

Training Manual

Centre for software Testing &
Training

SoftTest*Labs*

Your Career Architects in Software Testing

Training Manual

First Edition

© 2008 SoftTest Labs

ALL RIGHTS RESERVED. No part of this work covered by the copyright herein may be reproduced, transmitted, stored, or used in any form or by any means graphic, electronic, or mechanical, including but not limited to photocopying, recording, scanning, digitizing, taping, Web distribution, information networks, or information storage and retrieval systems.

For more information, contact us at

SoftTest Labs

A-19, Sector-59, Noida,

Uttar Pradesh-201301

Tel: +91-120-4294333, 4292222

<http://www.softtestlabs.net>

Email us at: training@softtestlabs.net

QA InfoTech India Pvt. Ltd

B-8, Sector 59,

Noida- 201301

Tell +91-120-4290222

Fax +91-120-2581692

<http://www.gainfotech.com>

Email: info@gainfotech.com



Contents

1 . SOFTWARE TESTING	7
Introduction to software testing	
Definitions	
Software Testing Myths	
Impossibility of complete testing	
Purpose of testing	
Criteria to stop testing	
Verification and Validation	
2 . SOFTWARE DEVELOPMENT LIFE CYCLE	17
What is SDLC?	
Understanding SDLC	
Phases in SDLC	
Traditional waterfall model	
Iterative Approaches	
Prototyping	
Rapid application development	
Evolutionary development	
Incremental model	
Spiral model	
Rational Unified Process	
3 . LEVELS OF TESTING	43
Types of testing	
Unit Testing	
Integration Testing	
System Testing	
Function Testing	
Performance Testing	
Regression Testing	
Beta testing	
Integrity and Release Testing	
Final Acceptance and Certification	
White Box Testing	
Code Coverage Analysis	
White Box Testing Strategies	
Black Box Testing	
Sequence of Events Involved In Black Box Testing	

4 . TESTING CYCLE 58

- Testing the requirements
 - Use Case
- Test Planning and Documentation
 - Types of Test Documents
- Test Case Development
 - Techniques to generate Test Cases
- Test Execution
- Software Error/Bug
 - Categories of Error/Bug
 - Bug Lifecycle
 - Bug Report

5 . INTERNATIONALIZATION 94

- Internationalization
 - Internationalization Testing
- Localization
 - Localization testing

6 . AUTOMATION 102

- Basics of Automation
- Test Tool requirement
- Advantages and disadvantages
- Automation Test Strategy
 - Decision to Automate
 - Test Tool acquisition
 - Setting the Entrance and Exit Criteria
 - Test approach
 - Test Execution
 - Test Program and Review Assessment

CHAPTER 1

Software Testing

“Software testing is a process used to identify the correctness, completeness and quality of developed computer software. Actually, testing can never establish the correctness of computer software, as this can only be done by formal verification. It can only find defects, not prove that there are none”

Courtesy: www.wikipedia.org

OVERVIEW OF THIS CHAPTER

This chapter deals with the basic understanding of software testing. It explains the common myths and misunderstood facts about software testing and also provides a general insight of the feasibility of complete testing.

- ✓ Traditional Definitions
- ✓ Software Testing Myths
- ✓ Objectives Of Testing
- ✓ Impossibility Of Complete Testing
- ✓ When to Stop Testing.

SOFTWARE TESTING

Software testing is a vital part of the software lifecycle.

It is the process of exercising or evaluating a system or system component by manual or automated means to verify that it satisfies specified requirements or to identify differences between expected and actual results.

Testing can be defined as the process of executing a program/system with the intent of finding errors.

GLENFORD MYERS

KEY CONCEPTS

- Testing cannot show the absence of defects; it can only show that software defects are present.
- Software testing is more than just error detection. Testing software is executing the software to
 1. Verify that it performs “as specified”
 2. To detect errors, and
 3. To validate that the specifications are in accordance with the user’s requirement.

Software possesses four basic capabilities which are combined to perform complicated tasks.

- Capability to accept input
- Capability to produce output
- Capability to store data
- Capability to perform computation



Software testing is basically driving the system to exercise its capabilities, then attacking each capability by forcing situations that are the cause of failure.

- Attacking inputs
- Attacking outputs
- Attacking data
- Attacking computation

Strategy to attack

- Decide the time to attack.
- Determine what makes the attack a success.
- Establish steps to determine if the attack exposes failures.
- Establish steps to conduct the attack.

Some Software Testing Myths

- It is possible to test a program completely.
- Testing demonstrates that defects are not present in the application.
- Testing is best done once the system is in the field and being used.
- Programmers can do all the testing, as they are the ones who wrote the code.

What do you mean by complete testing?

Ideally complete testing must mean that by the time testing is done; the tested system is devoid of all unknown bugs. If you test completely, then at the end of testing, there cannot be any undiscovered errors.

This is virtually impossible because

- The domain of possible inputs is too large.
- There are too many combinations of data to test.
- There are too many possible paths through the program to test.

Purpose of Testing

- The fundamental purpose of software testing is to find errors in the software.
- The intent of finding errors is to get them fixed.
- Testing can be used to show the presence of errors, but never their absence.
- Testing ensures that the software performs as required, and to enhance software quality, reliability and supportability.
- The mission of the test team is not just to perform testing, but to help minimize the risk of product failure.
- The role of the tester is to protect their organization from releasing software that is defective. Releasing the defected product not only mars the reputation of the company, it also comes with the dangers of legal actions taken by the customers.

Test Completion Criteria

When to stop testing?

Software Testing is considered finished when

- ➡ Certain percentage of the test case is passed.
- ➡ When the matrices involved (those are bug detection and bug fixing) reach a certain level i.e. bug detection decreases and bug fixing increases.
- ➡ Code review and the system functionality meet the required specification.
- ➡ Test phase time and resources are exhausted.



Verification and Validation

So far we have learned that Software testing is definitely something more than mere error detection.

It is executing the software for

- ➡ Verification
- ➡ Error detection
- ➡ Validation

Verification (Are We Building the Product Right?)

- ➡ Verification is defined in FDA's Quality System Regulation (21 CFR 820.3) as "confirmation by examination and provision of objective evidence that specified requirements have been fulfilled."
- ➡ It ensures that the software product is developed in the right manner.

Various phases involved in Verification

During the Verification, various documentations and the ready part of the software is reviewed in order to detect errors in it.

This process helps in prevention of potential bugs, which may cause in failure of the project.

It is achieved by conducting

- ➡ Inspection
- ➡ Walkthroughs
- ➡ Reviews



Validation (Are we building the right product?)

- Software validation is "establishing by objective evidence that all software requirements have been implemented correctly and completely and are traceable to system requirements" [Ref: NIST 500-234].
- Validation is the process of checking that what is specified is exactly what the user wanted.
- During validation process
- Test plan, test suits and test cases are developed.





CHAPTER 2

Software Development Life Cycle

“The sequence of phases through which the project will evolve. It is absolutely fundamental to the management of projects. It will significantly affect how the project is structured. The basic life cycle follows a common generic sequence: Opportunity, Design & Development, Production, Hand-over, and Post- Project Evaluation. The exact wording varies between industries and organizations”

Patel and Morris

OVERVIEW OF THIS CHAPTER

This chapter is concerned with various phases involved in the software production. Software development lifecycle provides a guideline for the successful software production by following various models or approaches. This chapter deals with such models.

- ✓ Traditional SDLC model
- ✓ Iterative approaches
 - ✎ Prototyping
 - ✎ Rapid application development
 - ✎ Evolutionary development
 - Incremental model
 - Spiral model

SOFTWARE DEVELOPMENT LIFE CYCLE

During its production the software goes through a series of development stages including initial visualization to assessing to introduction in the market. This whole sequence which a product undergoes is marked by a product's lifecycle.

- The Systems Development Life Cycle is a conceptual model that describes various phases the stages through which a software product passes from initial conceptualization through to the time it is eventually replaced.
- Every software development project is faced with two fundamental questions
 - What must the system do?
 - How should it be done?
- The answers to these questions are formulated during different phases of the systems development lifecycle (SDLC)–

- “what” during the analysis phase
- and, “how” during the design and implementation phases



VARIOUS PHASES IN SDLC ARE AS FOLLOWS

PLANNING

- Describes the vision of the project.
- Set the objective statement to provide a purpose to the development team.
- Develop the project plan including tasks, resources, and timeframes.

Project Plan - It is a guideline for the project that describes various activities of the product development including all the activities performed, the resources that will perform the activities and the timeline to wrap up the activities.



REQUIREMENT ANALYSIS

- Group the business requirements for the system.
- Understand and archive the business requirement for the proposed system.
- Determine customer's needs and product constraints.

Testing during the planning and requirement analysis stages

EVALUATE THE REQUIREMENT DOCUMENTS

Requirement analysis is done to check the following aspects

- ✓ Accuracy of the requirement
- ✓ Completeness and compatibility
- ✓ Testability

COMPARATIVE PRODUCT EVALUATION

It involves the analysis of the products that are already there in the market to get a hold of

- ✓ Product capabilities
- ✓ Missing features
- ✓ Drawbacks and benefits of the product
- ✓ Feedback From The Focus Group

A focus group is a small group selected by the reviewer and it is a part of the market sector which the product is targeted for.

PRODUCT TASK ANALYSIS

An analyst figure out the tasks a product will perform.

It helps to simplify, combine and eliminate the features if required.

DESIGN

- Design the technical architecture required to support the system.
- Design phase transforms the requirement specification to some form suitable for implementation.
- There are two types of designs

EXTERNAL DESIGN

- Product is described from the user's point.
- User interface is described in details.

INTERNAL DESIGN

- System's internal working is explained.
- Internal design is further divided in to the following

Structural Design

It explains how a code is sub divided in to various pieces.

Data Design

It gives an insight to the type of data the code will work with.

Logic Design

It describes the logical steps involved in performing the task.

CODING

- Requirements and system specifications defined in the analysis and design phases are used to create an actual system.

TESTING

- Testing can be defined as the process of executing a program/system with the intent of finding errors.
by Glenford Myers
- It is used to verify that the software is working in accordance with the system design.

IMPLEMENTATION

- This process involves working with the customer to move the system into production.
- This includes on-site technical and training support during the roll-out period.

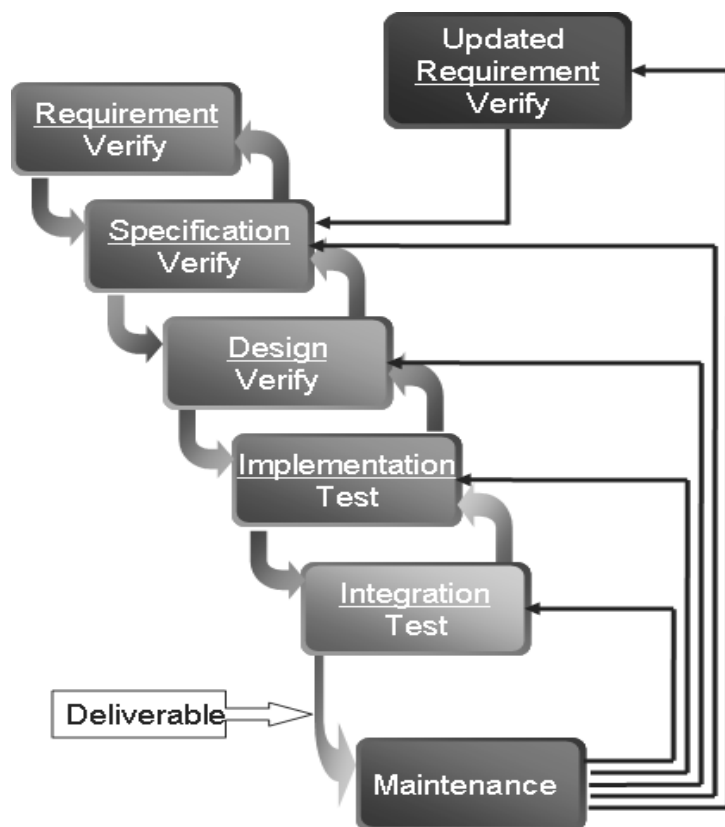
SUPPORT AND MAINTENANCE



- Testing during the maintenance will determine whether the system meets its specified objectives for maintainability.
- The system is monitored for continued performance in accordance with user requirements, and needed system modifications are incorporated.

DIFFERENT MODELS FOLLOWED IN SOFTWARE DEVELOPMENT LIFE CYCLE.

TRADITIONAL WATERFALL MODEL



CHARACTERISTICS

- Traditional waterfall model or linear sequential mode is characterized by its distinct phases.
- In a traditional model, each phase must be completed in its entirety before the next phase can begin.
- At the end of each phase, a review takes place to determine if the project is heading in the right direction.
- This systematic analysis helps to determine whether to retain or discard the project.

ADVANTAGES

- Clear development stages helps to track the progress in a systematic and controlled manner.
- Testing is an intrinsic part of every stage of this model.
- Milestones and deliverables can be clearly identified.
- The need to complete each stage before moving to the next smoothes the way for project management and control.

DISADVANTAGES

- The interval between requirements determination and delivery of complete system is too long to address evolving needs effectively.
- Inflexible partitioning of the project into separate phases makes it hard to deal with changing customer requirements.
- Formal planning and risk analysis is not taken in to consideration in the waterfall model.
- Estimating time and costs is difficult for each stage.

ITERATIVE DEVELOPMENT

The shortcomings of the traditional waterfall model created a demand for new methods of developing systems which ensures faster results entails less up-front information, and offer greater flexibility. In response, various new Iterative Development approaches came to fore where projects can be subdivided into small parts.

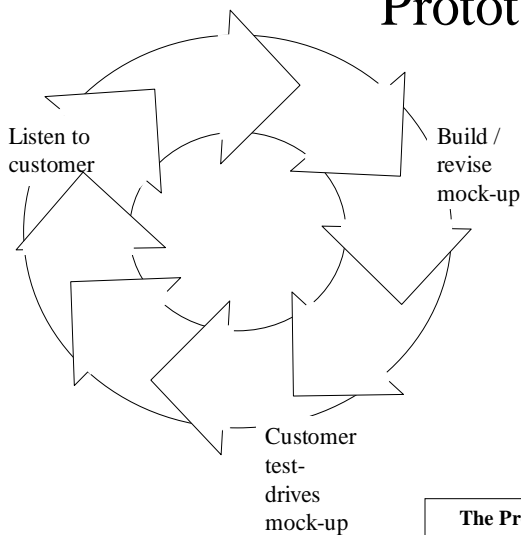
ITERATIVE APPROACHES TO TRADITIONAL WATERFALL

- Prototyping
- Rapid application development
- Evolutionary development
 - Incremental model
 - Spiral model



PROTOTYPING

Prototyping



**The Prototyping Paradigm
(Pressman 1997)**

CHARACTERISTICS

- This is a cyclic version of the linear model.
- In this model, once the requirement analysis is done and the design for a prototype is made, the development process gets started.
- Once the prototype is created, it is given to the customer for evaluation. The customer tests the package and gives his feed back to the developer who refines the product according to the customer's exact expectation. After a finite number of iterations, the final software package is given to the customer.

OBJECTIVES OF PROTOTYPING

- To explore requirements without having to build the entire product.
- To build systems that complies with the actual requirements.
- It shortens the time between requirement determination and delivery of a workable system.

PROTOTYPING APPROACHES IN SOFTWARE PROCESS

EVOLUTIONARY PROTOTYPING

- In this type of prototyping approach an initial prototype is created and refined through a number of stages to the final system.
- The main purpose of evolutionary prototyping is to deliver a working system to end-users.
- Here, development starts with those requirements which are best understood.



THROWAWAY PROTOTYPING

- In this type of prototyping a prototype is produced to help explore requirement problems and then discarded. The system is then developed using some other development process.
- Objective here is to identify and verify the system requirement.
- Here, development starts with those requirements which are poorly understood.

ADVANTAGES

- Potential for changing the system early in its development.
- Opportunity to stop development on an unworkable system.
- Problems are easier to trace early
- Possibility of developing a system that closely focuses on users' needs and expectations.

DISADVANTAGES

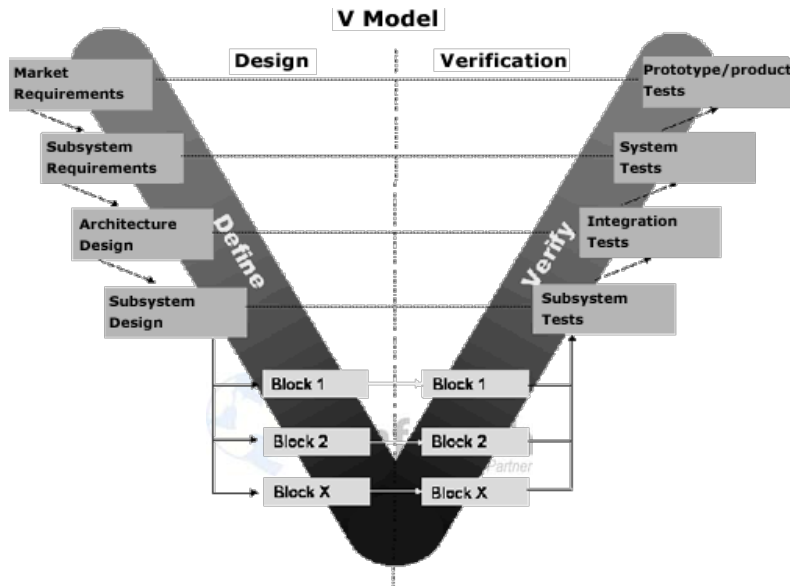
- Managing the prototyping process is difficult because of its rapid, iterative nature.
- Incomplete prototypes have the risk of being regarded as complete systems
- There is an absence of end phase reviews.
- Since the prime focus is on developing a prototype, system documentation takes a backseat.



V-SHAPED MODEL

V model is model in which testing is done in parallel with development.

V-SHAPED LIFE CYCLE MODEL



CHARACTERISTICS

- V model is model in which testing is done in parallel with development.
- The basic strategy here is to make testing in conjunction with V diagram development phases which helps to minimize the testing effort in later stages.
- Every phase of the testing in this model corresponds to some activity in the software development phase.
- The Requirement Analysis would correspondingly have an acceptance testing activity at the end.
- The design has Integration Testing and the System Integration Testing and so on.
- The testing activity formally starts at the fag end of the Coding phase or the Unit Testing activity by the developers.

ADVANTAGES

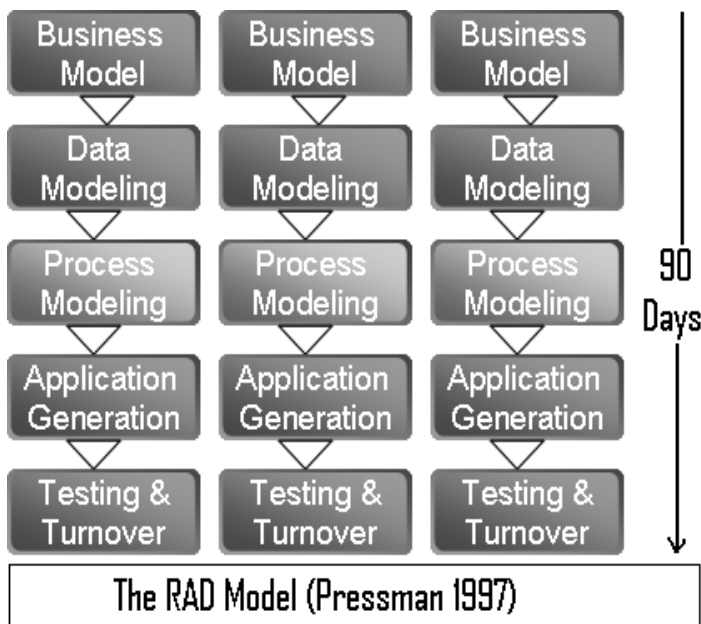


- Simple and easy to use.
- Each phase has specific deliverables.
- Higher chance of success over the waterfall model due to the development of test plans early on during the life cycle.
- Works well for small projects where requirements are easily understood.

DISADVANTAGES

- Very rigid, like the waterfall model.
- Little flexibility and adjusting scope is difficult and expensive.
- Software is developed during the implementation phase, so no early prototypes of the software are produced.
- Model doesn't provide a clear path for problems found during testing phases.

RAD MODEL



CHARACTERISTICS

- Rapid application development (RAD) is an object-oriented approach to systems development that includes a method of development as well as software tools.
- Rapid application development is a linear sequential software development process model that emphasizes on an extremely short development cycle.
- RAD and prototyping are conceptually very close.

THE RAD APPROACH ENCOMPASSES THE FOLLOWING PHASE

BUSINESS MODELING

The information flow in this process answers the following questions

What information drives the business process?

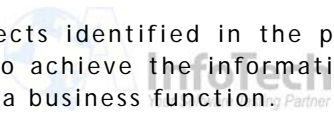
What information is generated?

Who generates it?

DATA MODELING

During this phase the information flow is into a set of data objects essential to support the business.

PROCESS MODELING

The data objects identified in the previous phase are transformed to achieve the information flow necessary to implement a business function. 

APPLICATION GENERATION

The RAD works to reuse existing program components (when possible) or create reusable components (when necessary). In all cases, automated tools are used to facilitate construction of the software.

TESTING AND TURNOVER

RAD process put a great emphasis on reuse which implies that many of the program components have already been tested.

This helps to minimize the testing and development time.

ADVANTAGES

- It aims at shortening the time in design and implementation.
- It is capable of dealing with the rapidly changing requirements.
- It involves users in each part of the development effort.

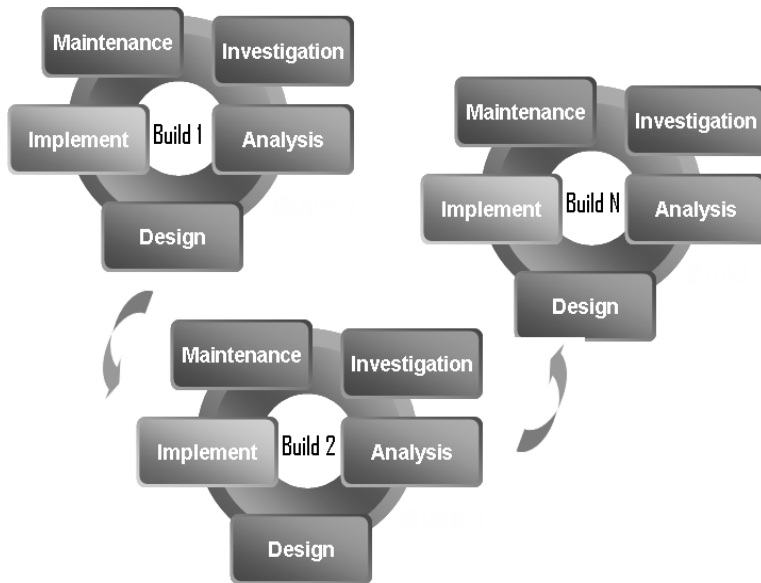
DISADVANTAGES

- May not be appropriate for large projects.
- The complete knowledge of RAD tools and techniques must be known to analysts and users.



EVOLUTIONARY DEVELOPMENT MODELS

INCREMENTAL MODEL



CHARACTERISTICS

- The Incremental model of development is an evolutionary model that is a combination of the elements of the linear sequential model (Waterfall model) and the iterative approach of Prototyping.
- It is ideal for a project that is complex by nature having large business components and interfaces with third party business applications.
- The Incremental Model divides the product into iterations which are developed and tested in parallel.
- Each iteration has its own requirements, design, implementation and testing phase.

- A working version of software is produced during the first iteration, and thus an early version of working software is produced.
- Subsequent iterations created on the basis of initial software produced during the first iteration.

ADVANTAGES

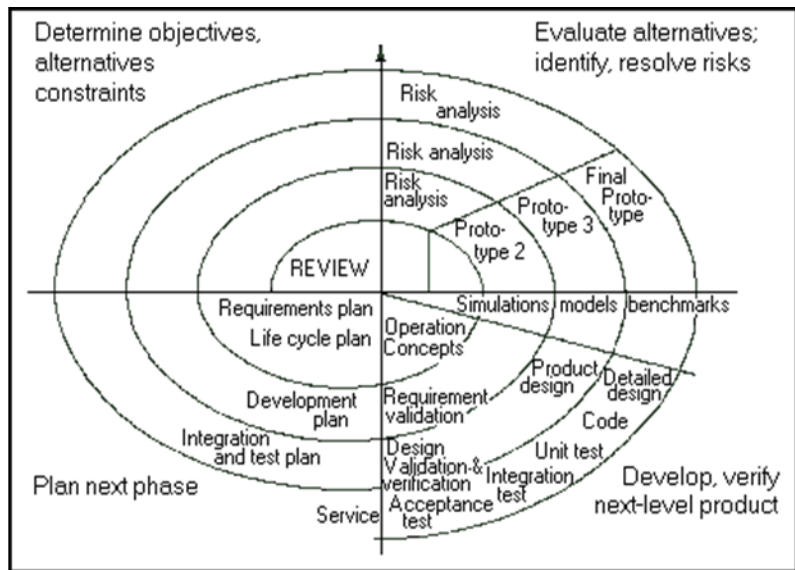
- This model is effective for projects that are too large to be developed as a unit.
- It facilitates changes to be made in user requirements during development, and provides some of the finished product at an early stage.
- It provides a flexible platform where it is less costly to change scope and requirements.
- It is definitely easier to test, detect and remove errors during a smaller iteration.



DISADVANTAGES

- This process is time consuming.
- It involves too much customer interaction and certain design decisions taken by the developer may not be approved by the customer.
- No visible end to the finished product.
- It can very easily degenerate into the build and fix approach.

SPIRAL MODEL



CHARACTERISTICS

- The spiral lifecycle model is the combination of the traditional waterfall model and an element called risk analysis. This model is very appropriate for large projects.
- The spiral model has four phases Planning, Risk Analysis, Engineering and Evaluation. A software project repeatedly passes through these phases in iterations (called Spirals in this model).
- The system requirements are described in details which results in creating a preliminary design for the
- A first prototype of the new system is developed from the preliminary design. Successive prototypes are evolved by testing the preceding ones.

- All the preceding prototypes are kept on refining until the customer is satisfied.
- Based on the refined prototype the final system is developed.
- The final system gets a methodical evaluation.

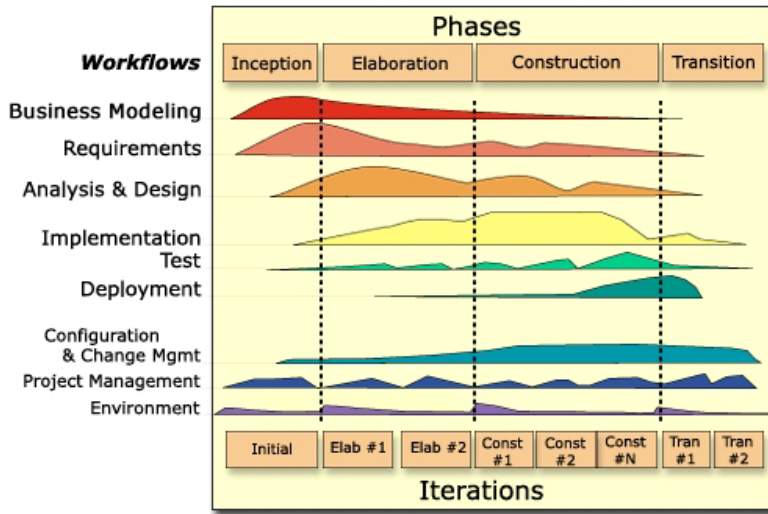
ADVANTAGES

- It is an effective approach for large and complex projects.
- It facilitates the risk analysis at each level. Direct consideration of risks at all levels helps in reducing problems.

DISADVANTAGES

- Needs considerable risk assessment and high amount of expertise.
- If a risk is not discovered, problems will surely occur.
- It doesn't work well for the projects that are smaller in scale.

RATIONAL UNIFIED PROCESS (RUP)



CHARACTERISTICS

- Develop software iteratively
- Divided in to 4 phases
 - Inception
 - Elaboration
 - Construction
 - Transition
- Each phase has its own goal
- Each phase is marked by a milestone that is to be completed before next phase starts.

INCEPTION PHASE

- Requirement and feasibility analysis
- Business case including success factors (expected revenue, market recognition, etc) is established
- Basic use case model, project plan and initial risk assessment are generated.
- Project must pass the Lifecycle objective milestone.

ELABORATION PHASE

- Project starts to take shape
- Problem domain analysis
- Architecture/design of the project gets its basic form.
- Goal is prove that the architecture works via an end-to-end technical prototype.
- This phase must pass the Lifecycle Architecture Milestone



CONSTRUCTION PHASE

- Build a working system that is ready to be put into production
- Development of components and other features
- Test the system
- Fix problems as they are found.
- Marked by the Initial Operational Capability (IOC) milestone.

TRANSITION PHASE

- Finish up the system and package it.
- Training of the end users and maintainers
- Beta testing of the system
- Product is checked against the quality level set in the Inception phase.

- Shipment
- Marked by Product Release (PR) milestone

ADVANTAGES

- Develop software iteratively
- Manage Requirements
- Continuously verify software quality
- Allows QA team to work all the time, not just at the end
- It is cheaper to fix problems sooner



CONCLUSION

Choosing the most appropriate development model for a particular development project depends on a number of factors, including the system being developed, the organizational environment, project budget, development resources, and other factors.





CHAPTER 3

Levels of Testing

“Testing is problem solving. Developing a good series of tests requires an ability to think through multiple possible paths to determine the important combinations. Tracking down bugs requires an ability to troubleshoot effectively, to eliminate unrelated variables, to reduce the problem to the simplest case. As a result, a good tester is both logical and intuitive.”

Elisabeth Hendrickson

OVERVIEW OF THIS CHAPTER

This chapter discusses the most common types of levels of testing and how these levels are further used in the basic techniques of software testing which are as follows:

- ✓ White box testing
- ✓ Black box testing

This chapter also talks about the strategies and steps involved in these two techniques of testing a software product.

LEVELS OF TESTING

Testing occurs at every stage of software development to verify that it is developed in the right manner or not. The further you move along in the software development life cycle, the harder and more expensive it becomes to find and correct the defects.

The development process involves various types of testing. Each test type addresses a specific testing requirement. The most common types of testing involved in the development process are as under

- ◆ Unit Testing
- ◆ Integration Testing
- ◆ System Testing
- ◆ Functional Testing
- ◆ Performance Testing
- ◆ Regression Testing
- ◆ Beta testing
- ◆ Integrity and Release Testing
- ◆ Final Acceptance and Certification

UNIT TESTING

- A unit test is a method of testing the correctness of a particular module of the code.
- The source code is divided into modules, which in turn are divided into smaller components called units. These units have specific behavior. The test done on these units of code is called unit test.
- Each module is tested separately to detect errors in the code.

INTEGRATION TESTING

- Integration testing is a method of testing two or more modules or functions together to find interface defects between them.
- It involves putting together groups of modules to verify that the system meets the specified requirements.

InfoTech
Your Software Testing Partner

INTEGRATION TESTING CAN BE CHARACTERIZED AS TOP DOWN TESTING OR BOTTOM UP TESTING.

- Top down testing
 - Highest levels of modules are tested first.
 - Test drivers are not needed.
 - Stubs are used which are eventually replaced by next level modules.
 - Major emphasis is on interface testing
- Bottom up testing
 - Lowest level of module is tested first.
 - Test drivers are needed to call the lowest module and passing the data.
 - Emphasis is on module functionality and performance.

SYSTEM TESTING

- Several modules constitute a project. Once all the modules are integrated, several errors may arise. The testing done at this stage is called a system test.
- System testing ensures that the entire integrated system meets the specified requirements.

PERFORMANCE TESTING

- Performance Testing is used to measure various system characteristics, such as processing speed, response time, resource consumption, throughput, device utilization, execution time and efficiency.
- Performance tests evaluate response time, memory usage.



REGRESSION TESTING

- Regression testing is a process of selective retesting of a software system that ensures that errors are fixed and the modified code still complies with the specified requirements.
- It also ensures that the unmodified code has not been affected as a result of the changes made.

Regression testing is initiated after a developer has attempted to fix a recognized problem or has added source code to a program that may have inadvertently introduced errors. This testing process involves the repetition of previous successful test any time there is a probability that subsequent changes could have affected the system.

WHILE TESTING A BUG FIX CHECK FOR THE FOLLOWING ASPECTS

- Verify that the concerned bug was actually tackled.
- Look out for the related bugs.
- Find the alternate pathways to reach the bug.
- Check whether this bug fix had any unfavorable consequence on the whole program.

WHAT IS THE SOURCE OF REGRESSION TEST CASES?

Regression test suite contains a battery of difficult tests which has the potential of failing a program if it is broken. Following is the list of common sources of regression tests

- Boundary tests
- Previously run test that led to the exposure of the bug in the program.
- Certain test groups that include bug reports communicated by customers or technical support staff.
- Collection of randomly generated test statistics that will exercise the program with different outputs.

Once you develop a collection of test cases, it's hard to decide which tests are to be run. Here are some useful tips to prioritize the test cases for regression testing

- Leave out the tests that look like the repetition of another.
- Once a bug has been fixed, select only few from the total mass of the tests run to get rid of that bug.
- If there is a possibility of certain test to be repeated very often, try to automate that test.

WHAT IS THE DIFFERENCE BETWEEN RETESTING AND REGRESSION TESTING?

- ➡ Regression testing is the process of re executing the application whether it affects the other parts of application or not.
- ➡ Retesting is just running the same tests and conditions over and over again, to verify reproducibility (precision).



BETA TESTING

- Beta testing is a process where a completed information system is released to a selected group of testers called beta testers and is tested in a real user environment.
- It is a test of the product prior to its commercial release.

ACCEPTANCE

- Acceptance testing is a formal testing conducted to determine whether software system satisfies the acceptance criteria.
- It ensures that the system provides all the advertised functions.



INTEGRITY AND RELEASE TESTING

INTEGRITY TESTING

- It is a last round of testing which involves a thorough testing of the system which is assumed to have all the designed features and functionalities.
- It is done by a customer service or by a different test group from the buyer's perspective.

RELEASE TESTING

- In the process of release testing all the materials that will go to the customers are organized and documented.
- Master disks are made and checked for viruses.



TECHNIQUES OF TESTING

WHITE BOX TESTING

- It is a testing technique where the internal workings of the system being tested are known and are used to select the test data.
- The tests exercise every path through a program.
- It involves the tester to look into the code and find out which unit of the code is broken and not working.
- Also called as glass, structural or clear box testing.

CODE COVERAGE ANALYSIS

- A path is a unique sequence of branches from the function entry to the exit.
- It measures whether each of the possible paths in each function have been followed.
- It is inherently a white box testing activity.

PATH TESTING AND COVERAGE CRITERIA

- It is not possible to test all the paths.
- Coverage criteria specify the paths to be tested. It describes three criteria
 - Line coverage
 - Branch coverage
 - Condition coverage
- Testing done in accordance to these criteria is called path testing.

WHITE BOX TESTING STRATEGIES

UNIT TESTING OR INCREMENTAL TESTING

The developer carries out unit testing in order to check if the particular module or unit of code is working fine.

BIG BANG TESTING

It is a technique where a program is tested as full system integration and not as individual pieces or modules.

STATIC TESTING

Code is examined and tested without being executed.

DYNAMIC TESTING

Dynamic analysis involves executing the code and analyzing the output. Code is tested without being examined.

STATEMENT COVERAGE

In this type of testing the code is implemented in a way to ensure that every statement of the application is executed at least once.

It helps in assuring that all the statements execute without any consequence.

BRANCH COVERAGE

Branch coverage helps in validation of all the branches in the code.

It makes sure that no branching leads to uncharacteristic behavior of the system.

SECURITY TESTING

Security Testing helps to determine how efficiently the system can protect itself from unauthorized access and Any code damage.



BLACK BOX TESTING

A software testing technique where the internal workings of the system being tested are not known.

SEQUENCE OF EVENTS INVOLVED IN BLACK BOX TESTING

- Test planning
- A detailed planning and designing is done in the first cycle of testing.
- Acceptance testing
- Function test and system test
- Verification and validation
- Beta testing.
- Final acceptance and certification
- Integrity and release testing





CHAPTER 4

Testing Cycle

Here are many approaches to software testing, but effective testing of complex products is essentially a process of investigation, not merely a matter of creating and following rote procedure.

OVERVIEW OF THIS CHAPTER

This chapter deals with various stages in the Testing cycle.

- ✓ Testing the requirements
- ✓ Test Planning
- ✓ Test case creation
- ✓ Test execution
- ✓ Defect logging process
- ✓ Analysis of defects



TESTING CYCLE

TESTING THE REQUIREMENTS

Requirement is a description of what a system should do.

A requirement or system analyst documents the requirements which are then passed on to QA team in the form of Software Requirements Specification (SRS) document.

QA team reads the specifications to understand the requirements. Requirement document is thoroughly reviewed for completeness, testability and traceability. Any query and uncertainty is documented in a **Review Comment form**.

QA team also starts writing the Requirement Based Test cases based on the received requirements.

Other activities that take place simultaneously are:

Test (strategy) Planning
Test Estimation

Now, the same requirements specification documents are also sent to a system design specialist who starts making high level and low level design. Based on those designs **Data Flow Diagrams, Entity Flow Diagrams** and **Use Cases** are developed.

These items are also passed on to QA team and are reviewed for completeness and traceability. Any query and uncertainty is again documented in a **Review Comment form**.

Document Details	
Document Name	
Document Version	
Author	
Reviewer Details	
Reviewer's name	
Date	
Priority(H=High; M=Medium; L=Low)	

To Be Completed By Reviewer						To Be Completed By Author	
No.	Reviewer's Initials	Section	Page	Reviewer's comment	Priority	Accept / Discuss / Duplicate	Author's comments

This effort is repeated till the time requirements are Baselined. At the same time Scenario Based Test Cases are written.

At the end of the process, the baselined requirements turn out to be the blue print of the product, a reference point for the client, the project manager, the tester and the designer.

USE CASE

WHAT IS A USE CASE?

A use case is a sequence of actions performed by a system, which together produce results required by users of the system.

A Use Case is a text description of how a system will be used and is usually supplemented with a graphic representation of system users, called Actors and the use of the system, called actions.

It is comprehensive description of a task or a series of tasks that users will accomplish using the software, and includes the responses of the software to user actions.

WHAT ARE THE OBJECTIVES OF USE CASE ANALYSIS?

- To document the requirements of the proposed system in an easy to read, easy to track text format and serve as an effective communication tool.
- To document the business process that is to be supported by the system under development.
- To identify business decisions, actions and constraints.
- Represents the goal of an interaction between an actor and the system.
- Records a set of paths (flow of events) that traverse an actor from a trigger event (start of the use case) to the goal (success scenarios).
- Records a set of scenarios that traverse an actor from a trigger event toward a goal but fall short of the goal (failure scenarios).

COMPONENTS OF A USE CASE

NAME

Each use case should have a name that indicates which process the use case describes.

OVERVIEW

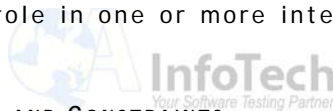
It provides a quick summary of the scope and purpose of the process documented in the use case. It gives a fair idea of the total document without having to dig in to the details.

ASSOCIATIONS / DEPENDENCIES

An association exists whenever an actor is involved with an interaction described by a use case. Associations are modeled as lines connecting use cases and actors to one another, with an optional arrowhead on one end of the line.

ACTORS

An actor is a person, organization, or external system that plays a role in one or more interactions with your system.



BUSINESS RULES AND CONSTRAINTS

These are the formal rules and limitations that a Use Case operates under.

PRE-CONDITIONS

Pre-conditions describe the expected state of the system before the use case is initiated. The use case will fail if any constraint specified by a pre-condition is not met. They are the guideline to what must be checked in the system before the process starts.

TRIGGERS

Triggers are the events that initiate the use case. A trigger could be an actor, another use case, or a temporal event.

FLOW OF EVENTS

These flows must be accompanied with the step no., actor action, system response, and business rules.

- The Basic Flow Of Events
- This flow basically envelopes the normal events that occur when a use case is executed.
- Alternate Flows of Events
- The alternate flows of events specify where the steps start in the basic flow and here they go when they end

End users of the completed system can go down many paths as they perform the functionality specified in the use case. This means that if the basic flow is followed, it is one scenario. But following the basic flow plus alternate flow 2 would be another. Similarly, the basic flow plus alternate flow 3 would be a third, and so on.

- Exception Flows
Describes the unusual situations that are not defined by the use case.

POST-CONDITIONS

Post-conditions describe the final state of the system after a successful completion of the typical course of events.

ASSOCIATED USE CASES

It is useful for a reader of the use cases to be able to reference all related use cases in one place. These associated use cases are the result of <includes> or <extends> relationship. It ensures that the use case reader have access to all the conditional processes which are documented in separate use case.

VISUAL REPRESENTATION

Use case diagrams are created to present the relationship between use cases and actors. It provides a graphical description of who will use the system and what kind of interactions are going to take place in that system.

EXAMPLE USE CASE

NAME: LOGIN.

Description: This use case handles the access to the system. If the user has no access (that is, is unregistered), he or she will be directed to a registration page.

ACTORS: CUSTOMER

Pre conditions: The user has browsed the home page, filled in the account details and must now log in.

POST CONDITIONS:

1. If the user was in the system, he is logged in and has access to the system;
2. The main menu is shown to the user;
3. If the user was not in the system an error message is shown with a possibility to register for the site.

FLOW OF EVENTS:

BASIC FLOW

1. The use case starts when the user browses into the Web store's front page;

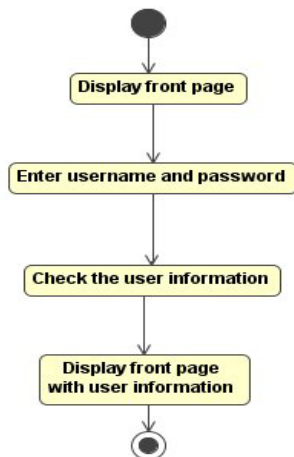
2. The system will display the front page with text boxes for username and password, a button for accepting the information, and a link for registration;
3. The user enters a username and password;
4. The system checks the information;
5. The system displays the front page with user-specific information;
6. The use case ends.

ALTERNATIVE FLOW

1. The use case starts when the user browses into the Web store's front page;
2. The system will display the front page with text boxes for username and password, a button for accepting the information, and a link for registration;
3. The user clicks the link to register;
4. The system displays a registration form with text boxes for name, address, telephone number, and e-mail;
5. The user fills in the information and accepts it;
6. The system loads a page where it shows the information again and asks the user to verify it;
7. The user verifies it;
8. The system gives the user guidance to accept the registration by checking his or her e-mail;
9. The user checks his or her e-mail and clicks the confirmation link;
10. The use case ends.

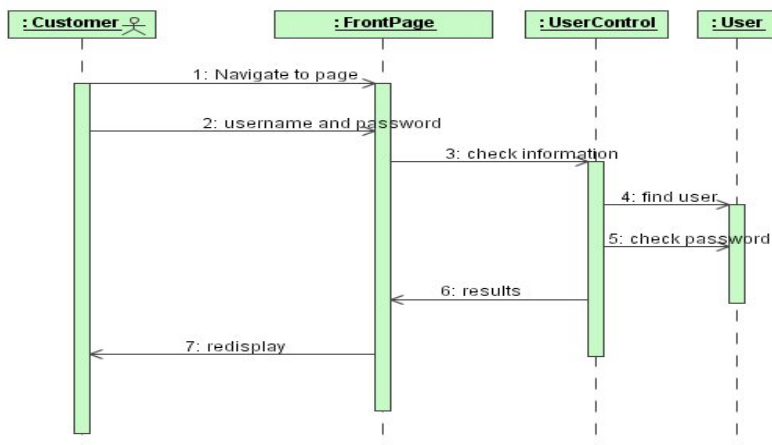
DIAGRAMMING THE LOGIN USE CASE

Following are the three diagrams called for in the documentation template. The activity diagram depicts the flow of events or some part of the flow of events.



A sequence diagram depicts the different application scenarios (use cases) as a sequence of actions.


Sequence diagram



TEST PLANNING AND DOCUMENTATION

Test Planning and Documentation is a systematic investigation of the program so as to develop a clear and efficient strategy to execute testing.

OBJECTIVES OF TEST PLANNING AND DOCUMENTATION

- It gives a clarity to the technical tasks of testing which enhances your ability to test in the following ways
 - Testing coverage is perfected.
 - With proper check lists, no item is forgotten.
 - Present a basic structure for the final test.
 - Check the completeness of your testing effort for any overlooked aspect.
- It acts as a bridge of communication between the product development team.
 - Brings out the feedback on testing accuracy and coverage.
 - With a clear test plan it is easier to reach specific agreements regarding size and timings of a testing job.
- It provides a basic guideline for organizing, scheduling and managing the test project. It helps in the following areas
 - Identification of the tasks
 - Plan a structure
 - Organizing the testing team
 - Measuring the project status

TYPES OF TEST DOCUMENTS

➤ Test plan

According to IEEE Std 829-1983

A document describing the scope, approach, resources, and schedule of intended testing activities. It identifies test items, the features to be tested, the testing tasks, who will be doing each task, and any risks requiring contingency planning.

Test plan is a formal document that covers the What, How, Who and When aspects of software testing.

- Purpose of the document
- Product description
- Test items
- Features to be tested
- Features to be tested
- Testing strategy
- Test approach
 - Test environment
 - Testing tools
 - Browser
- Test entry and exit criteria
- Test deliverables
- Criteria for success
 - Readiness criteria
 - Pass/fail criteria
 - Completion criteria
 - Acceptance criteria
- Dependencies and assumptions
- Risk Analysis and Contingency Plan
- Responsibilities
- Staff and training
- Schedule

➡ Test design specification

It explains how a feature or a set of feature will be tested. It contains the following sections

- Test design unique number
- Features to be tested
- Test techniques
- pass/fail criteria

➡ Test case specification

It is a document that defines a test case and specifies a set of test inputs or data, execution conditions, and expected results. It consists of the following parts

- Test case unique number
- Features to be tested
- Input specification
- Output specification
- Environmental needs
- Inter-case dependencies

➡ Test procedures specification

It gives out a stepwise description of executing a set of test cases and evaluating the results. It is comprised of the following sections

- Test procedure unique number
- Objective
- Steps followed

TEST CASE DEVELOPMENT

At this stage, we have a fully developed test plan that has

- The checklist identifying what areas/features are to be tested.
- Information about the environment conditions, resources and schedule.
- All other test-relevant information compiled in an organized manner.

This is when we start writing **Test Cases**.

WHAT DO WE MEAN BY A TEST CASE?

Test cases typically describe actions (steps) and the expected results of that action.

It includes test inputs, execution conditions, and expected results, identified for the purpose of making an evaluation of some particular aspect of a system under test.

TECHNIQUES TO GENERATE GOOD TEST CASES

- Equivalence Partitioning
- Boundary Value Analysis
- State-Transition Testing
- Race conditions and other time dependencies
- Error guessing

EQUIVALENCE PARTITIONING

The objective of the generation of equivalence classes is to detect errors with a minimum number of test cases.

The principle behind this is to partition all input data of a program in to a finite number of equivalence classes with the assumption that system behaves in the same manner for all tests of each partition.

TWO TESTS ARE CONSIDERED EQUIVALENT IF

- They have the same input variables.
- They result in similar operations.
- They affect the same output variables.

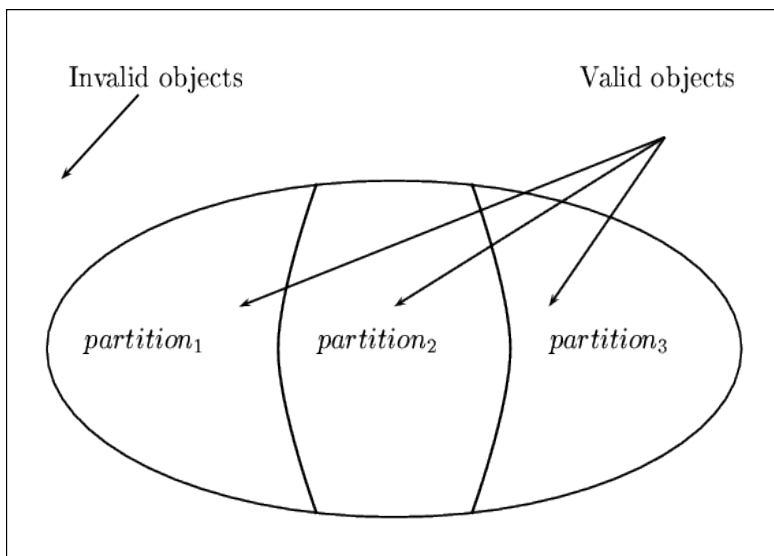
Equivalence partitions are designed so that every possible input belongs to one and only one equivalence partition.



When defining equivalence classes, two groups of equivalence classes have to be differentiated

- Valid equivalence classes
- Invalid equivalence classes

For valid equivalence classes, the valid input data are selected; in case of invalid equivalence classes erroneous input data are selected.



Example: If the input requires a number in the range 1-12,

Valid Equivalence classes: 1..12 Test case: 4

Invalid Equivalence classes: <1 Test case: -1

Invalid Equivalence classes: >12 Test Case: 20

BOUNDARY VALUE ANALYSIS

Boundary value analysis is a technique of Black box testing in which input values at the boundaries of the input domain are tested.

Boundaries mark the point or zone of transition from one equivalence class to another.

These are good members of equivalence classes to use because the program is more likely to fail at a boundary. It is a fact that input values at the extreme ends of, and just outside of, input domains has the tendency to cause errors.

To ensure proper functionality of the system, values at, and just beyond, the boundaries of the input domain are used to generate test.

Example: If the input requires a number in the range 1-12,

Valid Equivalence class: 1..12 Test Cases: 1,12

Invalid Equivalence class: <1 Test Case: 0

Invalid Equivalence class: >12 Test Case: 13

TESTING STATE TRANSITIONS

It is particularly useful technique for generating test cases. Test cases are designed to test the transitions between states by creating the events which lead to transitions.

For each test case define

- The starting state
- The input events that cause the transitions
- The output results of each transition
- The end state

For testing interactions between the paths

- Test all the paths that are likely to be followed by the user
- Test few random paths through the program.



RACE CONDITIONS AND OTHER TIME DEPENDENCIES

What is a race condition?

To understand race condition, let's assume there are two events taking place A and B.

In the normal courses of events A is performed first. But there might be a situation where B precedes the event A. this is the situation of a race condition when unexpected happens. The system is said to be having a race condition bug if the system fails because of this condition.

The task here is to ensure that the program work even under the unfavorable conditions. And also to check the conditions or the limits that make a program to crash.

- Try to interrupt the program during its transition state.
- Test the program by running it under a heavy load.
- Test all the limits of the program.



ERROR GUESSING

It is a test case design technique where the tester uses his experience and skills to assume what faults might occur and thereby design test cases specifically to reveal those faults.

A well placed error guess can show a bug which could easily be missed by many of the other test case design techniques presented in this paper. Conversely, in the wrong hands error guessing can be a waste of time.

While following this effective technique to design test cases, it is a good idea to build a check list of types of errors.

This check list can then be used to help "guess" where errors may occur within a unit. The check list is maintained to benefit from the experience gained in earlier unit tests, and thereby, improving the overall effective



TEST EXECUTION

This stage basically deals with the execution of test cases. Test cases/scripts are executed to find any kind of unexpected behavior shown by the system. The key action here is to look for any inconsistency in the actual and expected result. If any such behavior is noted it is called a bug/error. These errors are then reported to the development team.

Test ID	Function	Description	Pre Requisite	Input/actions	Expected result	Actual result	Result

Result is defined by Pass or Fail criteria.



WHAT IS A BUG/ERROR?

A software error, also called as bug or a defect in the program is a mismatch between the program and its specification if and only if the specification exists and is correct.

It is the deviation of the program's functionality from the expected. Test cases are executed to find any kind of unexpected behavior shown by the system. If any such behavior is noted it is called a bug.

WHY ARE THERE ERRORS?

Software error can exist due to many reasons. Some of them are as follows

- When programmers deal with tasks that aren't fully understood or fully defined.
- When late design changes are introduced.
- Failure to use source and version control tools.
- When one programmer tries to fix bugs, or otherwise modify another programmer's code.
- And, sometimes people just make mistakes.

Categories of Error

- User interface errors
- Error handling
- Boundary related errors
- Control flow
- Errors in handling and interpreting data
- Race conditions
- Load conditions
- Hardware / environment compatibility
- Source and version control
- Testing
- Documentation

Anyone can stumble upon bugs.

The key point is to

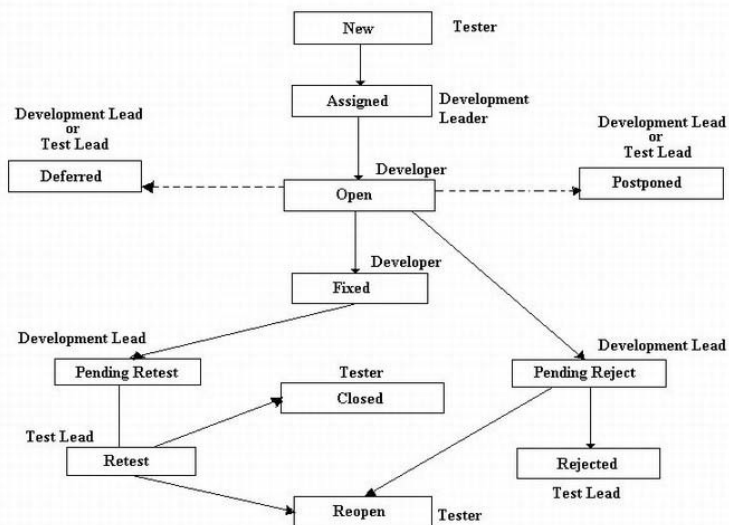
Find the right Bugs.

Find the right Bugs fast.

Find the right Bugs fast choosing the best Test Strategies.

WHAT IS A BUG LIFE CYCLE?

Bug life cycle marks the milestones from the time a bug is found (status new) to the time it is successfully closed (status closed). The bug can also be rejected, postponed or deferred. During this cycle a bug is assigned various statuses like New, Open, Assigned, Re-open, Deferred and Closed.



Bug Life Cycle

1. A Tester finds a bug and reports it to QA Lead.
2. The QA lead verifies if the bug is valid or not.
3. First scenario, Test lead finds that the bug is not valid and the bug is **'Rejected'**.
4. Second scenario, the bug is verified and reported to Development Lead with status as **'Open'**.
5. In case of step 4, the development lead verifies if it is a valid bug.
6. In case of Invalid bug, it is passed back to QA team with status marked **'Pending Reject'**.
7. If QA team confirms that a bug is invalid, the QA Lead marks the bug as **'Rejected'**.
8. In case the bug is valid, Development lead assigns a developer to fix it marking the status as **'Assigned'**.
9. When the developer is working on it, it is marked in **'Addressed'** state.
10. The developer solves the problem and marks the bug as **'Fixed'** and passes it back to the Development Lead, who gives it back to the QA team to retest. Bug in this state is marked in **'QA state'**.
11. If it is working fine, tester closes the bug and marks it as **'Closed'**.
12. It might happen that the after retesting it is found that the same problem persists. In this case QA team reopens the bug and marks it with **'Reopen'** status. And the bug is passed back to the Development team for fixing.
13. A bug is classified as **'Deferred'** when it won't be fixed in this release.

BUG REPORT FORM

After a bug is found, it needs to be communicated and assigned to developers who can fix it. After the problem is resolved, fixes should be re-tested, and determinations should be made regarding requirements for regression testing to check that fixes didn't create problems elsewhere. If a problem-tracking system is in place, it should encapsulate these processes. A variety of commercial problem-tracking/management software tools are available.

As soon as you find a bug, log it in the defect tracking tool, wherein a bug report form is filled.

Remember, before you log a bug in defect tracking tool, make sure it is not already logged.

FIELDS IN A BUG REPORT FORM

(The number of fields and their meaning may change from organizations to organizations).

Project: Name of the project under which the testing is being carried out.

Defect ID: This is a unique ID i.e. number created for the bug at the time of reporting, which identifies the bug uniquely.

Submitter: Name of the person entering the defect into the defect tracking system.

Status: This field displays the current status of the bug.

Type: This field describes the type of problem.

- Coding error
- Design issue
- Suggestion

- Documentation
- Hardware
- Query

Headline/Synopsis: It gives a brief description of what the bug is all about.

Product Version:

This field contains the version information of the software application in which the bug was detected.

Severity:

Severity is customer-focused. It quantifies the impact of a defect on functionality. An error is assigned a severity level based on its seriousness. Though the terminologies may vary from company to company, severity can be classified in to the following

Critical: The entire system or critical functionality is affected and is completely unusable, no work around is possible, fatal to the release.

For example infrastructure has failed (a server has crashed, the network is down, etc.), a functionality critical to the purpose of the website is broken, such as the search or commerce engine on a commerce site.

Major: A major part of the system or critical functionality is affected. A reasonable workaround is available, but requires a tremendous effort on the part of the user.

For example, a major functionality is broken or misbehaving, one or more pages is missing, a link on a major page is broken.

Average: Feature/Function is not working as expected. A workaround is available, and the users are able to complete their transactions despite the problem.

For example data transfer problems (like an include file error), page formatting problems, including slow pages and graphics, browser inconsistencies, such as table rendering or protocol handling.

Minor: A problem not affecting the actual function. (Cosmetic Defect)

For example display issues like font inconsistencies or color choice, text issues like typos, word choice, or grammar mistakes or page layout issues, like alignment or text spacing.

Priority:

Priority is business-focused. This value quantifies the necessity for the defect to be resolved. Again, the terminologies may vary from company to company but generally priority can be classified as following

1 or Critical priority: the priority is so high it must be fixed immediately.

2 or High priority: Any problem interfering with a major functionality is a high priority. It means to fix as soon as possible.

3 or Medium priority: These problems can usually wait until the more important problems are fixed. It means to fix before the next milestone.

4 or Low priority: These problems may not be affecting the actual functionality and can be postponed.

Severity of the bug is one of the factors for assigning priority for a bug. Other considerations are time and resource constraints. In most of the times High Severity bug is becomes High Priority bug, but it is not always the case. There are some instances where high Severity bugs will be low Priority and low Severity bugs will be high Priority. For example, if you have a simple typo on a page, that's minor severity because of its low scope -- but if that page is your home page and the typo is your company name, it becomes high priority.

Reported By: This field requires the name of the tester who detected/reported the bug.

Date: Date of submitting the report.

Assigned To/Owner: Name of the developer responsible for current action of change request depending on current state. Generally this field contains the name of developer group leader, who then delegates the task to member of his team, and changes the name accordingly.

Problem Summary: The ideal summary tells the reader what the bug is, what caused the bug, what part of the program it's from and what its worst consequence is.

Reproducible: Before writing a report you must always try to recreate the bug. Never say it is reproducible unless you have recreated the bug. If the error appears from time to time and you do not yet know why, say "intermittent" and explain.

Description: Submitter enters in full description of the problem. This field includes the symptoms and steps taken to re produce the problem.

- Describe all the steps from the start.
- Note what else you did before starting on the series of steps that led to this bug.
- Review similar problem reports you've come across before.
- Use tools such as capture/replay program, debugger, debug-logger that can help you identify things that you did before running into the bug.
- Mention actual and expected result.
- If you expect the reader to have any trouble reproducing the bug, be clear.

Resolution: Resolution emphasizes on the current state of the problem and Resolution version specifies what version contains the change.

Different types of Resolution

Pending fixed

Irreproducible

Deferred

As designed

Withdrawn by reporter

Disagree with suggestion

Duplicate

Attachment: It is important to attach screen-shots and audio/video files for the tested functionality as it helps in re-creating the similar testing condition. It's a good practice to take screen shots of execution of every step during software testing. Attachments may also contain a disk containing test data, a set of macros that will generate the test case and a memo explaining why you think the problem is important.

History: This field has the details of all the actions performed on a Defect or an enhancement. This is automatically populated.




EXAMPLE OF HOW A BUG IS LOGGED IN A DEFECT TRACKING TOOL (JIRA)

HOME BROWSE PROJECT FIND ISSUES CREATE NEW ISSUE ADMINISTRATION



Create Issue

Step 2 of 2: Enter the details of the issue...


Project: CompuTaught Ecommerce

Issue Type:  Bug

* Summary:

Priority: Major  

Component/s: Unknown

Affects Version/s: 
Unknown
Unreleased Versions
Basis
1404
1570
1422

Environment:

For example operating system, software platform and/or hardware specifications


Description:

Original Estimate:

An estimate of how much work remains until this issue will be resolved.
The format of this is ' *w *d *h *m ' (representing weeks, days, hours and minutes)
Examples: 4d, 5h 30m, 60m and 3w.


Attachment:

The maximum file upload size is 10.00 Mb. Please zip files larger than this.

External Partners:  None

Contracted work

EXAMPLE OF HOW A BUG IS LOGGED IN BUGZILLA



Bugzilla

Bugzilla – Enter Bug: AceQA

Home | New | Search | Find | Reports | My Requests | My Votes | Preferences | Log Out

Before reporting a bug, please read the [bug writing guidelines](#), please look at

Reporter: karandeepsingh@qainfotech.net

Version: 1.0

Severity: Major

Priority: Medium

Initial State: NEW

Assign To: Vijay Bahadur <vijaybahadur@qainfotech.net>

Abha Gupta <abhagupta@qainfotech.net>

abhishek <abhishekbhasin@qainfotech.net>

Cc: Abhishek <abhishektomar@qainfotech.net>

Abhishek Kumar <abhishekkumar@qainfotech.net>

Abhishek Mathur <abhishekmathur@qainfotech.net>

Default CC:

Estimated Hours: 0.0

Deadline: (YYYY-MM-DD)

URL:

Summary:

Description:

Attachment:

Depends on:

Blocks:

Only users in all of the selected groups can view this bug:
(Leave all boxes unchecked to make this a public bug.)

- ☒ Access to bugs in the Site AceQa product
- ☐ temp

Product: AceQA

Component:

Content

↑
↓

Platform:

▼

OS:

▼

EXAMPLE OF HOW A BUG IS LOGGED IN A DEFECT TRACKING TOOL
(CLEAR QUEST IN THIS CASE)

Submit Defect ECE200022206

[Main](#) [Environment](#) [Resolution](#) [Attachments](#) [Customer](#) [History](#)

Main			
Submitter	rtripathi	Defect ID	ECE200022206
State	New	Old ID	
		Type	<input type="text"/>
Headline	<input type="text"/>		
Platform	<input type="text"/>	Component	<input type="text"/>
Product	<input type="text"/>	Database	<input type="text"/>
ProductVersion	<input type="text"/>	OtherDatabase	<input type="text"/>
Severity	<input type="text"/>	FixPriority	<input type="text"/>
ReportedBy	<input type="text"/>	Reproducible	<input type="text"/>
Submit Date	11/22/2005 5:53:55 AM	Owner	<input type="text"/>
Description	<input type="text"/>		

[Main](#) [Environment](#) [Resolution](#) [Attachments](#) [Customer](#) [History](#)

Environment	
Environment Information	
Environment Accessed	<input type="text"/>
Browser	<input type="text"/>
Operating System	<input type="text"/>
Hardware	<input type="text"/>
ComponentVersion	<input type="text"/>
FoundInIteration	<input type="text"/>
Comments:	<input type="text"/>

[\[Main\]](#) [\[Environment\]](#) [Resolution](#) [\[Attachments\]](#) [\[Customer\]](#) [\[History\]](#)

Resolution			
FixedInProductVersion	<input type="text"/>	FixInBuild	<input type="text"/>
FixedInIteration	<input type="text"/>	VerifiedInIteration	<input type="text"/>
ResolutionNote	<div></div>		
Notify	<input type="text" value="login_name"/> <div> <input type="button" value="Search 'n' Add"/> <input type="button" value="Remove"/> </div>		
RootCause	<input type="text"/>	FixStatus	<input type="text" value="NotStarted"/>
Only Rel. Schedule	<input type="text"/>	TargetFixDate	<input type="text" value="12/6/2005"/>
Estimated Effort	<input type="text"/> hrs.	Actual Effort	<input type="text"/> hrs.
Duplicate of:			

[\[Main\]](#) [\[Environment\]](#) [\[Resolution\]](#) [Attachments](#) [\[Customer\]](#) [\[History\]](#)

Attachments		
Information	Actions	<input type="button" value="Add"/>

[\[Main\]](#) [\[Environment\]](#) [\[Resolution\]](#) [\[Attachments\]](#) [Customer](#) [\[History\]](#)

Customer			
Customer Details			
Name	..	Organization	LocationID
<div></div>			<input type="button" value="Search 'n' Add"/>
<div></div>			<input type="button" value="New"/>
<div></div>			<input type="button" value="Remove"/>
CallTrackingID	<input type="text"/>		
Customer Severity	<input type="text"/>		

[Main] [Environment] [Resolution] [Attachments] [Customer] History

History		
action_timestamp	user_name	action_name old_state new_state
UserComments		
UserName	DateEntered	Comments
<div style="border: 1px solid black; height: 100px; width: 100%;"></div>		
<input type="button" value="New"/>		

STRATEGY TO ANALYZE A REPRODUCIBLE BUG

- Look For The Follow Up Errors
- Keep using the program after you get problem. Does anything else happen?
- Check In Previous Program Versions For Errors
- Check if the error was present in the last version or was introduced as a part of a change.
- Look for Configuration Dependence
- Sometimes a bug will show itself as more serious if you run the program with less memory, a higher resolution output, more graphics on a page, etc.



CHAPTER 5

Internationalization

“Localization is the process of adapting software for a new place. It might involve a change of languages but it doesn’t have to. Along with changing the language you have to adapt to the culture too.”

Cem Kaner

OVERVIEW OF THIS CHAPTER

This chapter deals with the process of designing a product in such a way that it performs as per specifications even when it is modified for use in different locales and languages. These processes involve the following methods:

- ✓ Internationalization
- ✓ Localization

Internationalization is the process of designing and coding a product enabling it to perform accurately when it is modified for use in different languages and cultural conventions. Internationalization is also referred to as I18N because of the eighteen characters between I and N.

Internationalization includes processes and practices that make a product or a service compatible in majority of countries and economies.

Internationalization involves extracting language and cultural knowledge from the software source code. For example message translations, date/time format, text processing, character encoding, etc.

This is what an internationalized application does:

- Combines information that is language and culture specific into external resource files.
- Enables the user interface (UI) to display different language and cultural information.

FOR AN EFFECTIVE INTERNATIONALIZATION TESTING, IT IS IMPORTANT TO CONSIDER THE FOLLOWING ASPECTS

- Check whether time, date, numbers, and collated data show the correct, locale-specific formats.
- Check if the product requires any extra variables to be set in order to run in another surrounding.
- Ensure that the multi byte information get printed correctly.
- Verify that GUI and command line messages, objects, and labels are displayed correctly in all locales.
- Make sure that the product's email, Web transmissions, and registration information are processed properly.
- Make sure that systems and help database searching are working correctly.
- Verify that multi byte file and directory names, file contents, and other multi byte data is processed, viewed and saved correctly.
- Check whether certain source code tests and reviews generate the desired results.

Localization is the process of actually adapting internationalized software to the needs of users in a particular geographical or cultural area.

This involves translating and customizing a software product for a specific market.

Localization testing involves translating the product user interface and sometimes changing the initial settings to make it suited better for some local market. The test effort during localization testing focuses on areas affected by localization such as UI and content. It pays attention to culture-specific areas.

The process of Localization involves translation of UI elements such as strings used in buttons, labels, text messages etc, online help information, documentation, and inclusion of language specific components such as spell checkers etc.

Localization is also referred as L10N because of the ten characters between L and N.

FOR AN EFFECTIVE LOCALIZATION TESTING, IT IS IMPORTANT TO CONSIDER THE FOLLOWING FACTORS

- Check if there was a change made in the base code.
- Work with someone fluent in the language.
- Find out whether the text is independent of the code.
- Must be familiar that the translated text has the potential to take bigger space.
- Must have a thorough knowledge of the character sets categorization.
- Consistency checking of printed documentation, online help, messages, interface resources, command-key sequences, and so on.
- Confirmation of adherence to system, input and display environment standards.
- User interface usability.
- Assessment of cultural appropriateness.
- Check for the hyphenation, spelling, sorting and underscoring rule.
- Look out the difference in the printers, sizes of paper, data format and set-up options.
- Watch out for measurements and ruler differences.

WHAT IS UNICODE?

Unicode provides a unique number for every character,
no matter what the platform,
no matter what the program,
no matter what the language.

Basically, computers deal with numbers. Letters and other characters are stored by assigning each one of them a unique number.

Before Unicode was invented, these unique numbers were assigned by using hundreds of different encoding systems. But no single encoding could contain enough characters for example; the European Union alone requires several different encodings to cover all its languages. Even for a single language like English no single encoding was adequate for all the letters, punctuation, and technical symbols in common use. A single source code reduces time and costs to market, as there is no need for changing or recompiling it for different languages and locales.

This is where Unicode steps in

- Unicode is a universal character-encoding scheme that allows you to store information from any major language using a single character set.
- Unicode defines properties for each character, standardizes script behavior.
- Provides a standard algorithm for bidirectional text.
- Defines cross-mappings to other standards.

UNDERSTANDING UNICODE

Unicode is a character encoding standard developed by the Unicode Consortium. The aim of the standard is to provide universal way of encoding characters of any language, regardless of the platform, being used.

The core of Unicode, known as the Basic Multilingual Plane, contains space for over 65,000 characters. These include some 49000 characters from the world's languages; include letters from alphabets, ideographs used in writing systems such as Chinese, and other characters such as currency and mathematical symbols.

In addition to these, space is available for custom use, and supplementary code points are available for characters.

A 16-bit character encoding scheme allowing characters from Western European, Eastern European, Cyrillic, Greek, Arabic, Hebrew, Chinese, Japanese, Korean, Thai, Urdu, Hindi and all other major world languages to be encoded in a single character set.



CHAPTER 6

Automation

Automation testing reduces errors, as well as, testing costs. Human beings are error prone in the things they do. This is not bad thing, it is just a consequence of being human.

OVERVIEW OF THIS CHAPTER

This chapter deals with the basics of automation and the automated tool requirements. It gives an insight to the various phase involved in the automation test strategy.

- ✓ Basics of automation
- ✓ Automated tools
- ✓ Automation strategy
- ✓ Silk test

BASICS OF AUTOMATION

- Test automation is a technique of replicating the human action of testing applications; with the usage of test automation tools.
- The test automation tool
 - Provides the inputs to the application.
 - Compares the output with the expected output and generates the resulting report, thus ensuring that the application behaves as expected.
- User record the human action using these tool features and plays them back on the application

WHY AUTOMATE?

- Automation is the use of strategies and tools that reduces the need of manual interaction in unskilled, repetitive or redundant tasks.
- Automating testing reduces testing errors, as well as, testing costs.
- Automating testing removes the randomness of test cases as they are created according to intelligent guidelines enforced by the tool and stored in the tool's test pool. Thus, complete testing coverage is a more readily achievable goal with an automated suite of test tools.



SOME AUTOMATION MYTHS

- Automatic Test Plan Generation
- One Test Tool Fits All
- Immediate reduction in schedule
- Replaces manual testing
- Once record and playback is done, all is done
- Automation will pay for itself in no time
- Test automation is not software development
- 100% test automation.

THE FOLLOWING TYPES OF TESTING CAN BE AUTOMATED

- Functional testing
- Regression testing
- Stress testing
- Performance testing
- Load testing
- Acceptance testing



NEED FOR AUTOMATED TOOLS

Choosing the right tools for the job and targeting the right areas to install them is the most important consideration. The right areas where the automation fit must be chosen.

The following are the areas where automation is required

- Highly redundant tasks or scenarios
- Repetitive tasks that tend to cause human error
- Well-developed and well-understood use cases or scenarios first
- Stable areas of the application must be automated rather than choosing the unstable ones.

TEST TOOLS

TEST TOOL REQUIREMENTS

- The tool must support object-level record and playback.
- The Tool must have a robust high-level scripting language.
- The tool must support variable methods of comparing test results.
- The tool must run on all of the platforms you support.
- The tool must work in both 16 bit and 32 bit Windows environments.
- The tool must support foreign languages.
- The tool must support a range of test methods.
- The tool must support custom controls.
- The tool must support synchronized and unattended playback.
- The tool must be reliable.

AUTOMATED TESTERS MUST FOLLOW THE FOLLOWING GUIDELINES TO GET THE BENEFITS OF AUTOMATION

- Concise
- Self-Checking
- Repeatable
- Robust
- Sufficient
- Necessary
- Clear
- Efficient
- Specific
- Independent
- Maintainable
- Traceable

AUTOMATION TEST STRATEGY

Automation test strategy is same as any other software development effort.

It includes describing what should be automated, designing test automation, writing and testing the scripts. Other components of software development, like configuration management, are also included.

This building-block approach to the entire automation test strategy enables software test professionals to approach software testing in a methodical and repeatable fashion.

AUTOMATION LIFE CYCLE

1. DECISION TO AUTOMATE

This phase outlines

- Purpose of taking on a test automation effort.
- Test goals/objectives and strategies need to be defined and test process needs to be documented and communicated to the test team.
- Scope of testing.
- List what is and is not to be automated.

2. TEST TOOL ACQUISITION

This phase outlines

- A test tool proposal (There are several categories of testing tools each with its own purpose).
- Guidelines for the test engineer through the entire test tool evaluation and selection process, starting with confirmation of management support.

3. SETTING THE ENTRANCE AND EXIT CRITERIA.

4. TEST APPROACH

➡ Tools

Tool	Version
Silk Test	7
Internet Explorer	6.0+
MS Excel	Win2000

➡ Environment

➡ Automation Methodology

It involves a multi-step process, comprising the detailed and interrelated activities required to introduce and utilize an automated test tool.

To automate applications framework based architecture is implemented.

A Test Automation Framework is a way to organize the test process, methods, tools and product knowledge for consistent application and repeatable results. The Framework remains Flexible, Reusable, and Maintainable.

OBJECTIVE OF THIS ARCHITECTURE

- To provide an easy to understand excel spreadsheet that test Engineers can use to create test cases.
- To create a modular design for ease of expansion and maintenance.
- The automation scripts are re-usable and easy to maintain.

Automation Framework

The automation framework isolates the application under test from the test scripts by providing a set of functions in a shared function library. The test scriptwriters treat these functions as if they were basic commands of the test tool's programming language. They can thus program the scripts independently of the user interface of the software.

For example, a framework writer might create the function, open file (n). This function opens file n. It might operate by pulling down the file menu, selecting the Open command, copying the file name to the file name field, and selecting the OK button to close the dialog and do the operation. Or the function might be richer than this, adding extensive error handling. The function's definition might change from week to week. The scriptwriter will not be impacted, as long as open file (n) opens file n.

Most functions in this library will be useful in several applications, although not 100% portability. For example, one version of open file () might work for every application that uses the standard File Open dialog but you may need additional versions for programs that customize the dialog. Framework will include several types of functions, from very simple wrappers around simple application or tool functions to very complex scripts that handle an integrated task of performing search and looking for the search term in the documents.)

➡ Test Planning

During this phase, the test team identifies

- Test procedure creation standards and guidelines
- Hardware, software, and network required to support test environment
- Test Data requirements
- A preliminary test schedule
- Performance measure requirements
- A procedure to control test configuration and environment
- Defect-tracking procedure(s) and associated tracking tool(s).

The test plan will define

- Roles and responsibilities
- Project test schedule
- Test planning and design activities
- Test environment preparation
- Test risks and contingencies
- Acceptable level of thoroughness (test acceptance criteria).
- Test plan appendices may include test procedures, naming conventions, test procedure format standards.

➡ Test Design

The test design define

- The number of tests to be performed.
- The ways that testing will be approached (paths, functions).
- The test conditions that need to be exercised.
- Test design standards need to be defined and followed.

After test requirements have been derived using the described techniques, test procedure design begins.

- Test procedure design consists of the definition of logical groups of test procedures and a naming convention for the suite of test procedures.
- With a test procedure definition in place, each test procedure is then identified as either an automated or a manual test.

➡ Technical Environment

The test team needs to ensure that operational support activities have been properly scheduled and must monitor progress of these tasks.

5. TEST EXECUTION

- With the test plan and the test environment being operational, it's time to execute the tests defined for the test program.
- When executing test procedures, the test team must comply with a test procedure execution schedule. (The test procedure execution schedule implements the strategy defined within the test plan)
- Following test execution, test outcome evaluations are performed and test result documentation is prepared.

6. TEST PROGRAM REVIEW AND ASSESSMENT

The performance of the test program is reviewed to determine where changes can be implemented to improve the test program performance on the next project.



SILK TEST

Silk Test is an automated tool for testing the functionality of enterprise applications in any environment.

- It helps to verify application reliability by offering the accuracy, consistency and time-saving benefits.
- Application reliability is enhanced as test coverage and consistency increases.
- It helps to find and fix the bugs early thus preventing the redevelopment cost.

Accurate results analysis helps in optimizing the quality of the application.

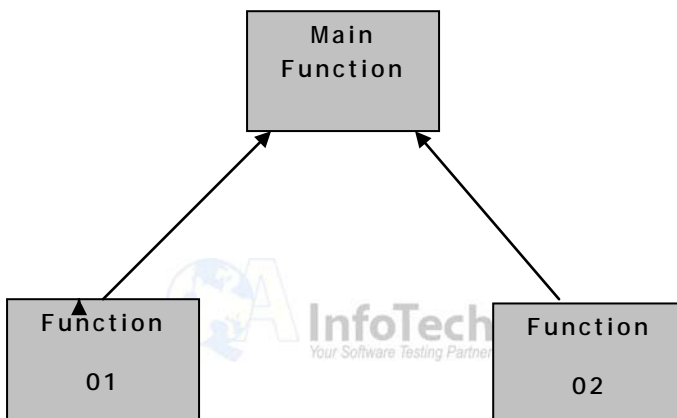


APPENDIX

TEST STUB

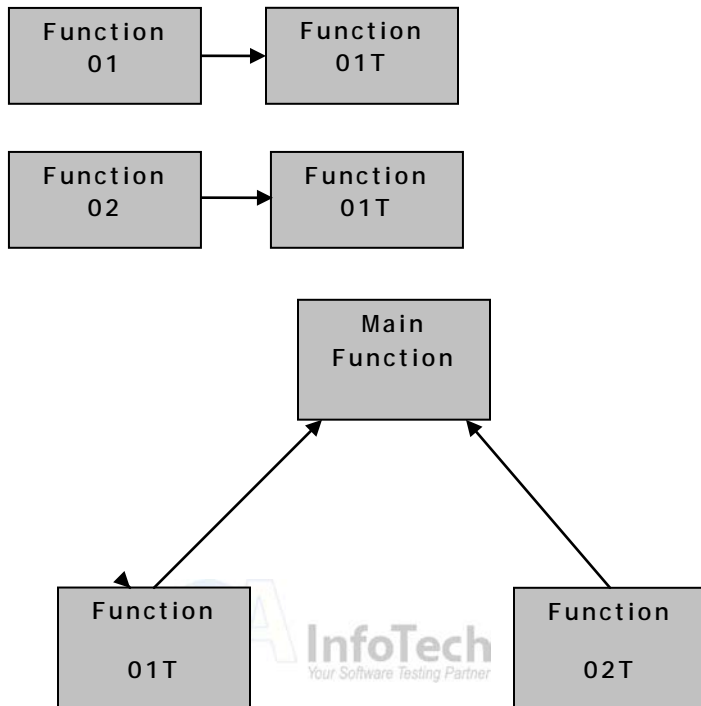
STUB IS A SIMPLIFIED VERSION OF A FUNCTION THAT RETURNS DESIRED INPUTS FOR UNTESTED FUNCTION.

A TEST STUB SENDS TEST DATA UP TO THE MODULE BEING TESTED.

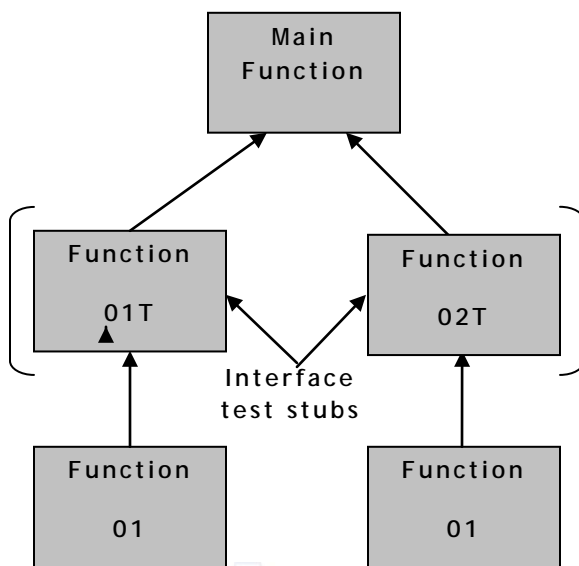


FIRST THE MAIN FUNCTION IS TESTED. AT THIS POINT OF TIME FUNCTION 01 AND 02 ARE NOT YET PREPARED.

SO, TO TEST THE MAIN FUNCTION TWO TEST DATA, I.E. FUNCTION 01T AND FUNCTION 02T ARE PREPARED.



FUNCTION 01T AND FUNCTION 02T, EVENTUALLY COULD BE SMALL PIECES OF CODE CALLED STUBS, WHICH ACTS LIKE AN INTERFACE BETWEEN FUNCTION 01T; FUNCTION 02T AND THE MAIN FUNCTION.



TEST DRIVER

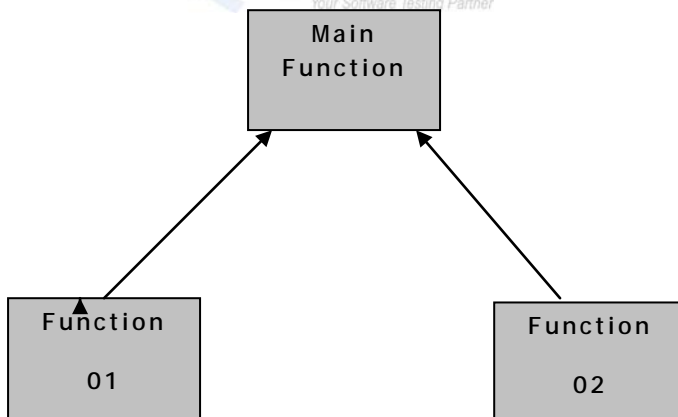
A program or test tool used to execute a test. Also known as a Test Driver.

A piece of code, which encompasses the module(s) under test, and acts as the channel for data into the code.

It typically contains a simple call-return leaf process for each unmatched procedure call in the code. It is used for the dynamic testing of an incomplete module or part of the application.

Driver function passes desired inputs to untested function

A test driver can replace the real software and more efficiently test a low-level module.



First function 01 and 02 are to be tested. At this point of time, main function is not yet prepared.

So, to test function 01 and 02, a driver function (or) module is written, which just calls function 01 and 02 respectively, ensuring that testing is done and finally verifies that both function works well.

