

MANUAL TESTING


TOPIC – 08 **TESTING PHASES**








01. Unit Testing

02. Integration Testing

-  **Incremental Integration Testing**
-  **Non-Incremental Integration Testing**

03. System Testing

-  **Performance Testing**
 - **Volume Testing**
 - **Stress Testing**
 - **Load Testing**
 - **Security Testing**

-  **Usability Testing**
-  **Backup Testing**
-  **Recovery Testing**
-  **Documentation Testing**
-  **Configuration Testing**
-  **Installation Testing**
-  **Compatibility Testing**

-  **Smoke Testing**
-  **Regression Testing**
-  **Sanity Testing**
-  **Retesting**
-  **Database Testing**
-  **Exploratory Testing**
-  **Localization Testing**

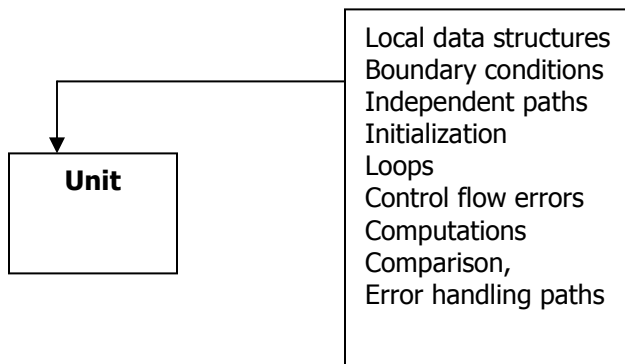
04. Acceptance Testing

-  **Alpha - Testing**
-  **Beta - Testing**

MANUAL TESTING

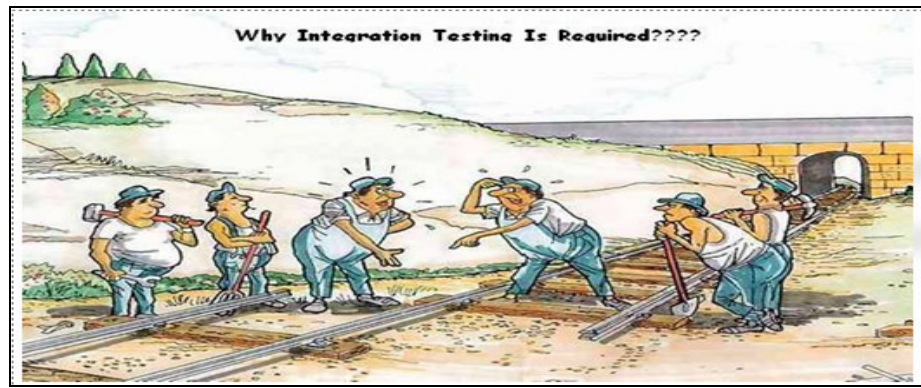
01. Unit Testing

- ❖ **Unit testing** is code-based and performed primarily by developers to demonstrate that their smallest pieces of executable code function suitably.
- ❖ The most 'micro' scale of testing to test particular functions, procedures or code modules. Also called as Module testing.
- ❖ Typically done by the programmer and not by Test Engineers, as it requires detailed knowledge of the internal program design and code.
- ❖ Purpose is to discover discrepancies between the unit's specification and its actual behavior.
- ❖ Testing a form, a class or a stored procedure can be an example of unit testing.
- ❖ Unit testing uncovers errors in logic and function within the boundaries of a component.



MANUAL TESTING

02. Integration Testing



- ❖ **Integration testing** demonstrates that two or more units or other integrations work together properly, and tends to focus on the interfaces specified in low-level design.
- ❖ Testing of combined parts of an application to determine if they function together correctly.
- ❖ The main three elements are interfaces, module combinations and global data structures.
- ❖ Attempts to find discrepancies between program & its external specification (program's description from the point of view of the outside world).
- ❖ Testing a module to check if the component of the modules are integrated properly is example of integration testing
- ❖ Modules are integrated by two ways.

Incremental Integration Testing

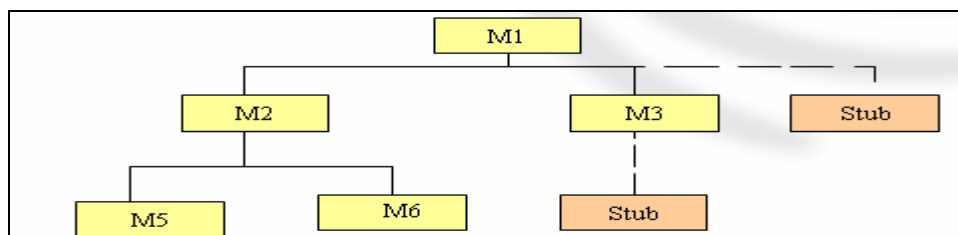
There are two types by which incremental module testing is achieved.

- A) Top down Approach**
- B) Bottom up Approach**

❖ **Top Down Incremental Module Integration:**

Firstly top module is tested first. Once testing of top module is done then any one of the next level modules is added and tested. This continues till last module at lowest level is tested.

- ❖ Integration approach can be done Depth first or Breadth-first.
- ❖ The main control module is used as a test driver
- ❖ Stubs are substituted for all components directly subordinate to the main control module.
- ❖ Depending on the approach subordinate stubs are replaced by actual components.



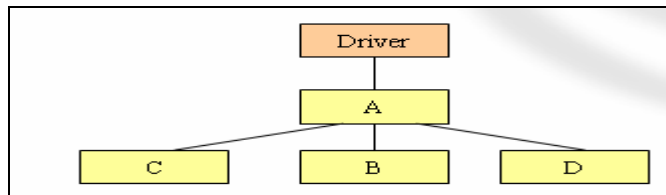
MANUAL TESTING

❖ **Bottom Up Incremental Module Integration:**

Firstly module at the lowest level is tested first. Once testing of that module is done then any one of the next level modules is added to it and tested.

This continues till top most modules is added to rest all and tested.

- ❖ Low-level components are combined into clusters (builds) that perform a specific sub function.
- ❖ A driver is written to coordinate test case input and output.
- ❖ Drivers are removed and clusters are combined moving upward in the program structure.



Top Down Testing	
<u>Advantages</u>	<u>Disadvantages</u>
1. Advantageous if major flaws occur toward the top of the program	1. Stub modules must be produced
2. Once the I/O functions are added, representation of test cases are easier	2. Stub Modules are often more complicated than they first appear to be.
3. Early skeletal Program allows demonstrations and boosts morale	3. Before the I/O functions are added, representation of test cases in stubs can be difficult
	4. Test conditions may be impossible, or very difficult, to create
	5. Observation of test output is more difficult
	6. Allows one to think that design and testing can be overlapped
	7. Induces one to defer completion of the testing of certain modules.
Bottom Up testing	
<u>Advantages</u>	<u>Disadvantages</u>
1. Advantageous if major flaws occur toward the bottom of the program	1. Driver Modules must be produced
2. Test conditions are easier to create	2. The program as an entity does not exist until the last module is added
3. Observation of test results is easier	

Non-Incremental Integration Testing (Big Bang Testing)

- ❖ Each Module is tested independently and at the end, all modules are combined to form an application.

MANUAL TESTING

03. System Testing

- ❖ **System testing** demonstrates that the system works end-to-end in a production-like environment to provide the business functions specified in the high-level design.
- ❖ Test the software in the real environment in which it is to operate. (Hardware, people, information, etc.)
- ❖ Observe how the system performs in its target environment, for example in terms of speed, with volumes of data, many users, all making multiple requests.
- ❖ Test how secure the system is and how can the system recover if some fault is encountered in the middle of procession
- ❖ System Testing, by definition, is impossible if the project has not produced a written set of measurable objectives for its product.
- ❖ Types of System Testing are :
 - ✓ Performance Testing
 - Volume Testing
 - Stress Testing
 - Load Testing
 - Security Testing
 - ✓ Usability Testing
 - ✓ Backup Testing
 - ✓ Recovery Testing
 - ✓ Documentation Testing
 - ✓ Configuration Testing
 - ✓ Installation Testing
 - ✓ Compatibility Testing
 - ✓ Smoke Testing
 - ✓ Regression Testing
 - ✓ Sanity Testing
 - ✓ Re-testing
 - ✓ Database Testing
 - ✓ Exploratory Testing
 - ✓ Localization Testing

Performance Testing

- ✓ **Performance** is the behavior of the system w.r.t. goals for time, space, cost and reliability.
- ✓ **Performance objectives:**
 - **Throughput:** The number of tasks completed per unit time. Indicates how much work has been done within an interval.
 - **Response time:** The time elapsed during input arrival and output delivery.
 - **Utilization:** The percentage of time a component (CPU, Channel, storage, file server) is busy.
- ✓ The objective of performance testing is to devise test case that attempts to show that the program does not satisfy its performance objectives.
- ✓ To ensure that the system is responsive to user interaction and handles extreme loading without unacceptable operational degradation.
- ✓ To test response time and reliability by increased user traffic.
- ✓ To identify which components are responsible for performance degradation and what usage characteristics cause degradation to occur.

MANUAL TESTING

■ **Volume Testing**

- ✓ This testing is subjecting the program to heavy volumes of data.

Objective:

- ✓ Find problems with max. Amounts of data.
- ✓ System performance or usability often degrades when large amounts of data must be searched, ordered etc.

Test Procedure:

- ✓ The system is run with maximum amounts of data.
- ✓ Internal tables, databases, files, disks etc. are loaded with a maximum of data.
- ✓ Maximal length of external input.
- ✓ Important functions where data volume may lead to trouble.

Result wanted:

- ✓ No problems, no significant performance degradation, and no lost data.

Considerations:

- ✓ Data generation may need analysis of a usage profile and may not be trivial. (Same as in stress testing.)
 - ✓ Copy of production data or random generation.
 - ✓ Use data generation or extraction tools.
 - ✓ Data variation is important!
 - ✓ Memory fragmentation important!
- ✓ **For e.g.**
 - A compiler would be fed a large source program to compile.
 - An operating systems job queue would be filled to full capacity.
 - A file system would be fed with enough data to cause the program to switch from one volume to another.
 - **Online system:** Input fast, but not necessarily fastest possible, from different input channels. This is done for some time in order to check if temporary buffers tend to overflow or fill up, if execution time goes down. Use a blend of create, update, read and delete operations.
 - **Database system:** The data base should be very large. Every object occurs with maximum number of instances. Batch jobs are run with large numbers of transactions, for example where something must be done for ALL objects in the data base. Complex searches with sorting through many tables. Many or all objects linked to other objects, and to the maximum number of such objects. Large or largest possible numbers on sum fields.
 - **File exchange:** Especially long files. Maximal lengths. Lengths longer than typical maximum values in communication protocols (1, 2, 4, 8, ... Mega- og Gigabytes). For example lengths that are not supported by mail protocols. Also especially MANY files, even in combination with large lengths. (1024, 2048 etc. files). Email with maximum number of attached files. Lengths of files that let input buffers overflow or trigger timeouts. Large lengths in general in order to tripper timeouts in communications.

MANUAL TESTING

- **Disk space:** Try to fill disk space everywhere there are disks. Check what happens if there is no more space left and even more data is fed into the system. Is there any kind of reserve like "overflow-buffers"? Are there any alarm signals, graceful degradation? Will there be reasonable warnings? Data loss? This can be tested by "tricks", by making less space available and testing with smaller volumes.
- **File system:** Maximal numbers of files for the file system and/or maximum lengths. Internal memory: Minimum amount of memory available (installed). Open many programs at the same time, at least on the client platform.

■ **Stress Testing**

- ✓ Stress testing deals with the quality of the application in the environment. The idea is to create an environment more demanding of the application than the application would experience under normal work loads. This is the hardest and most complex category of testing to accomplish and it requires a joint effort from all teams.
- ✓ A test environment is established with many testing stations. At each station, a script is exercising the system. These scripts are usually based on the regression suite. More and more stations are added, all simultaneous hammering on the system, until the system breaks. The system is repaired and the stress test is repeated until a level of stress is reached that is higher than expected to be present at a customer site.
- ✓ Race conditions and memory leaks are often found under stress testing. A race condition is a conflict between at least two tests. Each test works correctly when done in isolation. When the two tests are run in parallel, one or both of the tests fail. This is usually due to an incorrectly managed lock.
- ✓ A memory leak happens when a test leaves allocated memory behind and does not correctly return the memory to the memory allocation scheme. The test seems to run correctly, but after being exercised several times, available memory is reduced until the system fails.
- ✓ Stress testing involves subjecting the program to heavy loads or stresses. The idea is to try to "break" the system. That is, we want to see what happens when the system is pushed beyond design limits
- ✓ It is not same as volume testing. A heavy stress is a peak volume of data encounters over a short time
- ✓ In Stress testing a considerable load is generated as quickly as possible in order to stress the application and analyze the maximum limit of concurrent users the application can support
- ✓ Stress tests execute a system in a manner that demands resources in abnormal quantity, frequency, or volume

MANUAL TESTING

- ✓ Stress Tests should answer the following questions
 - Does the system degrade gently or does the server shut down
 - Are appropriate messages displayed ? E.g. Server not available
 - Are transactions lost as capacity is exceeded
 - Are certain functions discontinued as capacity reaches the 80 or 90 percent level
- ✓ **Examples :**
 1. Generate 5 interrupts when the average rate is 2 or 3
 2. Increase input data rate
 3. Test cases that require max. Memory

■ **Load Testing**

- ✓ Load testing is the process of subjecting a computer, peripheral, server, network or application to a work level approaching the limits of its specifications. Load testing can be done under controlled lab conditions to compare the capabilities of different systems or to accurately measure the capabilities of a single system. Load testing can also be done in the field to obtain a qualitative idea of how well a system functions in the "real world."
- ✓ Load testing is a part of a more general process known as performance testing. Examples of load testing include:
 - Downloading a series of large files from the Internet.
 - Running multiple applications on a computer or server simultaneously.
 - Assigning many jobs to a printer in a queue.
 - Subjecting a server to a large amount of e-mail traffic.
 - Writing and reading data to and from a hard disk continuously.
- ✓ Load testing can be conducted in two ways. Longevity testing, also called endurance testing, evaluates a system's ability to handle a constant, moderate work load for a long time. Volume testing, on the other hand, subjects a system to a heavy work load for a limited time. Either approach makes it possible to pinpoint bottlenecks, bugs and component limitations. For example, a computer may have a fast processor but a limited amount of RAM (random-access memory). Load testing can provide the user with a general idea of how many applications or processes can be run simultaneously while maintaining the rated level of performance.
- ✓ Load testing differs from stress testing, which evaluates the extent to which a system keeps working when subjected to extreme work loads or when some of its hardware or software has been compromised. The primary goal of load testing is to define the maximum amount of work a system can handle without significant performance degradation.

MANUAL TESTING

■ **Security Testing**

- ✓ Security Testing verifies that protection mechanisms built into the system will protect it from improper penetration.
- ✓ Security testing is the process of executing test cases that subvert the program's security checks.
- ✓ **Example :**
 - One tries to break the operating systems memory protection mechanisms
 - One tries to subvert the DBMS's data security mechanisms
 - The role of the developer is to make penetration cost more than the value of the information that will be obtained

Usability Testing

Usability is

- ✓ The effectiveness, efficiency and satisfaction with which specified users can achieve specified goals in a particular environment **ISO 9241-11**
- ✓ Effective— Accomplishes user's goal
- ✓ Efficient-- Accomplishes the goal quickly
- ✓ Satisfaction— User enjoys the experience
- ✓ **Test Categories and objectives**
 - Interactivity (Pull down menus, buttons)
 - Layout
 - Readability
 - Aesthetics
 - Display characteristics
 - Time sensitivity
 - Personalization
- ✓ Using specialized Test Labs a rigorous testing process is conducted to get quantitative and qualitative data on the effectiveness of user interfaces
- ✓ Representative or actual users are asked to perform several key tasks under close observation, both by live observers and through video recording
- ✓ During and at the end of the session, users evaluate the product based on their experiences

Backup Testing

- ✓ To determine whether there system or program (after failure) can be recovered without losing any data.

MANUAL TESTING

Recovery Testing

- ✓ A system test that forces the software to fail in variety of ways , checks performed
 - recovery is automatic (performed by the system itself)
 - reinitialization
 - check pointing mechanisms
 - data recovery
 - restarts are evaluated for correctness
- ✓ This test confirms that the program recovers from expected or unexpected events Events can include shortage of disk space, unexpected loss of communication

Documentation Testing

- ✓ This testing is done to ensure the validity and usability of the documentation
- ✓ This includes user Manuals, Help Screens, Installation and Release Notes
- ✓ Purpose is to find out whether documentation matches the product and vice versa
- ✓ Well-tested manual helps to train users and support staff faster

Configuration Testing

- ✓ Attempts to uncover errors that are specific to a particular client or server environment.
- ✓ Create a cross reference matrix defining all probable operating systems, browsers, hardware platforms and communication protocols.
- ✓ Test to uncover errors associated with each possible configuration

Installation Testing

Installer is the first contact a user has with a new software!!!

Installation testing is required to ensure:

- ✓ Application is getting installed properly.
- ✓ New program that is installed is working as desired.
- ✓ Old programs are not hampered.
- ✓ System stability is maintained.
- ✓ System integrity is not compromised.

Compatibility Testing

- ✓ It is done to verify the system is compatible with other system with which it should communicate. In this we are verifying the application is compatible with operating system, browser, database etc.
- ✓ **E.g.** There are instance where the website run's well in Internet Explorer but does not work with Netscape or Mozilla. So it becomes important to test the application on varied browser before it is released to the production.

MANUAL TESTING

Smoke Testing

- ✓ When a build is received, a smoke test is run to ascertain if the build is stable and it can be considered for further testing.
- ✓ Smoke testing can be done for testing the stability of any interim build.
- ✓ Smoke testing can be executed for platform qualification tests.
- ✓ Smoke Testing means the "**TEST BEFORE TESTING**"

Regression Testing

- ✓ Regression Testing is the testing of software after a modification has been made to ensure the reliability of each software release.
- ✓ Testing after changes have been made to ensure that changes did not introduce any new errors into the system.
- ✓ It applies to systems in production undergoing change as well as to systems under development
- ✓ Re-execution of some subset of test that have already been conducted
- ✓ Test suite contains
 - Sample of tests that will exercise all software functions
 - Tests that focus on software functions those are likely to be affected by the change
 - Tests for software components that have been changed

Sanity Testing

- ✓ Once a new build is obtained with minor revisions, instead of doing a through regression, sanity is performed so as to ascertain the build has indeed rectified the issues and no further issue has been introduced by the fixes. It's generally a **subset of regression testing** and a group of test cases are executed that are related with the changes made to the application.
- ✓ Generally, when multiple cycles of testing are executed, sanity testing may be done during the later cycles after through regression cycles.

	Smoke	Sanity
1	Smoke testing 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 fire and smoke. In software industry, smoke testing is a shallow and wide approach whereby all areas of the application without getting into too deep, is tested.	A sanity test is a narrow regression test that focuses on one or a few areas of functionality. Sanity testing is usually narrow and deep.
2	A smoke test is scripted--either using a written set of tests or an automated test	A sanity test is usually unscripted.
3	A Smoke test is designed to touch every part of the application in a cursory way. It's is shallow and wide.	A Sanity test is used to determine a small section of the application is still working after a minor change.
4	Smoke testing will be conducted to ensure whether the most crucial functions of a program work, but not bothering with finer details. (Such as build verification).	Sanity testing is a cursory testing; it is performed whenever a cursory testing is sufficient to prove the application is functioning according to specifications. This level of testing is a subset of regression Testing.
5	Smoke testing is normal health check up to a build of an application before taking it to testing in depth.	sanity testing is to verify whether requirements are met or not, Checking all features breadth-first.

MANUAL TESTING

Retesting

- ✓ It is done after the defect is rectified by the programmer, the tester re-test the application to verify the presence of any defect.
- ✓ Retesting the same module with additional test cases that were not considered at the initial period of testing.

Database Testing

- ✓ It is done to test data integrity and the validity of the data
- ✓ It is used for comparing data of the old DB with the new DB (Data Migration)

Exploratory Testing

- ✓ Also known as "**Random**" testing or "**Ad-hoc**" testing
- ✓ Exploratory testing is **simultaneous learning, test design, and test execution**. (...James Bach)
- ✓ A methodical approach-style is desirable
 - Test design Crafting
 - Careful Observation
 - Critical thinking
 - Diverse Ideas
 - Pooling resources (knowledge, learnings)

Localization Testing

- ✓ Localization translates the product UI and occasionally changes some settings to make it suitable for another region
- ✓ The test effort during localization testing focuses on
- ✓ Areas affected during localization, UI and content
- ✓ Culture/locale-specific, language specific and region specific areas

04. Acceptance Testing

- ❖ **Acceptance testing** is conducted by business owners and users to confirm that the system does, in fact, meet their business requirements.
- ❖ To test whether or not the right system has been created
- ❖ Usually carried out by the end user
- ❖ Two types are :
- ❖ **ALPHA TESTING & BETA TESTING**

Alpha – Testing

- ✓ Generally in the presence of the developer at the developers site

Beta - Testing

- ✓ Done at the customers site with no developer in site

====Topic - 8==== XXX ===TESTING PHASES===== XXX =====

MANUAL TESTING

TOPIC – 09

DEFECT MGT SYSTEM

01. Definition of Defect

02. Severity - Priority

03. Defect Entry Screen

04. Defect Life Cycle

05. Defect Analysis for Development Team

06. Defect Analysis for Testing Team

07. Defect Prevention

-  **Methods**
-  **Bug Report / Defect Report**
-  **Frequency distribution table for Errors**
-  **Pareto Chart**

08. Defect Tracking Tools

MANUAL TESTING

01. Definition of Defect

- ❖ A variance from a desired product attributes.
- ❖ Variance from product specifications
- ❖ Variance from customer expectations

02. Severity - Priority

❖ Severity Vs Priority

- Severity reflects the customer's or tester's perception of how bad the bug is, and priority reflects the project manager's assessment of how soon the bug should be fixed.
- Severity is the level of bug type specified by the Tester and Priority is the level of attention for fixing the bug given by the developer.

❖ Severity factors

- High : Main function of the software does not work
- Medium : Software does useful work but a degree of inconvenience is caused.
Correction is not deferrable and workaround exists
- Low : A tolerable defect, as corrections are deferrable

❖ Priority factors

- High : Problem is very important
- Medium : Problem is of an average importance
- Low : Problem is not very important.

03. Defect Entry Screen

Defect Details

Defect: 1 The list of flights is given even when past date set as Departing date

Page 1 Page 2

Details

Description

Attachments

History

* Category: Defect

* Project: Mercury Tours (HT)

* Subject: Flight Finder

* Reproducible: Y

* Detected By: alice_qc

Assigned To: james_qc

ITG Request Id:

* Status: Open

* Detected in Version: Version 1.0

* Detected on Date: 9/2/2003

Regression: N

* Severity: 3-High

Priority: 3-High

Planned

Planned Closing Version: Version 1.01

Estimated Fix Time: 5 (Days)

Actual

Closed in Version:

Actual Fix Time: (Days)

Closing Date:

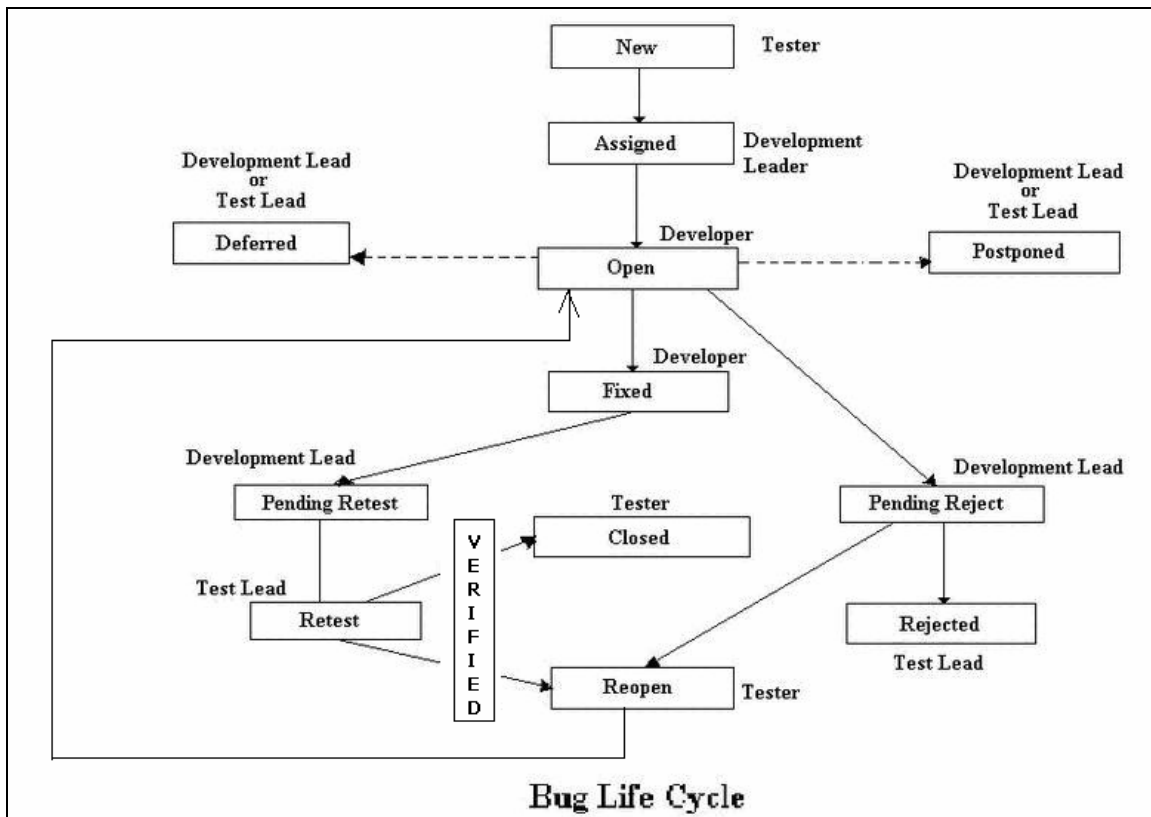
MANUAL TESTING

04. Defect Life Cycle

❖ What is a Bug Life Cycle?

The duration or time span between the first time bug is found ('New') and closed successfully (status: 'Closed'), rejected, postponed or deferred is called as 'Bug/Error Life Cycle'.

(Right from the first time any bug is detected till the point when the bug is fixed and closed, it is assigned various statuses which are New, Open, Postpone, Pending Retest, Retest, Pending Reject, Reject, Deferred, and Closed. For more information about various statuses used for a bug during a bug life cycle, you can refer to article 'Software Testing – Bug & Statuses Used During A Bug Life Cycle')



There are seven different life cycles that a bug can pass through:

< I > Cycle I:

- 1) A tester finds a bug and reports it to Test Lead.
- 2) The Test lead verifies if the bug is valid or not.
- 3) Test lead finds that the bug is not valid and the bug is 'Rejected'.

< II > Cycle II:

- 1) A tester finds a bug and reports it to Test Lead.
- 2) The Test lead verifies if the bug is valid or not.
- 3) The bug is verified and reported to development team with status as 'New'.
- 4) The development leader and team verify if it is a valid bug. The bug is invalid and is marked with a status of 'Pending Reject' before passing it back to the testing team.
- 5) After getting a satisfactory reply from the development side, the test leader marks the bug as 'Rejected'.

MANUAL TESTING

< III > Cycle III:

- 1) A tester finds a bug and reports it to Test Lead.
- 2) The Test lead verifies if the bug is valid or not.
- 3) The bug is verified and reported to development team with status as 'New'.
- 4) The development leader and team verify if it is a valid bug. The bug is valid and the development leader assigns a developer to it marking the status as 'Assigned'.
- 5) The developer solves the problem and marks the bug as 'Fixed' and passes it back to the Development leader.
- 6) The development leader changes the status of the bug to 'Pending Retest' and passes on to the testing team for retest.
- 7) The test leader changes the status of the bug to 'Retest' and passes it to a tester for retest.
- 8) The tester retests the bug and it is working fine, so the tester closes the bug and marks it as 'Closed'.

< IV > Cycle IV:

- 1) A tester finds a bug and reports it to Test Lead.
- 2) The Test lead verifies if the bug is valid or not.
- 3) The bug is verified and reported to development team with status as 'New'.
- 4) The development leader and team verify if it is a valid bug. The bug is valid and the development leader assigns a developer to it marking the status as 'Assigned'.
- 5) The developer solves the problem and marks the bug as 'Fixed' and passes it back to the Development leader.
- 6) The development leader changes the status of the bug to 'Pending Retest' and passes on to the testing team for retest.
- 7) The test leader changes the status of the bug to 'Retest' and passes it to a tester for retest.
- 8) The tester retests the bug and the same problem persists, so the tester after confirmation from test leader reopens the bug and marks it with 'Reopen' status. And the bug is passed back to the development team for fixing.

< V > Cycle V:

- 1) A tester finds a bug and reports it to Test Lead.
- 2) The Test lead verifies if the bug is valid or not.
- 3) The bug is verified and reported to development team with status as 'New'.
- 4) The developer tries to verify if the bug is valid but fails in replicate the same scenario as was at the time of testing, but fails in that and asks for help from testing team.
- 5) The tester also fails to re-generate the scenario in which the bug was found. And developer rejects the bug marking it 'Rejected'.

< VI > Cycle VI:

- 1) After confirmation that the data is unavailable or certain functionality is unavailable, the solution and retest of the bug is postponed for indefinite time and it is marked as 'Postponed'.

< VII > Cycle VII:

- 1) If the bug does not stand importance and can be/needed to be postponed, then it is given a status as 'Deferred'.

This way, any bug that is found ends up with a status of Closed.;

MANUAL TESTING

- 1. New:** When the bug is posted for the first time, its state will be "NEW". This means that the bug is not yet approved.
- 2. Open:** After a tester has posted a bug, the lead of the tester approves that the bug is genuine and he changes the state as "OPEN".
- 3. Assign:** Once the lead changes the state as "OPEN", he assigns the bug to corresponding developer or developer team. The state of the bug now is changed to "ASSIGN".
- 4. Test:** Once the developer fixes the bug, he has to assign the bug to the testing team for next round of testing. Before he releases the software with bug fixed, he changes the state of bug to "TEST". It specifies that the bug has been fixed and is released to testing team.
- 5. 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.
- 6. Rejected:** If the developer feels that the bug is not genuine, he rejects the bug. Then the state of the bug is changed to "REJECTED".
- 7. Duplicate:** If the bug is repeated twice or the two bugs mention the same concept of the bug, then one bug status is changed to "DUPLICATE".
- 8. Verified:** Once the bug is fixed and the status is changed to "TEST", the tester tests the bug. If the bug is not present in the software, he approves that the bug is fixed and changes the status to "VERIFIED".
- 9. Reopened:** If the bug still exists even after the bug is fixed by the developer, the tester changes the status to "REOPENED". The bug traverses the life cycle once again.

- 10. 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, he changes the status of the bug to "CLOSED". This state means that the bug is fixed, tested and approved.

While defect prevention is much more effective and efficient in reducing the number of defects, most organization conducts defect discovery and removal. Discovering and removing defects is an expensive and inefficient process. It is much more efficient for an organization to conduct activities that prevent defects.

05. Defect Analysis for Development Team

- ✓ Classify the Defect for future analysis
 - Category based on application layers, modules/components, Functionalities, features
- ✓ To accurately take the corrective and preventive actions for future developments
 - Category wise, severity wise defect status (Functionality, Modules, Layers)
 - Average turn around time for defect fixing
 - Defect density

06. Defect Analysis for Testing Team

- ✓ Plan retesting efforts
 - Tentative date when defect is expected to be fixed
- ✓ Analyze quality of defect reporting process
 - Defect Acceptance rate, Defect communication effectiveness
- ✓ Increase accuracy on future estimates
 - Defect reporting rate
 - Date wise defects planned to be fixed
- ✓ Generate accurate summaries for status reporting
 - Application module wise, Severity wise total/open defects
 - Functionality wise, severity wise total/open defect counts
- ✓ Monitor performance of the testers
 - Tester wise metrics

MANUAL TESTING


07. Defect Prevention

Methods

- Brain storming
- Fish Bone
- Pareto chart

Bug Report / Defect Report

QMSID : 12238				Defect Details	
Project Name :					
Project Manager : Mr. Robert					
From Date : 5/14/2007 To Date : 12/19/2007					
Defect Id.	Defect Description	Defect Severity	Stage Detected	Task	Assigned To
1	Test Scenario Description for PL_03 and PL_06 is not clear.	LOW	Test Case/Data Review	LSP/TC_TD_CR/TC_REV	Bob
2	Functionality is missing in Test Case.	MEDIUM	Test Case/Data Review	LSP/TC_TD_CR/TC_REV	Mayra
3	In TD6 - Purpose : Correct Letter No. needs to be mentioned.	LOW	Test Case/Data Review	LSP/TC_TD_CR/TC_REV	Andy
4	Formating related issues for CL29,31,32,24,20,05,04,03	MEDIUM	Test Case/Data Review	LSP/TC_TD_CR/TC_REV	Bob
5	CL03 - Prerequisites missing. CL05 - Input data in propre format. CL06 - Negative Data sets coverage.	LOW	Test Case/Data Review	LSP/TC_TD_CR/TC_REV	Mayra
6	The Purpose/Expected results for Test Data No. 6, 16, 19 & 21, have been directly copy-pasted from QTS. Need to be modified.	MEDIUM	Test Case/Data Review	LSP/TC_TD_CR/TC_REV	Andy
7	In TD6 - Purpose : Correct Letter No. needs to be mentioned.	LOW	Test Case/Data Review	LSP/TC_TD_CR/TC_REV	Bob
8	Test Scenario Description for PL_03 and PL_06 is not clear.	MEDIUM	Test Case/Data Review	LSP/TC_TD_CR/TC_REV	Mayra
9	Ujjwal's TC/TD - No Defect Found.	LOW	Test Case/Data Review	LSP/TC_TD_CR/TC_REV	Andy
10	Formating related issues for CL29,31,32,24,20,05,04,03	LOW	Test Case/Data Review	LSP/TC_TD_CR/TC_REV	Bob

Assigned To	Reported Date	Reported By	Stage Introduced	Review Type	Status	Defect Impact	Defect Priori
Bob	5/25/2007 0:00	Robin	Requirement	Self / Programmer Review	Fixed	N	HIGH
Mayra	5/25/2007 0:00	Pary	Coding	Self / Programmer Review	New	N	HIGH
Andy	6/1/2007 0:00	Hern	Testing	Peer To Peer Review	Postponed	N	HIGH
Bob	6/1/2007 0:00	D'costa	Maintanance	Peer To Peer Review	Deffered	N	Medium
Mayra	6/1/2007 0:00	Arnold	Requirement	Peer To Peer Review	Reopen	N	Medium
Andy	6/1/2007 0:00	Brute	Designing	Peer To Peer Review	Open	N	Medium
Bob	6/1/2007 0:00	Robin	Requirement	Peer To Peer Review	Fixed	N	Medium
Mayra	6/1/2007 0:00	Pary	Requirement	Peer To Peer Review	Verified	N	Medium
Andy	6/1/2007 0:00	Hern	Requirement	Peer To Peer Review	Verified	N	Medium
Bob	6/1/2007 0:00	D'costa	Test Case/Data Preparation	 Peer To Peer Review	Verified	N	LOW
Mayra	6/1/2007 0:00	Arnold	Test Case/Data Preparation	Peer To Peer Review	Verified	Y	LOW

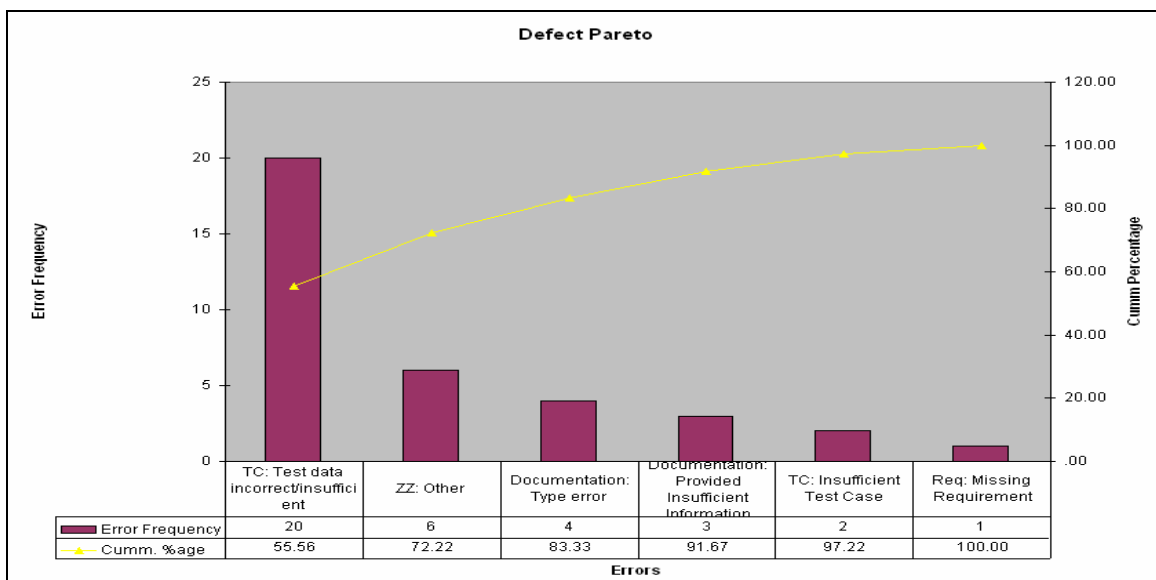
MANUAL TESTING

Defect Priority	Error Category	Root Cause	Test Case No.	Defect Count
HIGH	Documentation: Provided Insufficient Information	Inadequate inputs:Input documents inadequate/unclear	PC01-PC12	0
HIGH	Req: Missing Requirement	Competency : Test case preparation experience inadequate	PL01-PL16	0
HIGH	Documentation: Provided Insufficient Information	Personal : Cut-paste error	CL1,2,4,7	0
Medium	Documentation: Type error	Personal:Human Negligence	CL8,9,10,	0
Medium	Documentation: Type error	Competency : Test case preparation experience inadequate	CL11,14,15	0
Medium	TC: Insufficient Test Case	Inadequate inputs:Input documents inadequate/unclear	CL16,17,18	0
Medium	TC: Insufficient Test Case	Competency : Test case preparation experience inadequate	CL20,21,22	0
Medium	TC: Insufficient Test Case	Personal : Cut-paste error	CL23,25,30	0
Medium	TC: Insufficient Test Case	Personal:Human Negligence	CL33	0
LOW	ZZ: Other	Personal : Cut-paste error	Claims TC	3
LOW	TC: Insufficient Test Case	Personal:Human Negligence	Claims TC	2

Frequency distribution table for Errors

Defect Category	Frequency	Percentage	Cumm. Percentage
TC: Test data incorrect/insufficient	20	55.56	55.56
ZZ: Other	6	16.67	72.22
Documentation: Type error	4	11.11	83.33
Documentation: Provided Insufficient Information	3	8.33	91.67
TC: Insufficient Test Case	2	5.56	97.22
Req: Missing Requirement	1	2.78	100.00

Pareto Chart



MANUAL TESTING

08. Defect Tracking Tools

Product	Vendor	Description
Clear Quest	IBM Rational	A highly flexible defect and change tracking system that captures and tracks all types of change
TrackRecord	Compuware	An enterprise-wide change request management, defect and project-tracking tool.
SilkRadar	Segue Software, Inc.	A defect tracking product used to track and manage errors in your software projects
Bugzilla	Mozilla	Highly configurable Open source bug tracking system. It has Web-based database for bugs.
Bug Tracking	Bug-Track.com	Free bug tracking system. web based, fast and easy to use. Offers email notification, file attachment, tracking history, bilingual pages, 128-bit encryption connection and advance customization.

====Topic - 9==== XXX ===DEFECT MGT SYSTEM===== XXX =====

MANUAL TESTING

TOPIC – 10

METRICS

01. Metrics

MANUAL TESTING

01. Metrics

Page: 1 Metrics

* 1) Defect Density = $\frac{A+B}{C}$

A = Total No. of defects including both impact & non impact, found in all phases

B = Post Delivery Defect (Generally 3 Times)

C = Size

It has some range
+1 -1 OR +2 -2

total No. of Size = How many components are used in the App.
Ex: Form, buttons, Pages etc.
ex: Size = 10.

* 2) Avg. Defect Age = $\frac{\sum [C(A_i - B_i) * C_i]}{D}$

A = Defect Detection Phase No.

B = Defect Injection Phase No.

C = No. of Defects detected in the defect detection phase.

D = Total No. of Defects till Date

Take sum of each Defect

	1	2	3	4
1 Req. phase	1	0	0	0
2 Design phase	1	0	0	0
3 Coding	1	3	0	0
4 Testing	1	1	1	1
5 mainte				

2nd Det

$(A_1 = 3)$
 $(B_1 = 1)$
 $(C_1 = 0)$
 $(D_1 = 4)$

MANUAL TESTING

Page - (2)

Ques - 3

(*) 3) Detect Removal Efficiency : (DRE)

$$DRE = \left(\frac{A}{B} \times 100 \right)$$

A = No. of pre-delivery defects

B = Total No. of Defects

(*) 4) Review Effectiveness :

$$\text{Review Effectiveness} = \left(\frac{A}{B} \times 100 \right)$$

A = Total No. of defects found in Review

B = Total No. of Defects

(*) 5) Cost of finding Review Defect in Review :

$$CPDR = \left(\frac{A}{B} \right) \text{ Hr/Defect Unit.}$$

A = Total efforts spent on Reviews

B = No. of defect found in Reviews

MANUAL TESTING

Page ③

80% Defects are due to 20% of causes

Defect

⑥

Cost of finding a defect in Testing :-

$$CFDT = \left(\frac{A}{B} \right) \text{ HR/Defect/unit.}$$

A = Total efforts spent on Testing

B = Defects found in Testing

⑦

Cost of Quality :-

$$\% \text{ COQ} = \left(\frac{A+B+C}{D} \right) * 100$$

A = Total efforts spent on Prevention (Training)

B = Total efforts spent on Appraisal (Reviewing)

C = Total efforts spent on failure/Rework (Reworking)

D = Total efforts spent on Project.

⑧ Failure Cost :-

$$\text{Failure cost} = A + (3*B)$$

A = Efforts spent on fixing or reworking the pre-delivery defects

B = Efforts spent on fixing or reworking the post-delivery defects.

MANUAL TESTING

Page: (4)

(*) Test Case Effectiveness : Pre-8.10 Trap matrices in V&V Project

$$\% \text{ Test Case Eff.} = \left(\frac{A}{B} * 100 \right)$$

A = # of Detects using the test cases

B = # of Detects detected in Testing

(Effective ness of Test Cases)

(*) 10) Test Case Adequacy : Trap matrices - V&V

$$\% \text{ Test Case Adeq} = \left(\frac{A}{B} * 100 \right)$$

A = No. of actual Test cases

B = No. of Testcases estimated.

(*) 11) Detect Detection Index : Trap matrices V&V.

$$\% \text{ D.D.I} = \left(\frac{A}{B} \right)$$

A = # of detects detected in each phase

B = Total # of detects planned to be detected in each phase.

====Topic - 10==== XXX ===METRICS===== XXX =====

MANUAL TESTING

TOPIC – 11

Risks and Contingencies

01. Risks Identification

02. Risks Contingency

MANUAL TESTING

01. Risks Identification

- ❖ **Software Risks:** Defects tracking system crashes, Defect tracking system –database exceeds the limits, Security issues, Bug in the defect tracking system surfaced.
- ❖ **Business Risks:** License Expires for defect tracking tool, Defect metrics data not reported properly

02. Risks Contingency

- ❖ Back up the defect data periodically
- ❖ Have proper access control for the defect tracking tool
- ❖ Licensing agreements to be reviewed from time to time

====Topic - 11==== XXX ===Risks and Contingencies==== XXX =====

MANUAL TESTING

TOPIC – 12

TEST PLAN

01. Test Planning Overview

02. Test Strategy

03. Test Plan Elements

MANUAL TESTING

01. Test Planning Overview

- ❖ What is PLANNING ?
 - Developing a process & environment for achieving defined objectives
 - The success of a project will depend critically upon the effort, care and skill you apply in its initial planning
- ❖ What benefits ?

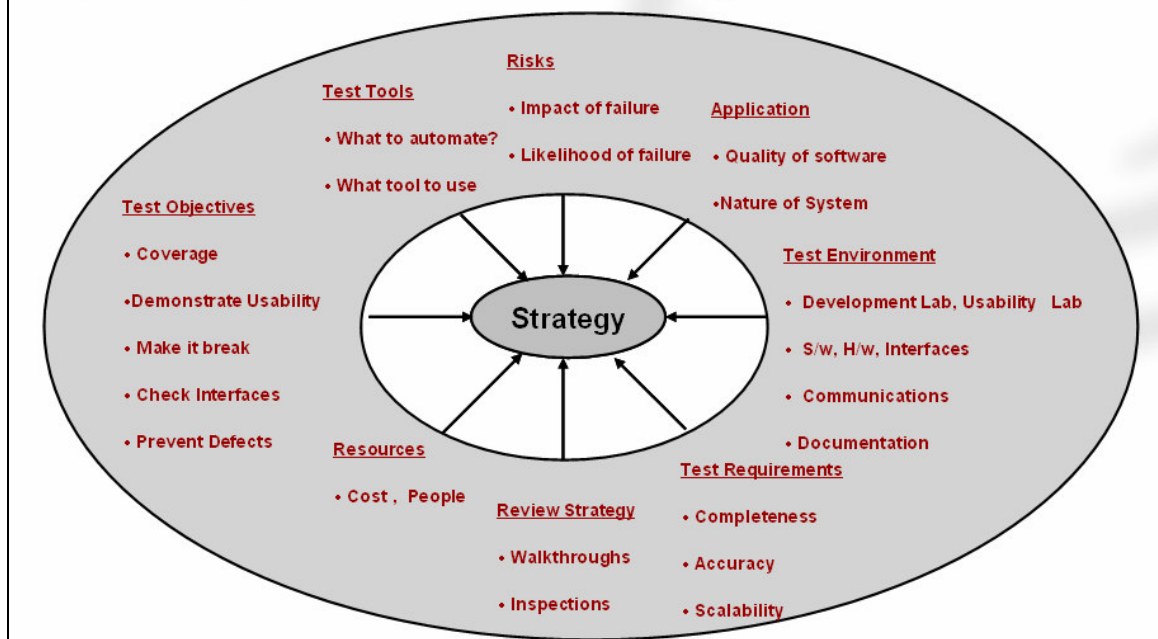
	Management	Marketing	Development	Test/QA Team
Receive From	Resources	Reasonable expectation	Quality Criteria	Realistic goals
Provide to	Understanding of the cost	Realistic dates	Clear expectation and acceptance criteria	Enhanced Approach

02. Test Strategy

- ❖ A **road map** that describes –
 - Test factors and testing objectives
 - The steps to be conducted as a part of testing
 - When these steps are planned & executed
- ❖ Test Strategy will give
 - Approach for testing including types and extent of testing
 - Resource allocation plan
 - Tools to be used
- ❖ Test strategy is based on a number of factors.
 - Product technology
 - Component selection
 - Product criticality
 - Product complexity
 - **The purpose is to clarify the major tasks and challenges of a testing project**

MANUAL TESTING

Influencing Factors on Test Strategy



❖ Steps in creating Test Strategy:

- Get the Test Development & Analyst Team together
- Identify quality goals
- Determine the quality attributes
- Understand the Release Roadmap
- Identify types of testing & coverage
- List the risks (environment, data, resources, tools)
- Identify defect management mechanism
- Identify metrics to be collected
- Identify project schedule constraints
- Identify tools that can be used
- Identify skill needs of the testing team

Test Plan Answers Questions Like	Test Strategy Answers Questions Like
What Will Be Tested?	How Is the Testing to Be Carried Out?
Who Will Test	Which Parts & Aspects of the System Needs Emphasis While Testing
Where?	What Should Be the Structured Approach to Testing?
When?	
Test Plan Gives Details of Logistics of Testing	Test Strategy Gives How Testing Will Be Carried Out in Project

MANUAL TESTING

03. Test Plan Elements

1. Introduction

- *Purpose*
 - State the purpose of the Plan, possibly identifying the level of the plan (master etc.).
 - Essentially the executive summary of the plan
- *Test Goals and Objectives*
 - This section defines Test management and execution procedure for desired test cycle of the project, adhering to Customer specific requirements.
- *Applicable document*
 - The reference documents required for testing purpose: Reference, location and comments on Business docs, User Interface docs, Test docs, Use case docs, Functional specification, Design specifications etc
- *Test Program Master Schedule*
 - This section specifies the Overall testing timeframes. All dates should be based on the most recent Milestone sheet published by the Development Team. The Schedule details such as Start-End date (on each activity level) should be included
- *Test Cycles*
 - This section of the test plan should spell out specific assumptions & estimates about the number, timing and arrangement of test cycles

2. Roles and Responsibilities

- *Project Roles and Responsibilities*
 - Specify the roles and responsibilities of Customer and Patni team
- *Test Task Structure*
 - Specify the key Testing tasks such as its owner, due date etc. E.g Ownership and sign off date of "test plan document creation, Test plan signoff, Test case review/signoff " etc.
- *Test Team Resources*
 - Reference to communication chart for T & M project not needed in FP project.

MANUAL TESTING

3. Test Program

- *Scope*
 - Essentially demarcating the elements that are to be tested/ not to be tested for a particular testing phase. Basically details on Scope-In
- *Test Approach*
 - Defining the test type & method shows how the test goals will be achieved by identifying the types of testing to use, such as unit, integration or regression testing.
 - Documenting the test types needed for this effort defines the testing activity to undertake. It gives an indication of the resources and test environments required for the testing activity
- *Entry Criteria*
 - Describe the condition that must be met or addressed to start testing. For e.g.: Is the system stable, in whatever form, is appropriate for the test phase ?
 - Are the supporting utilities, accessories and prerequisites available in forms that testers can use? Has developer testing been completed?
 - Is the test environment – lab, hardware, software and system administration support-ready?
- *Verification Methods*
 - Detailing on methods to check the traceability from Requirements to test case to verify the proper coverage and tracking that defects are pointing to the related requirement correctly.
- *Deliverables*
 - Specifies list of deliverable to Client.
- *Exclusion*
 - Specifies scope exclusions as per proposal
- *Test Process risk*
 - Describe details of Risks, mitigation and contingency. Can embed Risk management reports from ePMS.
 - Add any risk & issues for the testing effort
 - Identify what software is to be tested and what the critical areas are. There are some inherent software risks (such as complexity, Multiple Interfaces, Govt. rules & regulations). These needs to be identified

MANUAL TESTING

4. Test Environment

- *Test environment configuration*
 - Pre - Integration Test Environment
 - Integration Test Environment
 - Acceptance Test Environment etc...
- *Test Data*
 - Provide description on methodology of Test data creation/management.

5. Test Execution

- Describe test execution activities as they apply to test cases and/or test scripts.
- *Test Execution Reporting*
 - Specify different status reports for day-to-day reporting
 - Test Execution Metrics are captured during test execution.
- *Defect Tracking*
 - Testing tool
 - Testing Defect flow
 - Re-test execution flow
 - Defect Closure
 - Confirmation of Exit Criteria

6. Release Management

- One of the major interfaces between the overall project and testing occurs when new revisions, builds and components are submitted to the test team for testing
- Predefined release plan is needed
- Key elements of the test release process needs to be defined

7. Assumptions, Constraints & Dependencies

- Add assumptions made during Planning for Constraints
- Add constraints for this testing effort for Dependencies
- Add any external dependencies (outside of the test group) for this testing effort, like availability of resources, tools, environment, signoffs, etc.

8. References

- Acronyms
 - Identify & document list of terminology that may be used throughout the document and the test effort.
- Definitions
 - Document list of definitions that may be used throughout the document and the test effort
- Documents List all the documents that support this test plan

====Topic - 12==== XXX ===TEST PLAN===== XXX =====

MANUAL TESTING

TOPIC – 13

Configuration Mgt

- 01. What is Configuration Management ?**
- 02. Problems resulting from poor configuration Management**
- 03. Definition of Configuration Management**

MANUAL TESTING

01. What is Configuration Management ?

What is configuration management?

Our systems are made up of a number of items (or things). Configuration Management is all about effective and efficient management and control of these items.

During the lifetime of the system many of the items will change. They will change for a number of reasons; new features, fault fixes, environment changes, etc. We might also have different items for different customers, such as version A contains modules 1,2,3,4 & 5 and version B contains modules 1,2,3,6 & 7. We may need different modules depending on the environments they run under (such as Windows NT and Windows 2000).

An indication of a good Configuration Management system is to ask ourselves whether we can go back two releases of our software and perform some specific tests with relative ease.

02. Problems resulting from poor configuration Management

Problems resulting from poor configuration management

Often organizations do not appreciate the need for good configuration management until they experience one or more of the problems that can occur without it. Some problems that commonly occur as a result of poor configuration management systems include:

- the inability to reproduce a fault reported by a customer;
- two programmers have the same module out for update and one overwrites the other's change;
- unable to match object code with source code;
- do not know which fixes belong to which versions of the software;
- faults that have been fixed reappear in a later release;
- a fault fix to an old version needs testing urgently, but tests have been updated.

MANUAL TESTING

03. Definition of Configuration Management

Definition of configuration management

A good definition of configuration management is given in the ANSI/IEEE Standard 729-1983, Software Engineering Terminology. This says that configuration management is:

- "the process of identifying and defining Configuration Items in a system,
- controlling the release and change of these items throughout the system life cycle,
- recording and reporting the status of configuration items and change requests, and
- verifying the completeness and correctness of configuration items."

This definition neatly breaks down configuration management into four key areas:

- configuration identification;
- configuration control;
- configuration status accounting; and
- configuration audit.

Configuration identification is the process of identifying and defining Configuration Items in a system. Configuration Items are those items that have their own version number such that when an item is changed, a new version is created with a different version number. So configuration identification is about identifying what are to be the configuration items in a system, how these will be structured (where they will be stored in relation to each other) the version numbering system, selection criteria, naming conventions, and baselines. A baseline is a set of different configuration items (one version of each) that has a version number itself. Thus, if program X comprises modules A and B, we could define a baseline for version 1.1 of program X that comprises version 1.1 of module A and version 1.1 of module B. If module B changes, a new version (say 1.2) of module B is created. We may then have a new version of program X, say baseline 2.0 that comprises version 1.1 of module A and version 1.2 of module B.

Configuration control is about the provision and management of a controlled library containing all the configuration items. This will govern how new and updated configuration items can be submitted into and copied out of the library. Configuration control also determines how fault reporting and change control is handled (since fault fixes usually involve new versions of configuration items being created).

Status accounting enables traceability and impact analysis. A database holds all the information relating to the current and past states of all configuration items. For example, this would be able to tell us which configuration items are being updated, who has them and for what purpose.

Configuration auditing is the process of ensuring that all configuration management procedures have been followed and of verifying the current state of any and all configuration items is as it is supposed to be. We should be able to ensure that a delivered system is a complete system (i.e. all necessary configuration items have been included and extraneous items have not been included).

MANUAL TESTING

TOPIC – 14

Client Server Architecture / Testing

- 01. What is Client Server Computing ?**
- 02. Architecture for Client Server System**
- 03. Critical Issues involved in Client / Server Mgt**
- 04. Client Server Testing in different Layers**

MANUAL TESTING

01. What is Client Server Computing ?

Client/Server Software Testing

What is Client/Server Computing:

Client/Server computing is a style of computing involving multiple processors, one of which is typically a workstation and across which a single business transaction

02. Architecture for Client Server System

Architectures for Client/Server System:

Both traditional Client/Server as well as netcentric computing is tiered architectures. In both cases, there is a distribution of presentation services, application code, and data across clients and servers. In both cases, there is a networking protocol that is used for communication between clients and servers. In both cases, they support a style of computing where processes on different machines communicate using messages. In this style, the "client" delegates business functions or other tasks (such as data manipulation logic) to one or more server processes. Server processes respond to messages from clients.

A Client/Server system has several layers, which can be visualized in either a conceptual or a physical manner. Viewed conceptually, the layers are presentation, process, and database. Viewed physically, the layers are server, client, middleware, and network.

Client/Server 2-tiered architecture:

2-tiered architecture is also known as the client-centric model, which implements a "fat" client. Nearly all of the processing happens on the client, and client accesses the database directly rather than through any middleware. In this model, all of the presentation logic and the business logic are implemented as processes on the client. 2-tiered architecture is the simplest one to implement. Hence, it is the simplest one to test. Also, it is the most stable form of Client/Server implementation, making most of the errors that testers find independent of the implementation. Direct access to the database makes it simpler to verify the test results. The disadvantage of this model is the limit of the scalability and difficulties for maintenance. Because it doesn't partition the application logic very well, changes require reinstallation of the software on all of the client desktops.

3-tiered architecture:

For 3-tiered architecture, the application is divided into a presentation tier, a middle tier, and a data tier. The middle tier is composed of one or more application servers distributed across one or more physical machines. This architecture is also termed the "the thin client—fat server" approach. This model is very complicated for testing because the business and/or data objects can be invoked from many clients, and the objects can be partitioned across many servers. The characteristics make the 3-tiered architecture desirable as a development and implementation framework at the same time make testing more complicated and tricky.

MANUAL TESTING

03. Critical Issues Involved in Client / Server Mgt

Critical Issues Involved in Client/Server System Management:

Hurwitz Consulting Group, Inc. has provided a framework for managing Client/Server systems that identifies eight primary management issues.

- a. Performance
- b. Problem
- c. Software distribution
- d. Configuration and administration
- e. Data and storage
- f. Operations
- g. Security
- h. License

04. Client Server Testing in different Layers

Client/Server Testing in Different Layers:

Graphical user Interface testing: GUI testing involves the testing of the function that allows end-users to customize GUI objects. Many GUI development tools give the users the ability to define their own GUI objects. The ability to do this requires the underlying application to be able to recognize and process events related to these customer.

Testing on the Server Side:

- **Volume Testing:**

To find weaknesses in the system with respect to its handling of large amounts of data during short time periods.

Ensures that the system will process data across physical and logical boundaries such as across servers and across disk partitions on one server

- **Stress Testing**

To show that the system has the capacity to handle large numbers of processing transactions during peak periods.

Examples

- Everyone is logging back onto an on-line system after it has been down.

- Numerous jobs are fired up after down time in batch environment

- **Performance Testing**

Performance Testing can be in parallel with Volume and Stress testing to assess performance under all load conditions.

Performance is assessed in terms of response times and throughput rates under differing processing and configuration conditions.

Should be tested under emulations of each business processing cycle (e.g. month-end, Quarter-end, Semi-annual, and annual)

- **Data Recovery Testing**

Extremely important type of server testing.

Scripts which investigate data recovery and system restart capabilities can save a lot of money and time which could be lost when a production system fails.

Recovery testing is even more important because data stored on networked servers can be configured in many different ways.

MANUAL TESTING

- **Data Security**

Misuse of the database

Unauthorized access

Be ware of third party software which allows users to login into the database with privileges outside of the controlled C-S application environment.

- ? **Compatibilty Testing**

Functionally consistent interface across all platforms

Interface may appear in any physical form as long as the same basic user behavior results in same response

- ? **Configuration Testing**

Testing the system in all the hardware and software environments in which it is expected to operate.

- **Backup & Restore Testing**

All data base and file servers should have defined backup procedures and defined restore procedures.

These procedures should be tested as part of server testing.

Can provide a benchmark for time and resources required when restoring databases.

- **Error Trapping**

Capabilities which allow an orderly shut down of the application and not just allow operating system errors to be generated

Should complement data recovery and system restart procedures Error trapping logic can record the problem, by-pass the corrupt data, and continue processing as an alternative to an system shut down.

====Topic - 14==== XXX ===Client Server Architecture / Testing === XXX =====

MANUAL TESTING

TOPIC – 15

Web Testing

01. What is Web Testing?

MANUAL TESTING

01. What is Web Testing?

Anyone can test a web page or even an entire site for accessibility. The necessary knowledge isn't PhD level or even too vast. It does require familiarity with HTML and CSS, the ability to appreciate the unique challenges faced by users with various disabilities, and an understanding of the W3C Accessibility Guidelines. Beyond that, all you need is the desire and time.

Step 1 – Validate HTML and CSS

This step may come as a surprise to many. After all, wouldn't invalid code either not work or leave a visible bug? Actually, the answer is not necessarily.

The reason can be that some WYSIWYG editors generate invalid code and hard-core programmers who write their code by hand can easily omit some bit of HTML or CSS "grammar". This doesn't mean non-functioning code it just means it doesn't meet the standards. I won't go into specifics here, just think of it as sort of similar to formal collegiate writing. There is a particular standard which is expected. A paper could be written differently, more "free form", it could contain all the ideas and arguments, and it could be just as well thought out – but because it doesn't meet the standard it would not get a top grade.

Validating your code has a number of advantages. It decreases the probability of cross browser problems, it tends to eliminate or reduce so called code bloat, and valid code tends to be easier to maintain as well as being compatible with a broader range of assistive technologies used by people with disabilities.

Step 2 – Automated Accessibility Testing

Automated accessibility testing is an often misunderstood step in the overall process. To some it is everything that needs to be done. "My site is Bobby compliant. Doesn't that mean it's accessible?" To others it's a red herring and should be avoided all together. My take is that it's an invaluable step. When writing an article I rely on the spellchecker to catch my typos even though I know I still need to go through and check out the copy myself to make sure I have written "Dave" and not "Cave", for instance. Automated testing finds many issues which could easily be missed by reading the code and so I always begin with it.

Depending on the scale of your project you might be able to use one of several free web based validators or you may opt to buy one of the testing packages available on the market.

The report you will get will include tests which cannot be run by the validator but which it flags for manual examination. Make sure to go through these as well. Most of the tools will describe the issue enough for someone with the above mentioned prerequisites to test.

And lastly, make sure to have any issues raised, fixed before continuing. Doing so will greatly reduce the time required for the remainder of testing.

And please, I cannot emphasize enough that automated testing alone cannot assure accessibility. Please continue with the steps below.

Step 3 – Keyboard Testing

This a simple but very important step. Hide you mouse and navigate your web site using only your keyboard. If you have never done this then you are likely to learn something.

Various groups of people can't or don't want to use a mouse. For some it's just confusing or difficult, especially those with certain motor control problems or sometimes seniors. For others, like blind web users, it's impossible. Making sure every link, form field, button, or any other

MANUAL TESTING

functionality in the page is accessible via the keyboard is a basic necessity of web accessibility – but you may also find that to get to the main content or primary form on the page you need to click the Tab Key many times. Though technically accessible, this is extremely inconvenient.

Again, be sure to make any changes required which this phase of testing brought up before continuing.

Step 4 – Screen Reader Testing

To conduct screen reader testing you will need to install the necessary software . It will take some time to get used to and configure your screen reader so be patient. Begin by simply turning off your monitor and listening to your page. Does it make sense? Many web designs depend on visual cues and can get close to unintelligible when those cues aren't available.

Next, try to carry out one or more of the tasks your website was built for. If it's an online store, find a product and make a purchase. If it's an informational site then find key information. Remember – this is the reason you built the site and it is the reason you are making it accessible. If its core functionality depends on a complex form, can you tell which fields are required? If it's a shopping cart, can you see how much you have spent before making the purchase?

Step 5 – Target Audience Testing

Various conventions of web design have emerged in the course of the World Wide Web's short existence which we have grown used too and even depend on to help us navigate a new site. Links appear in a different color (often blue) and underlined. Site wide or global navigation is usually found along the top of the page. Small pictures can often be clicked to get a bigger one. Similarly, there are conventions used in quality accessible design but naturally those of us who aren't dependant on accessible design may not be aware of them. These might include links, sometimes invisible, along the top of a page which allow the user to skip to various parts of the page, colors with high contrast values or just consistent design throughout the site.

Web accessibility isn't just fulfilling a set of requirements or validating against predefined checkpoints. It also means quality design. And just as it's best to leave questions of browser based user interface design to an expert it's best to have your site checked over by an expert in screen reader user interface design when considering accessibility. And though in theory, there is no reason a sighted specialist couldn't become such an expert, one who is dependant on screen readers will more likely be intimate with their functions and use, the frustrations of poor web site design and solutions which ease or eliminate those frustrations in practice, not just in theory.

Conclusion

Website accessibility is something every company and organization needs to consider if they have or are planning to have a web presence. It's not only morally correct but it increases your potential customer base and it is increasingly becoming the law.

Testing doesn't need to be difficult or expensive. Depending on the scope of your site compliance can be achieved entirely in-house by following the steps above. If you haven't got the available resources or feel outside, expert help is required there are a number of consultants from which to choose. One should be aware, however, that the services rendered can range from automated testing only (perhaps appropriate for a web development team which is not particularly knowledgeable in accessibility issues but able to understand and implement post test recommendations) to full, regular audits costing 10's of thousands (possibly appropriate for a company with a large and regularly changing website such as an on-line auctioneer or a newspaper). It is vital, therefore, to understand one's needs and know which service to seek.

For any larger testing effort target audience testing is key. Unfortunately, most accessibility testing companies don't offer the service, and some don't even offer screen reader testing.

MANUAL TESTING

Web Testing:

Web sites are client/server application with web server & browsers client

Issues related to website:

- What are the expected loads on the server ?
- Who is the target audience?
- What kind of browsers will they be using?
- What kind of performance is expected on the client side ?
- What kinds of security will be required ?
- How reliable are the site's Internet connections required to be?
- How will internal and external links be validated ?
- How are browser caching, variations in browser option settings, dial-up, connection variability, and real-world internet 'traffic congestion' problems to be accounted for in testing?

Types of website testing:

1) Content Testing

Content Testing verifies a web site's content such as images, clip art and factual text.

2) Database Testing

Most web sites of any complexity store and retrieve information from some type of database. Clients often want to test the connection between their web site and database in order to verify data and display integrity.

3) Functionality Testing

Functionality testing ensures that the web site performs as expected.

4) Compatibility Testing

Compatibility testing tests your web site across a wide variety browser/operating system combinations

5) Server Side Testing

Server side testing tests the server side of the site, rather than the client side

6) Performance Testing

Performance Testing measures the web site's performance during various conditions

7) Stress and Load Testing

Load Testing verifies that a web site can handle a particular number of concurrent users while maintaining acceptable response times

Stress Testing – to find the Threshold point

====Topic - 15==== XXX === Web Testing ===== XXX =====

MANUAL TESTING

TOPIC – 16

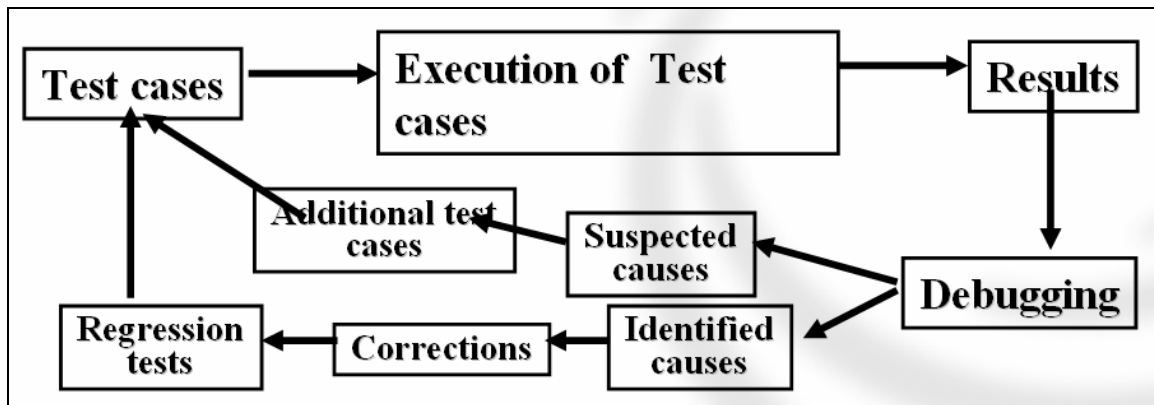
Debugging

01. Debugging

MANUAL TESTING

01. Debugging

- ❖ Occurs as a consequence of successful testing
- ❖ Is an action that results in the removal of the error
- ❖ Results of the test give a "symptomatic" indication of the software problem
- ❖ The symptoms may be caused by human errors.
- ❖ The symptom may be because of a timing problem rather than processing problem.
- ❖ The symptom may be intermittent.



- ❖ Testing is a structured process that identifies an error's "symptoms"
- ❖ Debugging is a diagnostic process that identifies an error's "cause"
- ❖ Methods/Approach used for debugging:
 - Brute force
 - Cause elimination – Induction or deduction
 - Backtracking
- ❖ Example of debugging by Brute Force are
 1. By studying Storage Dumps I.e. usually a crude display of storage location
 2. by invoking run-time traces
 3. by scattering print statements
 4. by use of automated debugging tools
- ❖ **Cause elimination**
- ❖ Debugging by Induction
 1. Locate data about what program did correctly/incorrectly
 2. Organize data
 3. Devise a hypothesis about the cause of the error
 4. Prove the hypothesis
- ❖ Debugging by deduction
 1. Enumerate the causes of error
 2. Eliminate each cause of error
- ❖ Debugging by Backtracking
 - Beginning at the place where the symptom is uncovered, the source code is traced backward until the site of the cause is found

MANUAL TESTING

- ❖ **Error Analysis - When Correcting the error , ask these three questions :**
 - Is the cause of the bug reproduced in another program?
 - What "next bug" might be introduced by the fix that I'm about to make?
 - What could we have done to prevent this bug in the first place?

====Topic - 16==== XXX === **Debugging** ===== XXX =====