing is performed at the end of every increment/iteration and the whole application is tested at the end.

Testing is done in different forms at every phase of SDLC:

- During the requirement gathering phase, the analysis and verification of requirements are also considered as testing.
- Reviewing the design in the design phase with the intent to improve the design is also considered as testing.
- Testing performed by a developer on completion of the code is also categorized as testing.

When to Stop Testing?

It is difficult to determine when to stop testing, as testing is a never-ending process and no one can claim that software is 100% tested. The following aspects are to be considered for stopping the testing process:

- Testing Deadlines
- Completion of test case execution

- Completion of functional and code coverage to a certain point
- Bug rate falls below a certain level and no high-priority bugs are identified
- Management decision

Verification & Validation

These two terms are very confusing for most people, who use them interchangeably. The following table highlights the differences between verification and validation.

S.N.	Verification	Validation
1	Verification addresses the concern: "Are you building it right?"	Validation addresses the concern: "Are You are building the right thing?"
2	Ensures that the software system Meets all the functionality.	Ensures that the functionalities meet The intended behavior.
3	Verification takes place first and includes the checking for documentation, code, etc.	Validation occurs after verification and mainly involves the checking of the overall product.
4	Done by developers.	Done by testers.
5	It has static activities, as it includes collecting reviews, walkthroughs, and Inspections to verify software.	It has dynamic activities, as it includes executing the software against the Requirements.
6	It is an objective process and no subjective decision should be needed To verify a software.	It is a subjective process and involves subjective decisions on how well a Software works.

GIPS Infotech

Software Development Life Cycle

Let us look at the Traditional Software Development life cycle Vs presently or most commonly used life cycle.

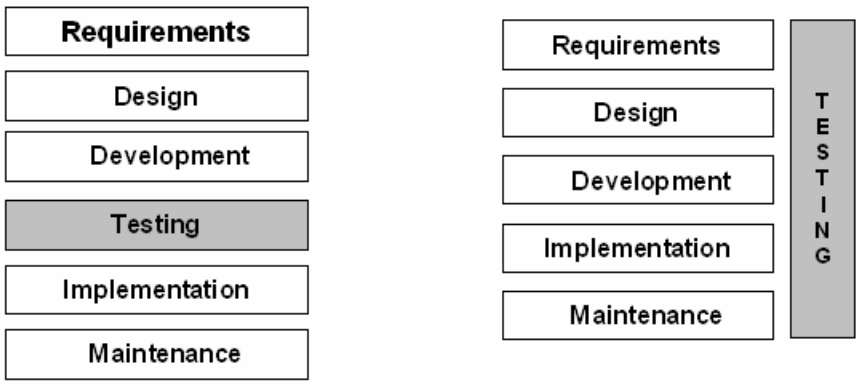


Fig A (Traditional)

Fig B (Most commonly used)

In the above Fig A, the Testing Phase comes after the Development or coding is complete and before the product is launched and goes into Maintenance phase. We have some disadvantages using this model - cost of fixing errors will be high because we are not able to find errors until coding is completed. If there is error at Requirements phase then all phases should be changed. So, total cost becomes very high.

The Fig B shows the recommended Test Process involves testing in every phase of the life cycle. During the Requirements phase, the emphasis is upon validation to determine that the defined requirements meet the needs of the customers. During Design and Development phases, the emphasis is on verification to ensure that the design and program accomplish the defined requirements. During the Test and Installation phases, the emphasis is on inspection to determine that the implemented system meets the system specification. During the maintenance phases, the system will be re-tested to determine that the changes work and that the unchanged portion continues to work.

If tester used the fig B approach most useful and significant always as because testing involves at early stage of SDLC which help to expose any ambiguities or inconsistencies in the requirement specification. It reduces the cost for fixing the defect since defects will be found in early stages

The thought process of designing tests early in the life cycle (verifying the test basis via test design) can help to prevent defects from being introduced into code. Reviews of documents (e.g. requirements) also help to prevent defects appearing in the code.

Involving software testing in all phases of the software development life cycle has become a necessity as part of the software quality assurance process. Right from the Requirements study till the implementation, testing is being involved at every phase. The V-Model of the Software Testing Life Cycle along with the Software Development Life cycle work as verification and validations

Fig A:

Advantages:

- ☐ Testing is inherent to every phase of the waterfall model
- ☐ It is an enforced disciplined approach
- ☐ It is documentation driven, that is, documentation is produced at every stage.

Disadvantages:

The disadvantage of waterfall development is that it does not allow for much reflection or revision. Once an application is in the testing stage, it is very difficult to go back and change something that was not well-thought out in the concept stage. Alternatives to the waterfall model include joint application development (JAD), rapid application development (RAD), synch and stabilize, build and fix, and the spiral model. It increases cost and time for fixing the defect since defect would be found at later stages.

Fig B

Advantages:

- ☐ In V model the testing is commenced at the beginning unlike in the traditional models where testing is given attention after the coding phase which causes failures and huge cost to repair.
- ☐ It's a proactive model.
- ☐ It reduces the cost and time for fixing the defect since defects will be found at early stages.

Software Testing Life Cycle STLC

Software Testing Life Cycle (STLC) is the testing process which is executed in systematic and planned manner. In STLC process, different activities are carried out to improve the quality of the product. Let's quickly see what all stages are involved in typical Software Testing Life Cycle (STLC).

Following steps are involved in Software Testing Life Cycle (STLC). Each step is having its own Entry Criteria and deliverable.

- Requirement Analysis
- Test Planning
- Test Case Development
- Environment Setup
- Test Execution
- Test Cycle Closure

Ideally, the next step is based on previous step or we can say next step cannot be started unless and until previous step is completed. It is possible in the ideal situation, but practically it is not always true.

GIPS Infotech

1. Requirement Analysis

Requirement Analysis is the very first step in Software Testing Life Cycle (STLC). In this step, Quality Assurance (QA) team understands the requirement in terms of what we will testing & figure out the testable requirements. If any conflict, missing or not understood any requirement, then QA team follow up with the various stakeholders like Business Analyst, System Architecture, Client, Technical Manager/Lead etc. to better understand the detail knowledge of requirement.

From very first step QA involved in the where STLC which helps to prevent the introducing defects into Software under test. The requirements can be either Functional or Non-Functional like Performance, Security testing. Also, requirement and Automation feasibility of the project can be done in this stage (if applicable)

Entry Criteria	Activities	Deliverable
Following documents should be available: – Requirements Specification. – Application architectural Along with above documents Acceptance criteria should be well defined.	Prepare the list of questions or queries and get resolved from Business Analyst, System Architecture, Client, Technical Manager/Lead etc. Make out the list for what all Types of Tests performed like Functional, Security, and Performance etc. Define the testing focus and priorities. List down the Test environment details where	List of questions with all answers to be resolved from business i.e. testable requirements Automation feasibility report (if applicable)

	testing activities will be carried out. Checkout the Automation feasibility if required & prepare the Automation feasibility report.	

2. Test Planning

Test Planning is most important phase of Software testing life cycle where all testing strategy is defined. This phase also called as Test Strategy phase. In this phase typically Test Manager (or Test Lead based on company to company) involved to determine the effort and cost estimates for entire project. This phase will be kicked off once the requirement gathering phase is completed & based on the requirement analysis, start preparing the Test Plan. The Result of Test Planning phase will be Test Plan or Test strategy & Testing Effort estimation documents. Once test planning phase is completed the QA team can start with test cases development activity.

Entry Criteria	Activities	Deliverable
Requirements Documents (Updated version of unclear or missing requirement). Automation feasibility report.	Define Objective & scope of the project. List down the testing types involved in the STLC. Test effort estimation and resource planning. Selection of testing tool if required. Define the testing process overview. Define the test environment required for entire project. Prepare the test schedules. Define the control procedures. Determining roles and responsibilities. List down the testing deliverable. Define the entry criteria, suspension criteria, resumption criteria and exit criteria. Define the risk involved if any.	Test Plan or Test strategy document. Testing Effort estimation document.

3. Test Case Development

The test case development activity is started once the test planning activity is finished. This is the phase of STLC where testing team write down the detailed test cases. Along with test cases testing team also prepare the test data if any required for testing. Once the test cases are ready then peer members or QA lead review these test cases.

Also, the Requirement Traceability Matrix (RTM) is prepared. The Requirement Traceability Matrix is an industry-accepted format for tracking requirements where each test case is mapped with the requirement. Using this RTM, we can track backward & forward traceability.

Entry Criteria	Activities	Deliverable
Requirements Documents (Updated version of unclear or missing requirement). Automation feasibility report.	Preparation of test cases. Preparation of test automation scripts (if required). Re-requisite test data preparation for executing test cases.	Test cases. Test data. Test Automation Scripts (if required).

4. Test Environment Setup

Setting up the test environment is vital part of the STLC. Basically, test environment decides on which conditions software is tested. This is independent activity and can be started parallel with Test Case Development. In process of setting up testing environment test team is not involved in it. Based on company to company may be developer or customer creates the testing environment. Meanwhile testing team should prepare the smoke test cases to check the readiness of the test environment setup.

Entry Criteria	Activities	Deliverable
Test Plan is available. Smoke Test cases are available. Test data is available.	Analyze the requirements and prepare the list of Software & hardware required to set up test environment. Setup the test environment. Once the Test Environment is setup execute the Smoke test cases to check the readiness of the test environment.	Test Environment will be ready with test data. Result of Smoke Test cases.

5. Test Execution

Once the preparation of Test Case Development and Test Environment setup is completed then test execution phase can be kicked off. In this phase testing team start executing test cases based on prepared test planning & prepared test cases in the prior step.

Once the test case is passed then same can be marked as Passed. If any test case is failed then corresponding defect can be reported to developer team via bug tracking system & bug can be linked for corresponding test case for further analysis. Ideally every failed test case should be associated with at least single bug. Using this linking we can get the failed test case with bug associated with it. Once the bug fixed by development team then same test case can be executed based on your test planning.

If any of the test cases are blocked due to any defect then such test cases can be marked as Blocked, so we can get the report based on how many test cases passed, failed, blocked or not run etc. Once the defects are fixed, same Failed or Blocked test cases can be executed again to retest the functionality.

Entry Criteria	Activities	Deliverable
Test Plan or Test strategy document. Test cases. Test data.	Based on test planning execute the test cases. Mark status of test cases like Passed, Failed, Blocked, Not Run etc. Assign Bug Id for all Failed and Blocked test cases. Do Retesting once the defects are fixed. Track the defects to closure.	Test case execution report. Defect report.

6. Test Cycle Closure

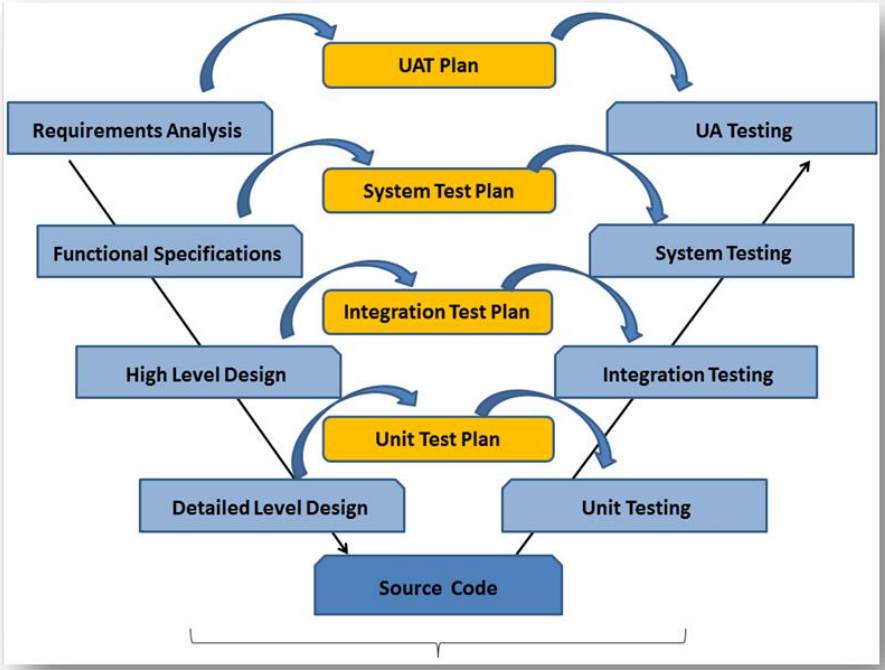
Call out the testing team member meeting & evaluate cycle completion criteria based on Test coverage, Quality, Cost, Time, Critical Business Objectives, and Software. Discuss what all went good, which area needs to be improve & taking the lessons from current STLC as input to upcoming test cycles, which will help to improve bottleneck in the STLC process. Test case & bug report will analyze to find out the defect distribution by type and severity. Once complete the test cycle then test closure report & Test metrics will be prepared. Test result analysis to find out the defect distribution by type and severity.

Entry Criteria	Activities	Deliverable
Test case execution is completed Test case Execution report Defect report	Evaluate cycle completion criteria based on Test coverage, Quality, Cost, Time, Critical Business Objectives, and Software Prepare test metrics based on the above parameters. Prepare Test closure report Share best practices for any similar projects in future	Test Closure report Test metrics

GIPS Infotech

The V- Model of Software Development

Software Testing has been accepted as a separate discipline to the extent that there is a separate life cycle for the testing activity. Involving software testing in all phases of the software development life cycle has become a necessity as part of the software quality assurance process. Right from the Requirements study till the implementation, there needs to be testing done on Software testing every phase. The V-Model of the Software Testing Life Cycle along with the Software Development Life cycle given below indicates the various phases or levels of testing.



Module-II

Myths

Myths about Software Testing

GIPS Infotech

Myths

Given below are some of the most common myths about software testing.

Myth 1: Testing is too expensive

Reality: There is a saying, pay less for testing during software development or pay more for maintenance or correction later. Early testing saves both time and cost in many aspects, however reducing the cost without testing may result in improper design of a software application rendering the product useless.

Myth 2: Testing is Time-Consuming

Reality: During the SDLC phases, testing is never a time-consuming process. However diagnosing and fixing the errors identified during proper testing is a time-consuming but productive activity.

Myth 3: Only Fully Developed Products are tested

Reality: No doubt, testing depends on the source code but reviewing requirements and developing test cases is independent from the developed code. However iterative or incremental approach as a development life cycle model may reduce the dependency of testing on the fully developed software.

Myth 4: Complete Testing is Possible

Reality: It becomes an issue when a client or tester thinks that complete testing is possible. It is possible that all paths have been tested by the team but occurrence of complete testing is never possible. There might be some scenarios that are never executed by the test team or the client during the software development life cycle and may be executed once the project has been deployed.

Myth 5: Tested Software is Bug-Free

Reality: This is a very common myth that the clients, project managers, and the management team believes in. No one can claim with absolute certainty that a software application is 100% bug-free even if a tester with superb testing skills has tested the application.

Myth 6: Missed Defects are due to Testers

Reality: It is not a correct approach to blame testers for bugs that remain in the application even after testing has been performed. This myth relates to Time, Cost, and Requirements changing Constraints. However the test strategy may also result in bugs being missed by the testing team.

GIPS Infotech

Myth 7: Testers are Responsible for Quality of Product

Reality: It is a very common misinterpretation that only testers or the testing team should be responsible for product quality. Testers' responsibilities include the

Identification of bugs to the stakeholders and then it is their decision whether they will fix the bug or release the software. Releasing the software at the time puts more pressure on the testers, as they will be blamed for any error.

Myth 8: Test Automation should be used wherever Possible to Reduce Time

Reality: Yes, it is true that Test Automation reduces the testing time, but it is not possible to start test automation at any time during software development. Test automaton should be started when the software has been manually tested and is stable to some extent. Moreover, test automation can never be used if requirements keep changing.

Myth 9: Anyone can Test a Software Application

Reality: People outside the IT industry think and even believe that anyone can test software and testing is not a creative job. However testers know very well that this is a myth. Thinking alternative scenarios, try to crash software with the intent to explore potential bugs is not possible for the person who developed it.

Myth 10: A Tester's Only Task is to find Bugs

Reality: Finding bugs in software is the task of the testers, but at the same time, they are domain experts of the particular software. Developers are only responsible for the specific component or area that is assigned to them but testers understand the overall workings of the software, what the dependencies are, and the impacts of one module on another module.

GIPS Infotech

Module-III

QA, QC and Testing

- ☐ Testing, Quality Assurance, and Quality Control
- ☐ Audit and Inspection
- ☐ Testing and Debugging

GIPS Infotech

QA, QC and Testing

Testing, Quality Assurance, and Quality Control

Most people get confused when it comes to pin down the differences among Quality Assurance, Quality Control, and Testing. Although they are interrelated and to some extent, they can be considered as same activities, but there exist distinguishing points that set them apart. The following table lists the points that differentiate QA, QC, and Testing.

Quality Assurance	Quality Control	Testing
QA includes activities that ensure the implementation of processes, procedures and standards in context to verification of developed software and intended requirements.	It includes activities that ensure the verification of a developed software with respect to documented (or not in some cases) requirements.	It includes activities that ensure the identification of bugs/error/defects in a software.
Focuses on processes and procedures rather than conducting actual testing on the system.	Focuses on actual testing by executing the software with an aim to identify bug/defect through implementation of procedures and process.	Focuses on actual testing.
Process-oriented activities.	Product-oriented activities.	Product-oriented activities
Preventive activities.	It is a corrective process.	It is a preventive process
It is a subset of Software Test Life Cycle (STLC).	QC can be considered as the Subset of Quality Assurance.	Testing is the subset of Quality Control.

Audit and Inspection

Audit: It is a systematic process to determine how the actual testing process is conducted within an organization or a team. Generally, it is an independent examination of processes involved during the testing of software. As per IEEE, it is a review of Documented processes that organizations implement and follow. Types of audit include Legal Compliance Audit, Internal Audit, and System Audit.

Inspection: It is a formal technique that involves formal or informal technical reviews of any artifact by identifying any error or gap. As per IEEE94, inspection is a formal evaluation technique in which software requirements, designs, or codes are examined in detail by a person or a group other than the author to detect faults, violations of development standards, and other problems. Formal inspection meetings may include the following processes: Planning, Overview Preparation, Inspection Meeting, Rework, and Follow-up.

Testing and Debugging

Testing: It involves identifying bug/error/defect in software without correcting it. Normally professionals with a quality assurance background are involved in bug identification. Testing is performed in the testing phase.

Debugging: It involves identifying, isolating, and fixing the problems/bugs. Developers who code the software conduct debugging upon encountering an error in the code. Debugging is a part of White Box Testing or Unit Testing. Debugging can be performed in the development phase while conducting Unit Testing or in phases while fixing the reported bugs.

GIPS Infotech

Module-IV

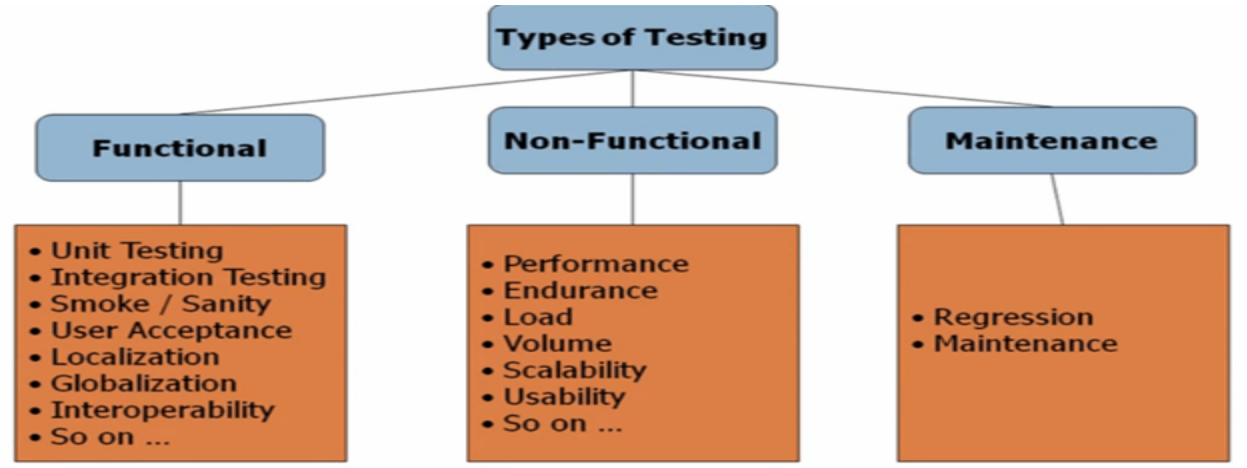
Types of Testing

- ☐ Functional Testing
- ☐ Non- Functional Testing
- ☐ Maintenance Testing
- ☐ Manual and Automation Testing

GIPS Infotech

Type of Testing

This section describes the different types of testing that may be used to test software during SDLC categorized in below manner.



Functional Testing

This is a type of black-box testing that is based on the specifications of the software that is to be tested. The application is tested by providing input and then the results are examined that need to conform to the functionality it was intended for. Functional testing of software is conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements.

There are five steps that are involved while testing an application for functionality.

Steps	Description
1	The determination of the functionality that the intended application is meant to perform.
2	The creation of test data based on the specifications of the application.
3	The output based on the test data and the specifications of the application.
4	The writing of test scenarios and the execution of test cases
5	The comparison of actual and expected results based on the executed test cases.

An effective testing practice will see the above steps applied to the testing policies of every organization and hence it will make sure that the organization maintains the strictest of standards when it comes to software quality.

Unit Testing

Testing of a unit or smallest piece of software to verify if it satisfies its functional specifications or its intended design structure.

Integration Testing

Testing which takes place as sub elements are combined (i.e., integrated) to form higher-level elements

Smoke Testing

Smoke Testing is a testing technique that is inspired from hardware testing, which checks for the smoke from the hardware components once the hardware's power is switched on. Similarly in Software testing context, smoke testing refers to testing the basic functionality of the build.

If the Test fails, build is declared as unstable and it is NOT tested anymore until the smoke test of the build passes.

Smoke Testing - Features:

- Identifying the business critical functionalities that a product must satisfy.
- Designing and executing the basic functionalities of the application.
- Ensuring that the smoke test passes each and every build in order to proceed with the testing.
- Smoke Tests enables uncovering obvious errors which saves time and effort of test team.
- Smoke Tests can be manual or automated.

Sanity Testing

Sanity testing is a software testing technique performed by the test team for some basic tests. The aim of basic test is to be conducted whenever a new build is received for testing. The terminologies such as Smoke Test or Build Verification Test or Basic Acceptance Test or Sanity Test are interchangeably used; however, each one of them is used under a slightly different scenario.

Sanity test is usually unscripted, helps to identify the dependent missing functionalities. It is used to determine if the section of the application is still working after a minor change. Sanity testing can be narrow and deep. Sanity test is a narrow regression test that focuses on one or a few areas of functionality.

Regression Testing

Selective re-testing of a system to verify the modification (bug fixes) have not caused unintended effects and that system still complies with its specified requirements

GIPS Infotech

System Testing

Testing the software for the required specifications on the intended hardware

Acceptance Testing

Formal testing conducted to determine whether or not a system satisfies its acceptance criteria, which enables a customer to determine whether to accept the system or not.

- **Alpha Testing**
  Testing of a software product or system conducted at the developer's site by the customer
- **Beta Testing**
  Testing conducted at one or more customer sites by the end user of a delivered software product system.

Non-Functional Testing

In order to ensure that the system is ready to go live it is necessary to go beyond just functional testing. Non-Functional Testing is designed to evaluate the readiness of the system according to several criteria not covered by functional testing. These criteria include but not limited to:

I. Performance Testing
II. Disaster Recovery
III. Security

I. What is Performance Testing

Performance testing is the process of determining the speed or effectiveness of a computer, network, software program or device. This process can involve quantitative tests done in a lab, such as measuring the response time or the number of MIPS (millions of instructions per second) at which a system functions

Following are the classification of Performance Testing

1. Load Testing
2. Stress Testing
3. Volume Testing

1. Load Testing

Load testing is subjecting a system to a statistically representative (usually) load. The two main reasons for using such loads are in support of software reliability testing and in performance testing. In performance testing, load is varied from a minimum (zero) to the maximum level the system can sustain without running out of resources or having, transactions suffer (application specific) excessive delay.

2. Stress Testing

Stress testing is subjecting a system to an unreasonable load while denying it the resources (e.g., RAM, disc, interrupts etc.) needed to process that load. The idea is to stress a system to the breaking point in order to find bugs that will make that break potentially harmful. The system is not expected to process the overload without adequate resources, but to behave (e.g., fail) in a decent manner (e.g., not corrupting or losing data). Bugs and failure modes discovered under stress testing may or may not be repaired depending on the application, the failure mode, consequences, etc. The load (incoming transaction stream) in stress testing is often deliberately distorted so as to force the system into resource depletion.

3. Volume Testing

Testing which confirms that any values that may become large over time (such as accumulated counts, logs, and data files), can be accommodated by the program and will not cause the program to stop working or degrade its operation in any manner

II. What is Disaster Recovery?

Disaster Recovery is a procedure defined for any application that is deployed in LIVE. It ensures that when a system breaks/stops abnormally it comes back to its original state and continues to function as it was before. The disaster recovery plan should be developed in such a way that the business is not impacted or least impacted due to system malfunctions.

III. What is Security Testing?

Techniques used to confirm the design and/or operational effectiveness of security controls implemented within a system. Examples: Attack and penetration studies to determine whether adequate controls have been implemented to prevent breach of system controls and processes Example Password strength testing by using tools (“password crackers”)

GIPS Infotech

Security testing involves testing software in order to identify any flaws and gaps from security and vulnerability point of view. Listed below are the main aspects that security testing should ensure:

- Confidentiality
- Integrity
- Authentication
- Availability
- Authorization
- Software data is secure
- Software is according to all security regulations
- Input checking and validation
- SQL insertion attacks
- Injection flaws
- Session management issues
- Cross-site scripting attacks
- Buffer overflows vulnerabilities
- Directory traversal attacks

Maintenance Testing

Regression Testing

If a piece of Software is modified for any reason testing needs to be done to ensure that it works as specified and that it has not negatively impacted any functionality that it offered previously. This is known as Regression Testing.

Regression testing attempts to verify:

- That the application works as specified even after the changes, additions, modification were made to it
- The original functionality continues to work as specified even after changes, additions, modification to the software application
- The changes, additions, modification to the software application have not introduced any new bugs

- **When to go for Regression Testing**

Regression Testing plays an important role in any Scenario where a change has been made to a previously tested software code. Regression Testing is hence an important aspect in various Software Methodologies where software changes enhancements occur frequently. Any Software Development Project is invariably faced with requests for changing Design, code, features or all of them. Some Development Methodologies embrace change.

For example 'Extreme Programming' Methodology advocates applying small incremental changes to the system based on the end user feedback.

Each change implies more Regression Testing needs to be done to ensure that the System meets the Project Goals.

Any Software change can cause existing functionality to break. Changes to a Software component could impact dependent Components. It is commonly observed that a Software fix could cause other bugs. All this affects the quality and reliability of the system. Hence Regression Testing, since it aims to verify all this, is very important.

Making Regression Testing Cost Effective

Every time a change occurs one or more of the following scenarios may occur:

- More Functionality may be added to the system
- More complexity may be added to the system
- New bugs may be introduced
- New vulnerabilities may be introduced in the system
- System may tend to become more and more fragile with each change

After the change the new functionality may have to be tested along with all the original functionality. With each change Regression Testing could become more and more costly. To make the Regression Testing Cost Effective and yet ensure good coverage one or more of the following techniques may be applied:

Test Automation: If the Test cases are automated the test cases may be executed using scripts after each change is introduced in the system. The execution of test cases in this way helps eliminate oversight, human errors; it may also result in faster and cheaper execution of Test cases. However there is cost involved in building the scripts.

Selective Testing: Some teams choose the test cases selectively to execute. They do not execute all the Test Cases during the Regression Testing. They test only what they decide is relevant. This helps reduce the Testing Time and Effort but there is also a risk of leaving out the impacted code which may be indirectly related.

- **Regression Testing – What to Test?**

Since Regression Testing tends to verify the software application after a change has been made everything that may be impacted by the change should be tested during Regression Testing.

Generally the following areas are covered during Regression Testing:

- Any functionality that was addressed by the change
- Original Functionality of the system
- Performance of the System after the change was introduced

GIPS Infotech

- **Regression Testing – How to Test?**

Like any other Testing Regression Testing Needs proper planning. For an Effective Regression Testing following steps are necessary:

Create a Regression Test Plan: Test Plan identified Focus Areas, Strategy, Test Entry and Exit Criteria. It can also outline Testing Prerequisites, Responsibilities, etc.

Create Test Cases: Test Cases that cover all the necessary areas are important. They describe what to Test, Steps needed to test, Inputs and Expected Outputs. Test Cases used for Regression Testing should specifically cover the functionality addressed by the change and all components affected by the change. The Regression Test case may also include the testing of the performance of the components and the application after the change(s) were done.

Defect Tracking: As in all other Testing Levels and Types it is important Defects are tracked systematically, otherwise it undermines the Testing Effort.

Manual Vs Automation Testing

Manual Testing

Manual testing includes testing a software manually, i.e., without using any automated tool or any script. In this type, the tester takes over the role of an end-user and tests the software to identify any unexpected behavior or bug. There are different stages for manual testing such as unit testing, integration testing, system testing, and user acceptance testing.

Testers use test plans, test cases, or test scenarios to test software to ensure the completeness of testing. Manual testing also includes exploratory testing, as testers explore the software to identify errors in it.

Automation Testing

Automation testing, which is also known as Test Automation, is when the tester writes scripts and uses software to test the product. This process involves automation of a manual process. Automation Testing is used to re-run the test scenarios that were performed manually, quickly, and repeatedly.

Apart from regression testing, automation testing is also used to test the application from load, performance, and stress point of view. It increases the test coverage, improves accuracy, and saves time and money in comparison to manual testing.

- **What to Automate?**

It is not possible to automate everything in software. The areas at which a user can make transactions such as the login form or registration forms, any area where large number of users can access the software simultaneously should be automated.

Furthermore, all GUI items, connections with databases, field validations, etc. can be efficiently tested by automating the manual process.

▪ When to Automate?

Test Automation should be used by considering the following aspects of software:

- Large and critical projects
- Projects that require testing the same areas frequently
- Requirements not changing frequently
- Accessing the application for load and performance with many virtual users
- Stable software with respect to manual testing
- Availability of time

▪ How to Automate?

Automation is done by using a supportive computer language like VB scripting and an automated software application. There are many tools available that can be used to write automation scripts. Before mentioning the tools, let us identify the process that can be used to automate the testing process:

- Identifying areas within a software for automation
- Selection of appropriate tool for test automation
- Writing test scripts
- Development of test suits
- Execution of scripts
- Create result reports
- Identify any potential bug or performance issues

GIPS Infotech

Module-V

Testing Methods

- Black-Box
- White-Box
- Grey-Box

GIPS Infotech

Testing Methods

There are different methods that can be used for software testing. This chapter briefly describes the methods available.

Black-Box Testing

The technique of testing without having any knowledge of the interior workings of the application is called black-box testing. The tester is oblivious to the system architecture and does not have access to the source code. Typically, while performing a black-box test, a tester will interact with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon.

Black Box Testing Methods

1. **Graph Based Testing Methods**

- Black-box methods based on the nature of the relationships (links) among the program objects (nodes), test cases are designed to traverse the entire graph
- Transaction flow testing (nodes represent steps in some transaction and links represent logical connections between steps that need to be validated)
- Finite state modelling (nodes represent user observable states of the software and links represent transitions between states)
- Data flow modelling (nodes are data objects and links are transformations from one data object to another)
- Timing modelling (nodes are program objects and links are sequential connections between these objects, link weights are required execution times)

2. **Equivalence Partitioning**

- Black-box technique that divides the input domain into classes of data from which test cases can be derived.
- An ideal test case uncovers a class of errors that might require many arbitrary test cases to be executed before a general error is observed
- Equivalence class guidelines:
    - If input condition specifies a range, one valid and two invalid equivalence classes are defined
    - If an input condition requires a specific value, one valid and two invalid equivalence classes are defined
    - If an input condition specifies a member of a set, one valid and one invalid equivalence class is defined

GIPS Infotech

- If an input condition is Boolean, one valid and one invalid equivalence class is defined

3. **Boundary Value Analysis**

   Black-box technique that focuses on the boundaries of the input domain rather than its centres.

4. **Comparison Testing**

- Black-box testing for safety critical systems in which independently developed implementations of redundant systems are tested for conformance to specifications
- Often equivalence class partitioning is used to develop a common set of test cases for each implementation

5. **Specialized Testing**

- Graphical user interfaces
- Client/server architectures
- Documentation and help facilities
- Real-time systems
  - Task testing (test each time dependent task independently)
  - Behavioural testing (simulate system response to external events)
  - Inter task testing (check communications errors among tasks)
  - System testing (check interaction of integrated system software and hardware)

Advantages of Black Box Testing

- More effective on larger units of code than glass box testing
- Tester needs no knowledge of implementation, including specific programming languages
- Tester and programmer are independent of each other
- Tests are done from a user's point of view
- Will help to expose any ambiguities or inconsistencies in the specifications
- Test cases can be designed as soon as the specifications are complete

Disadvantages of Black Box Testing

- Only a small number of possible inputs can actually be tested, to test every possible input stream would take nearly forever
- Without clear and concise specifications, test cases are hard to design
- There may be unnecessary repetition of test inputs if the tester is not informed of test
- cases the programme has already tried
- May leave many program paths untested

Gips Infotech

- Cannot be directed toward specific segments of code which may be very complex
- (and therefore more error prone)
- Most testing related research has been directed toward glass box testing

White-Box Testing

White-box testing is the detailed investigation of internal logic and structure of the code. In order to perform white-box testing on an application, a tester needs to know the internal workings of the code. The tester needs to have a look inside the source code and find out which unit/chunk of the code is behaving inappropriately.

Synonyms for white box testing
- Glass Box testing
- Structural testing
- Clear Box testing
- Open Box Testing

Types of White Box Testing

1. **Basis Path Testing**

A testing mechanism proposed by McCabe whose aim is to derive a logical complexity measure of
a procedural design and use this as a guide for defining a basic set of execution paths. These are test cases that exercise basic set will execute every statement at least once.

2. **Control Structure Testing**
   a. **Conditions Testing**
      Condition testing aims to exercise all logical conditions in a program module. They may define:
      - Relational expression: (E1 op E2), where E1 and E2 are arithmetic expressions.
      - Simple condition: Boolean variable or relational expression, possibly proceeded by a
      - NOT operator.
      - Compound condition: composed of two or more simple conditions, Boolean operators and parentheses.
      - Boolean expression: Condition without Relational expressions.
   b. **Data Flow Testing**
      Selects test paths according to the location of definitions and use of variables.
   c. **Loop Testing**
      Loops fundamental to many algorithms. Can define loops as simple, concatenated, nested, and unstructured.

Advantages of White Box Testing
- Forces test developer to reason carefully about implementation
- Approximate the partitioning done by execution equivalence
- Reveals errors in "hidden" code
- Beneficent side-effects

GIPS Infotech

Disadvantages of White Box Testing

- Expensive
- Cases omitted in the code could be missed out.

Grey-Box Testing

Grey-box testing is a technique to test the application with having a limited knowledge of the internal workings of an application. In software testing, the phrase the more you know, the better carries a lot of weight while testing an application.

Mastering the domain of a system always gives the tester an edge over someone with limited domain knowledge. Unlike black-box testing, where the tester only tests the application's user interface; in grey-box testing, the tester has access to design documents and the database. Having this knowledge, a tester can prepare better test data and test scenarios while making a test plan.

Advantages of Grey-Box Testing

- Offers combined benefits of black-box and white-box testing wherever possible.
- Grey box testers don't rely on the source code; instead they rely on interface definition and functional specifications.
- Based on the limited information available, a grey-box tester can design excellent test scenarios especially around communication protocols and data type handling.
- The test is done from the point of view if the user and not the designer

Disadvantages of Grey-Box Testing

- Since the access to source code is not available, the ability to go over the code and test coverage is limited.
- The tests can be redundant if the software designer has already run a test case.
- Testing every possible input stream is unrealistic because it would take an unreasonable amount of time; therefore, many program paths will go untested

GIPS Infotech

A Comparison of Testing Methods

The following table lists the points that differentiate black-box testing, grey-box testing, and white-box testing.

Black-Box Testing	Grey-Box Testing	White-Box Testing
The internal workings of an application need not be known.	The tester has limited knowledge of the internal workings of the application.	Tester has full knowledge of the internal workings of the application.
Also known as closed-box testing, data-driven testing, or functional testing.	Also known as translucent testing, as the tester has limited knowledge of the insides of the application.	Also known as clear-box testing, structural testing, or code-based testing.
Performed by end-users and also by testers and developers.	Performed by end-users and also by testers and developers.	Normally done by testers and developers.
Testing is based on external expectations - Internal behavior of the application is unknown.	Testing is done on the basis of high-level database diagrams and data flow diagrams.	Internal workings are fully known and the tester can design test data accordingly.
It is exhaustive and the least time-consuming.	Partly time-consuming and exhaustive.	The most exhaustive and time-consuming type of testing.
Not suited for algorithm testing.	Not suited for algorithm testing.	Suited for algorithm testing.
This can only be done by trial-and-error method.	Data domains and internal boundaries can be tested, if known.	Data domains and internal boundaries can be better tested.

GIPS Infotech

Module-VI

Levels of Testing

- Unit Testing
- Integration Testing
- System Testing
- Regression Testing
- Acceptance testing
- Non Functional Testing
- Migration/Upgrade Testing
- Portability Testing

GIPS Infotech

Levels of Testing

There are different levels during the process of testing. In this chapter, a brief description is provided about these levels.

Levels of testing include different methodologies that can be used while conducting software testing from functional and non functional point of views. The main levels of software testing are described below:

Unit Testing

Unit testing is the testing of individual hardware or software units or groups of related units. Using white box testing techniques, testers (usually the developers creating the code implementation) verify that the code does what it is intended to do at a very low structural level. For example, the tester will write some test code that will call a method with certain parameters and will ensure that the return value of this method is as expected. Looking at the code itself, the tester might notice that there is a branch (if then) and might write a second test case to go down the path not executed by the first test case. When available, the tester will examine the low-level design of the code; otherwise, the tester will examine the structure of the code by looking at the code itself. Unit testing is generally done within a class or a component.
The goal of unit testing is to isolate each part of the program and show that individual parts are correct in terms of requirements and functionality.

Benefits of Unit Testing

Testing a software Unit/component is basically done to resolve the following issues:

- Check whether the component meets its specification and fulfil its functional requirements.
- Check whether the correct and complete structural and interaction requirements, specified before the development of the component, are reflected in the implemented software system.
- Tests all paths – greater chance of detecting errors
- Best way to ensure good coverage
- Unearths logic errors and incorrect assumptions when coding for "special cases"
- Tests are designed to ensure that code conforms to design
- Boundary conditions get tested

Limitations of Unit Testing

Testing cannot catch each and every bug in an application. It is impossible to evaluate every execution path in every software application. The same is the case with unit testing.

There is a limit to the number of scenarios and test data that a developer can use to verify a source code. After having exhausted all the options, there is no choice but to stop unit testing and merge the code segment with other units.

Integration Testing

A system is made up of multiple components or modules that can comprise of hardware and software. Integration is defined as the set of interactions among components. Testing the interaction between the modules and interactions with other systems externally is called Integration Testing.

There may be more than one level of integration testing and it may be carried out on test objects of varying size. For example:

- Component integration testing tests the interactions between software components and is done after component testing;
- System integration testing tests the interactions between different systems and may be done after system testing

The three common strategies are as follows:

☐ Top-Down approach

☐ Bottom-Up approach

☐ Bi-Directional approach

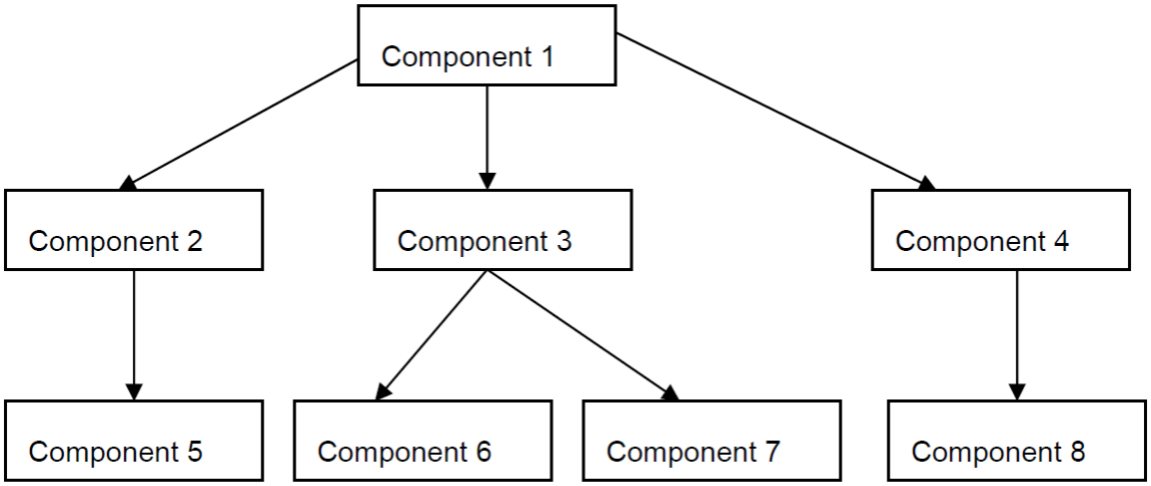
Top-Down Approach

Integration testing involves testing the topmost component interface with other components in the same order as you navigate from top to bottom, till you cover all the components at the bottom level.

Example:

To understand this methodology better, let us assume a new product/software development where components become available one after another in the order of component numbers specified in the Figure below. The integration starts with testing the interface between Component 1 and Component 2. To complete the integration testing, all interfaces mentioned in the figure below covering all the arrows, have to be tested together. The order in which the interfaces are to be tested is depicted in the table below.

GIPS Infotech

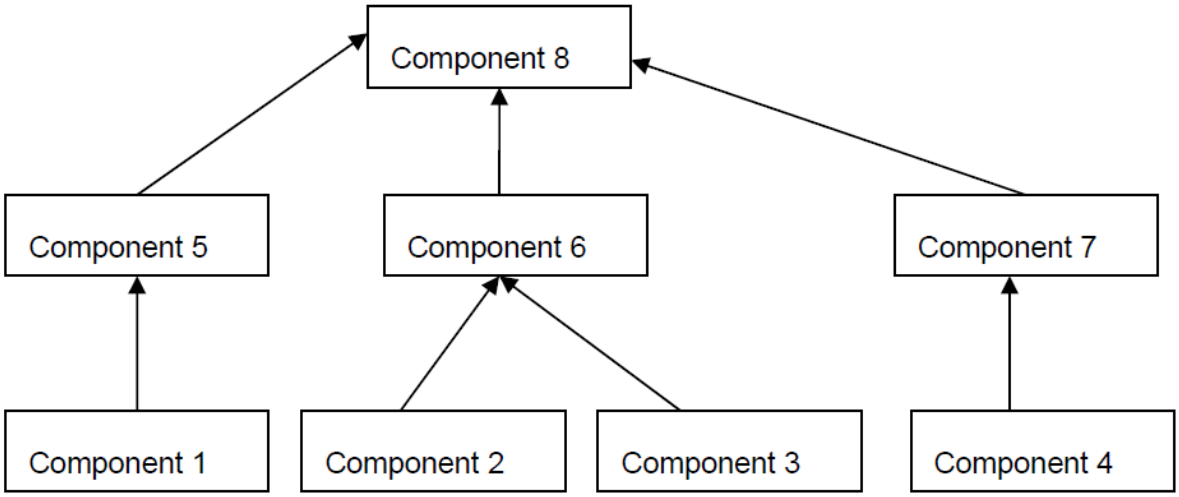


Order of testing interfaces for the above example

Step	Interfaces Tested
1	1-2
2	1-3
3	1-4
4	1-2-5
5	1-3-6
6	1-3-7
7	1-3-6-(3-7)
8	(1-2-5)-(1-3-6-(3-7))
9	1-4-8
10	(1-2-5)-(1-3-6-(3-7))-(1-4-8)

Bottom –Up Approach

Bottom-up integration is just the opposite of top-down integration, where the components for a new product development become available in reverse order, starting from the bottom. The navigation in bottom-up integration starts from component 1 covering all sub-systems, till component 8 is reached



Order of testing interfaces using bottom up integration for the above example

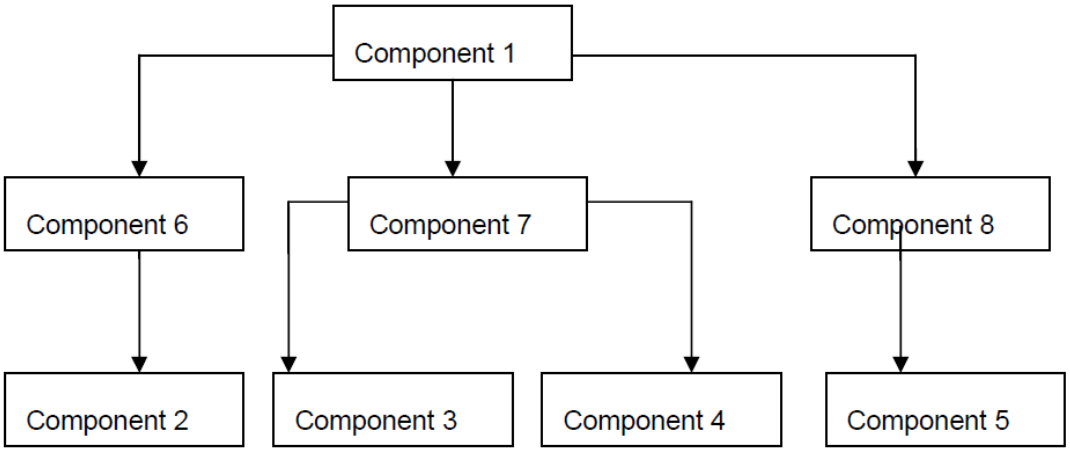
Step	Interfaces Tested
1	1-5
2	2-6,3-6
3	2-6-(3-6)
4	4-7
5	2-6-(3-6)-8
6	1-5-8
7	4-7-8
8	(1-5-8)(2-6-(3-6)-8)-(4-7-8)

Bi-Directional Approach

Bi-Directional integration is the combination of the top-down and bottom-up integration approaches used together to derive integration steps.

In this example, let us assume the software components become available in the order mentioned by the component numbers. The individual components 1,2,3,4 and 5 are tested separately and bi-directional integration is performed initially with the use of stubs and drivers. Drivers are used to provide upstream connectivity while stubs provide downstream connectivity. A driver is a function which redirects the requests to some other component and subs simulate the behaviour of a missing component. After the functionality of these integrated components is tested, the drivers and stubs are discarded. Once components 6,

7 and 8 become available, the integration methodology then focuses only on those components, as these are the components which need focus and are new. This approach is also called 'Sandwich integration'.



Order of testing interfaces using sandwich integration for the above example

Step	Interfaces Tested
1	6-2
2	7-3-4
3	8-5
4	(1-6-2)-(1-7-3-4)-(1-8-5)

Choosing Integration Method

Guidelines on selection of Integration method:

Sl.No	Factors	Suggested Integration Method
1	Clear requirements and design	Top-down
2	Dynamically changing requirements , design, architecture	Bottom-up
3	Changing architecture, stable design	Bi-direction

System Testing

System testing is defined as a testing phase conducted on the complete integrated system, to evaluate the system compliance with its specific requirements. A system is a complete set of integrated components that together deliver product functionality and features. A system can also be defined as a set of hardware, software, and other parts that together provide product features and solutions. In order to test the entire system, it is necessary to understand the product's behaviour as a whole. System testing helps in uncovering the defects that may not be directly attributable to a module or an interface. System testing brings out issues that are fundamental to design, architecture and code of the whole product.

System testing is the only phase of testing which tests the both functional and non-functional aspects of the product.
Functional testing involves testing a product's functionality and features. The testing focuses on real-time end user usage of the product and solutions.
Non-functional testing involves testing the product's quality factors. The performance, reliability, robustness, stability of the system is all testing under non functional testing.

Benefits of System Testing

- Bring in customer perspective in testing
- Provide a "fresh pair of eyes" to discover defects not found earlier by testing
- Test product behaviour in a holistic, complete and realistic environment
- Test both functional and non-functional aspects of the product
- Build confidence in the product
- Analyze and reduce the risk of releasing the product
- Ensure all requirements are met and ready the product for acceptance testing
- Better perspective can be provided if it is done by an testing team that is not part of the development team

Regression Testing

Whenever a change in a software application is made, it is quite possible that other areas within the application have been affected by this change. Regression testing is performed to verify that a fixed bug hasn't resulted in another functionality or business rule violation. The intent of regression testing is to ensure that a change, such as a bug fix should not result in another fault being uncovered in the application.

Regression testing is important because of the following reasons:

- Minimize the gaps in testing when an application with changes made must be tested.
- Testing the new changes to verify that the changes made did not affect any other area of the application.
- Mitigates risks when regression testing is performed on the application.
- Test coverage is increased without compromising timelines.
- Increase speed to market the product.

Acceptance Testing

Acceptance testing is a phase after system testing that is normally done by the customers or representatives of the customer. The customer defines a set of test cases that will be executed to qualify and accept the product. These test cases are executed by the customers themselves to quickly judge the quality of the product before deciding to buy/deploy the product. Acceptance test cases are normally small in number and are not written with the intention of finding defects.
Typical forms of Acceptance testing include the following:

User Acceptance Testing is often the final step before rolling out the application. Usually the end users who will be using the applications test the application before 'accepting' the application. This type of testing gives the end users the confidence that the application being delivered to them meets their requirements. This testing also helps nail bugs related to usability of the application

Operational Acceptance Testing (OAT) is becoming critically important to organizations that deliver and maintain complete software systems and complex IT infrastructures. In general, OAT addresses the non-functional attributes of a system. With the specific exclusions of performance and security, which we consider to be areas of specialty in their own right, OAT concentrates on such areas as maintainability, reliability, recoverability, install ability, compatibility and conformance, to name a few.

Contract Acceptance Testing is performed against a contract's acceptance criteria for producing custom developed software

Regulation Acceptance Testing is performed against any regulations that must be adhered to, such as governmental, legal or safety regulations

Alpha Testing is simulated or actual operational testing by potential users/customers or an independent test team at the developers' site. Alpha testing is often employed for off-the-shelf software as a form of internal acceptance testing, before the software goes to beta testing

Beta Testing comes after alpha testing. Versions of the software, known as beta versions, are released to a limited audience outside of the company. The software is released to groups of people so that further testing can ensure the product has few faults or bugs. Sometimes, beta versions are made available to the open public to increase the feedback field to a maximal number of future users.

GIPS Infotech

Non-Functional Testing

This section is based upon testing an application from its non-functional attributes. Non-functional testing involves testing software from the requirements which are nonfunctional in nature but important such as performance, security, user interface, etc.

Some of the important and commonly used non-functional testing types are below.

Performance Testing

It is mostly used to identify any bottlenecks or performance issues rather than finding bugs in software. Performance testing can be either qualitative or quantitative and can be divided into different sub-types such as **Load testing** and **Stress testing**.

Usability Testing

Usability testing is a black-box technique and is used to identify any error(s) and improvements in the software by observing the users through their usage and operation.

According to Nielsen, usability can be defined in terms of five factors, i.e. efficiency of use, learn-ability, memory-ability, errors/safety, and satisfaction. According to him, the usability of a product will be good and the system is usable if it possesses the above factors.

Security Testing

Security testing involves testing software in order to identify any flaws and gaps from security and vulnerability point of view

Migration/Upgrade Testing

Data migration testing is carried out with the intention of identifying errors in the database that occurs due to migration / during post migration. One of the key elements of a successful data migration is the conversion of the business data from the source format to the target format so that the format conversion does not impact the functionality of the application. This data is often extremely valuable and needs to be treated with care. Before we migrate data to the target environment the data is cleaned up to avoid incomplete, inconsistent or inaccurate data in the environment. Data clean-up is a key element of data migration. Our risk based approach ensures that the critical data conversion is prioritized by the impact and likelihood of failure.
There can be the following types of data migration:

One time bulk migration: This is done usually as a part of application migration from legacy to re-engineered, modern systems. This is also done when moving a database from a lower version to higher versions.

Ongoing phased migration: In some cases the data in legacy would be transferred in a phased manner to the modern systems. In these cases the migration would be a batch processes scheduled at a regular frequency to migrate a pre-determined size of data.
Data migration can happen between source and target of various types:

- Legacy to RDBMS: For instance from Mainframe VSAM files to Oracle
- Hierarchical Database to RDBMS: For instance from IMS to Oracle
- RDBMS to RDBMS: For instance DB2 to Oracle
- Different versions of the same DBMS: Oracle 8i to Oracle 9i
- Same version of DBMS with schema changes

Portability Testing

Portability testing includes testing software with the aim to ensure its reusability and that it can be moved from another software as well. Following are the strategies that can be used for portability testing:

- Transferring installed software from one computer to another.
- Building executable (.exe) to run the software on different platforms.

Portability testing can be considered as one of the sub-parts of system testing, as this testing type includes overall testing of a software with respect to its usage over different environments. Computer hardware, operating systems, and browsers are the major focus of portability testing. Some of the pre-conditions for portability testing are as follows:

- Software should be designed and coded, keeping in mind the portability requirements.
- Unit testing has been performed on the associated components.
- Integration testing has been performed.
- Test environment has been established.

GIPS Infotech

Module-VII

Test Management

- ☐ Test Planning
- ☐ Test Strategy
- ☐ Testing Tools and techniques
- ☐ Test progress monitoring and control
- ☐ Test Closure

GIPS Infotech

Test Management

Test Management

Test management, process of managing the tests. A test management is also performed using tools to manage both types of tests, automated and manual, that have been previously specified by a test procedure.
Test management tools allow automatic generation of the requirement test matrix (RTM), which is an indication of functional coverage of the application under test (SUT).
Test Management tool often has multifunctional capabilities such as test ware management, test scheduling, the logging of results, test tracking, incident management and test reporting.

Test Management Responsibilities:

- Test Management has a clear set of roles and responsibilities for improving the quality of the product.
- Test management helps the development and maintenance of product metrics during the course of project.
- Test management enables developers to make sure that there is fewer design or coding faults.

Test planning

Test planning, the most important activity to ensure that there is initially a list of tasks and milestones in a baseline plan to track the progress of the project. It also defines the size of the test effort.
It is the main document often called as master test plan or a project test plan and usually developed during the early phase of the project.

Test Plan Document includes details of all necessary parameters as showing below

S.No.	Parameter	Description
1.	Test plan identifier	Unique identifying reference.
2.	Introduction	A brief introduction about the project and to the document.
3.	Test items	A test item is a software item that is the application under test.
4.	Features to be tested	A feature that needs to tested on the software.
5.	Features not to be tested	Identify the features and the reasons for not including as part of testing.

Gips Infotech

6.	Approach	Details about the overall approach to testing.
7.	Item pass/fail criteria	Documented whether a software item has passed or failed its test.
8.	Test deliverables	The deliverables that are delivered as part of the testing process, such as test plans, test specifications and test summary reports.
9.	Testing tasks	All tasks for planning and executing the testing.
10.	Environmental needs	Defining the environmental requirements such as hardware, software, OS, network configurations, tools required.
11.	Responsibilities	Lists the roles and responsibilities of the team members.
12.	Staffing and training needs	Captures the actual staffing requirements and any specific skills and training requirements.
13.	Schedule	States the important project delivery dates and key milestones.
14.	Risks and Mitigation	High-level project risks and assumptions and a mitigating plan for each identified risk.
15.	Approvals	Captures all approvers of the document, their titles and the sign off date.

Test Planning Activities:

- Determine the scope and the risks that need to be tested and that are NOT to be tested.
- Documenting Test Strategy.
- Making sure that the testing activities have been included.
- Deciding Entry and Exit criteria.
- Evaluating the test estimate.
- Planning when and how to test and deciding how the test results will be evaluated, and defining test exit criterion.
- The Test artefacts delivered as part of test execution.
- Defining the management information, including the metrics required and defect resolution and risk issues.
- Ensuring that the test documentation generates repeatable test assets.

What is a Test Plan?

A Test Plan can be defined as a document that describes the scope, approach, resources and schedule of intended test activities. It identifies test items, the features to be tested, the testing tasks, who will do each task, and any risks requiring contingency planning.

GIPS Infotech

The main purpose of preparing a Test Plan is that everyone concerned with the project are in sync with regards to the scope, responsibilities, deadlines and deliverables for the project. It is in this respect that reviews and a sign-off are very important since it means that everyone is in agreement of the contents of the test plan and this also helps in case of any dispute during the course of the project (especially between the developers and the testers).

Contents of a Test Plan

- Purpose
- Scope
- Test Approach
- Entry Criteria
- Resources
- Tasks / Responsibilities
- Exit Criteria
- Schedules / Milestones
- Hardware / Software Requirements
- Risks & Mitigation Plans
- Tools to be used
- Deliverables
- References
    - Procedures
    - Templates
    - Standards/Guidelines
- Annexure
- Sign-Off

Contents (in detail)

Purpose
This section should contain the purpose of preparing the test plan

Scope
This section should talk about the areas of the application which are to be tested by the QA team and specify those areas which are definitely out of scope (screens, database, mainframe processes etc).

Test Approach
This would contain details on how the testing is to be performed and whether any specific strategy is to be followed (including configuration management).

Entry Criteria
This section explains the various steps to be performed before the start of a test (i.e.) pre-requisites. For example: Timely environment set up, starting the web server / app server, successful implementation of the latest build etc.

Resources
This section should list out the people who would be involved in the project and their designation etc.

Tasks / Responsibilities

GIPS Infotech

This section talks about the tasks to be performed and the responsibilities assigned to the various members in the project.

Exit criteria
Contains tasks like bringing down the system / server, restoring system to pre-test environment, database refresh etc.

Schedules / Milestones
This sections deals with the final delivery date and the various milestone dates to be met in the course of the project.

Hardware / Software Requirements
This section would contain the details of PC's / servers required (with the configuration) to install the application or perform the testing; specific software that needs to be installed on the systems to get the application running or to connect to the database; connectivity related issues etc.

Risks & Mitigation Plans
This section should list out all the possible risks that can arise during the testing and the mitigation plans that the QA team plans to implement incase the risk actually turns into a reality.

Tools to be used
This would list out the testing tools or utilities (if any) that are to be used in the project (e.g.) WinRunner, Test Director, PCOM, WinSQL.

Deliverables
This section contains the various deliverables that are due to the client at various points of time (i.e.) daily, weekly, start of the project, end of the project etc. These could include Test Plans, Test Procedure, Test Matrices, Status Reports, Test Scripts etc. Templates for all these could also be attached.

References
Procedures
Templates (Client Specific or otherwise)
Standards / Guidelines (e.g.) QView
Project related documents (RSD, ADD, FSD etc)

Annexure
This could contain embedded documents or links to documents which have been / will be used in the course of testing (e.g.) templates used for reports, test cases etc. Referenced documents can also be attached here.

Sign-Off
This should contain the mutual agreement between the client and the QA team with both leads / managers signing off their agreement on the Test Plan.

GIPS Infotech

Test Strategy

Test Strategy is also known as test approach defines how testing would be carried out. Test approach has two techniques:

- **Proactive -** An approach in which the test design process is initiated as early as possible in order to find and fix the defects before the build is created.
- **Reactive -** An approach in which the testing is not started until after design and coding are completed.

Different Test approaches:

There are many strategies that a project can adopt depending on the context and some of them are:

- Dynamic and heuristic approaches
- Consultative approaches
- Model-based approach that uses statistical information about failure rates.
- Approaches based on risk-based testing where the entire development takes place based on the risk
- Methodical approaches which is based on failures.
- Standard-compliant approach specified by industry-specific standards.

Factors to be considered:

- Risks of product or risk of failure or the environment and the company
- Expertise and experience of the people in the proposed tools and techniques.
- Regulatory and legal aspects, such as external and internal regulations of the development process
- The nature of the product and the domain

Testing Tools

Tools from a software testing context can be defined as a product that supports one or more test activities right from planning, requirements, creating a build, test execution, defect logging and test analysis.

Classification of Tools

Tools can be classified based on several parameters. They include:

- The purpose of the tool
- The Activities that are supported within the tool
- The Type/level of testing it supports
- The Kind of licensing (open source, freeware, commercial)
- The technology used

GIPS Infotech

Types of Tools:

S.No.	Tool Type	Used for	Used by
1.	Test Management Tool	Test Managing, scheduling, defect logging, tracking and analysis.	testers
2.	Configuration management tool	For Implementation, execution, tracking changes	All Team members
3.	Static Analysis Tools	Static Testing	Developers
4.	Test data Preparation Tools	Analysis and Design, Test data generation	Testers
5.	Test Execution Tools	Implementation, Execution	Testers
6.	Test Comparators	Comparing expected and actual results	All Team members
7.	Coverage measurement tools	Provides structural coverage	Developers
8.	Performance Testing tools	Monitoring the performance, response time	Testers
9.	Project planning and Tracking Tools	For Planning	Project Managers
10.	Incident Management Tools	For managing the tests	Testers

Test progress monitoring and control

Test progress monitoring

The purpose of test monitoring is to give feedback and visibility about test activities. Information to be monitored may be collected manually or automatically and may be used to measure exit criteria, such as coverage. Metrics may also be used to assess progress against the planned schedule and budget.

Common test metrics include:

- Percentage of work done in test case preparation (or percentage of planned test cases prepared).
- Percentage of work done in test environment preparation.
- Test case execution (e.g. number of test cases run/not run, and test cases passed/failed).
- Defect information (e.g. defect density, defects found and fixed, failure rate, and retest results).
- Test coverage of requirements, risks or code.

GIPS Infotech

- Subjective confidence of testers in the product.
- Dates of test milestones.
- Testing costs, including the cost compared to the benefit of finding the next defect or to run the next test.

Test Reporting

Test reporting is concerned with summarizing information about the testing endeavour, including:

- What happened during a period of testing, such as dates when exit criteria were met.
- Analyzed information and metrics to support recommendations and decisions about future actions, such as an assessment of defects remaining, the economic benefit of continued testing, outstanding risks, and the level of confidence in tested software. The outline of a test summary report is given in 'Standard for Software Test Documentation' (IEEE 829).

Metrics should be collected during and at the end of a test level in order to assess:

- The adequacy of the test objectives for that test level.
- The adequacy of the test approaches taken.
- The effectiveness of the testing with respect to its objectives.

Test control

Test control describes any guiding or corrective actions taken as a result of information and metrics gathered and reported. Actions may cover any test activity and may affect any other software life cycle activity or task.
Examples of test control actions are:

- Making decisions based on information from test monitoring.
- Re-prioritize tests when an identified risk occurs (e.g. software delivered late).
- Change the test schedule due to availability of a test environment.
- Set an entry criterion requiring fixes to have been retested (confirmation tested) by a developer before accepting them into a build.

Traceability Matrix (RTM)

What is Traceability Matrix

A traceability matrix is a document that co-relates any two-baseline documents that require a many-to-many relationship to check the completeness of the relationship.
It is used to track the requirements and to check the current project requirements are met.

What is RTM (Requirement Traceability Matrix)?

Requirement Traceability Matrix or RTM captures all requirements proposed by the client or development team and their traceability in a single document delivered at the conclusion of the life-cycle.

In other words, it is a document that maps and traces user requirement with test cases. The main purpose of Requirement Traceability Matrix is to see that all test cases are covered so that no functionality should miss while testing.

Requirement Traceability Matrix – Parameters include

- Requirement ID
- Risks
- Requirement Type and Description
- Trace to design specification
- Unit test cases
- Integration test cases
- System test cases
- User acceptance test cases
- Trace to test script

Types of Traceability Test Matrix

- **Forward traceability**: This matrix is used to check whether the project progresses in the desired direction and for the right product. It makes sure that each requirement is applied to the product and that each requirement is tested thoroughly. It maps requirements to test cases.
- **Backward or reverse traceability:** It is used to ensure whether the current product remains on the right track. The purpose behind this type of traceability is to verify that we are not expanding the scope of the project by adding code, design elements, test or other work that is not specified in the requirements. It maps test cases to requirements.
- **Bi-directional traceability (Forward + Backward):** This traceability metrics ensures that all requirements are covered by test cases. It analyzes the impact of a change in requirements affected by the defect in a work product and vice versa.

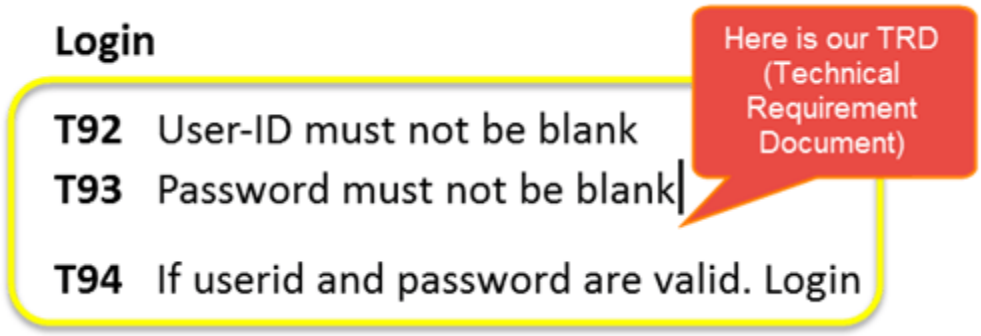
How to create Requirement Traceability Matrix

Below is a sample Business Requirement Document (BRD)

BR#	Module Name	Applicable Roles	Description
B1	Login and Logout	Manager Customer	**Customer:** A customer can login using the login page **Manager:** A manager can login using the login page f customer. Post Login homepage will show ifferent links based on role
B2	Enquiry	Customer	**ustomer:** 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
B3	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

Business Requirement # for Guru99 banking project

While the below table is our Functional/Technical Requirement Document (TRD).



Note: QA teams do not document the BRD and TRD. Also some companies use **Function Requirement Documents (FRD)** which are similar to Technical Requirement Document but the process of creating traceability matrix remains the same.

Let's Go Ahead and create RTM Testing

Step 1: Our Test Case is

"Verify Login, when correct ID and Password is entered, it should login successfully"

TestCase #	Test Case	Test Steps	Test Data	Expected Result
1	Verify Login	1) Go to Login Page 2) Enter UserID 3) Enter Password 4) Click Login	id= **GIPS** pass= 1234	Login Successful

When correct password and id entered, it should login successfully

Step 2: Identify the Technical Requirement that this test case is verifying. For our test case, the technical requirement is T94 is being verified.

T94 If userid and password are valid. Login

T94 is our technical requirement that verifies successful login

Step 3: Note this Technical Requirement (T94) in the Test Case.

TestCase #	TR #	Test Case	Test Steps	Test Data	Expected
1	T94	Verify Login	1) Go to Login Page 2) Enter UserID 3) Enter Password 4) Click Login	id= **GIPS** pass= 1234	Login Successful

Note the Technical Requirement in the test case

Step 4: Identify the Business Requirement for which this TR (Technical Requirement-T94) is defined

BR#	Module Name	Applicable Roles	Description
B1	Login and Logout	Manager Customer	**Customer:** A customer can login using the login page **Manager:** A manager can login using the login page of customer. Post Login homepage will show different links based on role

Identify the Business Requirement for which T94 is defined

Step 5: Note the BR (Business Requirement) in Test Case

TestCase #	BR #	TR #	Test Case	Test Steps	Test Data	Expe
1	B1	T94	Verify Login	1) Go to Login Page 2) Enter UserID 3) Enter Password 4) Click Login	id= GIPS pass= 1234	Login Successful

Step 6: Do above for all Test Cases. Later Extract the First 3 Columns from your Test Suite. RTM in testing is Ready!

Business Requirement #	Technical Requirement #	Test Case ID
B1	T94	1
B2	T95	3
B3	T96	3
B4	T97	4

Requirement Traceability Matrix

Advantage of Requirement Traceability Matrix

- It confirms 100% test coverage
- It highlights any requirements missing or document inconsistencies
- It shows the overall defects or execution status with a focus on business requirements
- It helps in analyzing or estimating the impact on the QA team's work with respect to revisiting or re-working on the test cases

Module-VIII

Defect Management

- Defect Definition
- Defect life cycle
- Severity and Priority
- Defect Clustering

GIPS Infotech

Defect Management

Defect Definition

A software bug arises when the expected result don't match with the actual results. It can also be error, flaw, failure, or fault in a computer program. Most bugs arise from mistakes and errors made by developers, architects.
Following are the methods for preventing programmers from introducing bugs during development:

- Programming Techniques adopted
- Software Development methodologies
- Peer Review
- Code Analysis

Common Types of Defects

Following are the common types of defects that occur during development:

- Arithmetic Defects
- Logical Defects
- Syntax Defects
- Multithreading Defects
- Interface Defects
- Performance Defects

Defect Logging and Tracking

Defect logging, a process of finding defects in the application under test or product by testing or recording feedback from customers and making new versions of the product that fix the defects or the clients feedback.

Defect tracking is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects gets multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier.
Examples - Hp Quality Center, IBM Rational Quality Manager

Defect Tracking Parameters

Defects are tracked based on various parameters such as:

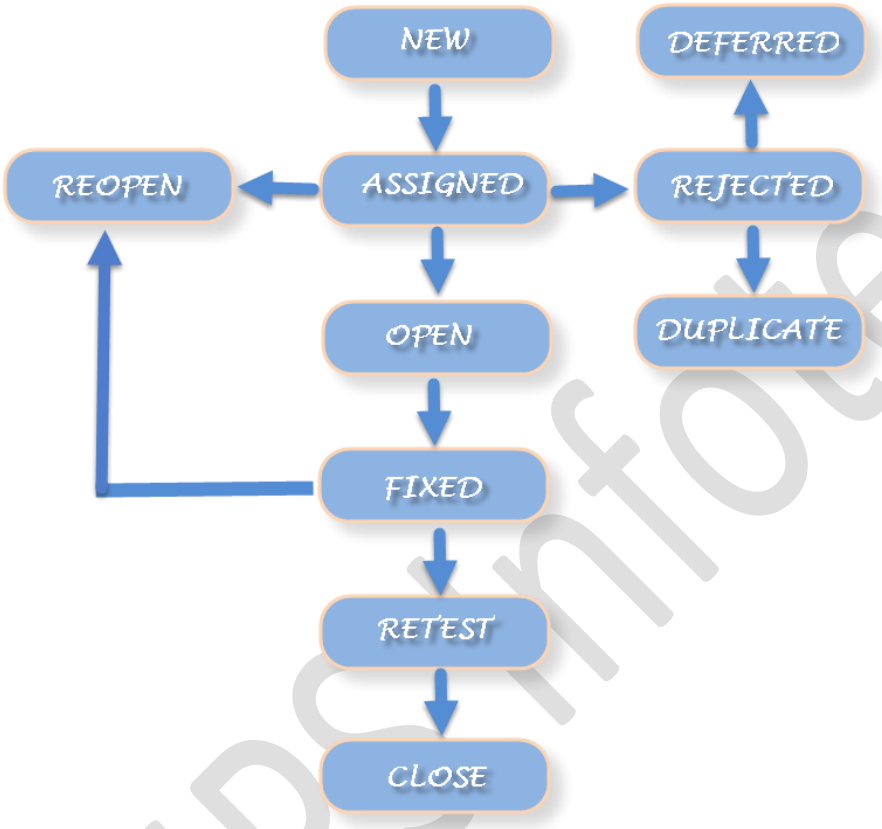
- Defect Id
- Priority
- Severity
- Created by
- Created Date
- Assigned to
- Resolved Date
- Resolved By
- Status

GIPS Infotech

Defect Life Cycle

Defect life cycle, also known as Bug Life cycle is the journey of a defect cycle, which a defect goes through during its lifetime. It varies from organization to organization and also from project to project as it is governed by the software testing process and also depends upon the tools used.

Defect Life Cycle - Workflow:



Defect Life Cycle includes following stages:
New: When a defect is logged and posted for the first time. Its state is given as new.

Assigned: Once the bug is posted by the tester, the lead of the tester approves the bug and assigns the bug to developer team. There can be two scenario, first that the defect can directly assign to the developer, who owns the functionality of the defect. Second, it can also be assigned to the Dev Lead and once it is approved with the Dev Lead, he or she can further move the defect to the developer.
Open: Its state when the developer starts analyzing and working on the defect fix.

Fixed: When developer makes necessary code changes and verifies the changes then he/she can make bug status as 'Fixed'. This is also an indication to the Dev Lead that the defects on Fixed status are the defect which will be available to tester to test in the coming build.

GIPSinfotech

Retest: At this stage the tester do the retesting of the changed code which developer has given to him to check whether the defect got fixed or not.
Once the latest build is pushed to the environment, Dev lead move all the Fixed defects to Retest. It is an indication to the testing team that the defects are ready to test.

Reopened: If the bug still exists even after the bug is fixed by the developer, the tester changes the status to "reopened". The bug goes through the life cycle once again.

Deferred: The bug, changed to deferred state means the bug is expected to be fixed in next releases. The reasons for changing the bug to this state have many factors. Some of them are priority of the bug may be low, lack of time for the release or the bug may not have major effect on the software.

Rejected: If the developer feels that the bug is not genuine, developer rejects the bug. Then the state of the bug is changed to "rejected".
Duplicate: If the bug is repeated twice or the two bugs mention the same concept of the bug, then the recent/latest bug status is changed to "duplicate".

Closed: Once the bug is fixed, it is tested by the tester. If the tester feels that the bug no longer exists in the software, tester changes the status of the bug to "closed". This state means that the bug is fixed, tested and approved.

Not a bug/Enhancement: The state given as "Not a bug/Enhancement" if there is no change in the functionality of the application. For an example: If customer asks for some change in the look and field of the application like change of color of some text then it is not a bug but just some change in the looks of the application.

Severity and Priority

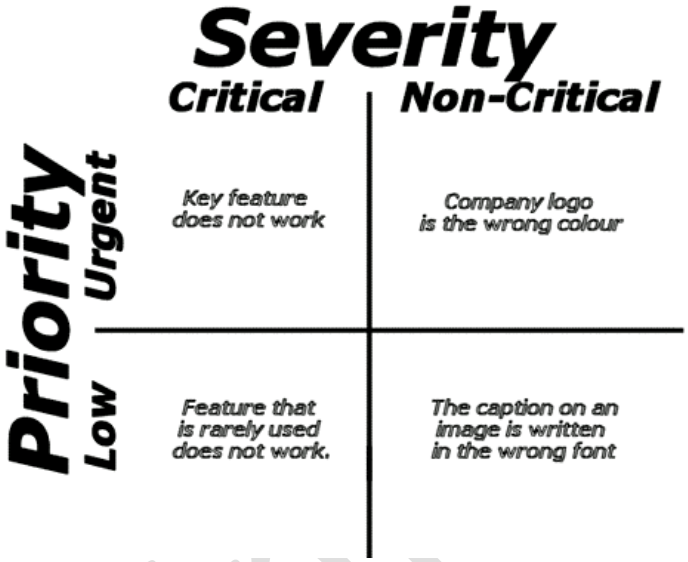
What is Severity?
Severity is defined as the degree of impact a defect has on the development or operation of a component application being tested.
Higher effect on the system functionality will lead to the assignment of higher severity to the bug. Quality Assurance engineer usually determines the severity level of defect
What is Priority?
Priority is defined as the order in which a defect should be fixed. Higher the priority the sooner the defect should be resolved.
Defects that leave the software system unusable are given higher priority over defects that cause a small functionality of the software to fail.

Defect severity can be categorized into four classes

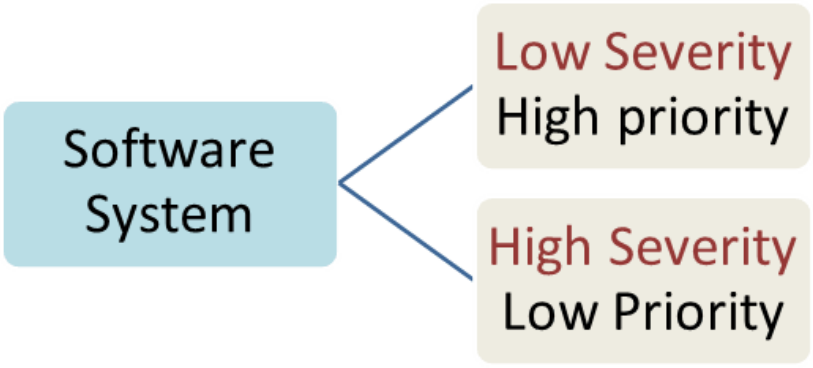
- **Critical**: This defect indicates complete shut-down of the process, nothing can proceed further
- **Major**: It is a highly severe defect and collapse the system. However, certain parts of the system remain functional
- **Medium**: It cause some undesirable behavior, but the system is still functional
- **Low**: It won't cause any major break-down of the system

Defect priority can be categorized into three classes

- **Low:** The defect is an irritant but repair can be done once the more serious defect have been fixed
- **Medium:** During the normal course of the development activities defect should be resolved. It can wait until a new version is created
- **High:** The defect must be resolved as soon as possible as it affects the system severely and cannot be used until it is fixed



A software system can have a



Let see an example of low severity and high priority and vice versa
A very low severity with a high priority: A logo error for any shipment website, can be of low severity as it not going to affect the functionality of the website but can be of high priority as you don't want any further shipment to proceed with wrong logo.

A very high severity with a low priority: Likewise, for flight operating website, defect in reservation functionality may be of high severity but can be a low priority as it can be scheduled to release in a next cycle.

Defect Clustering

Defect Clustering:

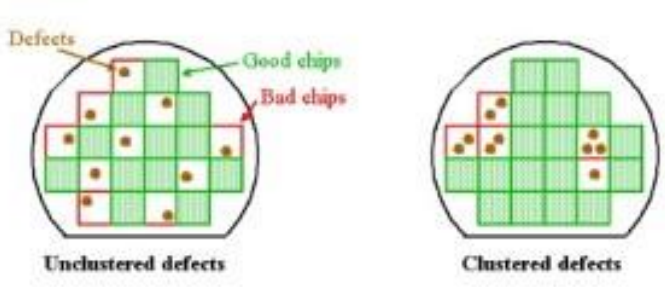
When a small number of modules contains most of the bugs detected or show the most operational failures.

Overview of Defect Clustering:

Defect clustering actually means that the distribution of defects is not across the application uniformly, but rather centralized in limited sections of the application. It is particularly true for large systems where the complexity, size, change and developer mistakes can impact the quality of the system and affect particular modules.

It is based on the Pareto principle, also known as the 80-20 rule, where it stated that approximately 80% of the problems are caused by 20% of modules.

While testing, most of the testers have observed this phenomenon which basically happens because an area of the code is complex and tricky. Test designers often use this information when making the risk assessments for planning the tests, and will focus on these known areas that may also be called as Hotspots.



Advantages:

- Tester can focus the same area in order to find the more number of defects.
- Helps in reducing the time and cost of finding defects.

GLOSSARY OF TESTING TERMINOLOGY

GIPS Infotech

Acceptance testing	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.
Ad hoc testing	Testing carried out informally; no formal test preparation takes place, no recognized test design technique is used, there are no expectations for results and arbitrariness guides the test execution activity.
Agile testing	Testing practice for a project using agile methodologies, such as extreme programming (XP), treating development as the customer of testing and emphasizing the test-first design paradigm.
Alpha testing	Simulated or actual operational testing by potential users/customers or an independent test team at the developers' site, but outside the development organization. Alpha testing is often employed for off-the-shelf software as a form of internal acceptance testing.
Back-to-back testing	Testing in which two or more variants of a component or system are executed with the same inputs, the outputs compared, and analyzed in cases of discrepancies.
Beta testing	Operational testing by potential and/or existing users/customers at an external site not otherwise involved with the developers, to determine whether or not a component or system satisfies the user/customer needs and fits within the business processes. Beta testing is often employed as a form of external acceptance testing for off-the-shelf software in order to acquire feedback from the market.
Big-bang testing	A type of integration testing in which software elements, hardware elements, or both are combined all at once into a component or an overall system, rather than in stages.
Black-box testing	Testing, either functional or non-functional, without reference to the internal structure of the component or system.
Black-box test design technique	Procedure to derive and/or select test cases based on an analysis of the specification, either functional or non-functional, of a component or system without reference to its internal structure.
Blocked test case	A test case that cannot be executed because the preconditions for its execution are not fulfilled.
Bottom-up testing	An incremental approach to integration testing where the lowest level components are tested first, and then used to facilitate the

	testing of higher level components. This process is repeated until the component at the top of the hierarchy is tested.
Boundary value	An input value or output value which is on the edge of an equivalence partition or at the smallest incremental distance on either side of an edge, for example the minimum or maximum value of a range.
Boundary value analysis	A black box test design technique in which test cases are designed based on boundary values.
Branch testing	A white box test design technique in which test cases are designed to execute branches.
Business process-based testing	An approach to testing in which test cases are designed based on descriptions and/or knowledge of business processes.
Capture/playback tool	A type of test execution tool where inputs are recorded during manual testing in order to generate automated test scripts that can be executed later (i.e. replayed). These tools are often used to support automated regression testing.
Certification	The process of confirming that a component, system or person complies with its specified requirements, e.g. by passing an exam.
Code coverage	An analysis method that determines which parts of the software have been executed (covered) by the test suite and which parts have not been executed, e.g. statement coverage, decision coverage or condition coverage.
Compliance testing	The process of testing to determine the compliance of the component or system.
Component integration testing	Testing performed to expose defects in the interfaces and interaction between integrated components.
Condition testing	A white box test design technique in which test cases are designed to execute condition outcomes.
Conversion testing	Testing of software used to convert data from existing systems for use in replacement systems.
Data driven testing	A scripting technique that stores test input and expected results in a table or spreadsheet, so that a single control script can execute all of the tests in the table. Data driven testing is often

	used to support the application of test execution tools such as capture/playback tools.
Database integrity testing	Testing the methods and processes used to access and manage the data (base), to ensure access methods, processes and data rules function as expected and that during access to the database, data is not corrupted or unexpectedly deleted, updated or created.
Defect	A flaw in a component or system that can cause the component or system to fail to perform its required function, e.g. an incorrect statement or data definition. A defect, if encountered during execution, may cause a failure of the component or system.
Defect masking	An occurrence in which one defect prevents the detection of another.
Defect report	A document reporting on any flaw in a component or system that can cause the component or system to fail to perform its required function.
Development testing	Formal or informal testing conducted during the implementation of a component or system, usually in the development environment by developers.
Driver	A software component or test tool that replaces a component that takes care of the control and/or the calling of a component or system.
Equivalence partitioning	A black box test design technique in which test cases are designed to execute representatives from equivalence partitions. In principle test cases are designed to cover each partition at least once.
Error	A human action that produces an incorrect result.
Error guessing	A test design technique where the experience of the tester is used to anticipate what defects might be present in the component or system under test as a result of errors made, and to design tests specifically to expose them.
Exhaustive testing	A test approach in which the test suite comprises all combinations of input values and preconditions.
Exploratory testing	An informal test design technique where the tester actively controls the design of the tests as those tests are performed and uses information gained while testing to design new and better tests.
Failure	Deviation of the component or system from its expected delivery, service or result.

Functional test design technique	Procedure to derive and/or select test cases based on an analysis of the specification of the functionality of a component or system without reference to its internal structure.
Functional testing	Testing based on an analysis of the specification of the functionality of a component or system.
Functionality testing	The process of testing to determine the functionality of a software product.
Heuristic evaluation	A static usability test technique to determine the compliance of a user interface with recognized usability principles (the so-called "heuristics").
High level test case	A test case without concrete (implementation level) values for input data and expected results. Logical operators are used; instances of the actual values are not yet defined and/or available.
ISTQB	International Software Testing Qualification Board. Click here for more details.
Incident management tool	A tool that facilitates the recording and status tracking of incidents. They often have workflow-oriented facilities to track and control the allocation, correction and re-testing of incidents and provide reporting facilities.
Installability testing	The process of testing the installability of a software product.
Integration testing	Testing performed to expose defects in the interfaces and in the interactions between integrated components or systems.
Isolation testing	Testing of individual components in isolation from surrounding components, with surrounding components being simulated by stubs and drivers, if needed.
Keyword driven testing	A scripting technique that uses data files to contain not only test data and expected results, but also keywords related to the application being tested. The keywords are interpreted by special supporting scripts that are called by the control script for the test.
Load testing	A test type concerned with measuring the behavior of a component or system with increasing load, e.g. number of parallel users and/or numbers of transactions to determine what load can be handled by the component or system.
Low level test case	A test case with concrete (implementation level) values for input data and expected results. Logical operators from high level test cases are replaced by actual values that correspond to the objectives of the logical operators.

Maintenance testing	Testing the changes to an operational system or the impact of a changed environment to an operational system.
Monkey testing	Testing by means of a random selection from a large range of inputs and by randomly pushing buttons, ignorant on how the product is being used.
Negative testing	Tests aimed at showing that a component or system does not work. Negative testing is related to the testers' attitude rather than a specific test approach or test design technique, e.g. testing with invalid input values or exceptions.
Non-functional testing	Testing the attributes of a component or system that do not relate to functionality, e.g. reliability, efficiency, usability, maintainability and portability.
Operational testing	Testing conducted to evaluate a component or system in its operational environment.
Pair testing	Two persons, e.g. two testers, a developer and a tester, or an end-user and a tester, working together to find defects. Typically, they share one computer and trade control of it while testing.
Peer review	A review of a software work product by colleagues of the producer of the product for the purpose of identifying defects and improvements. Examples are inspection, technical review and walkthrough.
Performance testing	The process of testing to determine the performance of a software product.
Portability testing	The process of testing to determine the portability of a software product.
Post-execution comparison	Comparison of actual and expected results, performed after the software has finished running.
Priority	The level of (business) importance assigned to an item, e.g. defect.
Quality assurance	Part of quality management focused on providing confidence that quality requirements will be fulfilled.
Random testing	A black box test design technique where test cases are selected possibly uses a pseudo-random generation algorithm, to match an operational profile. This technique can be used for testing non-functional attributes such as reliability and performance.
Recoverability testing	The process of testing to determine the recoverability of a software product.

Regression testing	Testing of a previously tested program following modification to ensure that defects have not been introduced or uncovered in unchanged areas of the software, as a result of the changes made. It is performed when the software or its environment is changed.
Requirements-based testing	An approach to testing in which test cases are designed based on test objectives and test conditions derived from requirements, e.g. tests that exercise specific functions or probe non-functional attributes such as reliability or usability.
Re-testing	Testing that runs test cases that failed the last time they were run, in order to verify the success of corrective actions.
Risk-based testing	An approach to testing to reduce the level of product risks and inform stakeholders on their status, starting in the initial stages of a project. It involves the identification of product risks and their use in guiding the test process.
Severity	The degree of impact that a defect has on the development or operation of a component or system.
Site acceptance testing	Acceptance testing by users/customers at their site, to determine whether or not a component or system satisfies the user/customer needs and fits within the business processes, normally including hardware as well as software.
Smoke test	A subset of all defined/planned test cases that cover the main functionality of a component or system, to ascertaining that the most crucial functions of a program work, but not bothering with finer details. A daily build and smoke test is among industry best practices.
Statistical testing	A test design technique in which a model of the statistical distribution of the input is used to construct representative test cases.
Stress testing	Testing conducted to evaluate a system or component at or beyond the limits of its specified requirements.
Stub	A skeletal or special-purpose implementation of a software component, used to develop or test a component that calls or is otherwise dependent on it. It replaces a called component.
Syntax testing	A black box test design technique in which test cases are designed based upon the definition of the input domain and/or output domain.
System integration testing	Testing the integration of systems and packages; testing interfaces to external organizations (e.g. Electronic Data Interchange, Internet).

System testing	The process of testing an integrated system to verify that it meets specified requirements.
Test automation	The use of software to perform or support test activities, e.g. test management, test design, test execution and results checking.
Test case specification	A document specifying a set of test cases (objective, inputs, test actions, expected results, and execution preconditions) for a test item.
Test design specification	A document specifying the test conditions (coverage items) for a test item, the detailed test approach and identifying the associated high level test cases.
Test environment	An environment containing hardware, instrumentation, simulators, software tools, and other support elements needed to conduct a test.
Test harness	A test environment comprised of stubs and drivers needed to execute a test.
Test log	A chronological record of relevant details about the execution of tests.
Test management tool	A tool that provides support to the test management and control part of a test process. It often has several capabilities, such as testware management, scheduling of tests, and the logging of results, progress tracking, incident management and test reporting.
Test oracle	A source to determine expected results to compare with the actual result of the software under test. An oracle may be the existing system (for a benchmark), a user manual, or an individual's specialized knowledge, but should not be the code.
Test plan	A document describing the scope, approach, resources and schedule of intended test activities. It identifies amongst others test items, the features to be tested, the testing tasks, who will do each task, degree of tester independence, the test environment, the test design techniques and entry and exit criteria to be used, and the rationale for their choice, and any risks requiring contingency planning. It is a record of the test planning process.
Test strategy	A high-level description of the test levels to be performed and the testing within those levels for an organization or programme (one or more projects).

Test suite	A set of several test cases for a component or system under test, where the post condition of one test is often used as the precondition for the next one.
Testware	Artifacts produced during the test process required to plan, design, and execute tests, such as documentation, scripts, inputs, expected results, set-up and clear-up procedures, files, databases, environment, and any additional software or utilities used in testing.
Thread testing	A version of component integration testing where the progressive integration of components follows the implementation of subsets of the requirements, as opposed to the integration of components by levels of a hierarchy.
Top-down testing	An incremental approach to integration testing where the component at the top of the component hierarchy is tested first, with lower level components being simulated by stubs. Tested components are then used to test lower level components. The process is repeated until the lowest level components have been tested.
Traceability	The ability to identify related items in documentation and software, such as requirements with associated tests. See also horizontal traceability, vertical traceability.
Usability testing	Testing to determine the extent to which the software product is understood, easy to learn, easy to operate and attractive to the users under specified conditions.
Use case	A sequence of transactions in a dialogue between a user and the system with a tangible result.
Use case testing	A black box test design technique in which test cases are designed to execute user scenarios.
Unit test framework	A tool that provides an environment for unit or component testing in which a component can be tested in isolation or with suitable stubs and drivers. It also provides other support for the developer, such as debugging capabilities.
Validation	Confirmation by examination and through provision of objective evidence that the requirements for a specific intended use or application have been fulfilled.
Verification	Confirmation by examination and through provision of objective evidence that specified requirements have been fulfilled.

Vertical traceability	The tracing of requirements through the layers of development documentation to components.
Volume testing	Testing where the system is subjected to large volumes of data.
Walkthrough	A step-by-step presentation by the author of a document in order to gather information and to establish a common understanding of its content.
White-box testing	Testing based on an analysis of the internal structure of the component or system.

GIPS Infotech