

Testing Methodology Training Manual

TABLE OF CONTENTS

1.0 INTRODUCTION TO TESTING	5
1.1 OVERVIEW ON TESTING.....	5
1.1.1 Economics of Testing.....	6
1.1.2 Lifecycle of Testing.....	7
1.2 TESTERS.....	8
1.3 TESTING TECHNIQUES.....	8
1.4 TYPES OF TESTING.....	9
1.5 INTEGRATION TESTING METHODS.....	10
1.6 UNIT TESTING CHECKPOINTS.....	12
1.7 EXAMPLE OF BUSINESS RULES.....	12
2.0 TEST PREPARATION PROCESS	13
2.1 BASELINE DOCUMENTS.....	13
2.1.1 Business Requirement (User Requirement Specification -URS).....	13
2.1.2 How to read a Business Requirement?.....	13
2.1.3 Functional Specification (Functional Requirement Specification-FRS).....	13
2.1.4 How to read a Functional Specification?.....	13
2.1.5 Tester's Reading Perspective.....	14
2.1.6 Design Specification (System Design Specification - SDS).....	15
2.2 PROTOTYPE.....	15
2.2.1 Scenarios in Prototype.....	15
2.2.2 Flow of Prototype.....	15
2.3 TEST STRATEGY.....	16
2.3.1 Testing Approach.....	16
2.3.2 Automation Strategy.....	16
2.3.3 Performance Strategy.....	16
2.3.4 Risk Analysis.....	17
2.3.6 Infrastructure.....	17
2.4 HIGH LEVEL TEST CONDITIONS.....	18
2.4.1 What is a High Level Test Condition.....	18
2.4.2 Understanding the maximum conditions for a specification.....	18
2.4.3 Queries on Functional Specification.....	18
2.5 TRACEABILITY.....	19
2.5.1 BR and FS.....	19
2.5.2 FS and Test Script.....	19
2.5.3 Gap Analysis.....	19
2.5.4 Tools Used for Traceability.....	19
2.6 TESTBED.....	20
2.6.1 What is Testbed?.....	20
2.7 TEST CASE.....	20
2.7.1 Test Case Formation.....	20
2.7.2 Explicit writing.....	20
2.7.3 Expected Results.....	21
2.7.4 Pre-Requirements.....	21
2.7.5 Data definition.....	22
2.8 TEST SCRIPT.....	23
2.8.1 Brief on Test Scripts.....	23

2.8.2	<i>Interaction with development team</i>	27
2.8.3	<i>Review of Test Scripts</i>	27
2.8.4	<i>Activity Report</i>	27
2.10	BACKEND TESTING	27
2.10.1	<i>Understanding the application</i>	28
2.10.2	<i>Data Feeds Management</i>	28
2.10.4	<i>Data Upload Process</i>	28
2.10.5	<i>Verification of Data</i>	28
2.11	NON CERTIFIED TESTING	28
2.11.1	<i>Documents</i>	28
2.11.2	<i>Screen Shots</i>	29
2.11.3	<i>Mapping</i>	29
3.0	TEST EXECUTION PROCESS	30
3.1	STAGES OF TESTING	30
3.1.1	<i>Three passes</i>	30
3.2	PRE- REQUIREMENTS FOR TESTING	30
3.2.1	<i>Version Identification</i>	30
3.2.2	<i>Interfaces for the application</i>	31
3.2.3	<i>Unit and Module test plan sign off</i>	31
3.3	TEST PLAN	31
3.3.1	<i>Test Execution Sequence</i>	31
3.3.2	<i>Allocation of test cases among the team</i>	32
3.3.3	<i>Targets for completion of Testing</i>	32
3.4	AUTOMATION OF TEST CASES	33
3.4.1	<i>Capturing during First pass</i>	33
3.4.1	<i>Intelligent Automation</i>	33
3.5	DEFECT MANAGEMENT	33
3.5.1	<i>What is a defect?</i>	33
3.5.2	<i>Types of Defects</i>	34
3.5.3	<i>Defect reporting by tester</i>	34
3.5.4	<i>Defect Tracking by Test Lead</i>	35
3.5.5	<i>Tools Used</i>	36
3.5.6	<i>Defects Meetings</i>	36
3.5.7	<i>Defect Metrics</i>	37
3.5.8	<i>Defects Publishing</i>	37
3.6	TEST DOWN TIMES	39
3.6.1	<i>Server problems</i>	39
3.6.2	<i>Problems on Testing side / Development side</i>	39
3.6.3	<i>Show Stopper</i>	39
3.7	FILE TRANSFER PROTOCOL	40
3.8	PERFORMANCE TESTING	40
3.8.1	<i>Execution</i>	40
3.8.2	<i>Performance Testing Tools</i>	41
4.0	POST TEST PROCESS	42
4.1	SIGN OFF	42
4.1.1	<i>Sign off Criteria</i>	42
4.2.2	<i>Authorities</i>	42
4.2	DELIVERABLES	42

4.2.1 <i>Internal</i>	42
4.2.2 <i>External</i>	43
4.5 ARCHIVING.....	44
4.5.1 <i>Tree</i>	44
4.5.2 <i>Collection of Details</i>	44
4.5.2 <i>Collection of Details</i>	45
4.5.3 <i>Document Warehouse</i>	45
4.5.4 <i>CD Recording</i>	45

1.0 Introduction to Testing

1.1 Overview on Testing

There are many published definitions of software testing; however, all of these definitions essentially convey the same statement:

Software testing is the process of executing software in a controlled manner, in order to answer the question “Does the software comply to specifications”.

Software testing is often used in association with the terms Verification and Validation. Verification is the checking or testing of items, including software, for conformance and consistency with an associated specification. Software testing is one kind of verification, which also uses techniques such as reviews, analysis, inspections and walkthroughs. Validation is the process of checking if what has been specified is what the user actually wanted.

- **Verification:** Are we building the product right?
- **Validation:** Are we building the right product?

Software testing should not be confused with debugging. Debugging is the process of analyzing and locating the bugs when software does not behave as expected. Although the identification of some bugs will be obvious from playing with the software, methodical approach to software testing is a much more thorough means of identifying bugs. Debugging is therefore an activity that supports testing, but cannot replace testing. However, no amount of testing can be guaranteed to discover all bugs.

What is testing?

Testing is the process of executing a program with the intent of finding an error. A successful test is one that uncovers an as-yet-undiscovered error.

Why testing?

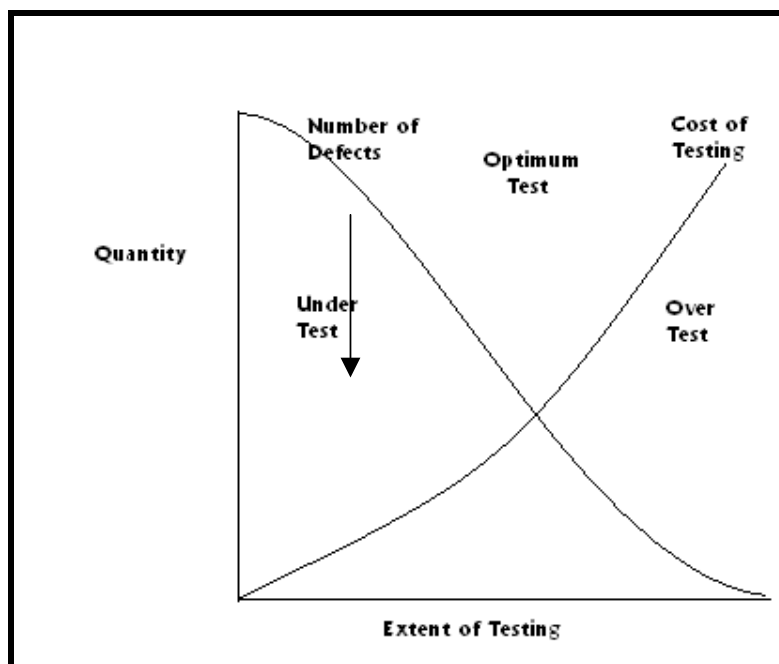
The development of software systems involves a series of production activities where opportunities for injection of human fallibilities are enormous. Errors may begin to occur at the very inception of the process where the requirements may be erroneously or imperfectly specified. Because of human inability to perform and communicate with perfection, software development is accompanied by a quality assurance activity.

1.1.1 Economics of Testing

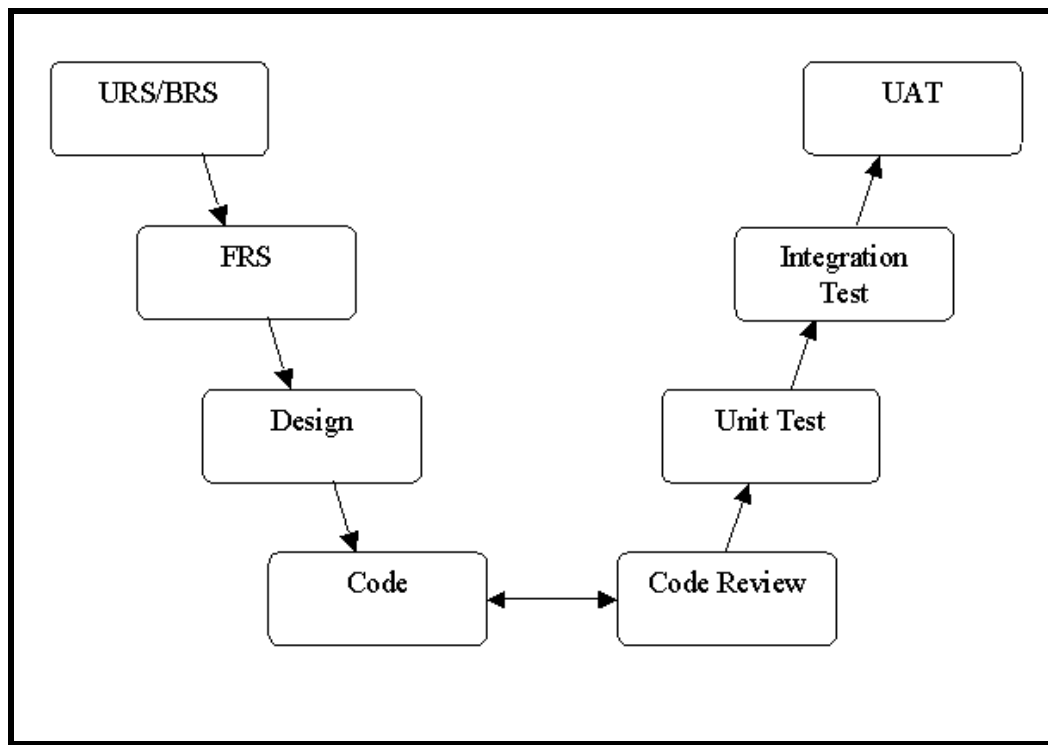
“Too little testing is a crime – too much testing is a sin”. The risk of under-testing is directly translated into system defects present in the production environment. The risk of over-testing is the unnecessary use of valuable resources in testing systems that have no defects, or very few defects that the cost of testing far exceeds the value of detecting the system defect.

Most of the problems associated with testing occur from one of the following causes:

- Failure to define testing objectives
- Testing at the wrong phase in the cycle
- Use of ineffective test techniques



1.1.2 Lifecycle of Testing



1.2 Testers

What does a tester do:

- Understand the Application Under Test (AUT)
- Prepare test strategy
- Assist with preparation of test plan
- Develop test scripts
- Understand the data involved
- Execute all assigned test cases
- Record defects in the defect tracking system
- Retest fixed defects
- Assist the test leader with his/her duties
- Provide feedback in defect analysis
- Automate test scripts
- Understanding of SQL

1.3 Testing Techniques

White Box Testing

Also known as *glass box, structural, clear box and open box testing*. A software testing technique whereby explicit knowledge of the internal workings of the item being tested are used to select the test data. Unlike black box testing, white box testing uses specific knowledge of programming code to examine outputs. The test is accurate only if the tester knows what the program is supposed to do.

Black Block Testing

Also known as *functional testing*. Software testing technique whereby the tester does not know the internal workings of the item being tested. The tester only knows the inputs and what the expected outcomes should be and not how the program arrives at those outputs. The tester does not ever examine the programming code and does not need any further knowledge of the program other than its specifications.

Path Testing

Path testing has been one of the first test methods, and even though it is a typical white box test, it is nowadays also used in black box tests. The procedure itself is similar to the walk-through. First, a certain path through the program is chosen. Possible inputs and the correct result are written down. Then the program is executed by hand, and its result is compared to the predefined. Possible faults have to be written down at once.

1.4 Types of Testing

Unit Testing – the most 'micro' scale of testing; to test particular functions or code modules. Typically done by the programmer and not by testers, as it requires detailed knowledge of the internal program design and code. Not always easily done unless the application has a well-designed architecture with tight code; may require developing test driver modules or test harnesses.

Integration Testing – testing of combined parts of an application to determine if they function together correctly. The 'parts' can be code modules, individual applications, client and server applications on a network, etc. This type of testing is especially relevant to client/server and distributed systems.

System Testing – black-box type testing that is based on overall requirements specifications; covers all combined parts of a system.

Regression Testing – re-testing after fixes or modifications of the software or its environment. It can be difficult to determine how much re-testing is needed, especially near the end of the development cycle. Automated testing tools can be especially useful for this type of testing.

Load testing – testing an application under heavy loads, such as testing of a web site under a range of loads to determine at what point the system's response time degrades or fails.

Stress testing – term often used interchangeably with 'load' and 'performance' testing. Also used to describe such tests as system functional testing while under unusually heavy loads, heavy repetition of certain actions or inputs, input of large numerical values, large complex queries to a database system, etc.

Performance testing – term often used interchangeably with 'stress' and 'load' testing. Ideally 'performance' testing (and any other 'type' of testing) is defined in requirements documentation or Test Plans.

Compatibility testing – testing how well software performs in a particular hardware/software/operating system/network/etc. environment.

User Acceptance Testing (UAT) – is performed by Users or on behalf of the users to ensure that the Software functions in accordance with the Business Requirement Document. UAT focuses on the following aspects:

- All functional requirements are satisfied
- All performance requirements are achieved
- Other requirements like transportability, compatibility, error recovery etc. are satisfied
- Acceptance criteria specified by the user are met.

Smoke Testing – A quick-and-dirty test that the major functions of a piece of software work. Originated in the hardware testing practice of turning on a new piece of hardware for the first time and considering it a success if it does not catch on fire. See also Sanity Testing below.

Sanity Checking/Testing – Brief test of major functional elements of a piece of software to determine if its basically operational. It is typically an initial testing effort to determine if a new software version is performing well enough to accept it for a major testing effort. For example, if the new software is crashing systems every 5 minutes, bogging down systems to a crawl, or destroying

databases, the software may not be in a 'sane' enough condition to warrant further testing in its current state.

Install/uninstall testing - testing of full, partial, or upgrade install/uninstall processes.

Recovery testing - testing how well a system recovers from crashes, hardware failures, or other catastrophic problems.

Security testing - testing how well the system protects against unauthorized internal or external access, willful damage, etc; may require sophisticated testing techniques.

End-to-End testing - similar to system testing; the 'macro' end of the test scale; involves testing of a complete application environment in a situation that mimics real-world use, such as interacting with a database, using network communications, or interacting with other hardware, applications, or systems if appropriate.

Alpha testing - testing of an application when development is nearing completion; minor design changes may still be made as a result of such testing. Typically done by end-users or others, not by programmers or testers.

Beta testing - testing when development and testing are essentially completed and final bugs and problems need to be found before final release. Typically done by end-users or others, not by programmers or testers.

Difference between Functional Test and UAT

Particulars	Functional Test	UAT
Baseline Document	Functional Specification	Business Requirement
Data	Simulated	Live Data
Environment	Controlled	Simulated Live
Perspective	Functionality	User style
Location	Off Site	On Site
Tester Composition	Tester	Test & Real Users
Purpose	Validation & Verification	User Needs

1.5 Integration Testing Methods

There are basically two types of Integration testing methods:

- (1) Incremental
- (2) Non- Incremental

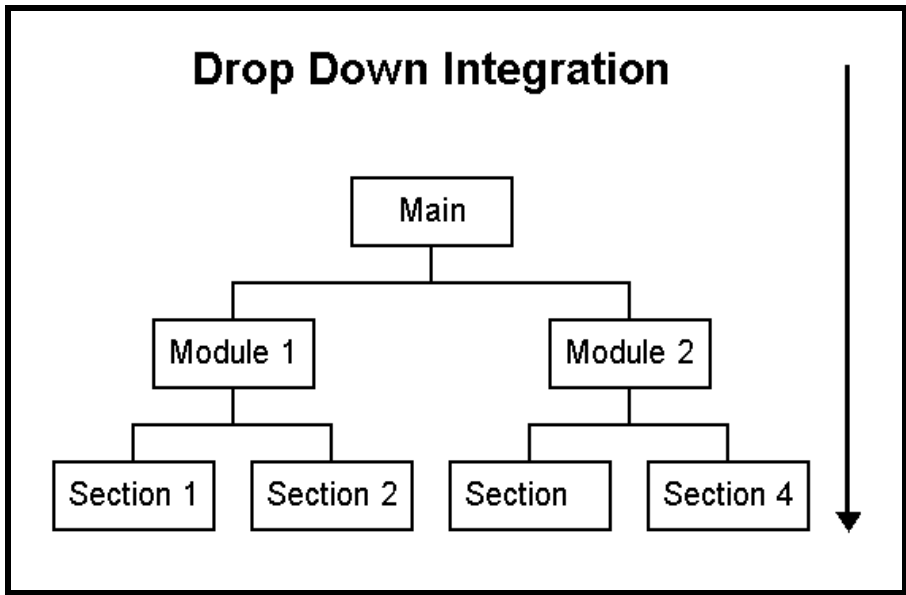
Incremental testing can have two sub-types, viz:

- Top-Down Incremental
- Bottom-Up Incremental

Top Down Integration:

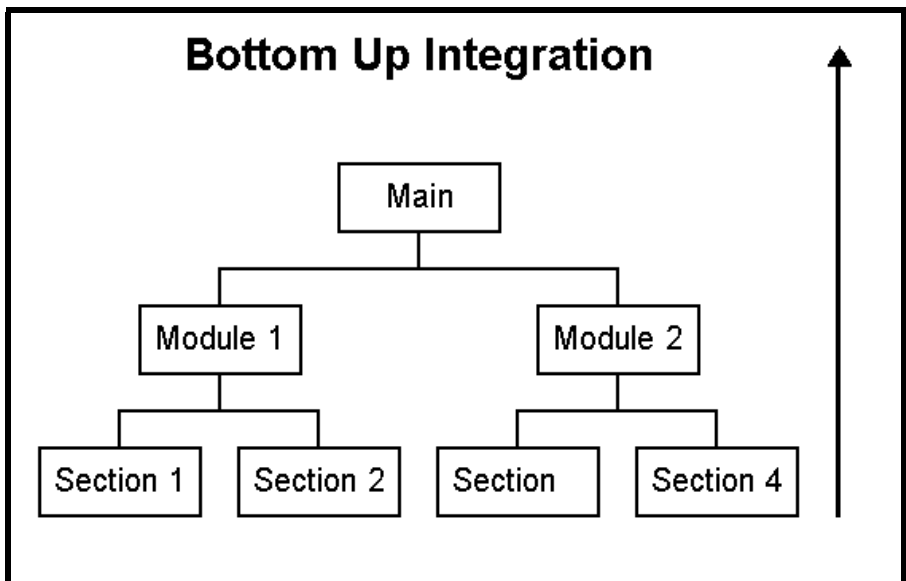
Modules are integrated by moving downward through the control hierarchy, beginning with the main control module. Modules subordinate to the main control module are incorporated into the structure in either a *depth-first* or *breadth-first* manner.

Top-down strategy sounds relatively uncomplicated, but in actual practice, logistical problems can arise. The most common of these problems occurs when processing at low-levels in the hierarchy is required to adequately test upper levels. No significant data can flow upward in the program structure.



Bottom up Integration:

As it's name implies, begins construction and testing with atomi modules i.e., modules at the lowest levels in the program structure).



'Big bang' (Non- Incremental) approach:

There is often a tendency to attempt non-incremental integration: that is, to construct the program using a 'big-bang' approach. All modules are combined in advance. The entire program is tested as a whole. And chaos usually results! Sets of errors are encountered. Correction is difficult because isolation of causes is complicated by the vast expanse of the entire program. Once these errors are corrected, new ones appear and the process continues in a seemingly endless loop.

1.6 Unit Testing Checkpoints

There are **three** main checkpoints in unit testing. They are at:

- (1) Compilation time.
- (2) Run time.
- (3) Application time.

Compilation time: The developer, after writing a code, compiles it and finds some errors. He goes back to the coding and identifies and fixes the errors, which may be logical or syntactical.

Run time: The developer may not find any errors at the first checkpoint, but during run time he may encounter certain errors, which are mostly logical errors. He has to go back to the coding and correct the errors.

Application time: Although compilation and run time errors may not be encountered in a program, still it may not give the expected output. This may be mainly due to wrong data type of the variables or some logical errors in the program. The developer should correct them before passing it on to the testing team/department.

1.7 Example of Business Rules

What is the Business Rules to add a customer?

The following table gives the business rules laid down for allowing persons in the New York / New Jersey area to enroll as new customers in a bank:

State	Status	Dep_amt(\$)
NY	A	>700
NY	B	>500 and <700
NY	C	>400 and <500
NJ	A	>400
NJ	B	>300 and <400

In the above table, a business rule has been laid down that a customer will be added only if he belongs to 'A', 'B', or 'C' status, which in turn will be decided by the Dep_amt that he has to offer. The business rule also says that for New York customers, the status 'A', 'B', 'C' conform to Dep_amt of \$700, \$500 and \$400 respectively, while the same status will be applicable to the New Jersey customers for \$100 less for each category. However, the bank does not allow any 'C' category for New Jersey, which is evident from the fact that status "C" is missing in the table for N.J.

2.0 Test Preparation Process

2.1 Baseline Documents

Construction of an application and testing are done using certain documents. These documents are written in sequence, each of it derived from the previous document.

2.1.1 Business Requirement (User Requirement Specification -URS)

This document describes users needs for the application. This is done over a period of time, and going through various levels of requirements. This should also portrays functionality that are technically feasible within the stipulated times frames for delivery of the application.

As this contains user perspective requirements, User Acceptance Test is based on this document.

2.1.2 How to read a Business Requirement?

In case of the Integrated Test Process, this document is used to understand the user requirements and find the gaps between the User Requirement and Functional Specification.

User Acceptance Test team should break the business requirement document into modules depending on how the user will use the application. While reading the document, test team should put themselves as end users of the application. This document would serve as a base for UAT test preparation.

2.1.3 Functional Specification (Functional Requirement Specification-FRS)

This document describes the functional needs; design of the flow and user maintained parameters. These are primarily derived from Business Requirement document, which specifies the client's business needs.

The proposed application **should adhere** to the specifications specified in this document. This is used henceforth to develop further documents for software construction and validation and verification of the software.

In order to achieve synchronisation between the software construction and testing process, Functional Specification (FS) serves as the Base document.

2.1.4 How to read a Functional Specification?

The testing process begins by first understanding the functional specifications. The FS is normally divided into modules. The tester should understand the entire functionality that is proposed in the document by reading it thoroughly.

It is natural for a tester at this point to get confused on the total flow and functionality. In order to overcome these, it is advisable for the tester to read the document multiple times, seeking clarifications then and there until clarity is achieved.

Testers are then given a module or multiple modules for validation and verification. These modules then become the tester's responsibility.

The Tester should then begin to acquire an in-depth knowledge of their respective modules. In the process, these modules should be split into segments like field level validations, module rules, business rules etc. In order to do the same module's importance and precisely the tester should interpret its role within the application.

A high level understanding of the data requirements for respective modules is also expected from the tester at this point.

Interaction with test lead at this juncture is crucial to draw a testing approach, like an end-to-end test coverage or individual test. *(Explained later in the document)*

2.1.5 Tester's Reading Perspective

Functional specification, is sometimes written assuming some level of knowledge of the Testers and constructors. We can categorize the explanations by

Explicit Rules: Functionality expressed as conditions clearly in writing, in the document.

Example

Date of a particular field should be system date

Implicit Rules: Functionality that is implied based on what is expressed as a specification/condition or requirement of a user.

Example

FS would mention the following for a deposit creation

Start Date Field: Should be = or > than the system date

Maturity Date Field: Should be = or > than the system date

Under this condition, the implied specification derived is that, Start date should not be equal to the maturity date

2.1.6 Design Specification (System Design Specification - SDS)

This document is prepared based on the functional specification. It contains the system architecture, table structures and program specifications. This is ideally prepared and used by the construction team. The Test Team should also have a detailed understanding of the design specification in order to understand the system architecture.

2.2 Prototype

This is look and feel representation of the application that is proposed. This basically shows the placement of the fields, modules and generic flow of the application. The main objective of the prototype is to demonstrate the understanding of the application to the users and obtain their buy-in before actual design and construction begins.

The development team also uses the prototype as a guide to build the application

This is usually done using HTML or MS PowerPoint with user interaction facility.

2.2.1 Scenarios in Prototype

The flow and positioning of the fields and modules are projected using several possible business scenarios derived from the application functionality.

Testers should not expect all possible scenarios to be covered in the prototype.

2.2.2 Flow of Prototype

The flow and positioning are derived from initial documentation on the project. A project is normally dynamic during initial stages, and hence tester should bear in mind the changes to the specification, if any, while using the prototype to develop test conditions.

It is a value addition to the project when tester can identify mismatches between the specifications and prototype, as the application can be rectified in the initial stages itself.

2.3 Test Strategy

Actual writing of a strategy involves aspects, which define other issues between the Testing organization and the client. Testers must basically understand some of the issues that are discussed in the strategy document.

2.3.1 Testing Approach

The testing process may take the form of an End-to-End approach or individual segment testing using various values.

End-to-End: The test path uses the entire flow provided in the application for completion of a specified task. Within this process various test conditions and values are covered and results analyzed. There may be a possibility of reporting several defects relating to the segments while covering the test path. The advantage of using this approach is to minimize combination and permutation of conditions/values and ensure coverage and integration.

Individual Segment Testing: Several conditions and values are identified for testing at the unit level for testing. These are tested as separate cases.

2.3.2. Automation Strategy

Automation of testing process is done to reduce the effort during regression testing. In some cases automating the entire testing process may not possible due to technical and time constraints. The possible automation strategies that could be adopted depending on the types of the project are.

Selective: Critical and complex cases are identified. These test cases are generally automated to simplify the testing process and save time.

Complete: As the term suggests, all test cases technically possible are automated.

2.3.3 Performance Strategy

The client specifies the standards for the performance testing. It generally contains

- Response time
- Number of Virtual Users

Using the above information, a Usage Pattern of the application is derived and documented in the strategy. Issues discussed in the performance strategy document are

Resources: Personnel trained in Performance testing tool identified.

Infrastructure: Generation of virtual users require huge amount of RAM. The performance team should be given machines, which are suitable for the performance tool.

Report: The type of report that will be generated after the tests are discussed. Generally reports are ideally in the form of graphs. Reports generated are:

- Detailed Transaction Report (By Virtual user)
- Throughput Graph
- Hits per second Graph
- Transaction per second
- Transaction Response Time Graph
- Transaction Performance Summary Graph
- Transaction Distribution Graph

2.3.4 Risk Analysis

Risks associated with projects are analyzed and mitigation's are documented in this document. Types of risk that are associated are

Schedule Risk: Factors that may affect the schedule of testing are discussed.

Technology Risk: Risks on the hardware and software of the application are discussed here

Resource Risk: Test team availability is discussed.

Support Risk: Clarifications required on the specification and availability of personnel for the same is discussed.

2.3.6 Infrastructure

Hardware and software requirements for the testing the application are documented. Apart from this, any other requirement should also be documented. Infrastructure that has to be provided by the client is also specified.

2.4 High Level Test Conditions

2.4.1 What is a High Level Test Condition

It represents the possible values that can be attributed to a particular specification.

2.4.2 Understanding the maximum conditions for a specification

At this point the tester will have a fair understanding of the application and his module. The functionality can be broken into

- Field level rules
- Module level rules
- Business rules
- Integration rules
- Processing logic

It may not be possible to segment the specifications into the above categories in all applications. It is left to the test team to decide on the applicable segmentation.

For the segments identified by the test team, the possible condition types that can be built are:

Positive condition: Polarity of the value given for test is to comply with the condition existence.

Negative condition: Polarity of the value given for test is **not to** comply with the condition existence.

2.4.3 Queries on Functional Specification

Preparation of test conditions would lead to certain queries arising because of

- Gap between the understanding of the tester and specification
- Implementation issues
- Design restrictions
- Contradictions within the functional specification document
- Contradictions between other application documents
- Contradictions between functional specification and real time applicability

These queries can be clarified using the following resources:

Domain Consultant: Normally the author of the FS, and an expert in the field of the application.

Test Manager: Person responsible for the management and co-ordination of the project.

Test Lead: Person responsible for the testing processes and team.

Peer Group: Other team members, who are generally in the test team.

2.5 Traceability

2.5.1 BR and FS

The requirements specified by the users in the business requirement document may not be exactly translated into a functional specification. Therefore, a trace on specifications between functional specification and business requirements is done on a one to one basis. This helps finding the gap between the documents. These gaps are then closed by the author of the FS, or deferred after discussions.

Testers should understand these gaps and use them as an addendum to the FS, after getting this signed off from the author of the FS. The final FS form may vary from the original, as deferring or taking in a gap may have ripple effect on the application. Sometimes, these ripple effects may not be reflected in the FS. Addendum may sometime affect the entire system and the test case development.

2.5.2 FS and Test Script

Test scripts built by the tester are traced with the FS to ensure full coverage of the baseline document. If gaps between the same are obtained, tester must then create test scripts for the gaps.

2.5.3 Gap Analysis

This is the terminology used on finding the difference between ‘what it should be’ and ‘what it is’. As explained, it is done on the Business requirement to FS and FS to test scripts. Mathematically, it becomes evident that Business requirements that are users needs are tested, as Business Requirement and tests are matched.

Simplifying the above,

A = Business Requirement
B = Functional Specification
C = Test

$A = B, B = C, \text{ Therefore } A = C$

Another way of looking at this process is to eliminate as many mismatches at every stage of the process, there by giving the customer an application, which will satisfy their needs.

In the case of UAT, there is a direct translation of specification from the Business Requirement to Tests (Test Scripts/Test Cases).

2.5.4 Tools Used for Traceability

The entire process of traceability is a time consuming process. In order to simplify, IBM/Rational has developed a tool (Rational RequisitePro), which will maintain the specifications of the documents. Then these are mapped correspondingly. The specifications have to be loaded into the system by the user. Even though it is a time consuming process, it helps in finding the ‘ripple’ effect on altering a specification. The impacts on test conditions can immediately be identified using the trace matrix.

2.6 Testbed

2.6.1 What is Testbed?

An execution environment configured for testing. May consist of specific hardware, OS, network topology, configuration of the product under test, other application or system software, etc. The Test Plan for a project should enumerated the test beds(s) to be used.

2.7 Test Case

A **test case** is a set of conditions or variables under which a tester will determine if a requirement upon an *application* is partially or fully satisfied. It may take many test cases to determine that a requirement is fully satisfied. In order to fully test that all the requirements of an application are met, there must be at least one test case for each requirement unless a requirement has sub requirements. In that situation, each sub requirement must have at least one test case.

What characterises a test case is that there is a *known input* and an *expected output*, which is worked out *before* the test. The known input should test a *precondition* and the expected output should test a *postcondition*. . A test case validates one or more system requirements and generates a pass or fail.

Under special circumstances, there could be a need to run the test, produce results - and a team of experts evaluate if the results can be considered as pass. This happens often on new products performance number determination. The first test is taken as the base line for subsequent test / product release cycles.

2.7.1 Test Case Formation

At this stage, the Tester has clarity on how the application is to be tested. It now, becomes necessary to aid the actual test action with test cases. Test cases are written based on the test conditions. It is the phrased form of test conditions, which becomes readable and understandable by all.

2.7.2 Explicit writing

There are three headings under which a test case is written. Namely,

Description: Here the details of the test on a specification or condition are written

Data and Pre-requirements: Here either the data for the test or specification is mentioned. Pre-requirements for the test to be executed should also be clearly mentioned.

Expected Results: The expected result on execution of the instruction in the description is mentioned. In general, it should reflect, in detail the result of the test execution.

While writing a case, to make the test case explicit, the tester should include the following

- Reference to the rules and specifications under test in words with minimal technical jargons
- Check on data shown by the application should refer to the table names if possible
- Location of the fields or if a new window displayed must be specified clearly
- Names of the fields and screens should also be explicit.

2.7.3 Expected Results

The out-come of executing an instruction would have a single or multiple impact on the application. The resultant behavior of the application after test execution is the expected result.

Single Expected Result: Has a single impact on the instruction executed

Example:

Description: Click on the hyperlink ‘New Deposit’ at the top left hand corner of the Main Menu Screen
Expected Result: New Time deposit Screen should be displayed

Multiple Expected Result: Has multiple impact on executing the instruction

Example:

Description: Click on the hyperlink ‘New Deposit’ at the top left hand corner of the Main Menu Screen
Expected Result: New Time deposit Screen should be displayed
 Customer contact date should be pre-filled with the system date

Language used in the expected results should not have ambiguity. The results expressed, should be clear and have only one interpretation possible. It is advisable to use the term ‘Should’ in the expected results.

2.7.4 Pre-Requirements

Test cases cannot generally be executed with normal state of the application. Below is a list of possible pre-requirements that could be attached to the test case:

1. Enable or disable external interfaces

Example:

“Foreign exchange rate information service server” to be connected to the application

2. Time at which the test cases is to be executed

Example:

Test to be executed after 2:30 p.m. in order to trigger a warning

3. Dates that are to be maintained (pre-date or post-date) in the database before testing, as its sometimes not possible to predict dates of testing, and populate certain date fields when they are to trigger certain actions in the application

Example

Maturity date of a deposit should be the date of test. So, it is difficult to give the value of the maturity date while data designing or preparing test cases.

4. Deletion of certain records to trigger an action by the application

Example

An “document availability indicator” field to be made null, so as to trigger an warning from the application

5. Change values if required to trigger an action by the application

Example

Change the value of the interest for a deposit so as to trigger a warning by the application

2.7.5 Data definition

Data for executing the test cases should be clearly defined in the test cases. They should indicate the values that will be entered into the fields and also indicate the default values of the field.

Example

Description: Enter client’s name

Data: John Smith

OR

Description: Check the default value of minimum deposit

Data: \$100

In the cases of calculations involved, the test cases should indicate the calculated value in the expected results of the test case.

Example

Description: Check the default value of the interest

Data: \$100

This value (\$100) should be calculated using the formula specified well in advance while data design.

In brief, the entire process should be within the control of the tester, and no action is outside the tester’s anticipation.

2.8 Test Script

A **test script** is either a written set of steps, or a short program written in a programming language used to test part of the functionality of a software System.

Written test scripts include a description of the functionality to be tested taken from the requirements and the preparation required to ensure that the test can be conducted. The use of written test scripts is defined as **manual testing**.

Any test that is written as a short program is regarded as an automated test.

2.8.1 Brief on Test Scripts

This will sequence the flow of an End-to-End test path or sequence of executing the individual test condition.

Test case specifies the test to be performed on each segment. Though the sequences of a path are analyzed, navigations to test conditions are not available in the test cases.

Test scripts should ideally start from the login screen of the application. This will help in two ways:

- Start conditions are always the same, and uniformity can be achieved
- Automation of test scripts requires start and end conditions i.e. the automation tool will look for the screen to be the same, as specified in its code. Then the tool will automatically run the series of cases without intervention by the user. So, the test scripts must start and end in the same originating screen.

The test scripts must explain the navigation paths very clearly and explicitly. The objective of this is to have flexibility on the person who would execute the cases.

Test scripts sequences must also take into account the impacts the previous cases i.e. in cases of deletion of certain record, the test should not flow by searching for details of the same.

In short, the test cases in series will form the test script in case of an End-to-End test approach. In individual test conditions, the navigation and the test instruction will be a test case and this will constitute a test script.

In practice, for End-to-End test approach, test scripts are written straightway incorporating the test cases. It is only for explanation; these were categorized into two steps.

Given below is the format used for writing the test script in Functional Testing and UAT.

TEST SCRIPT ID: <Unique No>
TEST SCRIPT TITLE:
TEST OBJECTIVE:
PRE-REQUISITE:

Step	Test Procedure	Expected Results	Actual Results	Pass/Fail	Defect Log No.	Comments

Last Page:

Comment(s)

Tested By: _____ Date: _____ Reviewed By: _____ Date: _____

<Project Name> Functional Test Plan
<insert application name and version>

<Company Logo>

Test Case ID: <Unique no>		Workstation ID:
Test Case Title: <Insert Functional Test Case Title>		
Test Objective:	To verify that the Application Under Test (AUT) successfully installs, open/closes and has basic functionality under the Windows 2000 operating environment	
Pre-Requisite:	Verify Windows 2000 Core Software is installed	

Step	Test Procedure	Test Data	Expected Results	Actual Results	Pass/Fail	Initials/ Date

Comment(s)

Tested By: _____ Date: _____ Reviewed By: _____ Date: _____

UAT Test Script Template

ACCEPTANCE TEST SCRIPT		
PROJECT		
USER DEPT.		PREPARED BY:
PASS NO:		SCRIPT NO:
FEATURE TESTED:		REQUIREMENT ID:
<u>CONDITIONS:</u>		
<u>INSTRUCTIONS TO EXECUTE TEST:</u>		
1. 2.		
<u>EXPECTED RESULTS:</u>		
1. 2.		
SUCCESS/FAILURE	TESTED BY:	DATE:

“Conditions” that are tested for this test script i.e. from the corresponding scenario are entered here. Test description is entered in sequence as steps in the ‘Instruction to execute test’ and the expected results are entered ‘Expected Results’ section in sequence corresponding to the test description.

In cases of data requirement they are filled either in test description or expected result depending on the case.

2.8.2 Interaction with development team

Interaction between the testing team and development team should begin while writing the test scripts. Any interaction prior to test case writing would ideally bias both the teams. Screen shots of the various screens should be obtained from the development team as well as interact on the design of the application.

The tester should not make any changes to the test script at this point based on what the development team has presented. The contradictions between the Test Scripts and actual application are left to Project Managers decisions only. Recommendations from the test team on the contradiction would be a value addition.

2.8.3 Review of Test Scripts

Test cases are given to project leader and managers for review. The test scripts are then sent to the Client for review. Based on the review, changes have to be made to the entire block that was built i.e. test scripts and test data. The Client then marks their comments in the 'comments' section in the test preparation script. Testers should understand that, if a change is made in the test script then it requires changes in the test scripts and data.

2.8.4 Activity Report

Test lead should report to his/her test manager on day to day activity of the test team. The report should basically contain plan for the next day, activities pending, activity for the day-completed etc.

2.10 Backend Testing

The process of testing also involves the management of data, which at times are required to flow as an input into the application referred to as feeds hence, from external data stores / applications or data that is generated within the scope of the same application. The required data is to be extracted and processed to produce information as per requirements specified by the business requirements document.

The process of absorbing data into an application could be varied, depending on factors like:

- Nature of data required
- Format of data received
- Limitations of the system supplying data, in presenting data in a required pattern.

2.10.1 Understanding the application

The understanding of the application, as specified in the Business requirements and Functional Specification, plays a key role in testing. The business requirements and the functional specifications draw a parallel between the requirements and the offering of the proposed system. This gives an understanding of the data requirements for the application.

2.10.2 Data Feeds Management

The process of Data Feeds Management involves the study of the data requirements of the business, the gathering of data in a format most suited to handle the process of upload and presentation of data. The Project Management team does this study, and the outcome is a document called the Data Acquisition Document, that identifies the data required.

2.10.4 Data Upload Process

Data feeds are uploaded and stored in the application database, by executing a sequence of programs. The System Design specification Document contains information about tables used in the application database.

2.10.5 Verification of Data

Before the process of testing is commenced, verification is to be carried out to check that the tables in the database are populated with the required data. This specification will be available as part of the program functionality and the task on checking its correctness and ensuring that it adheres to the specifications lies with the tester.

2.11 Non Certified Testing

Testing may sometimes have to be performed on application, which do not have baseline documents. This situation may arise when

- Application is already in use and not accepted by the users
- Enhancement to the application is proposed

Under these situations, conventional process explained may not be possible. Under those circumstances the following preparation methodology is used

2.11.1 Documents

All possible documents should be collected from the client regarding the application. Testers must go through these documents, understand and extract as much possible information as possible.

Most important of it would be the data base layout document. This document would explain data structure and layout.

2.11.2 Screen Shots

Screen shots from the application should be captured. These screen shots should represent the application screen by screen and maintaining the flow pattern of the application.

2.11.3 Mapping

Tester at this point will have both the database and front-end screen shots. Carefully data base should be mapped by understanding the entries made in front end (input) and values displayed in front end (output). The purpose of each field, screens and functionality should also be understood. The tester should arrive at clarity on the input and output of the application.

In these cases, tester should use SDS and/or FRS to decide the validations required at field, module and application level depending on the application purpose.

3.0 Test Execution Process

The preparation to test the application is now over. The test team should next plan the execution of the test on the application. In this section, we will see how test execution is performed.

3.1 Stages of Testing

3.1.1 Three passes

Tests on the application are done on stages. Generally, the test execution takes place in three passes or sometimes four passes depending on the state of the application. They are:

Comprehensive or Pass 1: All the test scripts developed for testing are executed. Some cases the application may not have certain module(s) ready for test; hence they will be covered comprehensively in the next pass. The testing here should not only cover all test cases but also **business cycles** as defined in the application.

Discrepancy or Pass 2: All Test scripts that have resulted in a defect during the comprehensive pass should be executed. In other words, all defects that have been fixed should be retested. Function points that may be affected by the defect should also be taken up for testing. Automated test scripts captured during the “Pass 1” are used here. This type of testing is called as **Regression** testing. Defects that are not fixed will be executed only after they are fixed.

Sanity or Pass 3: This is the final round in the test process. This is done either at the client’s site or at test lab depending on the strategy adopted. This is done in order to check if the system is sane enough for the next stage i.e. UAT or production as the case may be under an isolated environment. Ideally the defects that are fixed from the previous pass are checked and freeform testing done to ensure integrity is conducted.

UAT or Pass 4: UAT (*user acceptance testing*) typically the final phase in a software development process in which the software is given to the intended audience to be tested for functionality. UAT is either done by making the software available for a free trial, typically over the Internet or by using an in-house testing panel comprised of users who would be using the product in real-world applications. UAT is done in order to get feedback from users to make any final adjustments to the programming before releasing the product to the general public/user.

UAT also is called **beta testing**, end-user testing or application testing.

3.2 Pre- Requirements for Testing

3.2.1 Version Identification

The application would contain several program files for it to function. The version of these files and a unique identification number for these files is a must for change management.

These numbers will be generated for every program file on transfer from the development machine to the test environment. The number attributed to each program file is unique and if any change is made to the program file between the time it is transferred to the test environment and the time when it is transferred back to the development for correction, it can be detected by using these numbers. These identification methods vary from one client to another.

These values have to be obtained from the development team by the test team. This helps in identifying unauthorized transfers or usage of application files by both parties involved.

The responsibility of acquiring, comparing and tracking before and after transfer lies with the test team.

Version control software (Rational ClearCase/Visual SourceSafe) can be used to transfer/maintain program files.

3.2.2 Interfaces for the application

In some applications external interfaces may have to be connected or disconnected. In both cases the development team should certify that the application would function in an integrated fashion. Actual navigation to and from an interface may not be covered in black box testing.

3.2.3 Unit and Module test plan sign off

To begin an Integrated test on the application, development team should have completed tests on the software at Unit and module levels.

Unit and Module Testing: Unit testing focuses verification effort on the smallest unit of software design. Using the Design specification as a guide, important control paths and field validations are tested. This is normally a white box testing.

Clients and the development team must sign off this stage, and hand over the test plan and defect report for the test to the testing team.

In cases of the UAT, the System Test Summary Report must be signed off before commencement of UAT.

3.3 Test Plan

This document is a deliverable to client. It contains actual plan for test execution with details to the minute.

3.3.1 Test Execution Sequence

Test scripts can either be executed in a random format or in a sequential fashion. Some applications have concepts that would require sequencing of the test cases before actual execution. The details of the execution are documented in the test plan.

Sequencing can also be done on the modules of the application, as one module would populate or formulate information required for another.

3.3.2 Allocation of test cases among the team

The Test team should decide on the resources that would execute the test scripts. Ideally, the tester who designed the test script for the module executes the test. In some cases, due to shortage of time or resource at that point of time, additional test scripts might have to be executed by some members of the team.

Clear documentation of responsibilities is done in the test plan.

3.3.3 Targets for completion of Testing

Time frames for the testing have to be decided and committed to the clients well in advance to the start of test. Some of the factors considered for doing so are

Number of cases/scripts: Depending on the number of test scripts and the resource available, completion dates are prepared

Complexity of Testing: In some cases the number of test cases may be less but the complexity of the test may be a factor. The testing may involve time-consuming calculations or responses from external interfaces etc

3.4 Automation of Test Cases

3.4.1 Capturing during First pass

As, we now understand, the tool has to capture the test sequence, inputs, outputs and calculations involved in the test cases. While executing the test cases, testers should capture them using the tool. Automation experts in the team will provide guidance.

3.4.1 Intelligent Automation

Like intelligent testing, automation can also be chosen carefully. Depending on the strategy i.e. Complete or selective the test cases are automated.

In cases of selective, only critical and complex test cases are automated. Time-consuming test cases are also automated, as it would take less time for regression testing.

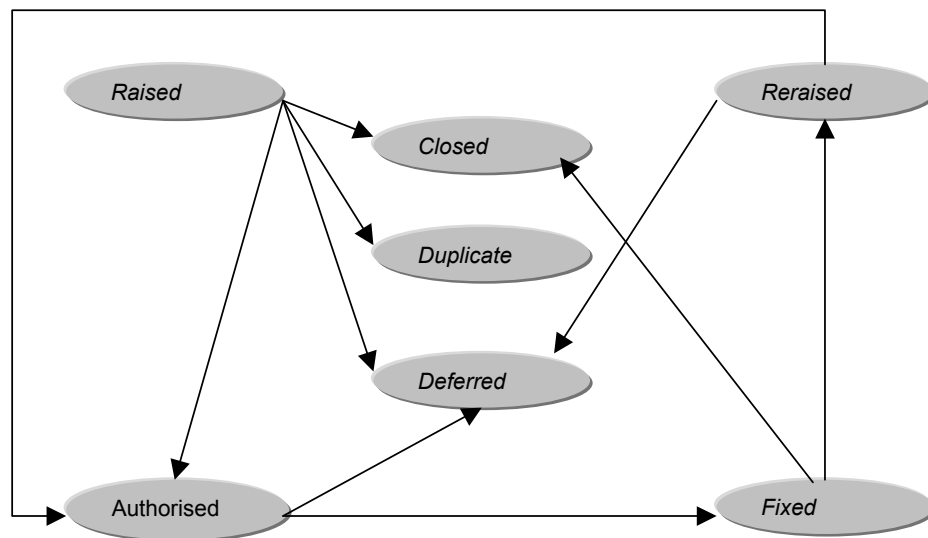
3.5 Defect Management

3.5.1 What is a defect?

A **Defect** is a product anomaly or flaw. Defects include such things as omissions and imperfections found during testing phases. Symptoms (flaws) of faults contained in software that is sufficiently mature for production will be considered as defects. Deviations from expectation that is to be tracked and resolved is also termed a defect.

An evaluation of defects discovered during testing provides the best indication of software quality. Quality is the indication of how well the system meets the requirements. So in this context defects are identified as any failure to meet the system requirements.

Life cycle of a defect is explained diagrammatically below:



3.5.2 Types of Defects

Defects that are detected by the tester are classified into categories by the nature of the defect. The following are the classification

Showstopper (X): The impact of the defect is severe and the system cannot go into the production environment without resolving the defect since an interim solution may not be available.

Critical (C): The impact of the defect is severe, however an interim solution is available. The defect should not hinder the test process in any way.

Non critical (N): All defects that are not in the X or C category are deemed to be in the N category. These are also the defects that could potentially be resolved via documentation and user training. These can be Graphic User Interface (GUI) defects are some minor field level observations.

3.5.3 Defect reporting by tester

Defects or Bugs when detected in the application by the tester must be duly reported through an automated tool. Particulars that have to be filled by a tester are

Defect Id: Number associated with a particular defect, and henceforth referred by its ID. In this case, an auto-incremented number can be generated by defect tracking tool.

Date of execution: The date on which the test script which resulted in a defect was executed

Severity: As explained, it can be Critical, Non-Critical and Showstopper

Module ID: Module in which the defect occurred

Status: Raised, Authorised, Deferred, Fixed, Re-raised, Closed and Duplicate.

Defect description: Description as to how the defect was found, the exact steps that should be taken to simulate the defect, other notes and attachments if any.

Test Script Reference No: The number of the test script which resulted in the defect

Owner: The name of the tester who executed the test case

Test Script description: The instructions in the test cases for the step in which the error occurred

Expected Result: The expected result after the execution of the instructions in the test script descriptions.

History of the defect: Normally taken care of the automated tool used for defect tracking and reporting.

Attachments: The screen shot showing the defect should be captured and attached

Responsibility: Identified team member of the development team for fixing the defect.

3.5.4 Defect Tracking by Test Lead

The test lead, categorizes the defects after meetings with the clients as,

Modify Cases: Test cases to be modified. This may arise when the testers understanding may be incorrect.

Discussion Items: Arises when there is a difference of opinion between the test and the development team. This is marked to the Domain consultant for final verdict.

Change Technology: Arises when the development team has to fix the bug.

Data Related: Arises when the defect is due to data and not coding.

User Training: Arises when the defect is not severe or technically not feasible to fix, it is decided to train the user on the defect. This should ideally not be critical.

New Requirement: Inclusion of functionality after discussion

User Maintenance: Masters and Parameter maintained by the user causing the defect.

Observation: Any other observation, which is not classified in the above categories like a user perspective GUI defect.

Reporting is done for defect evaluation and also to ensure that the development team is aware of the defects found and is in the process of resolving the defects. A detailed report of the defects is generated everyday and given to the development team for their feedback on defect resolution. A summary report is generated for every report to evaluate the rate at which new defects are found and the rate at which the defects are tracked to closure.

Defect counts are reported as a function of time, creating a **Defect Trend** diagram or report, and as a function of one or more defect parameters like category or status, creating a **Defect Density** report. These types of analysis provide a perspective on the trends or distribution of defects that reveal the system's reliability, respectively.

It is expected that defect discovery rates will eventually diminish as the testing and fixing progresses.

Defects included in an analysis of this kind are confirmed defects. Not all reported defects report an actual flaw, as some may be enhancement requests, out of the scope of the system, or describe an already reported defect. However, there is a value to looking at and analysing why there are many defects being reported that are either duplicates or not confirmed defects.

3.5.5 Tools Used

Tools that are used to track and report defects are,

ClearQuest (CQ): It belongs to the Rational Test Suite and it is an effective tool in Defects Management. CQ functions on a native access database and it maintains a common database of defects. With CQ the entire Defect Process can be customized. For e.g., a process can be designed in such a manner that a defect once raised needs to be definitely authorized and then fixed for it to attain the status of retesting. Such a systematic defect flow process can be established and the history for the same can be maintained. Graphs and reports can be customized and metrics can be derived out of the maintained defect repository.

Test Director (TD): TestDirector is an Automated Test Management Tool developed by Mercury Interactive for Test Management to help to organize and manage all phases of the software testing process, including planning, creating tests, executing tests, and tracking defects. TestDirector enables us to manage user access to a project by creating a list of Authorized users and assigning each user a password and a user group such that a perfect control can be exercised on the kinds of additions and modifications an user can make to the project. Apart from Manual Test Execution, the WinRunner automated test scripts of the project can also be executed directly from TestDirector. TestDirector activates WinRunner, runs the tests, and displays the results. Apart from the above, it is used for

- To report defects detected in the software.
- As a sophisticated system for tracking software defects.
- To monitor defects closely from initial detection until resolution.
- To analyze Testing Process by means of various graphs and reports.

3.5.6 Defects Meetings

Meetings are conducted at the end of everyday between the test team and development team to discuss test execution and defects.

Before meetings with the development team, test team should have internal discussions with the test lead on the defects reported to the test lead. This process ensures that all defects are accurate and authentic to the best knowledge of the test team.

3.5.7 Defect Metrics

Analysis on the defect report is done for management and client information. These are categorized as

Defect Age: Defect age is the time duration between the point of introduction of defect to the point of closure of the defect. This would give a fair idea on the defect set to be included for smoke test during Regression

Defect Analysis: The analysis of the defects can be done based on the severity, occurrence and category of the defects. As an example **Defect Density** is a metric which gives the ratio of defects in specific modules to the total defects in the application. Further analysis and derivation of metrics can be done based on the various components of the defect management.

3.5.8 Defects Publishing

Defects that are authorized are published in a mutually accepted media like Internet, Intranet, email etc. These are published in the Intranet and depending on the clients' requirements, defects are published either in their Intranet or Internet.

Reports that are published are

- Daily defect report
- Summarized defect report for the individual passes
- Final defect report

Format used for publishing the defects are given below with some examples.

Defect Report Sample:

Defect ID	Date of Testing	Test Script ID	Test Script Description	Defect Description	Expected Results	Severity	Status	Resp.	Comments
1	28-Jun	xx/002/003	Navigate to New Item screen through Account detail SKr, book xx	Order Template is not displayed, when New item screen is reached through 'Action' in the account details screen.	Order Template should be displayed to select the source account	Critical	Closed	Sundar	Tested and closed on 22nd july
2	28-Jun	xy/005/031	Memo Balance Incorrect	The available balance in the security account decreased by no of units * price entered in the Sell Security template	Settlement account is decreasing by the sell order amount (no of units * price entered in the sell Security template). Security account, the amount should decrease by no of units * price (original price).	Critical	Closed	Raj	Reraised on 15th July. Closed/UT by CIT 20th July. As per specification .[19-07-2000] Rajaram has clarified that the amount should be calculated on the input price. This has been closed last week itself.
3	28-Jun	xz/010/007	Sell Security with account status =900.	After the NOGO message the system was still in the sell security template.	Sell security with security status = 900 is a NOGO status. After the NOGO message the system should go the security search screen.	Non Critical	Closed	Sundar	The error message is not right 'Error in obtaining template' Reraised on 15th July. [19-07] Message has been changed. Transferred. Retest. Tested and reraised on 27th july. Tested and closed on 8th Aug.
4	28-Jun	PINT/002/016	In PINT, change the value date and check the trade date	The trade date is changing if the value date is changed.	The trade date should not change based on the change in value date	Non Critical	Closed	Raj	Changed to UT and Closed after discussion with Raj (5th July)

3.6 Test Down Times

During the execution of the test, schedules prepared earlier may slip based on certain factors. Time lost due to these should be recorded duly by the test team.

3.6.1 Server problems

Test team may come across problems with the server, on which the application is planted. Possible causes for the problems are

- Main server on which the application may have problems with number of instances on it slowly down the system
- Networking to the main server or internal network may get down
- Software compatibility with application and middleware if any may cause concerns delaying the test start
- New version of databases or middleware may not be fully compatible with the application
- Improper installation of system applications may cause delays

3.6.2 Problems on Testing side / Development side

Delays can also be from the test or development teams like

- Data designed may not be sufficient or compatible with the application (missing some parameters of the data)
- Version transferred for testing may not be the right one

3.6.3 Show Stopper

Schedule may not only slip because of the above-mentioned reason.

Module Show Stopper: Testing on some components of the application may not be possible due to fundamental errors in it, stopping further testing

Application Show Stopper: Testing the application may not be possible due to fundamental errors in it, stopping further testing

Given below is a sample of a test down time log with examples

Downtime Log

Date	Time			Application	Machine	Description	Referred to	Action taken
	From	To	Hrs					
04-Jul-00	9:45	12:20	2.35	Unable to upload DGT data, SMS Server Down	All machines	K1 values received 34 files without matching K1 values.	Clients	IST – could not be started.
05-Jul-00	10:23	11:02	.39	RM Module failure, deletion of files from control machine, Server down, SMS Error	All machines		Clients	IST - could not proceed with execution of test cases
	12:41	1:07	.26					
	4:45	5:15	.30					

3.7 File Transfer Protocol

The Acronym of FTP is File Transfer Protocol. File Transfer is used to transfer one or multiple files from one system to another. Files can also be transferred one system to another immaterial of the operating system there are functioning on. FTP supports two different modes of transfer: Binary and ASCII

3.8 Performance Testing

Performance testing is designed to test run time performance of software within the context of an integrated system. It is not until all systems elements are fully integrated and certified as free of defects the true performance of a system can be ascertained.

Performance tests are often coupled with stress testing and often require both hardware and software infrastructure. That is, it is necessary to measure resource utilization in an exacting fashion.

3.8.1 Execution

To execute the Performance Test, test cases are planned and developed for the application. The test cases constitute simulation of series of activities carried out by the user. These are called virtual scripts. Virtual scripts run in the background creating test code using the specific navigation steps that the user performs. Several such sets of scripts (scenarios) running simultaneously at different load conditions form the basis for Performance testing.

Analysis of response times provides a result on the behavior of application and of specific transactions, under different load conditions. A successful transaction passes the criteria outlined for it in the User Acceptance defined in the specifications. A stable transaction responds to the loading of extra concurrent users in a predictable manner with regard to its operational norm of success as established in the User Acceptance criteria. A successful and stable transaction fulfills both the above conditions.

3.8.2 Performance Testing Tools

LoadRunner enables you to test your system under controlled and peak load Conditions. To load your system, LoadRunner simulates an environment where Multiple users work concurrently. To generate load, LoadRunner runs multiple of Virtual Users that are distributed over a network. Using a minimum of hardware resources, these Virtual Users provide consistent, repeatable, and measurable load to exercise the system just as the real users would. While the system is under load, LoadRunner accurately measures, monitors, and analyzes the system's Performance. LoadRunner's in- depth reports and graphs provide the information that needed to evaluate the performance of the system.

4.0 Post Test Process

4.1 Sign Off

4.1.1 Sign off Criteria

In order to acknowledge the completion of the test process and certify the application, the following has to be completed

- All passes have been completed
- All test cases should have been executed
- All defects raised during the test execution have either been closed or deferred
- Show-stoppers in the last pass of the test have been rectified

4.2.2 Authorities

The following personnel have the authority to sign off the test execution process

Client: The owners of the application under test.

Project Manager: Personnel who managed the project

Project Lead: Personnel who managed the test process

4.2 Deliverables

4.2.1 Internal

The following are the internal deliverables

Test Preparation Scripts: Test script that was sent to clients and the corrections made

Data Sheets: Sheets used for designing the data for test

Minutes of meetings/discussions: Team meetings, meetings with client's etc

Project archives: These are explained further in the document

4.2.2 External

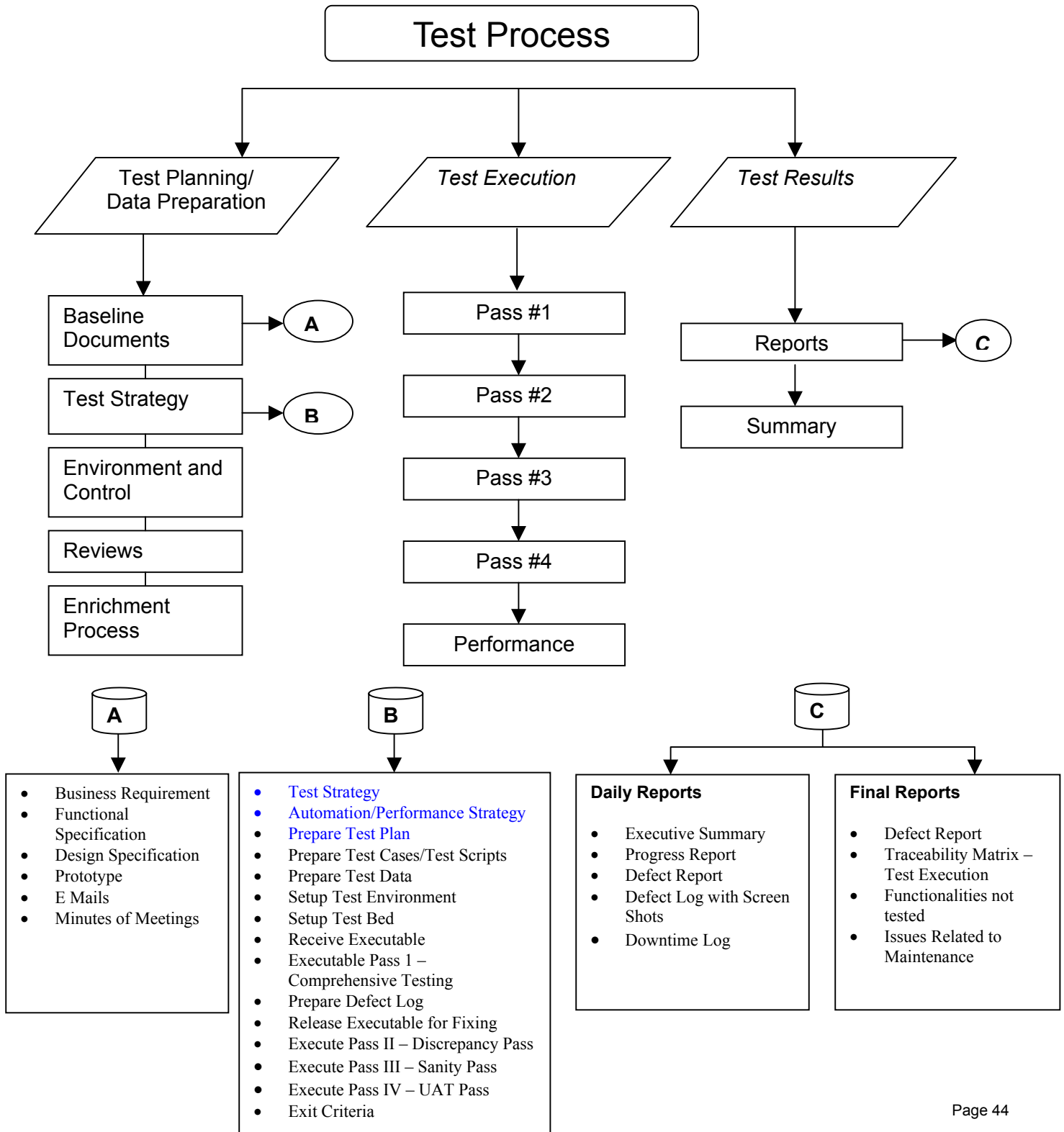
The following are the deliverables to Clients

- Test Strategy
- Effort Estimation
- Test Plan (includes Test scripts)
- Test Results
- Traceability matrix
- Pass defect report
- Final Defect report
- Final Summary Report
- Defect Analysis

4.5 Archiving

4.5.1 Tree

All documents and paper work should be archived. An archive tree is created which will help in easy retrieval of information. The diagram given below explains the tree and is quite self-explanatory.



4.5.2 Collection of Details

Details that have to be collected and personnel to be contacted are

<i>Sl No</i>	<i>Documents</i>	<i>Client</i>	<i>QA Manager</i>	<i>QA Lead</i>	<i>QA Team</i>	<i>Development Team</i>
1	Test Strategy		X	X		
2	Test Plan		X	X		
4	Test Data				X	
5	Test Scripts				X	
6	Mails on the project		X	X		
7	Defect Metrics		X	X		
9	Defect Reports		X	X		
10	FRS		X	X	X	
11	BRD/URS		X	X	X	
12	SDS			X	X	X
13	Prototype			X	X	X
14	Minutes of the meetings		X	X	X	

4.5.3 Document Warehouse

External deliverables that were discussed earlier are then put into clients and Intranet. Hence these become published documents. This can be used for future references.

4.5.4 CD Recording

All the documents collected above are then recorded into a CD-ROM using the achieving tree described above. This becomes reference material for all further clarifications, which may arise in future.