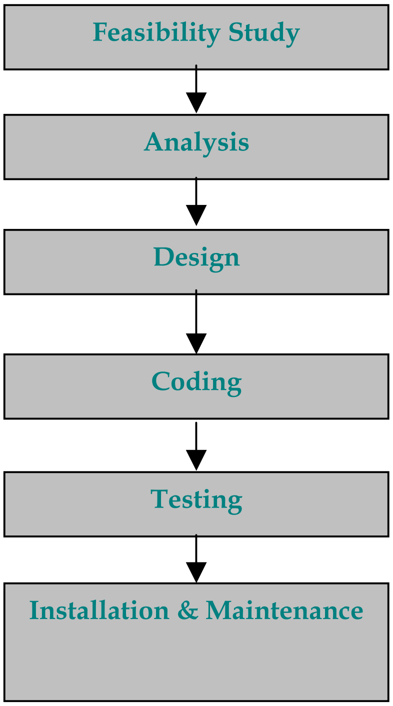
Testing Manual

CONTENTS

1. The Software life cycle

All of the stages from start to finish that take place when developing new Software.



- **Feasibility Study and problem Analysis**

– What exactly is this system supposed to do?
Determine and spell out the details of the problem.

- **Design**

– How will the system solve the problem?

- **Coding**

– Translating the design into the actual system.

- **Testing**

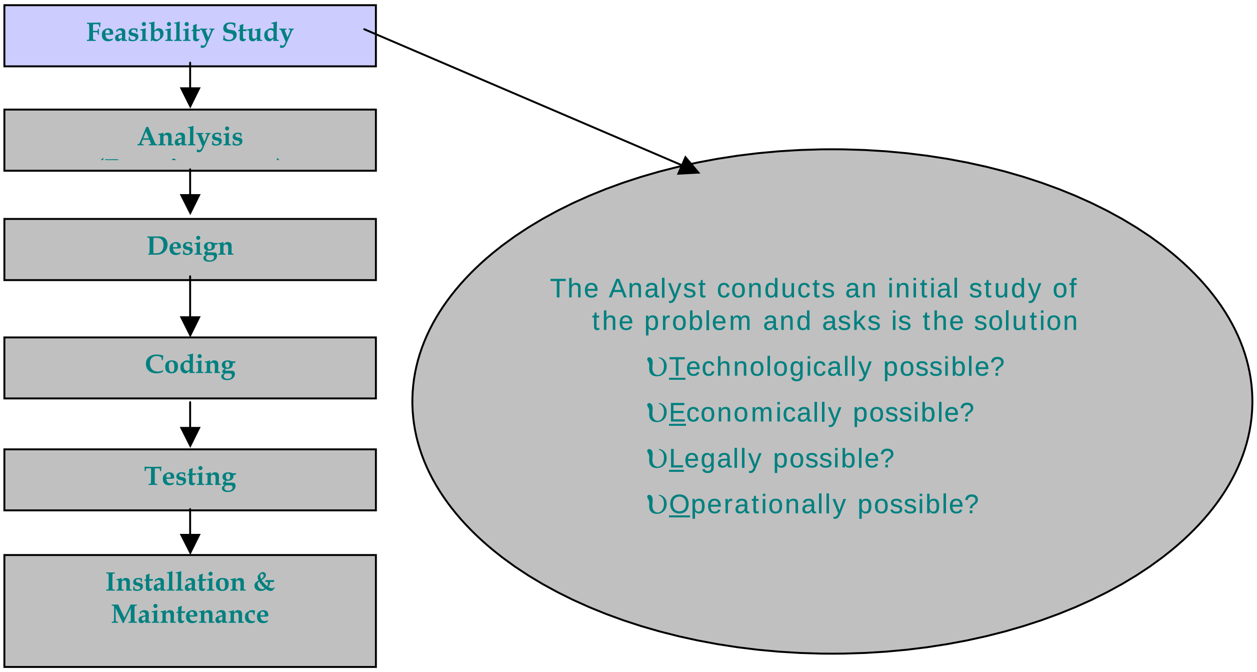
– Does the system solve the problem?
– Have the requirements been satisfied?
– Does the system work properly in all situations?

- **Maintenance**

– Bug fixes

- The software life-cycle is a description of the events that occur between the birth and death of a software project inclusively.
- Defines the concrete strategy to engineer some software artifact
- SDLC is separated into phases (steps, stages)
- SLDC also determines the order of the phases, and the criteria for transitioning from phase to phase

1.1 Feasibility Study:



The Analyst conducts an initial study of the problem and asks is the solution

ʋTechnologically possible?

ʋEconomically possible?

ʋLegally possible?

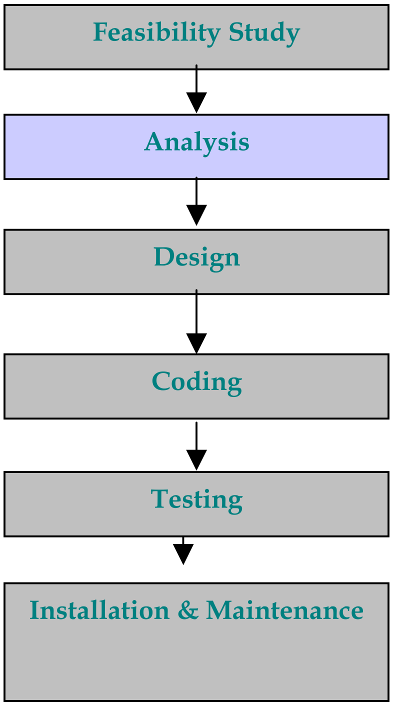
ʋOperationally possible?

The feasibility report

- Applications areas to be considered eg
- Stock control, purchasing, Accounts etc
- System investigations for each application
- Cost estimates
- System requirements
- Timescale for implementation
- Expected benefits

1.2 Systems Analysis:

- System analysis and design is the process of………
- Investigating a business…….
- With a view to determining how best to manage the various procedures and information processing tasks that it involves.



1.2.1 The Systems Analyst

- Performs the investigation…..
- and might recommend the use of a computer to improve the efficiency of the information system being investigated.

1.2.2 Systems Analysis

- The intention……..
- to determine how well a business copes with its current information processing needs,
- and whether it is possible to improve the procedures in order to make it more efficient or profitable.

The System Analysis Report

- BRS(Business Requirement Document)
- FRS(Functional Requirement Document) Or Functional specifications
- Use Cases(User action and system Response)
- [These 3 are the Base documents for writing Test Cases]
- Documenting the results
  - systems flow charts
  - data flow diagrams
  - organization charts
  - report

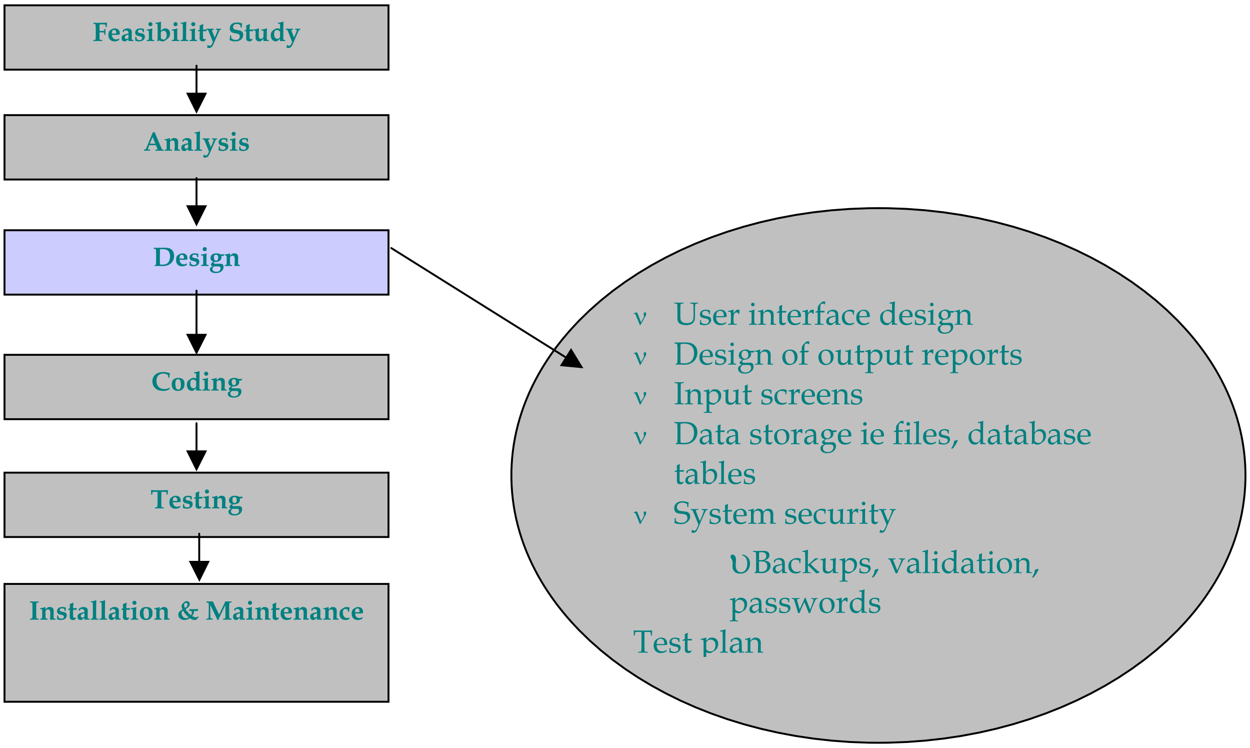
Note: FRS contains Input, Output, process but no format.
Use Cases contains user action and system response with fixed format.

1.3 Systems Design:

•Planning the structure of the information system to be implemented.

•Systems analysis determines what the system ***should do***...

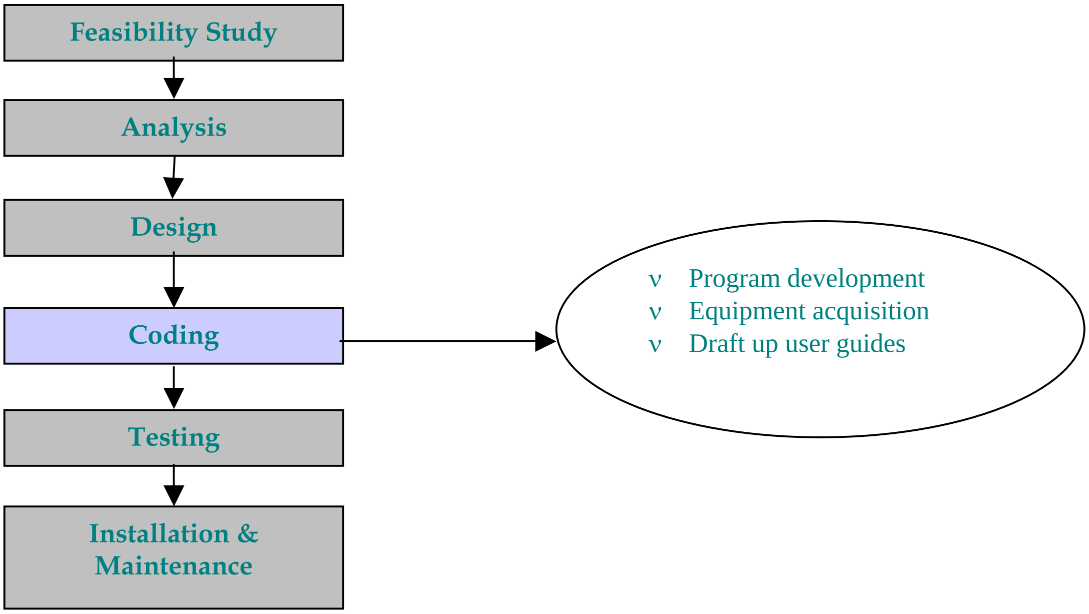
• and design determines how it ***should be done***.


Feasibility Study
Analysis
Design
Coding
Testing
Installation & Maintenance
ν User interface design
ν Design of output reports
ν Input screens
ν Data storage ie files, database tables
ν System security
υBackups, validation, passwords
Test plan

System Design Report

- Design Document that consist of Architectural Design, Database Design, Interface Design

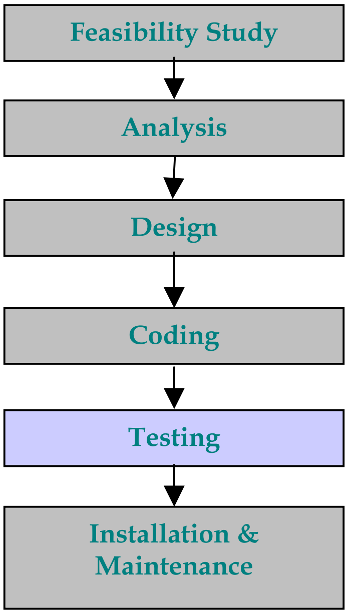
1.4 Coding:



Coding Report

- All the programs, Functions, Reports that related to Coding.

1.5 Testing:



1.5.1 What Is Software Testing?

IEEE Terminology: An examination of the behavior of the program by executing on sample data sets.

Testing is Executing a program with an intention of finding defects

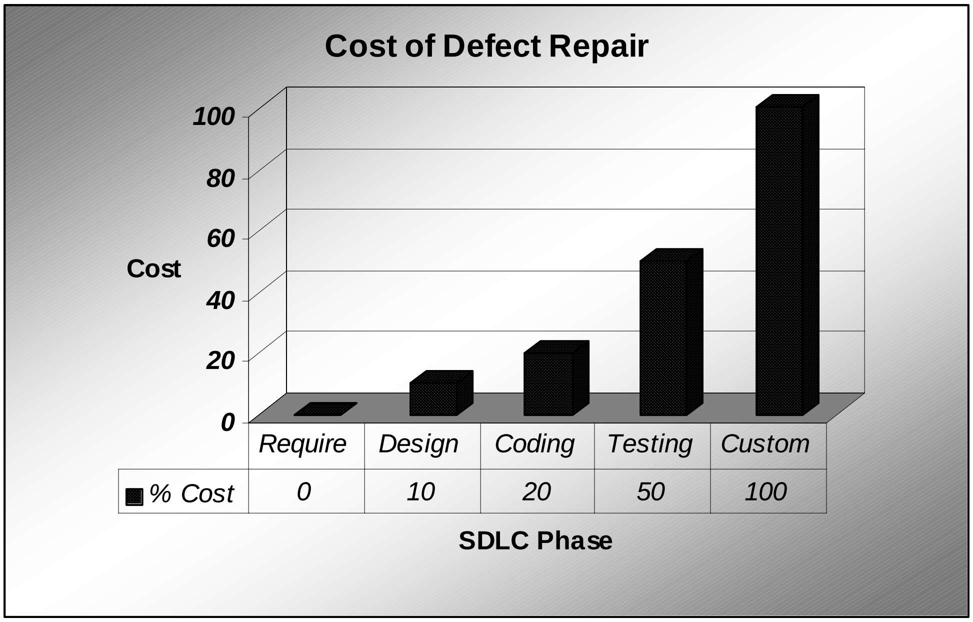
Testing is executing a program with an indent of finding Error/Fault and Failure.
Fault is a condition that causes the software to fail to perform its required function.
Error refers to difference between Actual Output and Expected Output.
Failure is the inability of a system or component to perform required function according to its specification.
Failure is an event; fault is a state of the software, caused by an error.

Why is Software Testing?

To discover defects.
To avoid user detecting problems
To prove that the software has no faults
To learn about the reliability of the software.
To ensure that product works as user expected.
To stay in business
To avoid being sued by customers
To detect defects early, which helps in reducing the cost of defect fixing.

COST OF DEFECT REPAIR

Phase	% Cost
Requirements	0
Design	10
Coding	20
Testing	50
Customer Site	100



How exactly Testing is different from QA/QC

Testing is often confused with the processes of quality control and quality assurance.
Testing is the process of creating, implementing and evaluating tests.
Testing measures software quality.
Testing can find faults; when they are removed, software quality is improved.
QC is the process of Inspections, Walk-troughs and Reviews.
QA involves in Monitoring and improving the entire SDLC process, making sure that any agreed-upon standards and procedures are followed, and ensuring that problems are found and dealt with.

Why should we need an approach for testing?

Yes, We definitely need an approach for testing.
To over come following problems, we need a formal approach for Testing.

Incomplete functional coverage: Completeness of testing is difficult task for Testing team with out a formal approach. Team will not be in a position to announce the percentage of testing completed.

No risk management: This is no way to measure overall risk issues regarding code coverage and quality metrics. Effective quality assurance measures quality over time and starting from a known base of evaluation.

Too little emphasis on user tasks: Because testers will focus on ideal paths instead of real paths. With no time to prepare, ideal paths are defined according to best guesses or developer feedback rather than by careful consideration of how users will understand the system or how users understand real-world analogues to the application tasks. With no time to prepare, testers will be using a very restricted set input data, rather than using real data (from user activity logs, from logical scenarios, from careful consideration of the concept domain).

Inefficient over the long term: Quality assurance involves a range of tasks. Effective quality assurance programs expand their base of documentation on the product and on the testing process over time, increasing the coverage and granularity of tests over time. Great testing requires good test setup and preparation, but success with the kind Test plan-less approach described in this essay may reinforce bad project and test methodologies. A continued pattern of quick-and-dirty testing like this is a sign that the product or application is unsustainable in the long run.

Areas of Testing:

Black Box Testing

Black box testing is also called as Functionality Testing. In this testing user will be asked to test the correctness of the functionality with the help of Inputs and Outputs. User doesn’t require the knowledge of software code.

Approach:

EQUIVALENCE CLASS:

- For each piece of the specification, generate one or more equivalence Class
- Label the classes as “Valid” or “Invalid”
- Generate one test case for each Invalid Equivalence class
- Generate a test case that Covers as many
- Valid Equivalence Classes as possible

Boundary Value Analysis

- Generate test cases for the boundary values.
- Minimum Value , Minimum Value + 1, Minimum Value -1
- Maximum Value, Maximum Value + 1, Maximum Value - 1

Error Guessing.

- Generating test cases against to the specification.

White Box Testing

White box testing is also called as Structural testing. User does require the knowledge of software code.
Structure = 1 Entry + 1 Exit with certain Constraints, Conditions and Loops.
Why do white box testing when white box testing is used to test conformance to requirements:
Logic Errors and incorrect assumptions most are likely to be made while coding for "special cases". Need to ensure these execution paths are tested.
May find assumptions about execution paths incorrect, and so make design errors.
Typographical errors are random. Just as likely to be on an obscure logical path as on a mainstream path.

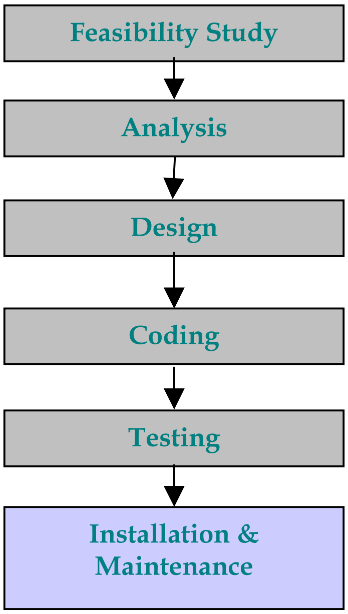
Approach
Basic Path Testing:
Cyclomatic Complexity and Mc Cabe Method
Structure Testing:
Condition Testing, Data Flow Testing and Loop Testing

Grey Box Testing

Grey box Testing is the new term, which evolved due to the different behaviors of the system. This is just a combination of both Black box & white box testing. Tester should have the knowledge of both the internals and externals of the function.
Even though you probably don't have full knowledge of the internals of the product you test, a test strategy based partly on internals is a powerful idea. We call this gray box testing. The concept is simple: If you know something about how the product works on the inside, you can test it better from the outside. This is not to be confused with white box testing, which attempts to cover the internals of the product in detail. In gray box mode, you are testing from the outside of the product, just as you do with black box, but your testing choices are informed by your knowledge of how the underlying components operate andinteract.

Gray box testing is especially important with Web and Internet applications, because the Internet is built around loosely integrated components that connect via relatively well-defined interfaces. Unless you understand the architecture of the Net, your testing will be skin deep. Hung Nguyen's Testing Applications on the Web (2000) is a good example of gray box test strategy applied to the Web**.**

1.6 Installation & Maintenance:



Installation:

- File conversion
- System testing
- System changeover
- New system becomes operational
- Staff training

Maintenance:

- corrective maintenance
- perfective maintenance
- adaptive maintenance

Table format of all the phases in SDLC:

PHASE	INPUT	OUTPUT
Analysis	BRS	SRS
Design	SRS	Design Doc
Coding	Design	.exe File/Application/ Website
Testing	All the above Doc's	Defect Report

2. Software Development Life Cycles

Life cycle: Entire duration of a project, from inception to termination

Different life cycle models:

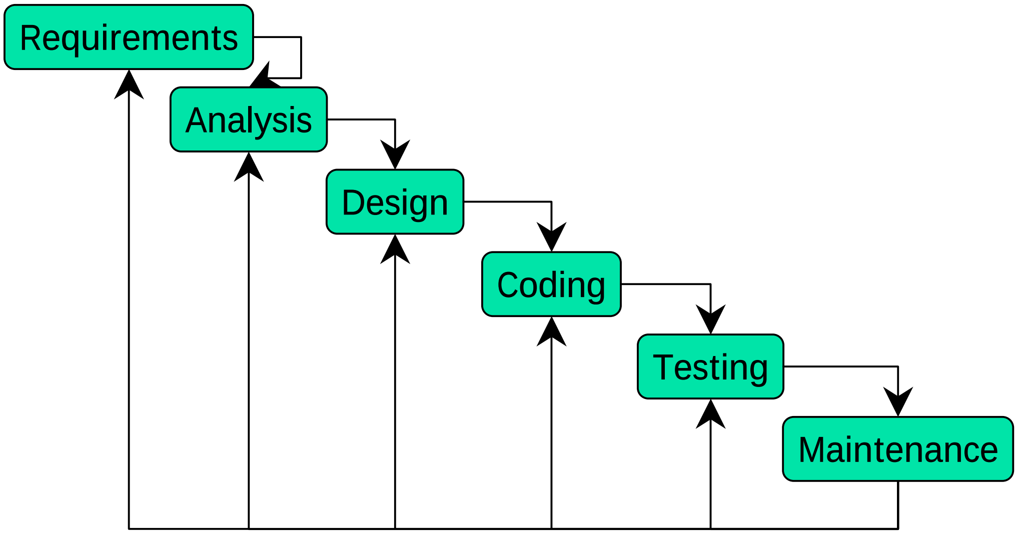
2.1. Code-and-fix model:

_ Earliest software development approach (1950s)
_ Iterative, programmers' approach
_ Two phases: 1. coding, 2. fixing the code
No provision for:
_ Project planning
_ Analysis
_ Design
_ testing
_ Maintenance
Problems with code-and-fix model:
1. After several iterations, code became very poorly structured; subsequent fixes became very expensive
2. Even well-designed software often very poorly matched users' requirements: were rejected or needed to be redeveloped (expensively!)
3. Changes to code were expensive, because of poor testing and maintenance practices
Solutions:
1. *Design* before coding
2. *Requirements analysis* before design
3. Separate *testing* and *maintenance* phases after coding

2.2. Waterfall model:

_ Also called the *classic life cycle*
_ Introduced in 1956 to overcome limitations of code-and-fix model
_ Very structured, organized approach, suitable for planning
Main phases:
1. Feasibility study
2. Analysis
3. Design (overall design & detailed design)
4. Coding
5. Testing (unit test, integration test, acceptance test)
6. Maintenance

Classic Waterfall



_ Waterfall model is a *linear* approach, quite inflexible
_ At each phase, feedback to previous phases is possible (but is discouraged in practice)
_ Still is the *most widespread model* today

Problems with Waterfall Model:

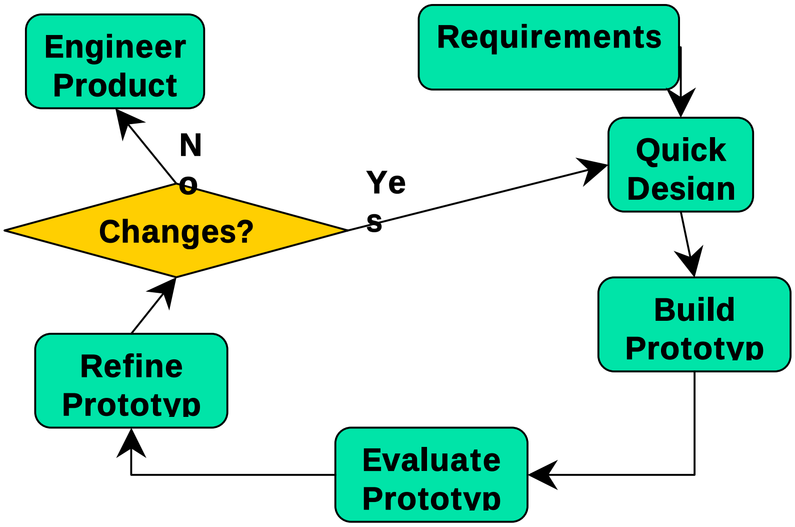
- It doesn't happen(Requirements are frizzed)
- Real projects tend not to follow a sequential flow
- Activities are done opportunistically during all "phases"
- Delivery only at the end (long wait)

2.3. Prototyping model:

_ Introduced to overcome shortcomings of waterfall model
_ Suitable to overcome problem of requirements definition
_ Prototyping builds an *operational model* of the planned system, which the customer can evaluate

Main phases:
1. Requirements gathering
2. Quick design
3. Build prototype
4. Customer evaluation of prototype
5. Refine prototype
Iterate steps 4. and 5. to "tune" the prototype
6. Engineer product

Prototyping



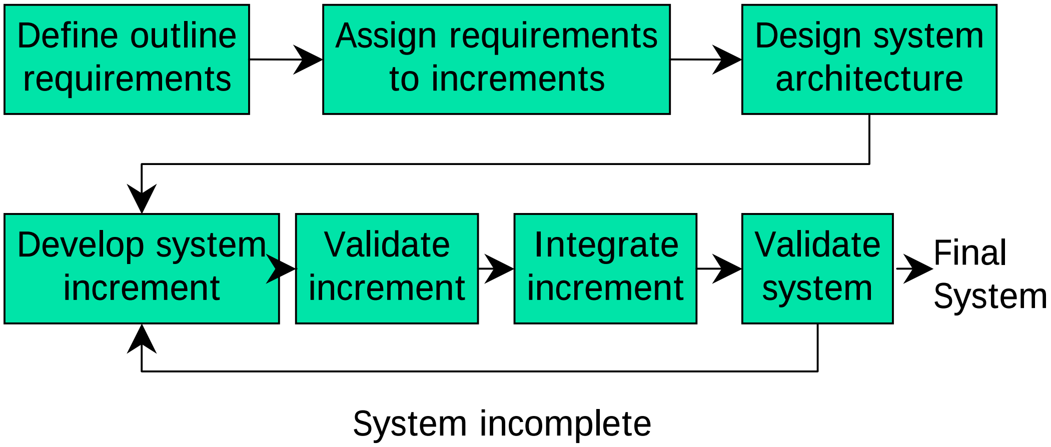
Mostly, the prototype is discarded after step 5. and the actual system is built from scratch in step 6. (*throw-away prototyping)*

Possible problems:
_ Customer may object to prototype being thrown away and may demand "a few changes" to make it working (results in poor software quality and maintainability)
_ Inferior, temporary design solutions may become permanent after a while, when the developer has forgotten that they were only intended to be temporary (results in poor software quality)

2.4 Incremental:

During the first one-month phase, the development team worked from static visual designs to code a prototype. In focus group meetings, the team discussed users' needs and the potential features of the product and then showed a demonstration of its prototype. The excellent feedback from these focus groups had a large impact on the quality of the product.
Main phases:
1. Define outline req uirements
2. Assign requirements to increments
3. Design system architecture
4. Develop
5. Integrate
6. Validate

Incremental

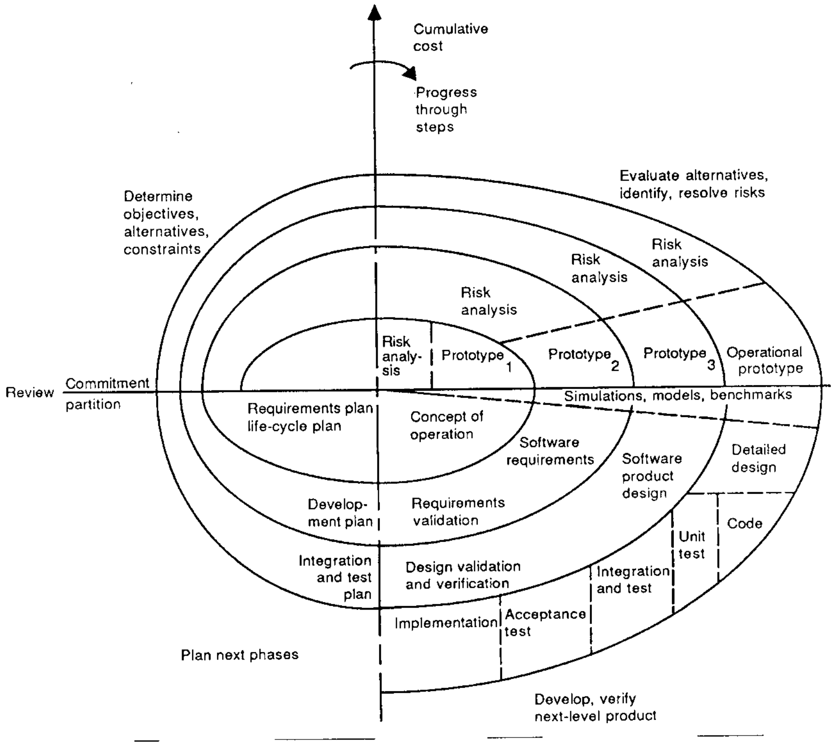


After the second group of focus groups, the feature set was frozen and the product definition complete. Implementation consisted of four-to-six-week cycles, with software delivered for beta use at the end of each cycle. The entire release took 10 months from definition to manufacturing release. Implementation lasted 4.5 months. The result was a world-class product that has won many awards and has been easy to support.

2.5 Spiral model:

_ Objective: overcome problems of other models, while combining their advantages
_ Key component: risk management (because traditional models often fail when risk is neglected)
_ Development is done *incrementally*, in several *cycles* _ Cycle as often as necessary to finish
Main phases:
1. Determine objectives, alternatives for development, and constraints for the portion of the whole system to be developed in the current cycle
2. Evaluate alternatives, considering objectives and constraints; identify and resolve risks
3. Develop the current cycle's part of the system, using evolutionary or conventional development methods (depending on remaining risks); perform validation at the end
4. Prepare plans for subsequent phases

Spiral Model



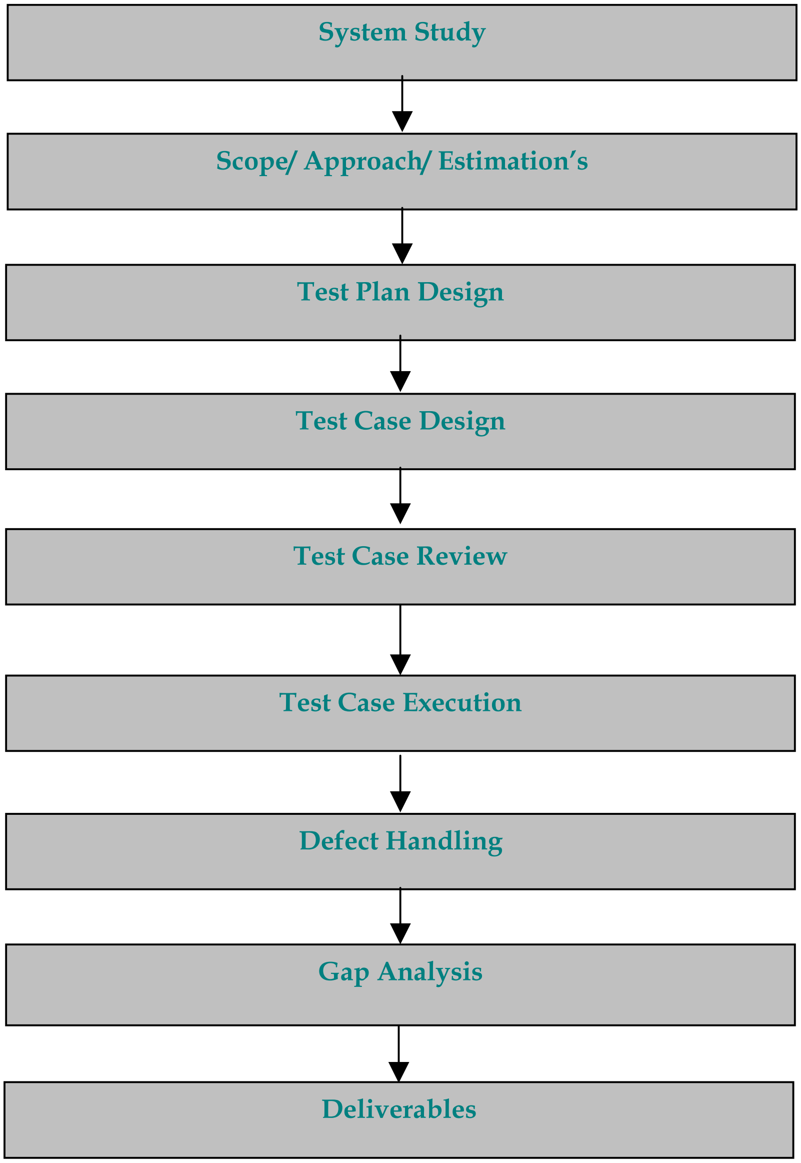
Advantages of spiral model:

_ Most realistic approach for *large* systems envelopment
_ Allows identification and resolution of risks *early* in the development

Problems with spiral model:

_ Difficult to convince customer that this approach is controllable
_ Requires significant risk assessment expertise to succeed
_ Not yet widely used: efficacy not yet proven

3. Testing Life Cycle



3.1 System Study:

Making documents of

1. **Domain Knowledge :-** Used to know about the client business
   Banking / Finance / Insurance / Real-estates / ERP / CRM / Others

2. **Software : -**

Front End	(GUI) CB / JAVA/ FORMS / Browser
Process	Language witch we want to write programmes
Back End	Database like Oracle, SQL Server etc.

3. **Hardware: -** Internet/ Intranet/ Servers which you want to install.
4. **Functional Points: -** Ten Lines Of Code (LOC) = 1 Functional Point.
5. **Number of Pages: -** The document which you want to prepare.
6. **Number of Resources : -**Like Programmers, Designers, and Managers.
7. **Number of Days: -** For actual completion of the Project.
8. **Numbers of Modules**
9. **Priority:-** High/ Medium/ Low importance for Modules

3.2 Scope/ Approach/ Estimation:

Scope:- What to be tested
What not to test

Approach:- Testing Life Cycle

Estimation:- (Formula = LOC/ FP/Resources)

- 1000 = 100 FP (10 LOC = 1 FP)
- 100 x 3 = 300 (FP x 3 Tech. = Test Cases) The 3 Tech are
    1. Equivalence Class:
    2. Boundary Value Analysis
    3. Error Guessing.
- 30 TC Par Day => 300/30 = 10 Days to Design Test Cases
- Test Case Review => ½ of Test Case Design (5 Days)
- Test Case Execution = 1 ½ of Test Case Design(15 Days)
- Defect Headlining = Test Case Design (5 Days)
- Test Plan = 5 days (1 week)
- Buffer Time = 25% of Estimation

3.3 Test Plan Design:

The Test Plan Design document helps in test execution it contain

1. About the client and company
2. Reference document (BRS, FRS and UI etc.)
3. Scope (What to be tested and what not to be)
4. Overview of Application
5. Testing approach (Testing strategy)
6. For each testing
    - Definition
    - Taconic
    - Start criteria
    - Stop criteria
7. Resources and there Roles and Responsibilities
8. Defect definition

9. Risk / Contingency / Mitigation Plan
10. Training Required
11. Schedules
12. Deliverables

3.4 Test Cases Design:

What is a test case?

Test case is a description of what to be tested, what data to be given and what actions to be done to check the actual result against the expected result.

WHAT ARE THE ITEMS OF TEST CASE?

Test case items are:
Test Case Number
Pre-Condition
Description
Expected Result
Actual Result
Status (Pass/Fail)
Remarks.

Can this test cases reusable?

Yes, Test cases can be reusable.

Test cases developed for functionality testing can be used for Integration/System/Regression testing and performance testing with few modifications.

What are the characteristics of good test case?

A good test case should have the following:
TC should start with “what you are testing”.
TC should be independent.
TC should not contain “If” statements.
TC should be uniform.
Eg. <Action Buttons> , “Links”…

Are there any issues to be considered?
Yes there are few Issues:

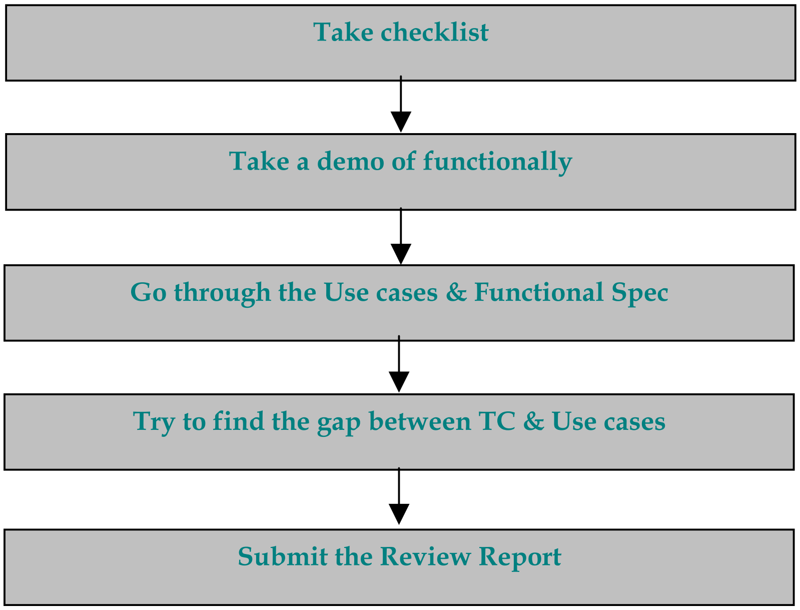
All the TC’s should be traceable.
There should not be too many duplicate test cases.
Out dated test cases should be cleared off.
All the test cases should be executable.

TC ID	Pre-Condition	Description	Expected Result	Actual Result	Status	Remarks
Unique TestCase number	Condition to satisfied	1. What to be tested 2. what data to provided 3. what action to be done	As pear FSR	System response	Pass or Fail	If any
Yahoo-001	Yahoo web page should displayed	1. Check inbox is displayed 2. User ID/PW 3. Click on Submit	System should mail box	System response		

3.5 Test Case Review:

Peer to peer Reviews
Team Lead Review
Team Manager Review

Review Process


Take checklist
Take a demo of functionally
Go through the Use cases & Functional Spec
Try to find the gap between TC & Use cases
Submit the Review Report

Review Format

Review-ID	Origin	Description	Status	Priority
Unique ID	Birth place From where it starts	Defect description	Open/ Close	Major Medium Minor

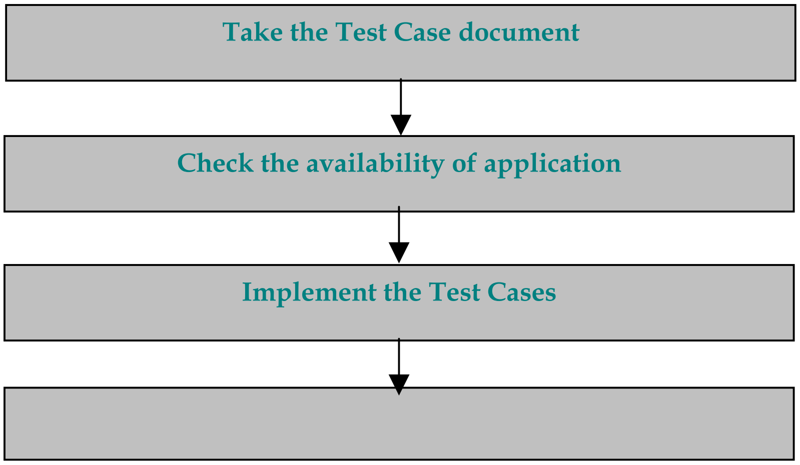
3.6 Test Case Execution:

Execution and execution results plays a vital role in the testing. Each and every activity should have proof.

The following activities should be taken care:

1. Number of test cases executed.
2. Number of defects found
3. Screen shots of successful and failure executions should be taken in word document.
4. Time taken to execute.
5. Time wasted due to the unavailability of the system.

Test Case Execution Process:


Take the Test Case document
Check the availability of application
Implement the Test Cases

Inputs
-Test Cases
-System Availability
-Data Availability
Process
-Test it.
Output
-Raise the Defects
-Take screen shot & save it

3.7 Defect Handling

What is Defect?

In computer technology, a Defect is a coding error in a computer program. It is defined by saying that “A software error is present when the program does not do what its end user reasonably expects it to do”.

Who can report a Defect

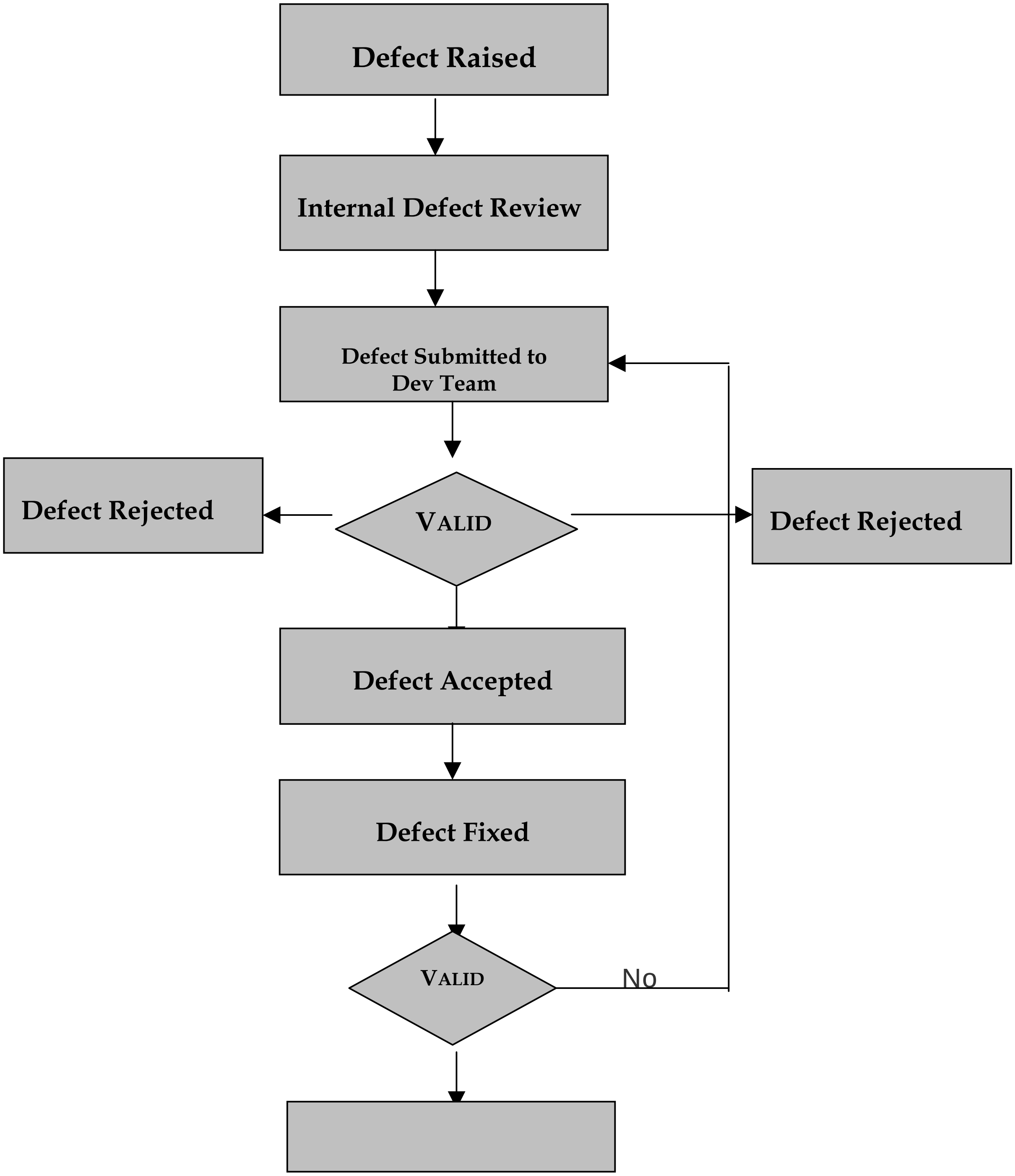
Anyone who has involved in software development life cycle and who is using the software can report a Defect. In most of the cases defects are reported by Testing Team.

A short list of people expected to report bugs:

Testers / QA Engineers
Developers
Technical Support
End Users
Sales and Marketing Engineers

Defect Life Cycle

Defect Life Cycle helps in handling defects efficiently. This DLC will help the users to know the status of the defect.



Types of Defects

Cosmetic flaw
Data corruption
Data loss
Documentation Issue
Incorrect Operation
Installation Problem
Missing Feature
Slow Performance
System Crash
Unexpected Behavior
Unfriendly behavior

How do u decide the Severity of the defect

Severity Level	Description	Response Time or Turn-around Time
High	A defect occurred due to the inability of a key function to perform. This problem causes the system hang it halts (crash), or the user is dropped out of the system. An immediate fix or work around is needed from development so that testing can continue.	Defect should be responded to within 24 hours and the situation should be resolved test exit
Medium	A defect occurred which severely restricts the system such as the inability to use a major function of the system. There is no acceptable work-around but the problem does not inhibit the testing of other functions	A response or action plan should be provided within 3 working days and the situation should be resolved before test exit.
Low	A defect is occurred which places minor restrict on a function that is not critical. There is an acceptable work-around for the defect.	A response or action plan should be provided within 5 working days and the situation should be resolved before test exit.
Others	An incident occurred which places no restrictions on any function of the system. No immediate impact to testing. A Design issue or Requirements not definitively detailed in project. The fix dates are subject to negotiation.	An action plan should be provided for next release or future enhancement

Defect Severity VS Defect Priority

The General rule for the fixing the defects will depend on the Severity. All the High Severity Defects should be fixed first.

This may not be the same in all cases some times even though severity of the bug is high it may not be take as the High priority.
At the same time the low severity bug may be considered as high priority.

Defect Tracking Sheet

Defect No	Description	Origin	Severity	Priority	Status
Unique No	Dec of Bug	Birth place of the Bug	Critical Major Medium Minor Cosmetic	High Medium Low	Submitted Accepted Fixed Rejected Postponed Closed

Defect Tracking Tools

Bug Tracker -- BSL Proprietary Tools

Rational Clear Quest
Test Director

3.8 Gap Analysis:

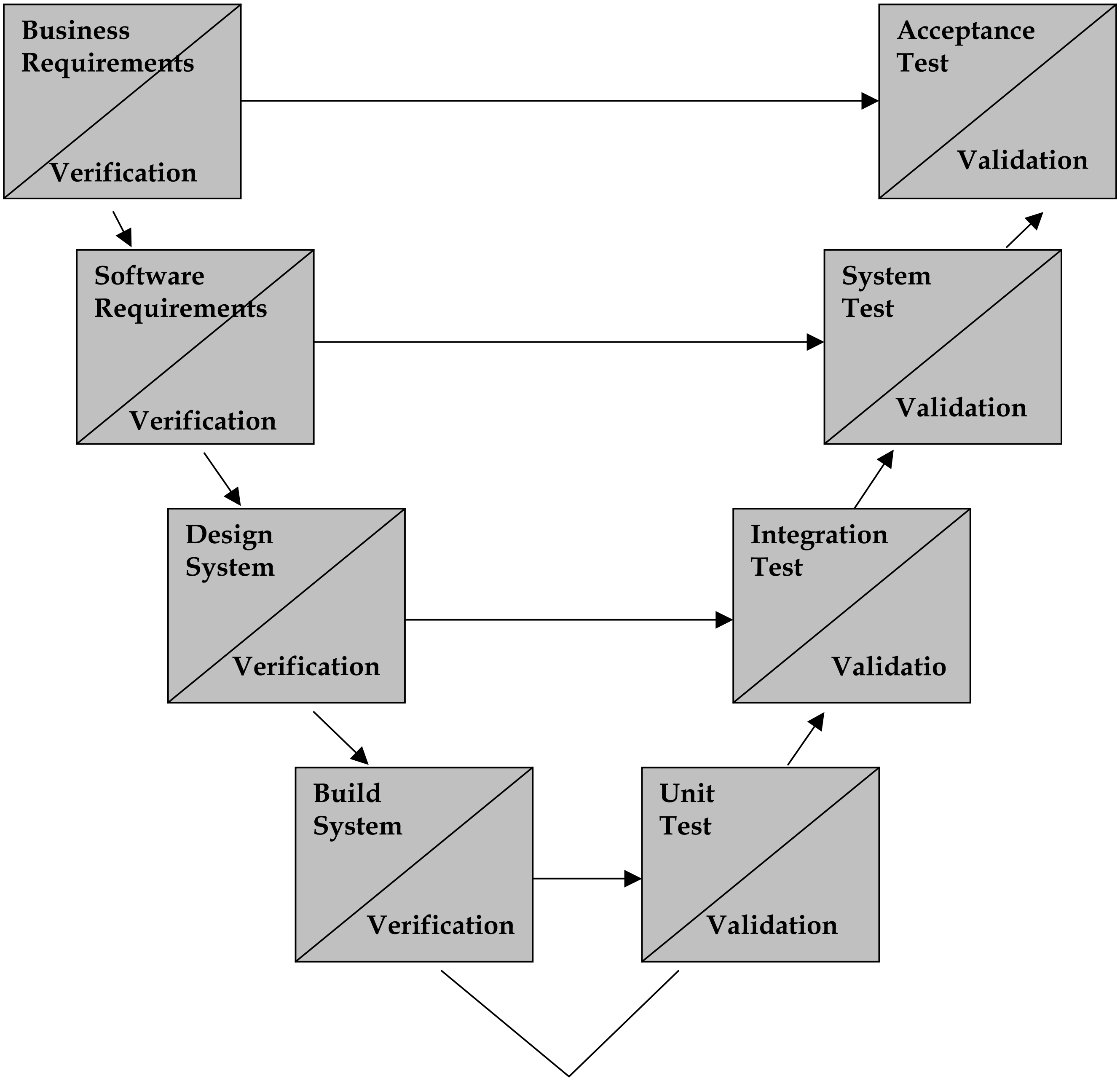
1. BRS Vs SRS
   BRS01 – SRS01
   -SRS02
   -SRS03
2. SRS Vs TC
   SRS01 – TC01
   - TC02
   - TC03
3. TC Vs Defects
   TC01 – Defects01
   – Defects02

3.9 Deliverables:

4. Testing Phases – The V Model

Verification → Static System – Doing Right Job

Validation → Dynamic System - Job Right



4.1 Unit Testing:

In Unit testing user is supposed to check each and every micro function. All field level validations are expected to test at the stage of testing.
In most of the cases Developer will do this.

Approach:

EQUIVALENCE CLASS:

- For each piece of the specification, generate one or more equivalence Class
- Label the classes as “Valid” or “Invalid”
- Generate one test case for each Invalid Equivalence class
- Generate a test case that Covers as many Valid Equivalence Classes as possible

Boundary Value Analysis

- Generate test cases for the boundary values.
- Minimum Value , Minimum Value + 1, Minimum Value -1
- Maximum Value, Maximum Value + 1, Maximum Value - 1

Error Guessing.

– Generating test cases against to specification

4.2 Integration Testing:

The primary objective of integration testing is to discover errors in the interfaces between Modules/Sub-Systems (Host & Client Interfaces).

Approach:

Top-Down Approach

The integration process is performed in a series of 5 steps

1. The main control module is used as a test driver, and stubs are substituted for all modules directly subordinate to the main control module.
2. Depending on the integration approach selected (depth or breadth-first) subordinate stubs are replaced at a time with actual modules.
3. Tests are conducted as each module is module is integrated.
4. One completion of each set of tests, another stub is replaced with the real-module.
5. Regression testing may be conducted to ensure that new errors have not been introduced.

Bottom-Up Approach.

A bottom-up integration strategy may be implemented with the following steps:

1. Low level modules are combined into clusters (Some times called builds) that perform a specific software sub function.
2. A driver (control program for testing) is written to coordinate test case input and output.
3. The cluster is tested.
4. Drivers are removed and clusters are combined upward in the program structure

An integration testing is conducted, the tester should identify critical modules. A critical module has one or more of the following characteristics:

1. Address several software requirements.

2. Has a high-level of control. (resides relatively high in the program structure)
3. Complex & Error-Phone.
4. Have definite performance requirements.

4.3 System Testing:

The primary objective of system testing is to discover errors when the system is tested as a hole. System testing is also called as End-End Testing. User is expected to test from Login-To-Logout by covering various business functionalities.

Approach: IDO Model

Identifying the End-End/Business Life Cycles.
Design the test and data.
Optimize the End-End/Business Life Cycles.

4.4 Acceptance Testing:

The primary objective of acceptance testing is to get the acceptance from the client. Client will be using the system against the business requirements.

Pre-user acceptance testing will be conducted to ascertain the stability and to check whether the complete functionality of the system is checked during system testing. After the first round of system testing, test engineers will go through the test cases (Test Scripts) sent by the users. They will ascertain whether a particular condition (functionality) is covered and the test case number will be entered against each condition. If a particular condition test case sent by the user is not covered because of the changes in the requirement, that particular test case will be documented (refer the tabular format) and the existing behavior of the system will be mentioned in the remarks column.
When a particular condition is not covered, a new test case is prepared along with the test data and it is executed to ensure the system is working accordingly. If there are any test cases which are not covered during system testing and when there is no supportive document for that particular test case it is named as an invalid test case. After the mapping the whole document will be sent back to the user.

Approach: BE

- Building a team with real-time user, functional users and developers.
- Execution of business Test Cases.

When Should we start writing Test Cases/ Testing
V Model is the most suitable way to start writing Test Cases and conduct Testing.

SDLC Phase	**Requirements Freeze**	**Requirements Build**
Business Requirements Docs	Acceptance Test Cases	Acceptance Testing
Software Requirements Docs	System Test Cases	System testing
Design Requirements Docs	Integration test Cases	Integration Testing
Code	Unit Test Cases	Unit Testing

5 Testing Methods – FURRPSC Model

5.1 Functionality Testing:

Objective:

1. Test against system requirements.
2. To confirm all the requirements are covered.

Approach:

EQUIVALENCE CLASS
Boundary Value Analysis
Error Guessing.

5.2 Usability Testing:

To test the Easiness and User-friendliness of the system.

Approach:

Qualitative & Quantitative
Qualitative Approach:

1. **E**ach and every function should available from all the pages of the site.
2. **U**ser should able to submit each and every request with in 4-5 actions.
3. **C**onfirmation message should be displayed for each and every submit.

Quantitative Approach:
Heuristic Checklist should be prepared with all the general test cases that fall under the classification of checking.
This generic test cases should be given to 10 different people and ask to execute the system to mark the pass/fail status.
The average of 10 different people should be considered as the final result.
Example: Some people may feel system is more users friendly, If the submit is button on the left side of the screen. At the same time some other may feel its better if the submit button is placed on the right side.

Classification of Checking:
Clarity of communication.
Accessibility
Consistency
Navigation
Design & Maintenance
Visual Representation.

5.3 Reliability Testing:

RT is property, which defines how well the software meets its requirements. Objective is to find Mean Time between failure/time available under specific load pattern. Mean time for recovery.

Approach: RRT(Rational Real Time) for continuous hours of operation.
More then 85% of the stability is must.
Reliability Testing helps you to confirm:
Business logic performs as expected
Active buttons are really active
Correct menu options are available
Reliable hyper links
(Why is load runner used for reliability testing -reason)
Virtual Users can be created using Load Runner. Load Scenerios, which are a mix of business, processes and the number of virtual users, will run on each load server. User can quickly compose multi-user test scenarios using Load Runner's Controller. The Controller's interactive capability provides an interactive environment in which user can manage and drive the load test scenario, as well as create repeatable and consistent load. Load Runner's graphical interface helps to organize and control scenarios during load test setup and execution.(Write some sentences that can be correlated with load runner. Need to put this sentence in an appropriately)

5.4 Regression Testing:

Objective is to check the new functionalities has incorporated correctly with out failing the existing functionalities.
RAD – In case of Rapid Application development Regression Test plays a vital role as the total development happens in bits and pieces.
The term "regression testing" can be applied two ways. First, when a code problem has been fixed, a regression test runs tests to verify that the defect is in fact fixed.; "Imagine finding an error, fixing it, and repeating the test that exposed the problem in the first place. This is a regression test". Second, regression testing is the counterpart of integration testing: when new code is added to existing code, regression testing verifies that the existing code continues to work correctly, whereas integration testing verifies that the *new* code works as expected. Regression testing can describes the process of testing new code to verify that this new code hasn't broken any old code

Approach: Automation tools

5.5 Performance Testing:

Primary objective of the performance testing is "to demonstrate the system works functions as per specifications with in given response time on a production sized database.
Objectives:

Assessing the system capacity for growth.
Identifying weak points in the architecture
Detect obscure bugs in software
Tuning the system
Verify resilience & reliability

Performance Parameters:
Request-Response Time
Transactions per Second
Turn Around time
Page down load time
Through Put

APPROACH: USAGE OF AUTOMATION TOOLS

Classification of Performance Testing:
Load Test
Volume Test
Stress Test

Load Testing
Approach: Load Profile

Volume Testing
Approach: Data Profile

Stress Testing
Approach: RCQE Approach

Repeatedly working on the same functionality
Critical **Q**uery Execution (Join Queries)
To **E**mulate peak load.

Load Vs Stress:
With the Simple Scenario (Functional Query), N number of people working on it will not enforce stress on the server.
A complex scenario with even one less number of users will stress the server.

5.6 Scalability Testing:

Objective is to find the maximum number of user system can handle.

Classification:
Network Scalability
Server Scalability
Application Scalability

Approach: Performance Tools

5.7 Compatibility Testing:

Compatibility testing provides a basic understanding of how a product will perform over a wide range of hardware, software & network configuration and to isolate the specific problems.
Approach: ET Approach

Environment Selection.
TEST BED CREATION

I. Selection of environment

There are many types of Operating systems, Browsers, JVMs used by wide range of audience around the world. Compatibility testing for all these possible combinations is exhaustive and hence, optimizing the combination of environment is very critical.

Many times the customer may give the environment details for the compatibility testing. Incase if it is not given, the following strategy may be adopted for selecting the environment.

- **By understanding the end users.**

  List the possible end users for the proposed software application to be tested. Analyze their requirement of environment on the basis of the previous experience (Region wise or type of the application). Select the possible combination of operating system & browser from this input.

- **Importance of selecting both old browser and new browsers**

  Many end users use the default browser, which are part of the operating system & may not upgrade for new versions. Where as some end-users may tend to go for the latest versions of the browsers. Hence importance should be given for both old & new versions of the browsers for compatibility testing.

- **Selection of the Operating System**

  The operating system of Microsoft has wide range of user compared to other operating system. However many also use Macintosh and Unix operating system. The compatibility of the application with different operating system is very important. The details of operating system versus browsers supported are given vide Table-3 of section 2.0.

II. Test Bed Creation

Following steps are adopted for creation of test bed for different version of browsers and Microsoft operating system. This procedure is not applicable for Macintosh and Unix operating systems.

When the user wants to look for compatibility on different Microsoft operating systems and different version of browsers, following steps helps to reduce time and cost.

1) Partition of the hard disk.
2) Creation of Base Image

1) Partition of the hard disk

Partition helps in installing more than one operating system on a single hard disk. Hard disk is available with two partition namely primary partition and extended partition. The first sector of hard disk contains a partition table. This partition table has room to describe four partitions these are called primary partitions. One of these primary partitions can point to a chain of additional partitions. Each partition in this chain is called a logical partition & one partition is visible at a time.

Using partition magic software the primary partition of the hard disk can be configured into maximum of four parts.

Following are the steps involved while partitioning:

a) Create one primary partition of required size.
b) Make it active.
c) Load the particular operating system.
d) Using partition magic hide that partition.
e) After installing each operating system steps a, b, c, d are repeated.

With this the primary partition of the hard disk can be configured with WinNT, Win95,
Win98 & Win2k respectively.

2) Creation of Base Image

Base image is a clone of hard disk. It is possible to create the base image of all the four operating system of Microsoft along with IE lower version and office 97.

Incase of Internet Explorer, it is not possible to change from higher version to lower version. With the help of base image it is possible to rewrite the hard disk with the required Operating system, which contains lower version of IE. Norton ghost software helps to take the base image of the partitioned hard disk along with the required operating system. Incase of Netscape Navigator there is no problem of changing from higher version to lower version and vice versa.
Following is the comparison of the time required for installing operating systems with & without Norton ghost.

Without using Norton ghost	With Norton ghost
1) Win95, IE & office97 is 60 minutes. 2) Win98, IE & office97 is 70 minutes. 3) Win2K, IE & office97 is 70 minutes. 4) WinNT,IE & office97 is 50 minutes.	1) It takes 7 minutes to write one operating system with IE & office97. 2) It takes 18 minutes to write base image of all the four operating systems with lower. versions of Internet Explorer.

6. Performance Life Cycle

6.1 What is Performance Testing:

Primary objective of the performance testing is “to demonstrate the system works functions as per specifications with in given response time on a production sized database

6.2 Why Performance Testing:

-To assess the system capacity for growth

The load and response data gained from the tests can be used to validate the capacity planning model and assist decision making.

-To identify weak points in the architecture

The controlled load can be increased to extreme levels to stress the architecture and break it bottlenecks and weak components can be fixed or replaced

-To detect obscure bugs in software

Tests executed for extended periods can cause failures caused by memory leaks and reveal obscure contention problems or conflicts

-To tune the system

Repeat runs of tests can be performed to verify that tuning activities are having the desired effect – improving performance.

-To verify resilience & reliability

Executing tests at production loads for extended periods is the only way to access the systems resilience and reliability to ensure required service levels are likely to be met.

6.3 Performance-Tests:

Used to test each part of the web application to find out what parts of the website are slow and how we can make them faster.

6.4 Load-Tests:

This type of test is done to test the website using the load that the customer expects to have on his site. This is something like a “real world test” of the website.

First we have to define the maximum request times we want the customers to experience, this is done from the business and usability point of view, not from a technical point of view. At this point we need to calculate the impact of a slow website on the company sales and support costs.

Then we have to calculate the anticipated load and load pattern for the website (Refer Annexure I for details on load calculation) which we then simulate using the Tool.

At the end we compare the test results with the requests times we wanted to achieve.

6.5 Stress-Tests:

They simulate brute force attacks with excessive load on the web server. In the real world situations like this can be created by a massive spike of users – far above the normal usage – e.g. caused by a large referrer (imagine the website being mentioned on national TV…).
The goals of stress tests are to learn under what load the server generates errors, whether it will come back online after such a massive spike at all or crash and when it will come back online.

6.6 When should we start Performance Testing:

It is even a good idea to start performance testing before a line of code is written at all! Early testing the base technology (network, load balancer, application-, database- and web-servers) for the load levels can save a lot of money when you can already discover at this moment that your hardware is to slow. Also the first stress tests can be a good idea at this point.
The costs for correcting a performance problem rise steeply from the start of development until the website goes productive and can be unbelievable high for a website already online.
As soon as several web pages are working the first load tests should be conducted and from there on should be part of the regular testing routine each day or week or for each build of the software.

6.7 Popular tools used to conduct Performance Testing:

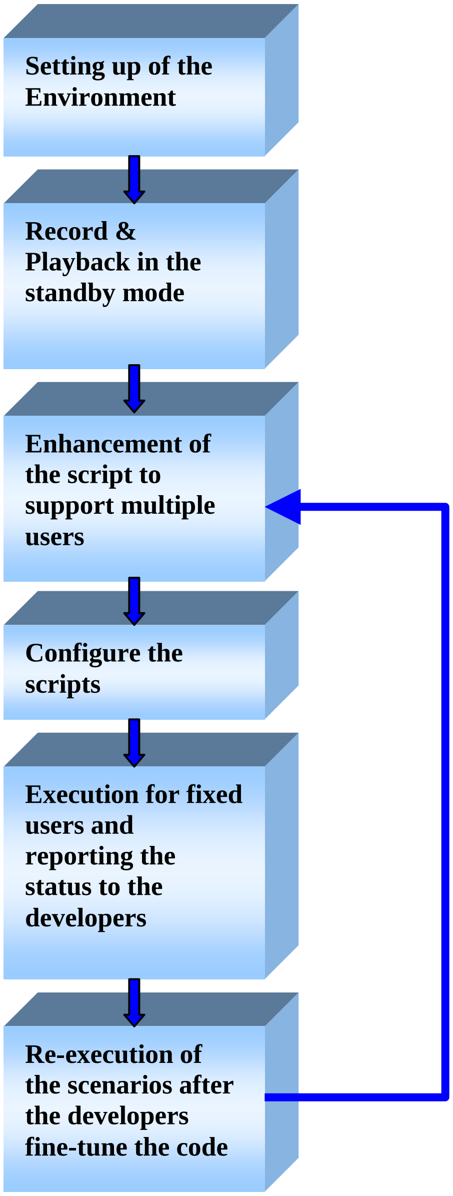
Some of the popular industry standard tools used to conduct performance test are

- LoadRunner from Mercury Interactive
- AstraLoad from Mercury Interactive
- Silk Performer from Segue
- Rational Suite Test Studio from Rational
- Rational Site Load from Rational
- OpenSTA from Cyrano
- Webload from Radview
- RSW eSuite from Empirix
- MS Stress tool from Microsoft

6.8 Performance Test Process:

This is a general process for performance Testing. This process can be customized according to the project needs. Few more process steps can be added to the existing process, deleting any of the steps from the existing process may result in Incomplete process. If Client is using any of the tools, In this case one can blindly follow the respective process demonstrated by the tool.

General Process Steps:


Setting up of the Environment
Record & Playback in the standby mode
Enhancement of the script to support multiple users
Configure the scripts
Execution for fixed users and reporting the status to the developers
Re-execution of the scenarios after the developers fine-tune the code

Setting up of the test environment

The installation of the tool, agents, directory structure creation for the storage of the scripts and results and installation additional software if essential to collect the server statistics (like SNMP agent). It is also essential to ensure the correctness of the environment by implementing the dry run.

Record & playback in the stand by mode

The scripts are generated using the script generator and played back to ensure that there are no errors in the script.

Enhancement of the script to support multiple users
The variables like logins, user inputs etc. should be parameterised to simulate the live environment. It is also essential since in some of the applications no two users can login with the same id.

Configuration of the scenarios
Scenarios should be configured to run the scripts on different agents, schedule the scenarios, distribute the users onto different scripts, collect the data related to database etc.

- Hosts

The next important step in the testing approach is to run the virtual users on different host machines to reduce the load on the client machine by sharing the resources of the other machines.

- Users

The number of users who need to be activated during the execution of the scenario.

- Scenarios

A scenario might either comprise of a single script or multiple scripts. The main intention of creating a scenario to simulate load on the server similar to the live/production environment.

- Ramping

In the live environment not all the users login to the application simultaneously. At this stage we can simulate the virtual users similar to the live environment by deciding -

1. How many users should be activated at a particular point of time as a batch?
2. What should be the time interval between every batch of users?

Execution for fixed users and reporting the status to the developers
The script should be initially executed for one user and the results/inputs should be verified to check it out whether the server response time for a transaction is less than or equal to the acceptable limit (bench mark).
If the results are found adequate the execution should be continued for different set of users. At the end of every execution the results should be analysed.

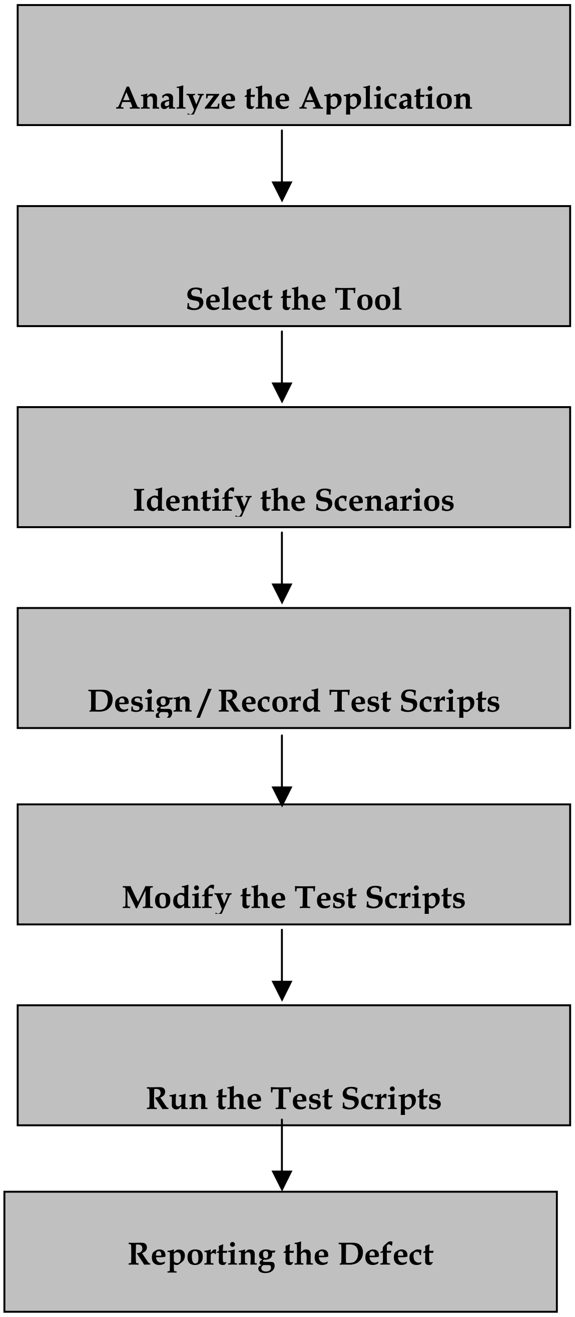
If a stage reaches when the time taken for the server to respond to a transaction is above the acceptable limit, then the inputs should be given to the developers.

Re-execution of the scenarios after the developers fine tune the code
After the fine-tuning, the scenarios should be re-executed for the specific set of users for which the response was inadequate. If found satisfactory, then the execution should be continued until the decided load.

Final report
At the end of the performance testing, final report should be generated which should comprise of the following –

- Introduction – about the application.
- Objectives – set / specified in the test plan.
- Approach – summary of the steps followed in conducting the test
- Analysis & Results – is a brief explanation about the results and the analysis of the report.
- Conclusion – the report should be concluded by telling whether the objectives set before the test is met or not.
- Annexure – can consist of graphical representation of the data with a brief description, comparison statistics if any etc.

7. Life Cycle of Automation



7.1 What is Automation?

A software program that is used to test another software program, This is referred to as “automated software testing”.

Why Automation

Avoid the errors that humans make when they get tired after multiple repetitions. The test program won’t skip any test by mistake.

Each future test cycle will take less time and require less human intervention.

Required for regression testing.

7.2 Benefits of Test Automation:

Allows more testing to happen
Tightens / Strengthen Test Cycle
Testing is consistent, repeatable
Useful when new patches released
Makes configuration testing easier
Test battery can be continuously improved.

7.3 False Benefits:

Fewer tests will be needed
It will be easier if it is automate
Compensate for poor design
No more manual testing.

7.4 What are the different tools available in the market:

Rational Robot
WinRunner
QTP
SilkTest
QA Run
WebFT

8. Testing Limitations

- We can only test against system requirements
- May not detect errors in the requirements.
- Incomplete or ambiguous requirements may lead to inadequate or incorrect testing.
- Exhaustive (total) testing is impossible in present scenario.
- Time and budget constraints normally require very careful planning of the testing effort.
- Compromise between thoroughness and budget.
- Test results are used to make business decisions for release dates.

8.1 Test Stop Criteria:

Minimum number of test cases successfully executed.
Uncover minimum number of defects (16/1000 stm)
Statement coverage
Testing uneconomical
Reliability model

9. Tester Responsibilities

Follow the test plans, scripts etc. as documented.
Report faults objectively and factually
Check tests are correct before reporting s/w faults
Assess risk objectively
Prioritize what you report
Communicate the truth.

9.1 How to Prioritize Tests:

We can't test every thing. There is never enough time to do all testing you would like, so what testing should you do?

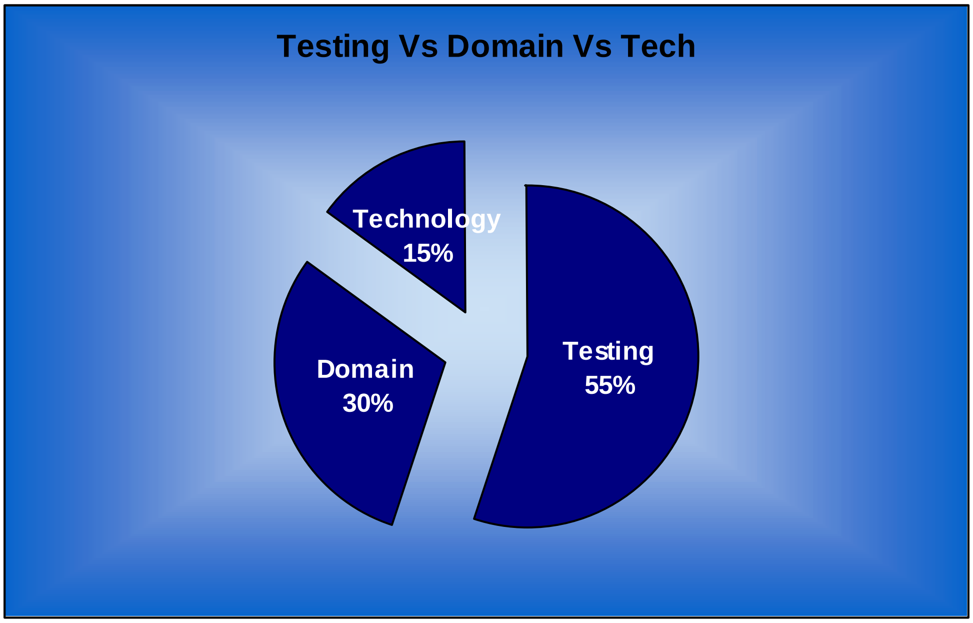
Prioritize Tests. So that, whenever you stop testing, you have done the best testing in the time available.

Tips
Possible ranking criteria (all risk based)
Test where a failure would be most serve.
Test where failures would be most visible.
Take the help of customer in understanding what is most important to him.
What is most critical to the customers business.
Areas changed most often.
Areas with most problems in the past.
Most complex areas, or technically critical.

10. How can we improve the efficiency in testing?

In the recent year it has show lot of outsourcing in testing area. Its right time to think and create process to improve the efficiency of testing projects. The best team will result in the efficient deliverables. The team should contain 55% hard core test engineers, 30 domain knowledge engineers and 15% technology engineers.

How did we arrive to this figures? The past projects has shown 50-60 percent of the test cases are written on the basis of testing techniques, 28-33% of test cases are resulted to cover the domain oriented business rules and 15-18% technology oriented test cases.


Testing Vs Domain Vs Tech
Technology
Domain
Testing