Manual & Automated Testing

Minal Sehgal , Shikha Sharma , Deepa Gupta Mam(Internal Guide)
Amity University,Amity Institute of Information Technology

Abstract

*Software testing is both a discipline and a process. The primary objective is to understand the concept of Software Testing and the Software Testing Tools which companies use to achieve Quality Assurance. In this contribution we will also discuss how **Beatrix Software Factory** & **Three Rivers Technologies** shifted from Manual to Automated testing & analyzing the contribution of Automated Testing Tools to IT Industry.*

***Keywords**-Win Runner, Load Runner, Quick Test Professional, Rational Robot, TAO.*

1. Introduction

Software testing is both a discipline and a process. Though software testing is part of the software development process, it should not be considered part of software development. It is a separate discipline from software development. Software development is the process of coding functionality to meet defined end-user needs. Software testing is an iterative process of both validating functionality, and, even more important, attempting to break the software. The iterative process of software testing consists of Designing tests, Executing tests, Identifying problems, getting problems fixed. The objective of software testing is to find problems and fix them to improve quality. Software testing typically represents 40% of a software development budget.

1.1. Types of Software Testing

Software testing consists of several subcategories, each of which is done for different purposes, and often using different techniques. Software testing categories include: Functionality testing, to verify the proper functionality of the software, including validation of system and business requirements, validation of formulas and calculations, as well as testing of user interface functionality. Forced error testing, or attempting to break and fix the software during testing so that customers do not break it in production. Compatibility testing, to ensure that software is compatible with various hardware platforms, operating systems, other software packages, and even previous releases of the same software. Performance testing, to see how well software performs in terms of the speed of computations and responsiveness to the end-user. Stress testing, to see how the system performs under extreme conditions, such as a very large number of simultaneous users. Usability testing, to ensure that the software is easy and intuitive to use.

1.2. Methods of Software Testing

There are two basic methods of performing software testing: Manual Software Testing -As the name would imply, manual software testing is the process of an individual or individuals manually testing software. Automated Software Testing-Automated software testing is the process of creating test scripts that can then be run automatically, repetitively, and through much iteration. Achieving the Right Blend of Software Testing An effective software testing process is typically a mix of test types, executed through a combination of manual and automated testing. The mix and number of tests is determined by the quality requirements of the application. Each method (automated or manual) is used for what is appropriate. Manual testing is best leveraged for those tests which require spontaneity and creativity, as well a good deal of subjectivity, user interface or usability testing & exploratory/ad hoc testing .While automated testing is best used for tests which are explicit and repetitive, general QA and functionality tests (i.e. does each module do what the requirements say it should? How does the application respond to incorrect inputs?) ,'End to end' scenario tests (simulating a 'real world' use of the software in a production environment), performance, load, and stress testing.

2. Research Methodology

2.1. Purpose of the study

The purpose of the study is to understand the various reasons why companies go in for Automated testing from Manual Testing and what are the factors which are fuelling this conversion and various hurdles which organization has to come across while implementing it.

2.2. Research objectives

Primary objectives are understanding the concept of Software Testing & understanding the Software Testing Tools which companies use to achieve Quality Assurance. Secondary objectives understand how Belatrix Software Factory & Three Rivers Technologies shifted from Manual to Automated testing & analyzing the contribution of Automated Testing Tools to IT Industry.

2.3. Research design

This research is the most commonly used and the basic reason for carrying out descriptive research is to identify the cause of something that is happening. Method of data collection is secondary sources.

2.4 Scope/ relevance of the study

The study will help the individual to understand that the primary purpose of testing is to detect software failures so that defects may be discovered and corrected. Testing cannot establish that a product functions properly under all conditions but can only establish that it does not function properly under specific conditions. The scope of software testing often includes examination of code as well as execution of that code in various environments and conditions as well as examining the aspects of code: does it do what it is supposed to do and do what it needs to do. Instruments of data collection are Books & Internet.

2.5 Limitations

Since only secondary data has been used and not the primary data so we have to rely on the information which is being provided by these sources which sometimes does not give accurate and reliable information.

3. Concepts of Testing

3.1. Definition of Testing

Software testing is performed to verify that the completed software package functions according to the expectations defined by the requirements/specifications. The overall objective to not to find every software bug that exists, but to uncover situations that could negatively impact the customer, usability and/or maintainability.

3.2. History

The separation of debugging from testing was initially introduced by Glenford J. Myers in 1979. Although his attention was on breakage testing ("a successful test is one that finds a bug") it illustrated the desire of the software engineering community to separate fundamental development activities, such as debugging, from that of verification. Dave Gelperin and William C. Hetzel classified in 1988 the phases and goals in software testing in the following stages:-

- Until 1956 - Debugging oriented
- 1957–1978-Demonstration oriented
- 1979–1982 - Destruction oriented
- 1983–1987 - Evaluation oriented
- 1988–2000 - Prevention oriented

3.3. Successful Testing Strategies

Below are some of the tips which every tester should keep in mind before Testing any application:-

3.3.1. Learn to analyze our test results thoroughly. Do not ignore the test result. The final test result may be 'pass' or 'fail' but troubleshooting the root cause of 'fail' will lead us to the solution of the problem. Testers will be respected if they not only log the bugs but also provide solutions.

3.3.2. Learn to maximize the test coverage every time we test any application.

3.3.3. To ensure maximum test coverage break our application under test (AUT) into smaller functional modules.

3.3.4. Make our test cases available to developers prior to coding.

3.3.5. Keep developers away from test environment. This is required step to detect any configuration changes missing in release or deployment document. Sometimes developers do some system or application configuration changes but forget to mention those in deployment steps. If developers don't have access to testing environment they will not do any such changes accidentally on test environment and these missing things can be captured at the right place.

3.3.6. It's a good practice to involve tester's right from software requirement and design phase. These way testers can get knowledge of application dependability resulting in detailed test coverage.

3.3.7. If possible identify and group our test cases for regression testing**.** This will ensure quick and effective manual regression testing.

3.4. Advantages of Testing

3.4.1. To improve quality

As computers and software are used in critical applications, the outcome of a bug can be severe. Bugs can cause huge losses. Bugs in critical systems have caused airplane crashes, allowed space shuttle missions to go awry, halted trading on the stock market, and worse. For Example: The so-called year 2000 (Y2K) bug has given birth to a cottage industry of consultants and programming tools dedicated to making sure the modern world doesn't come to a screeching halt on the first day of the next century. Quality means the conformance to the specified design requirement. Being correct, the minimum requirement of quality, means performing as required under specified circumstances.

3.4.2. For Verification & Validation (V&V) Another important purpose of testing is verification and validation (V&V). It is heavily used as a tool in the V&V process. Testers can make claims based on interpretations of the testing results, which either the product works under certain situations, or it does not work.

3.4.3. For reliability estimation

Software reliability has important relations with many aspects of software, including the structure, and the amount of testing it has been subjected to. Testing is required to make sure that Application developed is in accordance with Client and have implemented with all the requirements from the client. Testing cannot establish that a product functions properly under all conditions but can only establish that it does not function properly under specific conditions

3.5. Scope of Testing

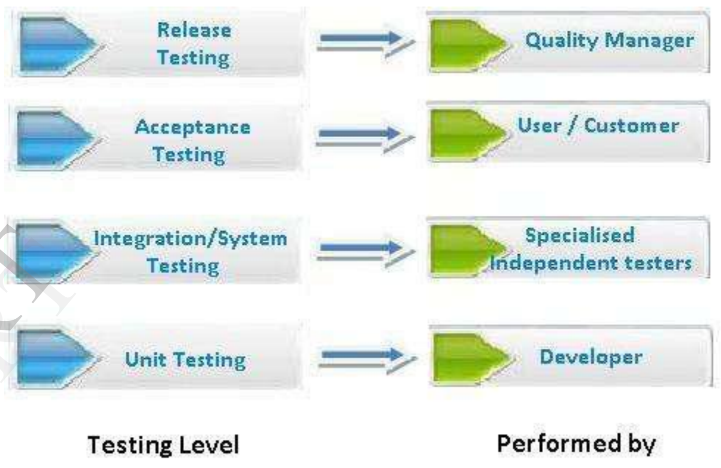
The purpose of software testing is to assess and evaluate the quality of work performed at each step of the software development process. Although it sometimes seems that way, the purpose of testing is NOT to use up all the remaining budget or schedule resources at the end of a development effort. The goal of testing is to ensure that the software performs as intended, and to improve software quality, reliability and maintainability. Testing software is operating the software under controlled conditions, to (1) verify that it behaves "as specified"; (2) to detect errors, and (3) to validate that what has been specified is what the user actually wanted.

Verification is the checking or testing of items, including software, for conformance and consistency by evaluating the results against pre-specified requirements.

Error Detection: Testing should intentionally attempt to make things go wrong to determine if things happen when they shouldn't or things don't happen when they should.

Validation looks at the system correctness – i.e. is the process of checking that what has been specified is what the user actually wanted.

Testing must be done by different persons at different levels. Different purposes are addressed at the different levels of testing. Factors which decide who will perform testing include the size and context of the system, the risks, the development methodology used, the skill and experience of the developers. Below Figure shows persons involved at different levels of software testing.



3.6 Testing Limitations

3.6.1. It can only identify the known issues or errors. It gives no idea about defects still uncovered. Testing cannot guarantee that the system under test is error free.

3.6.2. Testing provides no help when we have to make a decision to either "release the product with errors for meeting the deadline" or to "release the product late compromising the deadline".

3.6.3. Testing cannot establish that a product functions properly under all conditions but can only establish that it does not function properly under specific conditions.

3.6.4. Software testing does not help in finding root causes which resulted in injection of defects in the first place.

4. Methods of Software Testing

4.1. Manual Testing

Software development is a complex process and involves a series of tasks to be carried out including code testing. Testing can be carried out through automated or Manual Testing Method. Testing is a specialized skill or domain; it can be carried out by only seasoned professionals who are well versed with various testing methods and tools. Numerous automated testing tools and methods are available for testers; however, Manual Testing method has its own importance. It is the process of manually testing the software functionality for errors in programming. In this method of testing a tester plays the role of an end user and works on all the options provided by the application to ensure correct behaviour.

4.1.1. Definition of Manual Testing is a process where in a tester often follows a written test plan that leads them through a set of important test cases. A test case in software testing is a set of conditions under which a tester will determine whether an software application is working correctly or not. In order to fully test that all the requirements of an application are met, there must be at least two test cases for each requirement: one positive test and one negative test.

4.1.2 Advantages of Manual Testing

Good for all projects: Manual testing can be use for both small and big projects. We can easily add and remove the test cases according to project movements.
Cost Effective: It is very cost effective. It is very cheaper to manage.
Easy to Understand: Fresh tester can understand very easily the process of manual testing.
Reliable: Manual testing is more reliable than automation testing (in many cases automated not cover all cases).
It allows the tester to perform more ad-hoc (random testing). More bugs are found via random testing than via automation. Manual testing is not related with any programming languages.

4.1.3 Drawbacks of Manual Testing

- Time Consuming: Manual testing can be very time consuming as everything has to be done manually.
- It does not have concept of re-usability.
- More human involvement.
- Each time there is a new build, the tester must re-run all required tests - which after a while would become very dull and tiresome.
- In manual testing, the concept of repeatability not so accurate

4.2. Automated Testing

The principle of automated testing is that there is a program (which could be a job stream) that runs the program being tested, feeding it the proper input, and checking the output against the output that was expected. Once the test suite is written, no human intervention is needed, either to run the program or to look to see if it worked; the test suite does all that, and somehow indicates whether the program's output was as expected. We, for instance, have over two hundred test suites, all of which can be run overnight by executing one job stream submission command; after they run, another command can show which test suites succeeded and which failed.

4.2.1 Definition of Automated Testing: Test automation is the use of software to control the execution of tests, the comparison of actual outcomes to predicted outcomes, the setting up of test preconditions, and other test control and test reporting functions. Commonly, test automation involves automating a manual process already in place that uses a formalized testing process.
The following types of testing can be automated:-
Functional-testing that operations perform as expected.
Regression-testing that the behaviour of the system has not changed.
Stress-determining the absolute capacities of the application and operational infrastructure.
Performance-providing assurance that the performance of the system will be adequate for both batch runs and online transactions in relation to business projections and requirements.
Load-determining the points at which the capacity and performance of the system become degraded to the situation that hardware or software upgrades would be required.

4.2.3. Advantages of Automated Testing

a) Reliable: Tests perform precisely the same operations each time they are run, thereby eliminating human error
b) Repeatable: You can test how the software reacts under repeated execution of the same operations.
c) Comprehensive: You can build a suite of tests that covers every feature in your application.
d) Reusable: You can reuse tests on different versions of an application, even if the user interfaces changes.
e) Better Quality Software: Because you can run more tests in less time with fewer resources

f) Fast: Automated Tools run tests significantly faster than human users.
g) Cost Reduction: As the number of resources for regression test are reduced.

4.2.4. Disadvantages of Automated Testing
Though the automation testing has many advantages, it has its own disadvantages too. Some of the disadvantages are:

- Proficiency is required to write the automation test scripts.
- Debugging the test script is major issue. If any error is present in the test script, sometimes it may lead to deadly consequences.
- Test maintenance is costly in case of playback methods. Even though a minor change occurs in the GUI, the test script has to be rerecorded or replaced by a new test script.
- Maintenance of test data files is difficult, if the test script tests more screens.

5. Life Cycle of Testing

Software testing has its own life cycle that meets every stage of the SDLC. The software testing life cycle diagram can help one visualize the various software testing life cycle phases. They are:

1) Requirement Stage
2) Test Planning
3) Test Analysis
4) Test Design
5) Test Verification and Construction
6) Test Execution
7) Result Analysis
8) Bug Tracking
9) Reporting and Rework
10) Final Testing and Implementation
11) Post Implementation

1) Requirement Stage: This is the initial stage of the life cycle process in which the developers take part in analyzing the requirements for designing a product. Testers can also involve themselves as they can think from the users' point of view which the developers may not. Thus a panel of developers, testers and users can be formed. Formal meetings of the panel can be held in order to document the requirements discussed which can be further used as software requirements specifications or SRS.

2) Test Planning: Test planning is predetermining a plan well in advance to reduce further risks. Without a good plan, no work can lead to success be it software-related or routine work. A test plan document plays an important role in achieving a process-oriented approach. Once the requirements of the project are confirmed, a test plan is documented. The test plan structure is as follows:

Introduction: This describes the objective of the test plan.
Test Items: The items that are referred to prepare this document will be listed here such as SRS, project plan.
Features to be tested: This describes the coverage area of the test plan, i.e. the list of features that are to be tested that are based on the implicit and explicit requirements from the customer.
Features not to be tested: The incorporated or comprised features that can be skipped from the testing phase are listed here. Features that are out of scope of testing, like incomplete modules or those on low severity e.g. GUI features that don't hamper the further process can be included in the list.
Approach: This is the test strategy that should be appropriate to the level of the plan. It should be in acceptance with the higher and lower levels of the plan.
Item pass/fail criteria: Related to the show stopper issue. The criterion which is used has to explain which test item has passed or failed.
Suspension criteria and resumption requirements: The suspension criterion specifies the criterion that is to be used to suspend all or a portion of the testing activities, whereas resumption criterion specifies when testing can resume with the suspended portion.
Test deliverable: This includes a list of documents, reports, charts that are required to be presented to the stakeholders on a regular basis during testing and when testing is completed.
Testing tasks: This stage is needed to avoid confusion whether the defects should be reported for future function. This also helps users and testers to avoid incomplete functions and prevent waste of resources.
Environmental needs: The special requirements of that test plan depending on the environment in which that application has to be designed are listed here.
Responsibilities: This phase assigns responsibilities to the person who can be held responsible in case of a risk.
Staffing and training needs: Training on the application/system and training on the testing tools to be used needs to be given to the staff members who are responsible for the application.
Risks and contingencies: This emphasizes on the probable risks and various events that can occur and what can be done in such situation.

Approval: This decides who can approve the process as complete and allow the project to proceed to the next level that depends on the level of the plan.

3) Test Analysis

Once the test plan documentation is done, the next stage is to analyze what types of software testing should be carried out at the various stages of SDLC.

4) Test Design

Test design is done based on the requirements of the project documented in the SRS. This phase decides whether manual or automated testing is to be done. In automation testing, different paths for testing are to be identified first and writing of scripts has to be done if required. There originates a need for an end to end checklist that covers all the features of the project.

5) Test Verification and Construction

In this phase test plans, the test design and automated script tests are completed. Stress and performance testing plans are also completed at this stage. When the development team is done with a unit of code, the testing team is required to help them in testing that unit and reporting of the bug if found. Integration testing and bug reporting is done in this phase of the software testing life cycle.

6) Test Execution

Planning and execution of various test cases is done in this phase. Once the unit testing is completed, the functionality of the tests is done in this phase. At first, top level testing is done to find out top level failures and bugs are reported immediately to the development team to get the required workaround. Test reports have to be documented properly and the bugs have to be reported to the development team.

7) Result Analysis

Once the bug is fixed by the development team, i.e. after the successful execution of the test case, the testing team has to retest it to compare the expected values with the actual values, and declare the result as pass/fail.

8) Bug Tracking

This is one of the important stages as the Defect Profile Document (DPD) has to be updated for letting the developers know about the defect. Defect Profile Document contains the following:

Defect Id: Unique identification of the Defect.
Test Case Id: Test case identification for that defect.
Description: Detailed description of the bug.
Summary: This field contains some keyword information about the bug, which can help in minimizing the number of records to be searched.
Defect Submitted By: Name of the tester who detected/reported the bug.
Date of Submission: Date at which the bug was detected and reported.
Build No.: Number of test runs required.
Version No.: The version information of the software application in which the bug was detected and fixed.
Assigned To: Name of the developer who is supposed to fix the bug.
Severity: Degree of severity of the defect.
Priority: Priority of fixing the bug.
Status: This field displays current status of the bug.
The contents of a bug well explain all the above mentioned things.

9) Reporting and Rework

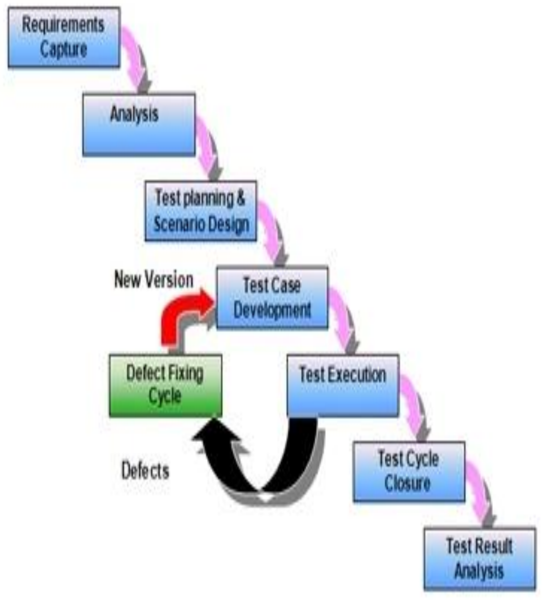
Testing is an iterative process. The bug once reported and as the development team fixes the bug, it has to undergo the testing process again to assure that the bug found is resolved. Regression testing has to be done. Once the Quality Analyst assures that the product is ready, the software is released for production. Before release, the software has to undergo one more round of top level testing. Thus testing is an ongoing process.

8) Final Testing and Implementation

This phase focuses on the remaining levels of testing, such as acceptance, load, stress, performance and recovery testing. The application needs to be verified under specified conditions with respect to the SRS. Various documents are updated and different matrices for testing are completed at this stage of the software testing life cycle.

9) Post Implementation

Once the tests are evaluated, the recording of errors that occurred during various levels of the software testing life cycle, is done. Creating plans for improvement and enhancement is an ongoing process. This helps to prevent similar problems from occurring in the future projects. In short, planning for improvement of the testing process for future applications is done in this phase.



6. Testing Tools

Automation testing of the project is done through the Automation Testing Tools. There are many Automation Tools in use developed by different companies. In that mainly, a) Mercury Interactive b) IBM c) SAP. Mercury Interactive is developed by Mercury International Corporation
The main Tools from this are:-Win Runner, Load Runner, and Quick test professional. The Tools from IBM are: - Rational Robot **.**The Tools from SAP are:- TAO

6.1. Win Runner: Win Runner is the most used Automated Software Testing Tool. Main Features of Win Runner are:

- Developed by Mercury Interactive
- Functionality testing tool
- Supports C/s and web technologies such as (VB, VC++, D2K, Java, HTML, Power Builder, Delphe, Cibell (ERP))
- To Support .net, xml, SAP, PeopleSoft, Oracle applications, Multimedia we can use QTP.
- Win runner run on Windows only.
- Xrunner run only UNIX and Linux.
- Tool developed in C on VC++ environment.
- To automate our manual test win runner used TSL (Test Script language like c)

The main Testing Process in Win Runner is:-

1) **Learning**- Recognazation of objects and windows in our application by win runner is called learning. Win runner 7.0 follows Auto learning.
2) **Recording**- Win runner records over manual business operation in TSL
3) **Edit Script**- Depends on corresponding manual test, test engineer inserts check points in to that record script.
4) **Run Script**- During test script execution, win runner compare tester given expected values and application actual values and returns results.
5) **Analyze Results**- Tester analyzes the tool given results to concentrate on defect tracking if required.

6.2. Load Runner

Load Runner is an industry-leading performance and load testing product by Hewlett-Packard (since it acquired Mercury Interactive in November 2006) for examining system behaviour and performance, while generating actual load. Load Runner can emulate hundreds or thousands of concurrent users to put the application through the rigors of real-life user loads, while collecting information from key infrastructure components (Web servers, database servers etc). The results can then be analysed in detail, to explore the reasons for particular behaviour. Consider the client-side application for an automated teller machine (ATM). Although each client is connected to a server, in total there may be hundreds of ATMs open to the public. There may be some peak times such as 10 a.m. Monday, the start of the work week during which the load is much higher than normal. In order to test such situations, it is not practical to have a test bed of hundreds of ATMs. So, given an ATM simulator and a computer system with Load Runner, one can simulate a large number of users accessing the server simultaneously.

6.2.1. The Load Runner Testing Process

There are five steps in Load Runner Testing Process:
Step I: Planning the Test
Successful load testing requires that we develop a thorough test plan. A clearly defined test plan will ensure that the Load Runner scenarios that we develop will accomplish our load testing objectives.
Step II: Creating the Vuser scripts
Vusers emulate human users interacting with our client/server system. A Vuser script contains the actions that each virtual user performs during scenario execution.
In each Vuser script we determine the tasks that will be: Performed by each Vuser, performed

simultaneously by multiple Vusers, measured as transactions

Step III: Creating the Scenario

A scenario describes the events that occur during a client/server testing session. A scenario includes a list of machines that "host" Vusers; a list of Vuser scripts that the Vusers run; and a list of Vusers that run during the scenario. We create scenarios using the LoadRunner Controller.

Creating the List of Hosts to Run Vusers: For each scenario, we create a list of hosts--machines configured to execute Vuser scripts.

Creating the List of Vuser Scripts: For each scenario, we create a list of scripts that Vusers run during scenario execution.

Creating the Vusers: To each Vuser in a scenario, we assign a Vuser script and a host to run the script.

Step IV: Running the Scenario: We emulate user load on the server by instructing multiple Vusers to perform tasks simultaneously. We can set the level of load by increasing and decreasing the number of Vusers that perform tasks at the same time. Before we run a scenario, we set the scenario configuration. This determines how all the hosts and Vusers behave when we run the scenario. For each scenario, we create a list of scripts that Vusers run during scenario execution. We can run the entire scenario, individual Vusers, or groups of Vusers (Vuser Groups). While a scenario runs, Load Runner measures and records the transactions that we defined in each Vuser script.

Step V: Analyzing Test Results

During scenario execution, Load Runner records the performance of the client/server system under different loads. We use Load Runner's graphs and reports to analyze the server's performance.

6.3. QTP

Quick Test Professional (QTP) is an automated functional Graphical User Interface (GUI) testing tool that allows the automation of user actions on a web or client based computer application.

It is primarily used for functional regression test automation. QTP uses a scripting language built on top of VBScript to specify the test procedure, and to manipulate the objects and controls of the application under test.

6.3.1. Testing Process

QTP (Quick Test Professional) lets us create tests and business components by recording operations as we perform them in our application.

1) **First** step is **Planning-** Before starting to build a test; we should plan it and prepare the required infrastructure. For example, determine the functionality we want to test, short tests that check specific functions of the application or complete site. Decide how we want to organize our object repositories.

2) **Second** step in QTP is **Creating Tests or Components**

We can create a test or component by

a) Either recording a session on our application or Web site.

As we navigate through the application or site, Quick Test graphically displays each step we perform as a row in the Keyword View. The Documentation column of the Keyword View also displays a description of each step in easy-to-understand sentences. A step is something that causes or makes a change in our site or application, such as clicking a link or image, or submitting a data form.

OR b) Build an object repository and use these objects to add steps manually in the Keyword View or Expert View. We can then modify our test or component with special testing options and/or with programming statements.

3) **Third** step is **inserting checkpoints** into our test or component. A checkpoint is a verification point that compares a recent value for a specified property with the expected value for that property. This enables us to identify whether the Web site or application is functioning correctly.

4) **Fourth** step is **Broaden the scope of our test** or component by replacing fixed values with parameters. To check how our application performs the same operations with different data we can parameterize our test or component. When we parameterize our test or component, Quick Test substitutes the fixed values in our test or component with parameters

5) **Fifth** step is **running the test** After creating test or component, we run it. Run test or component to check the site or application. When we run the test or component, Quick Test connects to our Web site or application and performs each operation in a test or component, checking any text strings, objects, or tables we specified. If we parameterized the test with Data Table parameters, Quick Test repeats the test (or specific actions in your test) for each set of data values we defined. Run the test or component to debug it.

6) **Sixth** step is **analyzing the results** After running the test or component, we can view the results of the run in the Test Results window. We can view a summary of the results as well as a detailed report.

6.4. Rational Robot

Rational Robot is a test automation tool for functional testing of client/server applications.

- Provides a general-purpose test automation tool for QA teams for functional testing of client/server applications.
- Lowers learning curve for testers discovering the value of test automation processes.
- Enables test-automation engineers to detect defects by extending test scripts and to define test cases.
- Provides test cases for common objects and specialized test cases to development environment objects.
- Includes built-in test management, integrates with IBM Rational Unified Process tools. Aids in defect tracking, change management and requirements traceability.
- Supports multiple UI technologies. Operating systems supported: Windows

6.4.1.Features and benefits

IBM Rational® Robot v2003 automates regression, functional and configuration testing for e-commerce, client/server and ERP applications. It's used to test applications based upon a wide variety of user interface technologies, and is integrated with the IBM Rational® TestManager® solution to provide desktop management support for all testing activities.

Simplifies configuration testing - Rational Robot can be used to distribute functional testing among many machines, each one configured differently. The same functional tests can be run simultaneously, shortening the time to identify problems with specific configurations.

Tests many types of applications - Rational Robot supports a wide range of environments and languages, including HTML and DHTML, Java™, VS.NET, Microsoft Visual Basic and Visual C++, Oracle Developer/2000, PeopleSoft, Sybase PowerBuilder and Borland Delphi.

Tests custom controls and objects - Rational Robot allows you to test each application component under varying conditions and provides test cases for menus, lists, alphanumeric characters, bitmaps and many more objects.

Provides an integrated programming environment - Rational Robot generates test scripts in SQABasic, an integrated MDI scripting environment that allows you to view and edit your test script while you are recording.

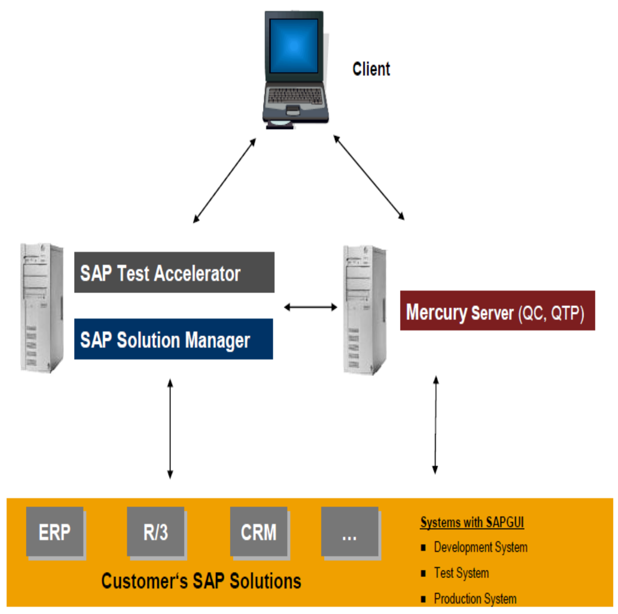
Helps you analyze problems quickly - Rational Robot automatically logs test results into the integrated Rational Repository, and color codes them for quick visual analysis. By double-clicking on an entry, you are brought directly to the corresponding line in your test script, thereby ensuring fast analysis and correction of test script errors.

Enables reuse - Rational Robot ensures that the same test script, without any modification, can be reused to test an application running on Microsoft Windows XP, Windows ME, Windows 2003, Windows 2000, Windows 98 or Windows NT.

6.5. TAO

SAP Test Acceleration and Optimization to generate automatic tests during regression testing of SAP solutions, quickly. SAP Test Acceleration and Optimization creates components from the screens of a transaction and parameterizes them. These tests are for a single transaction and can be combined into a scenario test. SAP Test Acceleration and Optimization also supports maintenance of components and tests by integration into the Business Process Change Analyzer in SAP Solution Manager. We can only test business processes of SAP ABAP ERP products (SAP GUI only).It breaks down a test into components, using SAP Test Acceleration and Optimization, and SAP Quality Center application by HP, and do the following: Drag these components into tests, regenerate the components or tests whenever there is a change in screen, data, or business process.

.



6.5.1Integration

SAP Test Acceleration and Optimization interacts with the SAP Solution Manager System, managed systems and SAP Quality Center. SAP Test Acceleration and Optimization uses the SAP Solution Manager repository to store information, such as results of analysis. You use the SAP Quality Center to compose the tests using the components generated by SAP Test Acceleration and Optimization.

6.5.2.Features
SAP Test Acceleration and Optimization with SAP Quality Center has the following features:
Test creation: It record and execute tests quickly. You can use the default components available with SAP Test Acceleration and Optimization to build tests. The subject matter expert of a product has to spend less time with quality professionals, to explain the business process.
Regeneration of components affected: It regenerates any number of components whenever there is a change to the business process.
Test Consolidation: It consolidates a test into a component and use it in a scenario test, to speed up test execution.
Test Execution: It executes the tests available in SAP Quality Center. You can also create or update technical bills of material for associated SAP Solution Manager Items.

6.5.3.Release Notes SAP Test Acceleration and Optimization 2.0 SP04
The SAP Test Acceleration and Optimization™ (SAP TAO ™) software streamlines the creation and maintenance of ERP business process testing. It helps Quality Analyst specialists to break down application into components. Assemble test cases through a simple interface using drag and drop components in Quality Center. Test script can be parameterized for flexible reuse. Maintained easily and inexpensively, even when screens, flows, or service packs change.
It provides an overview of new and changed functions in SAP Test Acceleration and Optimization 2.0 SP04.

6.5.4.Features of SAP TAO ™ 1.0 version released in 2007:
Inspect: Captures the data in a screen or transaction and determines its validity. It enables you to create and maintain a list of transactions and screens.
Import/Export: Primarily runs in background mode to export and import data from the SAP Test Acceleration and Optimization™ client to the SAP Quality Center.
Consolidator: Gathers all the objects and data in an SAP Quality Center test script and creates a single component.
Connect: Connection settings for SAP and Quality Center

6.5.5.Enhancements in SAP TAO ™ 2.0 versions released in 2009:
PFA (Process flow Analyzer): It records user interactions and the sequence of screens to execute a business process, in the SAP TAO ™ repository. It automates inspection and creation of the test components and a parameterized draft transition test case. It automatically creates the data table spreadsheet with the DT columns and values used during the recording process.
Repository: The SAP Test Acceleration and Optimization™ repository is a part of the SAP Solution Manger system and is used to store: User interaction and sequence of the screens in a business process. Information specific to SAP Test Acceleration and Optimization™ that cannot be retrieved by other tools.
Change Analyzer: It helps you to analyze the impact of changes due to upgrades, SAP patches or Custom development on a test, components or consolidated component.

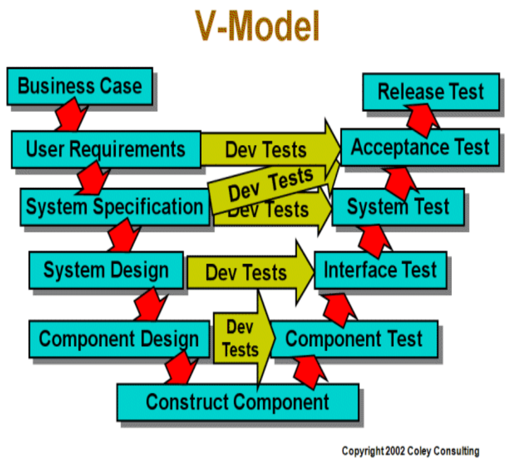
7. Testing Models

7.1. V-Model

As you get involved in the development of a new system a vast number of software tests appear to be required to prove the system. Below we describe the various types of software testing and how they fit into the V-Model. The main software testing types are:

- Component.
- Interface.
- System.
- Acceptance.
- Release

To put these types of software testing in context requires an outline of the development process.



Development Process
The development process for a system is traditionally as a Waterfall Model where each step follows the next, as if in a waterfall.

Business Case
The first step in development is a business investigation followed by a "Business Case" produced by the customer for a system. This outlines a new system, or change to an existing system, which it is anticipated will deliver business benefits, and outlines the costs expected when developing and running the system.

User Requirements
The next broad step is to define a set of "User Requirements", which is a statement by the customer of what the system shall achieve in order to meet the need

System Specification
"Requirements" are then passed to developers, who produce a "System Specification". This changes the focus from what the system shall achieve to how it will achieve it by defining it in computer terms, taking into account both functional and non-functional requirements.

System Design
Other developers produce a "System Design" from the "System Specification". This takes the features required and maps them to various components, and defines the relationships between these components.

Component Design
Each component then has a "Component Design", which describes in detail exactly how it will perform its piece of processing.

Component Construction
Finally each component is built, and then is ready for the test process.
The level of test is the primary focus of a system and derives from the way a software system is designed and built up. Conventionally this is known as the "V-Model", which maps the types of test to each stage of development.

Component Test
Starting from the bottom the first test level is "Component Test", sometimes called Unit Testing. It involves checking that each feature specified in the "Component Design" has been implemented in the component.

Interface Test
As the components are constructed and tested they are then linked together to check if they work with each other. It is a fact that two components that have passed all their tests, when connected to each other produce one new component full of faults. These tests can be done by specialists, or by the developers

System Test
Once the entire system has been built then it has to be tested against the "System Specification" to check if it delivers the features required. It is still developer focussed, although specialist developers known as systems testers are normally employed to do it.

Acceptance Test
Acceptance Testing checks the system against the "User Requirements". It is similar to systems testing in that the whole system is checked but the important difference is the change in focus:

- Systems testing checks that the system that was specified has been delivered.
- Acceptance Testing checks that the system delivers what was requested.

The customer and not the developer should always do acceptance testing.

Release Test
Even if a system meets all its requirements, there is still a case to be answered that it will benefit the business. The linking of "Business Case" to Release Testing is looser than the others, but is still important. Release Testing is about seeing if the new or changed system will work in the existing business environment. Mainly this means the technical environment, and checks concerns such as:

- Does it affect any other systems running on the hardware?
- Is it compatible with other systems?
- Does it have acceptable performance under load?

Regression Tests
With modern systems one person's system, becomes somebody else's component. It follows that all the above types of testing could be repeated at many levels in order to deliver the final value to the business. In fact every time a system is altered.

7.2 Butterfly Model
The **Butterfly Model** focuses on verification and validation of software products and is therefore a good fit for software testing tasks that are incorporated into the V-model of software development. This model provides a graphic picture of the complexity of test tasks using the outline of a butterfly. The areas occupied by the wings and body are approximately related to the level of effort afforded to each of the activities included in the model. The model establishes three general areas of test activities that are illustrated by the butterfly's graphic outline. They are:

Test Analysis (butterfly's left wing): The left wing of the butterfly represents test analysis – the investigation, quantization, and/or re-expression of a facet of the software to be tested.

Test Design (right wing) : The right wing of the butterfly represents the act of designing and implementing the test cases needed to verify the design artefact as replicated in the implementation.
Test Execution (butterfly's body): Test execution includes only the formal running of the designed tests. Informal test execution is a normal part of test design, and in fact is also a normal part of software design and development.

7.3. Waterfall Model

The waterfall model derives its name due to the cascading effect from one phase to the other . In this model each phase well defined starting and ending point, with identifiable deliveries to the next phase. This model is sometimes referred to as the linear sequential model or the software life cycle.

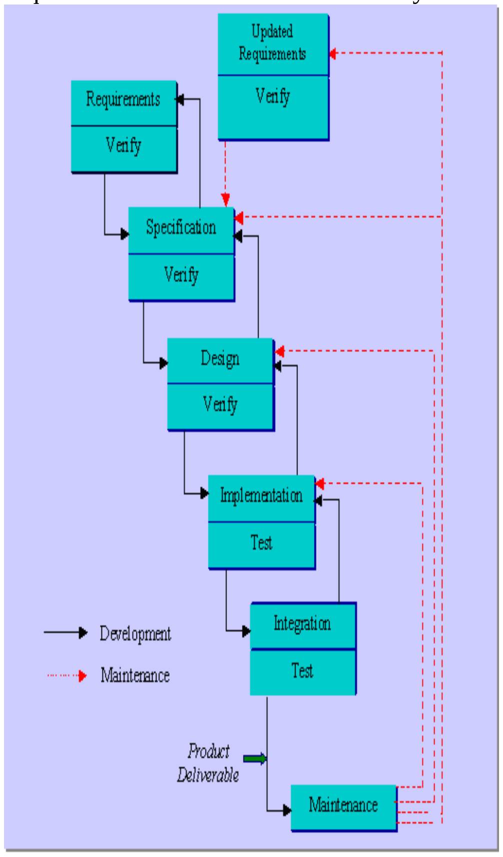


Fig 1.2 - Schematic illustrating the Waterfall Model

The model consists of six distinct stages, namely:
1. In the requirements analysis phase
(a) The problem is specified along with the desired service objectives (goals)
(b) The constraints are identified
2. In the specification phase the system specification is produced from the detailed definitions of (a) and (b) above.

This document should clearly define the product function.

Note that in some text, the requirements analysis and specifications phases are combined and represented as a single phase.
3. In the system and software design phase, the system specifications are translated into a software representation. The software engineer at this stage is concerned with:
a) Data structure
b) Software architecture
c) Algorithmic detail and
d) Interface representations
The hardware requirements are also determined at this stage along with a picture of the overall system architecture. By the end of this stage should the software engineer should be able to identify the relationship between the hardware, software and the associated interfaces. Any faults in the specification should ideally not be passed 'down stream'
4. In the implementation and testing phase stage the designs are translated into the software domain
a) Detailed documentation from the design phase can significantly reduce the coding effort.
b) Testing at this stage focuses on making sure that any errors are identified and that the software meets its required specification.
5. In the integration and system testing phase all the program units are integrated and tested to ensure that the complete system meets the software requirements. After this stage the software is delivered to the customer [Deliverable- The software product is delivered to the client for acceptance testing.]
6. The maintenance phase the usually the longest stage of the software. In this phase the software is updated to:
a) Meet the changing customer needs
b) Adapted to accommodate changes in the external environment
c) Correct errors and oversights previously undetected in the testing phases
d) Enhancing the efficiency of the software
Observe that feed back loops allow for corrections to be incorporated into the model. For example a problem/update in the design phase requires a 'revisit' to the specifications phase. When changes are made at any phase, the relevant documentation should be updated to reflect that change.

8. Analysis

8.1. Case Study

8.1.1. From Manual Testing to Automated Testing Belatrix Software Factory

The Company. This Client is a leading supplier of Enterprise solutions, desktop software, scientific databases, and professional services for biotechnology, drug discovery and chemical research, including software, databases, and web sites which enable customers to create, analyze and communicate chemical, biological, and scientific information more effectively.

The Products. This Client's products are used primarily in the pharmaceutical, biotechnology and chemical industries. They are also used in higher and academic education and government research. The company's principal software is the de facto standard and primary communication tool on the chemist's desktop. The enterprise version of this solution enables information research organizations to deploy application and information solutions using Internet, intranet, and extranet technologies. These solutions are now in use by companies such as Abbot Laboratories, Johnson & Johnson, Merck, etc.

The Challenge: With over 1,500 manual test cases, the quality assurance process for the Client's main desktop application was quickly becoming very challenging to maintain. The application was constantly growing (more features were added) and we had to find a way to reduce testing times, increase the amount of test cases and improve the system quality overall without introducing more additional testing time. Based on this experience, the Belatrix Quality Assurance team proposed to automate our testing process. Due to the complexity of the application, we had to decide what the best testing architecture would be. These are the problems that have arisen:
• There are several automation tools available (free and open source), but none of them provides a complete solution. Even the licensed ones can prove to be quite expensive and they confine you in their specific architecture and scripting languages.
• We could spend too much time writing complex test scripts, so reusability was very important.
• We could spend too much time maintaining scripts.
Based on these restrictions, we realized we had a few options:

Select a licensed tool

- **Benefits:** We could start creating scripts faster, since we do not need to develop anything.
- **Risks***:* The licensed tools could not meet our requirements and we may not be able to adapt the tool's behavior due to the fact that we do not have the source code. If the tool turned out to be ineffective, much work would have been wasted as it would have probably been developed on a proprietary technology or language.

Select a specific free and Open Source tool:

- **Benefits**: they are free and we could add more features as needed because we have the source code. They tend to use popular scripting languages, preventing future lock-in.
- **Risks:** we could spend too much time creating scripts and maintaining them and we should also develop several features to cover our needs.

Use and combine several Open Source test automation tools:

- **Benefits:** they have a low cost and we could leverage best-of-breed approaches.
- **Risks:** some tools may not work well with the others. However, this can be mitigated with appropriate prototyping.

The Solution: Along with the client, Belatrix decided that the best option was to follow the best approach and select multiple open source tools combined with a custom framework that allows us to:

- Reduce the time to create scripts.
- Reduce the time to maintain the scripts.
- Combine several automation tools in order to get the benefits of each one of them while circumventing their weaknesses.
- Develop special features that could be reused by all scripts:

✓ Take screenshots
✓ Connect with a data base
✓ Create automated reports

Some of the tools and technologies that were chosen include:

✓ Python as a general scripting language
✓ Open STA
✓ DummyNet
✓ PyWinAuto
✓ Selenium

The Results. Belatrix helped the Client reduce the test processing time by 40%, add hundreds of additional test cases without affecting schedules and keep test cycles under control. Belatrix was also able to help the Client automate the system deployment process, as well as the system installation, installation testing, smoke tests, feature testing (including integration with other tools), performance testing and all the report generation to produce reliable quality metrics.

8.1.2. Automated Testing Case Study

THREE RIVERS TECHNOLOGIES

Background: A Fortune 500 client of has invested significantly in the development of a custom Manufacturing Fulfilment system for their 33 worldwide business units. The system is used to manage the production of manufactured products tailored to their customers' needs. The system was originally developed by another software services provider over the course of several years. It is a traditional client/server application developed in Visual C++ and accesses an Oracle database running in a Unix environment. Their client was receiving weekly updates of the system in the latter phases of development - prior to implementation. They found it virtually impossible to perform the necessary testing and validation of these updates before the next update was received. They also realized that a more consistent approach was needed to properly test each update of the system.

Objectives: Their client wanted to find a way to perform the testing and validation of these updates to ensure the system was meeting all the functional requirements and was going to be reliable upon deployment. Three Rivers Technologies was asked to help develop a way to automatically test the system quickly and efficiently. The objectives of the project were as follows:

- Define a testing strategy that would automate as much testing as possible
- Develop Test Plans that would ensure core functionality would be reliably tested using automated testing tools
- Develop an automated test suite that would give their client the ability to fully regression test the system quickly during the final phases of development as well as after deployment (as enhancements were being added)
- Design and build test scripts for system validation that would be highly reusable and inexpensive to maintain

Methodology/Approach: The first step that Three Rivers Technologies' QA consultants took was to familiarize themselves with the system and its intended functionality. This was accomplished by talking to and working with users of the system, by reviewing requirements documents, and by participating in the existing manual testing activities. The main goal of their early involvement was to gain an overall understanding of the system in order to develop a comprehensive testing strategy that would take maximum advantage of automated testing tools. Once the overall testing strategy had been established the following tasks were performed:

- A Test Plan was developed
- A system testing environment was set up that could be restored to its initial state after running scripts - which allowed for repeated execution of the regression tests
- Automated test scripts were built using Rational's Team Test product
- Libraries of reusable functions were designed and built so that they could be incorporated into the test scripts in order to minimize maintenance efforts
- Automated test scripts that covered 80 percent of the system's functionality were developed over a six month period of time

Results
This project has been very successful. All of the project objectives were met and their client has been extremely happy with the results.

- Their client now has the ability to regression test their Manufacturing Fulfilment system in approximately 4 hours using the test scripts our consultants developed. This level of testing would take several weeks to perform manually and would have an increased possibility of missing defects.
- Three years after initiating this project, all of the original scripts are still being used to test new releases of the system, and only minimal maintenance has been required to keep scripts updated to work with the new releases.
- The automated testing strategy that was envisioned and developed for this system has saved thousands and thousands of hours of manual testing. And because so much of the testing effort has been standardized and automated, testers are able to focus their manual testing efforts for new releases on critical areas of the system that have been enhanced with new functionality.

- Automated testing has reduced the time required to deploy new system functionality, thereby increasing users' overall productivity.
- The investment made in automated testing of this system has also resulted in improved quality. The system was deployed in the field last year and subsequent updates have been released without any major regression errors being found after release.
- The automated test scripts have been leveraged for other types of testing like performance and load testing, environment testing, and integration testing.

Since initiating this project three years ago, their client has gained efficiencies in testing, improved system quality, and realized significant cost savings. As a result, this Fortune 500 client has increased their investment in automated testing and has brought on additional Three Rivers Technologies' QA consultants to design and build automated testing solutions for many of their critical business systems.

9. Findings

[1] One important reason for doing testing is user/production environment will be completely **different from development environment.**
For example, a webpage developer may be using FireFox as browser for doing his webpage development. But the user may be using different browser such as Internet Explorer, Safari, Chrome and Opera. The web page appearing good in FireFox may not appear good in other browsers. So ultimately, user will not be happy even if the developer puts more efforts to develop the webpage. As we know that **Users** satisfaction is more important for growth of any business, testing becomes more important.

[2] Quality can be ensured by testing only. In the competitive market, **only Quality product can exist for long time.**

[3] Prevent Defect Migration:- The majority of errors are usually introduced in the software requirements gathering phase. If the errors are detected early, they can be prevented from migrating to the subsequent development phase. Early detection and debugging of errors leads to huge savings in software development costs.

[4] For Reliability Estimation:-From the user point of view, reliability means how dependable the software product is. In medical diagnosis, an incorrect suggestion to the doctor can result in the loss of lives. Critical software products are thoroughly checked for all aspects of its functionality.

[5] Prove Usability and Operability:- One very important aim of software testing is to prove the software is both usable and operable. Usability testing is where the software is released to a select group of users and their working with the product is observed. All aspects of a user's interaction with the software, like ease of use and where users are facing problems, are recoded and analyzed.

10. Suggestions

[1] Rely more heavily on high volume automation techniques.

[2] Retrain software testers to help them focus more effectively on risk.

[3] Identify all the platforms on which application will be run.

[4] Keep track of bug fixing time taken by development team.

[5] Improve requirements management process – business & functional requirements should be well documented. Involve testing team in requirement gathering phase.

[6] Create a document / list of all possible scenarios before writing test cases. Include it into test planning.

[7] Keep developers away from test environments.

[8] Analyse test results thoroughly. Try to identify root cause from functional perspective.

11. Conclusion

[1] Software testing is an art. Most of the testing methods and practices are not very different from 20 years ago. It is nowhere near maturity, although there are many tools and techniques available to use. Good testing also requires a tester's creativity, experience and intuition, together with proper techniques.

[2]Testing is more than just debugging. Testing is not only used to locate defects and correct them. It is also used in validation, verification process, and reliability measurement.

[3]Testing is expensive. Automation is a good way to cut down cost and time. Testing efficiency and effectiveness is the criteria for coverage-based testing techniques.

[4]Complete testing is infeasible. Complexity is the rootof the problem. At some point, software testing has to be stopped and product has to be shipped. The stopping time can be decided by the trade-off of time and budget. Or if the reliability estimate of the software product meets requirement.

Acknowledgement

Success is a sweet fruit which requires hard work and for that one takes all the encouraging and helping hands of the people. I am indebted and obliged to thank my Research paper guide **Ms. Deepa Gupta** for her proper guidance and helpful hints to ensure my research paper was on the right track. Without her constant support, valuable supervision and encouragement I would have not been able to complete this work on time. I would also like to thank my friends who have been always ready to help me at every step.

Bibliography/ References

Book:

[1] R.S. Pressman & Associates

References:

- Section 1 in [1]
- Section 3 in [2]
- Automated testing case study –three rivers in [3]
- About Load Runner in [5]
- Software testing lifecycle in [4]
- Manual testing in [6]
- About software testing in [7]

REFERENCES

[1]http://www.logigear.com/newsletter-2006/271-introduction-to-software-testing.html

[2]http://www.ece.cmu.edu/~koopman/des_s99/sw_testing/#introduction

[3] http://www.3riverstech.com/atcasestudy.php

[4] http://www.onestoptesting.com/qtp/testing-process.asp

[5]http://www.softwaretestinggenius.com/download/introlr.pdf

[6]http://www.outsourcingdotnetdevelopment.com/manual-testing.html

[7] http://bazman.tripod.com/what_testing.html