Effective Testing: A case study approach for improving test efficiency

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Award of the Degree of

Master of Philosophy
in
Computer Science

by
Abdul Rauf E.M
(Reg. No. 1135008)

Under the Guidance of
Balaji V
Associate Professor

Department of Computer Science

CHRIST UNIVERSITY
BANGALORE, INDIA
March 2012

Approval of Dissertation

Dissertation entitled **"Effective Testing: A case study approach for improving test efficiency"** by Abdul Rauf E.M, Reg. No.1135008 is approved for the award of the degree of Master of Philosophy in Computer Science

Examiners:

1. _____ _____

2. _____ _____

3. _____ _____

Supervisor(s):

_____ _____

Chairman:

_____ _____

Date: _____

(Seal)

Place: Christ University

DECLARATION

I Abdul Rauf E.M hereby declare that the dissertation, titled 'Effective Testing: A case study approach for improving test efficiency' is a record of original research work undertaken by me for the award of the degree of Master of Philosophy in Computer Science. I have completed this study under the supervision of Ms. Balaji V, Associate Professor, Department of Computer Science

I also declare that this dissertation has not been submitted for the award of any degree, diploma, associate ship, fellowship or other title. It has not been sent for any publication or presentation purpose.

Place: Christ University

Date: ……

Signature of the candidate

Abdul Rauf E.M
Reg. No. 1135008
Department of Computer Science
Christ University, Bangalore

CERTIFICATE

This is to certify that the dissertation submitted by Abdul Rauf E.M(Reg. No. 1135008) titled 'Effective testing: A case study approach for improving test efficiency' is a record of research work done by him during the academic year **2011-2012** under my supervision in partial fulfillments for the award of Master of Philosophy in Computer Science.

This dissertation has not been submitted for the award of any degree, diploma, associate ship, fellowship or other title. It has not been sent for any publication or presentation purpose.

Place: Christ University

Date: ……

Signature of the Guide

Balaji V

Associate Professor

Department of Computer science

Christ University, Bangalore

Abstract

The study presented in this thesis investigates the methods for improving the software test efficiency. Test efficiency measures the cost-effectiveness of a test organisation and it is measured by dividing the number of defects found in a test by the effort needed to perform the test. A review of the literature suggests that software test efficiency improvement depends on direct and indirect success factors like test process, test management, test tools, test object delimitation, test case determination, test infrastructure, configuration management, release management etc. This thesis was a case study approach for improving the test efficiency of an existing test setup in a database environment. Most of the thesis work followed an action based research approach by giving importance to the test setup. Work started with an analysis of the initial test environment, identified the issues and improvement areas in existing test setup and given an implementation proposal for the identified problems. Based on the proposal, team implemented the solutions, which lead to a test environment containing number of actions like automation using standard framework, risk based testing, parallel execution, modularization, avoiding code redundancy and proper test management.

The results of the case study suggest that the software products that has multiple releases should seriously consider the test improvement factors like regression environment, risk based testing, light weight test automation etc., in the initial stages of the testing. This will lead to cost savings, quality, flexibility and higher productivity. The investigation further identifies the issues in test management and introduced new method called “test point” method for proper test execution tracking. Based on the implementation results and their discussions, this study presents a new approach and practical guidelines for improving test efficiency of a software test project. IBM has recognised this case study by giving eminence and excellence award for saving one person year of testing effort in their indexing tool test environment.

Table of contents

List of figures

Abbreviation notation and nomenclature

SDLC	Software Development Life Cycle
STLC	Software Test Life Cycle
DB	Data Base
SQL	Structured Query Language
GUI	Graphical User Interface
LDTP	Linux Desktop Testing Project
FIT	Federated Integrated Test
IP	Intellectual Property
TIAT	Test Install Automation Tool
TTT	Test Tracking Tool
TP	Test Point

1. Introduction

1.1 Project background

Even though software development industry spends more than half of its budget on software testing and maintenance related activities; software testing has received little attention in our curricula. This suggests that most software testers are then either self taught or they acquire needed skills on the job perhaps through formal and informal mechanisms used commonly in the industry. Lack of proper attention in acquiring testing skills is resulting in less utilization of test resources and thus results in less test efficiency of organisation. Review of extant literature on software testing lifecycle (STLC) identifies various software testing activities and ways in which these activities can be carried out in conjunction with the software development process. This literature also identifies various skills that software testers need to possess in order to perform activities effectively in a given phase of STLC. Similar to development lifecycle (SDLC), STLC also suggests the phases of analysis, design, implementation, execution, and evaluation in software testing lifecycle. The V - model, which is the most popular testing model, provides a basis for the identification of various testing activities. Based on the V- model, Vijay (2001), Waligora and Coon (1996) suggest the need to conduct testing in parallel with many of the SDLC phases so that testing efforts in later stages can be minimized.

With this case study, we are targeting on how we can improve the test efficiency of database indexing tool and thus to prepare generic guidelines for improving the test efficiency of an organisation. Database indexing tool, which provides users and application programmers a fast, versatile, and quick method of searching full-text documents stored in DB and file systems using SQL queries. The initial test environment of this tool was not fully

automated, that results in lot of manual intervention for executing system test cases and thus results in lot of manpower utilization. This tool has multiple releases and service packs, each service pack is consuming around 100 man days of system testing due to the execution of regression scenarios in test cycle. This case study was targeted to come out with new regression environment that can use the existing test setup and test tools for reducing 30-50 % of system test effort. Case study was conducted in database environment, but the solution will be generic and can be used in other test environments after customization.

1.2 Purpose, scope and hypothesis

The purpose of the case study was to evaluate the various software testing techniques used in software industry, make a proposal for improving the test efficiency of an existing test environment and implement the same in a data base domain. Thesis work comprised the following activities.

- Identify and evaluate "state of the art" testing techniques and processes followed in standard software industry. This is done based on my experience and the informal interviews with various testing professionals.
- Evaluate the test process and methods followed in real time environment: As per the industry standards.
- Identify the improvement areas: Details are mentioned in chapter 6.
- Prepare an implementation proposal and implement the same: Details are mentioned in chapter 7.

During the studies we cut down the scope of the work to a specific environment for getting a clear understanding of the work and also decided to come out with a general solution that can extend to any environment for improving the test efficiency. We selected the data base environment and decided to conduct case study on a data base indexing tool. The current test environment of the indexing tool does not have automated regression setup and this had negative impact on the effectiveness of the testing of the product. Test team used to run regression test cases manually in system test cycle. Due to the time limit and resource shortage, only selected regression scenarios were considered in system test cycle, which resulted in a risk of less coverage for regression scenarios. Below are some of the issues we identified in our case study and based on that prepared a proposal and implemented the same.

- Manual intervention for running the test suites was resulting in more number of days of effort
- Manual download and installation of DB and indexing tool drivers was a repetitive task and thus causing redundancy issues to test engineers.
- Execution and monitoring of test scenarios from various test machines was resulting in more effort and confusion.
- Lack of scheduled test execution.
- Existing test management process was not sufficient to track the test progress of various service packs.
- Issues in tracking the test status of the individual test team member.
- Under utilisation of available hardware resources.
- Less regression test coverage.
- No GUI interface.
- No central console for analysing the test outputs.

2. Review of Literature

2.1 Literature

While doing the research, many numbers of books, journals, articles, and technical websites are referred. Names of the important references are mentioned in the reference section of this document. Following are the important types of documents referred for this case study.

- Testing fundamentals.
- Test automation frameworks and related works.
- Test management documents.
- Combinatorial testing documents.
- Security testing documents.
- Software quality and productivity improvement documents.
- IEEE documents related to software quality.
- Operational excellence documents.
- Rational build forge documents.
- Software metrics related documents.
- Test effort estimation documents.
- Orthogonal testing documents.
- Data base and indexing documents.

2.2 Findings

While doing the case study, we evaluated automation frame works like rational build forge (IBM Rational build forge V 7.13), LDTP (Linux desktop testing project), Federated Integrated Test frame work (FIT) etc. Out of this we selected the rational build forge because of the various reasons like flexibility of the framework, support from IBM etc. The

selected case study was done on a data base indexing tool. As a first step, a detailed analysis on existing test environment is done and we came out with hypothesis. The hypothesis is validated based on the input we got from the reference document mentioned in the reference section.

Manual intervention in SVT execution was one of the main bottlenecks identified in the evaluation of the case study. For addressing this issue, various testing automation documents, test management documents and best practices documents mentioned in the reference section were referred and were summarised the various solutions and customized the same for our case study evaluation. Automation framework documents for the LDTP, build forge, FIT projects etc., played a major role in the selection of frameworks based on the case study requirements. VM ware concepts helped us to solve the issue related to hardware and the maximum utilisation of the available hardware. Parallel processing and scheduling of the work implemented in the case study was a major step in efficient utilisation. The articles published in this area and the common methods used in industry were the main input for this task. Test documents published by IEEE were very useful for planning the test strategy, test plan, test management and test execution. Test management method using test point system will give a graphical representation of the test tracking and test status. Combinatorial approach and best practices published in this area were used for proper selection of inputs to the case study. We were able to reduce the testing effort by selecting proper input with minimal execution effort and maximum releasing of defects. Papers published in operational excellence area give a clear guidance for planning the proper operational activities and test execution. Intellectual Property (IP) confidentiality is one of the hot areas in software testing. Introducing proper security testing was one of the challenging activities during testing. By following the security guidelines of the industry we were able to give maximum attention to this area.

3. Method of Research

This chapter describes research methods followed in the case study. It starts with a roadmap that describes the overall structure of the thesis followed by a discussion concerning possible research methods to select.

3.1 Road Map

Through out the thesis preparation we followed an action based research methodology.
Figure 3.1 shows the approach used for studying and implementing the setup.

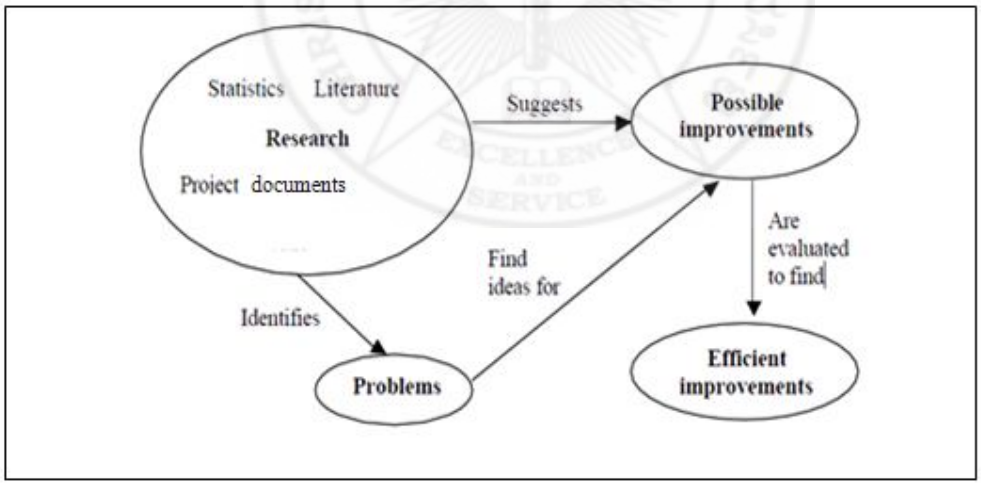


Figure 3.1

The main activities we followed in our case study approach comprises the following steps

- Study current research in software testing industry.
- Evaluate test processes on various real time environments (Like LDTP, FIT, Device anywhere, Build forge etc) and select one environment for implementation.
- Execute a detailed investigation study on the selected environment.

- Prepare an improvement proposal.
- Implement the approved proposal.

3.2 Research method selection

While conducting the case study, it was possible to use two different approaches for collecting information: the qualitative, and the quantitative method. The methods are applicable in different situations and on different sources of information. The main difference between these two methods is the way they approach the objects to investigate. The quantitative method makes an assumption and then examines a set of representative objects to see if they are valid, whereas the qualitative method seeks answers by reviewing as many sides of the object as possible. Since the quantitative method gives numerical data, it can provide better scientific results than the qualitative method; the main approach we employed in this case study was the exploratory approach by mixing both qualitative and quantitative method. In this method we gave more importance to action research as a supporting method for gathering project information by testing theory in an on-going project. The main benefit with action research was to monitor the result of a change while actively undergoing the changes in the existing environment. A drawback is that action research requires allowance by the company to conduct live experiments since it might interfere with the daily work and also it requires more efforts than other types of research since it is hard to conduct on larger samples.

4. Testing Fundamentals

This chapter tries to give a basic knowledge about various testing activities that software testers need to possess in order to perform activities effectively in a given phase of STLC.

4.1 Testing phases

Software testing is the process of verifying, validating and defect finding in a software application or program. In verification we are ensuring that the construction steps are done correctly (are we building the product right), where as in validation we are checking that deliverable (code) is correct (are we building the right product). In software testing a defect is the variance between the expected and actual result. During defects finding, its ultimate source may be traced to a fault introduced in specification, design or development phases. Following are the different levels of testing doing in STLC

- Unit test.
- Integration test.
- System test.
- Acceptance test.
- Regression testing.

Defects can be categorized in to different groups based on severity and priority. Below list shows the common defect category used in software industry.

- Show stopper - Not possible to continue testing because of the severity of the defect
- Critical – Testing can proceed but the application cannot be released until the defect is fixed.

- Major – Testing can continue but the defects may results in serious impacts in business requirements if the software is released for production.
- Medium - Testing can continue and the defect will cause only minimal deviations from the business requirements when in production.
- Minor –Testing can continue and the defect will not affect release.
- Cosmetic - Minor cosmetic issues like colours, fonts, and pitch size that do not affect testing or production release.

Figure 4.1 shows the V-model of software development cycle. V- Model incorporates testing in to the entire SDLC cycle and highlights the existence of different levels of testing and depicts the way each relates to a different development phase. Figure 4.2 shows 5 different testing phases each with a certain type of test associated with it. Each phase has entry criteria that must be met before testing starts and specific exit criteria that should be met before certification of the test. Entry and exit criteria are defined by the test owners listed in the test plan.

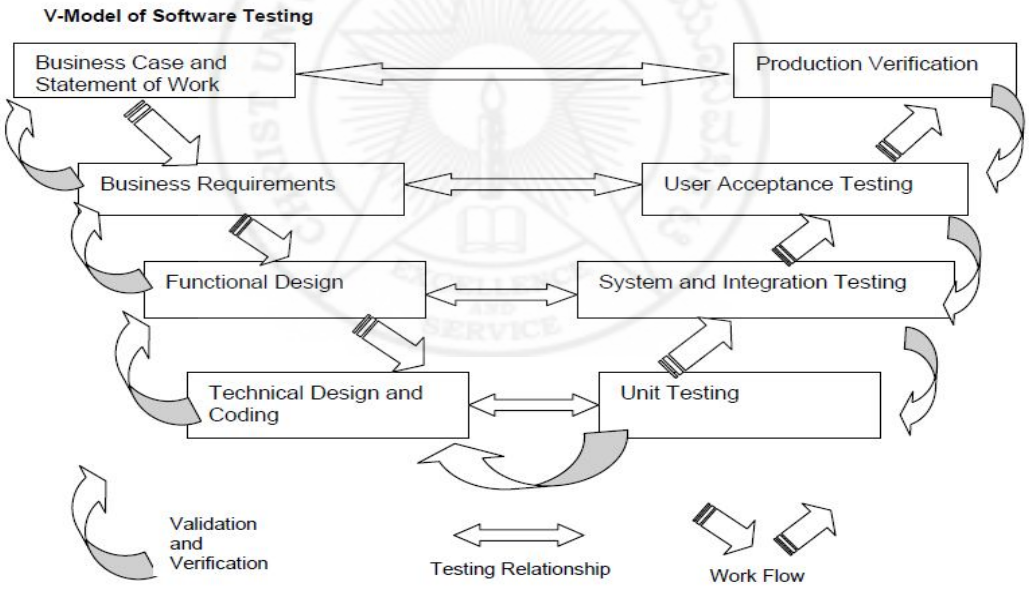


Figure 4.1

Phase	Guiding Document	Test Type
Development Phase	Technical Design	Unit Testing
System and Integration Phase	Functional Design	System Testing Integration Testing
User Acceptance Phase	Business Requirements	User Acceptance Testing
Implementation Phase	Business Case	Product Verification Testing

Figure 4.2

4.1.1 Unit testing

Unit testing test the functionality of basic software units. A unit is the smallest piece of software that does something meaningful. It may be a small function, a statement or a library. Unit test is also called module test where the developer tests the code he/she has produced. Unit tester is mainly looking whether the code was implemented as per low level design document (LLD or functional requirements) and the code structure. Following are some of the faults that are uncovered during unit testing.

- Unit implementation issues – Checking that the unit has implemented the algorithm correctly.
- Input/output data validation errors – Unit's input/output are validated properly.
- Exception handling – Checking whether unit handles the entire environment related errors/exceptions.
- Dynamic resource related errors –Verify whether the dynamic resources (memory, handles, etc.) are allocated and deallocated.
- UI formatting errors – Verify UI is consistent, correct user interface (tabs, spelling, colours etc).
- Basic performance issues – Each unit is critical to overall system performance. Unit tester will ensure that the unit's performance is as per the requirements specification.

Design of unit test cases is done using functional specification or LLD of the units. Any techniques like white box/black box/ grey box can be applied to design unit test cases. Also the structure of the code can be used as another input for improving the quality of the unit test cases. During test cases design, some test cases may come as common to many units; such test cases can be considered as a standard check list and can be used as reusable test suite. If the unit

is not a user interface (UI), it is necessary to write test driver (drives the unit under test with inputs and stores the outcome of the test) and test stubs (dummy storage module used for replacing the unit not available) to automate the unit testing.

4.1.2 Integration testing

Integration testing starts as soon as a few modules are ready and the developers integrate their code for testing the interfaces implemented by their code. High level design document (HLD) is the main input for designing the test cases for integration testing. Following are some of the faults that are uncovered during integration testing:

- Interface integrity issues – Test whether the unit comply to the agreed upon interface specification.
- Data sharing issues – Verifying the common data is handled properly, synchronization issues etc.
- Exception handling – Handles all the environment related errors/exceptions.
- Resource hogging issues – Check whether any unit consumes excessive resources.
- Build issues – Cases like multiple units use a version of common unit that each depends upon.
- Error handling and bubbling of errors – Check that the error returned by a unit is handled by the higher unit appropriately.
- Functionality errors – Functionality formed by the integration of unit(s) work.

Integration testing is proceeded based on integration strategy (order of integration of module) that the project follows. Since testing is an act to find issues that pose severe risk as early as possible, it is preferable to test those interfaces that pose the high risk. Mainly four types of integration strategy employed in software industry.

- Top-down – Integration starts from highest chain of control (top-most module) and this kind of integration uses where upper level interfaces are important.
- Bottom-up – Integration starts from lowest chain of control (bottom-most module) and this kind of integration uses where lower level interfaces are important.
- Sandwich – Approach uses when not all on the top or not all at the bottom are important, this will be a mixture of top-down and bottom-up approach.
- Big bang – This is pretty dumb strategy but this will find issues, the main problem of this approach is the difficulty in debugging.

The approach will be decided based on the criticality of the interfaces and the most critical interfaces should be tested first and the others later. The criticality of the interface can decide once the architecture of the project is ready. Normally most of the projects will follow sandwich approach.

4.1.3 System testing

A system is not a just our code that we developed but that will be a collection of developed code, supporting libraries , data bases (if any) , Web/App servers (if any), operating system and hardware. In system testing phase we test the systems as a whole. For ensuring the maximum benefit of the system test, it is preferable to perform system testing in an environment that is similar to the target environment. Following are the types of faults discovered in system testing.

- Functional errors – Verification of the system that it has implemented the functionality correctly.
- Performance issues – Making sure that the system is fast enough .
- Load-handling capability – Ensuring that the system handling the real life situation with stated resources.
- Usability issues – Verify that the system is friendly and easy to use.
- Volume handling – Verify that the system is capable of handling large volume of data.
- Installation errors – Making sure that the system is able to install correctly using the installation documents.
- Documentation errors – Checking that the documentation done for the system is correct.
- Language handling issues – Verify that the system is implemented the multiple locales correctly. Localization and internationalization testing is performing in this stage.

4.1.4 Acceptance testing

Acceptance testing is the final testing done by the test team and the customer together before the system put in to operation. Acceptance testing starts after completing the system test. The purpose of the acceptance test is to give confidence in that the system is working, rather than trying to find defects. Acceptance testing is mostly performed in contractual development to verify that the system satisfies the agreed requirements. Acceptance testing is sometimes integrated into the system testing phase.

4.1.5 Regression testing

Regression testing is doing for building the confidence of the system that has undergone some changes like modification of the code, defects fixing or added some new module etc. In this test user will rerun the existing test suites/test cases and make sure that the recent changes has not impacted the functionality of the system. Regression test selection is one important task in this phase and need to do carefully for avoiding unnecessary execution. Regression testing is a repeated task and one of the most expensive activities doing in STLC. For saving the effort, it is always good to look for automation so that we can save lot of manual effort. (Harrold 2000) According to Harrold, some studies indicate that regression testing can account for as much as one-third of the total cost of a software system.

4.1.6 Sanity test

Sanity testing will be performed whenever cursory testing is sufficient to prove that the system is functioning according to specifications. A sanity test is a narrow regression test that focuses on one or a few areas of functionality. Sanity testing is usually narrow and deep. It will normally include a set of core tests of basic GUI functionality to demonstrate connectivity to the database, application servers, printers, etc.

4.1.7 Alpha testing

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

4.1.8 Beta testing

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

4.2 Test technique

Effective test cases are the heart of the software testing. For designing test cases testers will use various test techniques in industry and also uses options like domain knowledge, history of past issues etc. Following are some of the test techniques used in industry.

4.2.1 Positive and Negative testing

Positive testing – Check that software performs its intended function correctly and execute programs to check that it meets requirements.

Negative testing – Execute programs with an intent to find defects and discover defects in the system. Negative testing involves testing of special circumstances that are outside the strict scope of the requirements specification, and will therefore give higher coverage.

4.2.2 Risk based testing

Risk is the possibility of a negative or undesirable outcome, quality risk is a possible way that something about your organization's products or services could negatively affect stakeholder satisfaction. Through risk based testing we can reduce quality risk level. This type of testing has number of advantages.

- Finding defects earlier in the defect cycle and thus avoid the risk in schedule delay.
- Finding high severe and priority bugs than unimportant bugs.
- Providing the option of reducing the test execution period in the event of a schedule crunch without accepting unduly high risks.

4.2.3 Defect testing

Defect testing or fault based testing is doing to ensure that certain types of defects are not there in the code. It is a negative testing approach to discover defects in the system. Normally testing team will identify and classify the defects that have occurred in the previous release of the product. Based on this classification test team will decide where to add more testing efforts and also will decide how deeply need to conduct testing on those areas. Test team will use defect tracking tool or defect database as an input for this activity. The root cause analysis available in the defect or that is prepared will play a major role in defect classification.

4.2.4 White box testing

White box testing or glass box testing or structural testing method uses the code structure to come up with test cases. For doing effective white box testing tester need to have a good understanding of the code. Normally there is a miss-understanding that white box testing can apply only in unit level testing. It can definitely be applied at the unit level. It can be applied at the higher levels like

integration level, system level. Unit testing becomes difficult as the size of the code rapidly increases at higher levels.

4.2.5 Black box testing

In black box testing or functional testing, the tester should have a clear understanding of the specification of the product/project that he is testing. Specification covers both data (input and out put specification) as well as business logic specification (processing logic involved).Requirement specification is one of the major input doc for doing black box testing. Black box testing can apply at any levels of testing. Some of the black box techniques detect functionality issues while some of them help in detecting non-functional issues.

4.2.6 Grey box testing

Gray box testing is combination of white and black box testing. This testing will identify the defects related to bad design or bad implementation of the product. Test engineer who executes gray box testing has some knowledge of the system and design test cases based on that knowledge. Tester applies a limited number of test cases to the internal working of the software under test. Remaining part of the execution will do based on data specification and business logic. The idea behind the gray box testing is that one who knows something about how the products works on the inside, one can test it better.

4.2.7 Statistical testing

The purpose of statistical testing is to test the software according to its operational behaviour, i.e. by running the test cases with the same distribution as the users intended use of the software. By developing operational profiles that describes the probability of different kinds of user input over time; it is possible to select a suitable distribution of test cases. Developing operational profiles is a time consuming task but a proper developed profile will help to make a system with a high reliability. In short a statistical test will help to make a quantitative decision about a process.

4.2.8 Clean room software engineering

Clean room software engineering is more of a development process than a testing technique. The idea clean room will help to avoid high cost defects by writing source code accurately during early stages of development process and also employ formal methods for verifying the correctness of the code before testing phase. Even though the clean room process is time

consuming task but helps to reduce the time to market because the precision of the development helps to eliminate rework and reduces testing time. Clean room is considered as a radical approach to quality assurance, but has become accepted as a useful alternative in some systems that have high quality requirements.

4.2.9 Static testing

Testing is normally considered as a dynamic process, where the tester will give various inputs to the software under test and verify the results. But static testing is of different kind of testing that is used for evaluating the quality of the software without executing the code. Static testing is fall in the verification process that ensures the construction steps are done correctly with out executing the code. One commonly used technique for static testing is the static analysis-functionality that the compilers for most modern programming languages have. Reviews and inspection are the most commonly used static testing method in almost all software development organizations. Static testing is applicable to all stages but particularly appropriate in unit testing, since it does not require interaction with other units.

4.2.10 Review and inspection

Each author has there on definition for the terms review and inspection. As per IEEE Std. 610.12-1990 the terms are defined as

***Review*:** A process or meeting during which a work product, or set of work products, is presented to project personnel, managers, users, customers, or other interested parties for comment or approval. Types include code review, design review, formal qualification review, requirements review, and test readiness review' (IEEE 1990). IEEE standard says that, the purpose of a technical review is to evaluate a software product by a team of qualified personnel to determine its suitability for its intended use and identify discrepancies from specifications and standards. Following are some of the inputs to the technical review:

- A statement of objectives for the technical review (mandatory).
- The software product being examined (mandatory).
- Software project management plan (mandatory).
- Current anomalies or issues list for the software product (mandatory).
- Documented review procedures (mandatory).
- Relevant review reports (should).

- Any regulations, standards, guidelines, plans, and procedures against which the software product is to be examined (should).
- Anomaly categories (See IEEE Std 1044-1993 [B7]) (should).

Inspection: A static analysis technique that relies on visual examination of development standards, and other problems. Types include code inspection; design inspection' (IEEE 1990). Inspection has many names, some called software inspection that could cover design and documentation, and some others will call it as code inspection that relates more on source code written by developer. Fagan inspection is another name that came as the name of the person who invented QA and testing method. Code inspection is a time consuming task but statistics telling that it may cover up to 90% of the contained errors if we apply that in a systematic way. Figure 4.3 below shows time, employee relationship in a software development process.

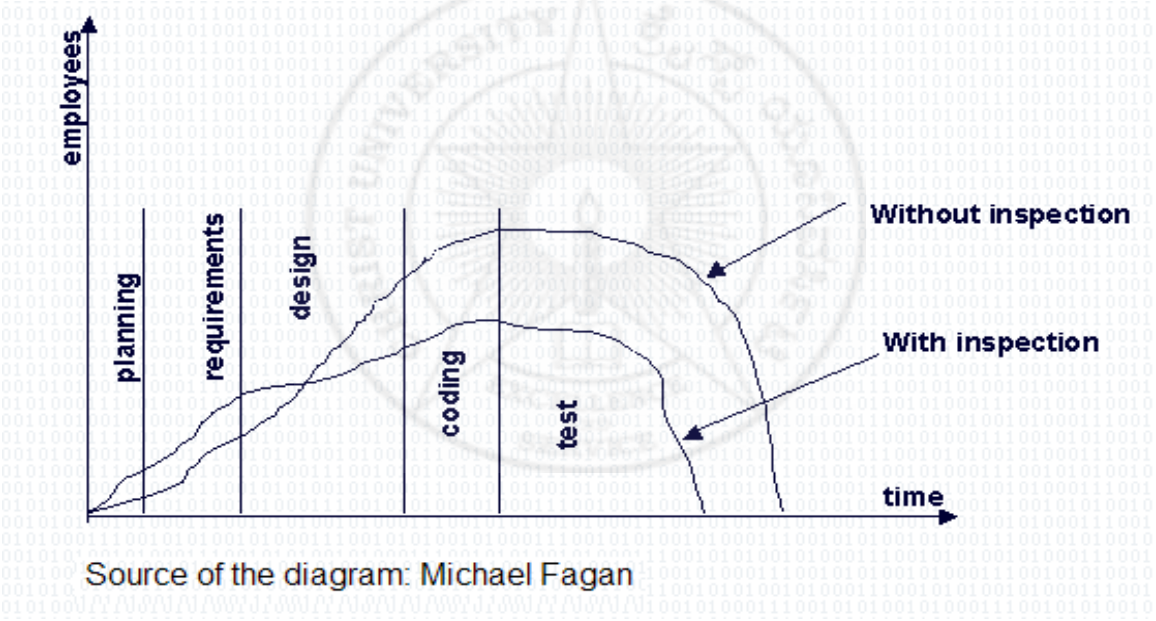


Figure 4.3

IEEE Standard for Software Reviews (IEEE 1028-1997 standard) is talking about manual static testing methods like inspections, reviews and walkthroughs.

4.2.11 Walk-throughs

Walk-throughs are techniques used in software development cycle for improving the quality of the product. It helps to detect anomalies, evaluate the conformance to standards and specifications etc. It is considering as a techniques for collecting ideas and inputs from team members during the design stage of the software product and also as for exchanging techniques and conduct

training to the participants , thus to raise the level of team mates to same programming style and details of the product. Walk-through leader, recorder, author of the product under development and team members are some of the roles defined in walk-through method.

4.3 Test case design techniques

A test case is a set of data and test programs (scripts) and their expected results. Test case validates one or more system requirements and generates a pass or fail. The Institute of Electrical and Electronics Engineers defines test case as "A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement." Selecting adequate test case is an important task to testers other wise that may result in too much testing, or too little testing or testing wrong things. Following are the characteristics of a good test.

- A test case has a reasonable probability of catching an error
- It is not redundant
- It's the best of its breed
- It is neither too simple nor too complex

While doing test case design, designer should have an intension to find errors so that he can start searching ideas for test cases and try working backwards from an idea of how the program might fail. Following are some of the techniques we use in industry for designing effective test cases.

4.3.1 Equivalence classes

It is essential to understand equivalence classes and their boundaries. Classical boundary tests are critical for checking the program's response to input and output data. You can consider test cases as equivalent, if you expect same result from two tests. A group of tests forms an equivalent class if you believe that

- They all test same thing
- If one test catch catches a bug , the others probably will too
- If one test doesn't catch a bug, the others probably won't either.

Tests are often lumped into the same equivalence classes when

- They involve the same input variables
- They result in similar operations in the program

- They affect the same output variables
- None force the program to do error handling or all of them do

Different people will analyse programs in different way and comes up with different list of equivalent classes. This will help you to select test cases and avoid wasting time repeating what is virtually the same test. You should run one or few of the test cases that belongs to an equivalence class and leave the rest aside. Below are some of the recommendations for looking equivalence classes:

- Don't forget equivalence classes for invalid inputs
- Organize your classification into a table or an outline
- Look for range of numbers
- Look for membership in a group
- Analyse responses to lists and menus
- Look for variables that must be equal
- Create time-determined equivalence classes
- Look for variable groups that must calculate to a certain values or range
- Look for equivalent output events
- Look for equivalent operating environments

4.3.2 Boundaries of equivalence classes

Normally we use to select one or two test cases from each equivalence class. The best ones are the class boundaries, the boundary values are the biggest, smallest, soonest, shortest, loudest, fastest ugliest members of the class i.e., the most extreme values. Program that fail with non-boundary values usually fail at the boundaries too. While analysing program boundaries it is important to consider all outputs. It is good to remember that input boundary values might not generate output boundary values.

4.3.3 Black box test techniques

This type of techniques can be categorized in to three broad types

- Those useful to design test scenarios (High level test design techniques).
- Those useful to generate test values for each input(Low level test design techniques).
- Those useful in combining test values to generate test cases.

4.3.3.1 High level test design techniques

Some of the commonly used high level test design techniques are

- Flowchart – Represent flow based behaviour (Each scenarios has a unique flow in the flow chart).
- Decision table – Represent rule based behaviour (Each scenario is an unique rule in the decision table).
- State machine – Represent state based behaviour (Each scenario is an unique path in the state transition diagram.

4.3.3.2 Low level test design techniques

Following are some of the some of the low level test design techniques:

- Boundary value analysis - Generate test values on and around boundary
- Equivalence partitioning – Ensues that all representative values have been considered
- Special value - generate interesting test values based on experience/guess
- Error based vales - Generate test values based on past history of issues

4.3.3.3 Combinational test design techniques

This technique will combine test values to generate test cases, some of the combinational test design techniques are mentioned below:

- Exhaustive testing – Combine all vales exhaustively (All combination of all test inputs are considered).
- All-pairs /Orthogonal – Combine to form minimal yet complete combinations. This will ensures that all distinct pairs of inputs have been considered.
- Single-fault – Combine such that only a single input in a test case is faulty (Generate negative test cases where only one input is incorrect).

4.3.4 White box test techniques

This technique uses the structure of the code for designing test cases; following are some of the aspects of the code that constitutes the code structure:

- Flow of control – Is the code sequential / recursive / concurrent
- Flow of data – Where is the data initialized and where it is used
- Resource usage – What dynamic resources are allocated , used and released

4.3.5 Coverage based testing

Statement coverage is an oldest structural test technique that targets to execute every statement and branch during a set of tests. Statement coverage will give an idea about the percentage of total statements executed. Since programs with for example loops contain an almost infinite number of different paths, complete path coverage is impractical. Normally, a more realistic goal is to execute every statement and branch at least once. This technique can be varied in several ways and is usually tightly knit to coverage testing.

- Branch coverage – Measuring the number of conditions / branches executed as a percentage of total branch.
- Multiple condition coverage – Measuring the number of multiple conditions executed as a percentage of total multiple conditions.
- Statement coverage – Measuring the number of statements executed as a percentage of total statements.

4.3.6 Random input testing

Rather than explicitly subdividing the input in to a series of equal sub ranges, it is better to use a series of randomly selected input values, that will ensues that input value is likely as any other , any two equal sub ranges should be about equally represented in your tests. When ever you cannot decide what vales to use in test cases, choose them randomly. A random input doesn't mean "what ever inputs come to your mind" but a table of random numbers or a random number generating function. Random testing using random inputs can be very effective in identifying rarely occurring defects, but is not commonly used since it easily becomes a labour-intensive process.

4.3.7 Syntax testing

This is a data-driven test technique where well-defined syntax rules validate the input data Syntax testing can also be called grammar -based testing since grammars can define the syntax rules. An example of a grammar model is the Backus Naur Form, which can represent every combination of valid inputs.

4.4 Types of tests

Testing can be broadly classified as two types, namely functional tests and non-functional tests. Functional Testing is the process by which expected behaviour of an application can be tested. We already discussed many functional test techniques in previous section. In this section we will try to give a brief description about various non functional tests that used to execute in IT industry.

4.4.1 Load test

Load testing is used for verifying the software product is able to handle real life operations with the stated resources. It can be done in controlled lab conditions or in a field. Load test in a lab will help to compare the capabilities of different systems or to measure the actual capability of a single system. The main aim of the load testing is to determine the maximum limit of the work that can handle with out significant performance degradation.

4.4.2 Stress test

This test will check that worst load it can handle is well above real life extreme load. The stress test process can involve quantitative test done in a lab , such as measuring the frequency of errors or system crashes. It can also use for evaluating the factors like availability of the system, resistance to denial of service attacks.

4.4.3 Performance test

Check that the key system operations perform with in the stated time. Performance testing is very difficult to conduct because the performance requirements often are poorly specified and the test requires a realistic operational environment to get reliable results. Automated tool support is required for doing proper performance evaluation of the software.

4.4.4 Scalability test

Check that the system is able to handle more loads with more hardware resources. We can consider scalability testing as an extension of performance testing. Scalability is the factor that needs to consider in the beginning of the project planning and designing. The architect of the product should have a proper picture about the product before he plans the scalability of the product under development. For making sure that the products is truly scalable and for identify

major work loads and mitigate bottlenecks, it is very important to rigorously and regularly test it for scalability issues. The results from the performance test can consider as the baseline, and we can compare the results of the performance test results to know the application is scaled up or not.

4.4.5 Reliability test

This test will check that the system when used in an extended manner is free from failures. In systems with strict reliability requirements, the reliability of the system under typical usage should be tested. Several models for testing and predicting reliability exist but in reality, the exact reliability is more or less impossible to predict.

4.4.6 Volume test

Check that the system can handle large amounts of data. Volume test is mainly concentrating about the concept of throughput instead of response time on other testing. Capacity drivers are the key to do effective volume testing for the application like messaging systems, batch systems etc. A capacity driver is something that directly impacts on the total processing capacity. For a messaging system, a capacity driver may well be the size of messages being processed.

4.4.7 Usability test

Check whether the system is easy to operate by its end users. When the system contains a user interface, the user-friendliness might be important. However, it is hard to measure usability since it is difficult to define and most likely require end-user interaction when being tested. Nevertheless, it is possible to measure attributes like for example learn-ability and handling ability by monitoring potential users and record their speed of conducting various operations in the systems.

4.4.8 Security test

This test will ensure that the integrity of the system is not compromised. Security test is also called penetration testing and used to test how well the system protects against unauthorized internal or external access, wilful damage, etc; may require sophisticated testing techniques. Testers must use a risk-based approach, grounded in both the system's architectural reality and the attacker's mindset, to gauge software security adequately. By identifying risks in the system and creating tests driven by those risks, a software security tester can properly focus on areas of

code in which an attack is likely to succeed. This approach provides a higher level of software security assurance than possible with classical black-box testing.

4.4.9 Recovery test

Recovery test will verify that that the system is able to recover from erroneous conditions graciously. It also tests how well a system recovers from crashes, hardware failures, or other catastrophic problems.

4.4.10 Storage test

Check that the system complies with the stated storage requirements like disk/memory.

4.4.11 Internationalization test (I18N)

This test will verify the ability of the system to support multiple languages. Internationalization test is also called as I18N test. I18N testing of the products is targeted to uncover the international functionality issues before the system's global release. Mainly this will check whether the system is correctly adapted to work under different languages and regional settings like the ability to display correct numbering system – thousands, decimal separators, accented characters etc. I18N testing is not same as the L10N testing. In I18N testing product functionality and usability are the focus, where as L10N testing focuses on linguistic relevance and verification that functionality has not changed as a result of localization.

4.4.12 Localization test (L10N)

Check that the strings, currency, date, time formats for this language version has been translated correctly. Localization testing is also called L10N testing. Localization is the process of changing the product user interface and modification of some initial settings to make it suitable for another region. Localization testing checks the quality of a product's localization for a particular target culture/locale. Localization test is based on the results of I18N testing, which verifies the functional support for that particular culture/locale. L10N testing can be executed only on the localized version of a product.

4.4.13 Configuration test

Check that the system can execute on different hardware and software configuration.

4.4.14 Compatibility test

Check that the system is backward compatible to its prior versions.

4.4.15 Installation test

Check that the system can be installed correctly following the installation instructions. The installation test for a release will be conducted with the objective of demonstrating production readiness.

4.4.16 Documentation test

Documentation test will make sure that the user documentation, online help is inline with software functionality. Testing of user documentation and help-system documentation is often overlooked because of a lack of time and resources (Watkins 2001). However, Watkins claims that accurate documentation might be vital for successful operation of the system and reviews are in that case probably the best way to check the accuracy of the documents.

4.4.17 Compliance test

Check that the software has implemented the applicable standard correctly.

4.4.18 Accessibility test

Accessibility test will check that the product under test is accessibility complaint or not. With this test we are targeting four types of users namely people with visual impairments, hearing impairments, motor skills(Inability to use keyboard or mouse) and cognitive abilities (reading difficulties, memory loss). Normally we plan separate testing cycle for accessibility testing. Inspectors or web checkers are some example of tools available in market for doing accessibility testing.

4.5 Test Strategy

A Test Strategy document is a high level document that talks about the overall approach for testing and normally developed by project manager. This document is normally derived from the Business Requirement Specification document. This static document contains standards for testing process and will not undergo changes frequently. This is acting as an input document for test plan. A good test strategy will answer the below questions.

- Where should I focus?
- On what features?
- On what type of potential issues?
- What test technique should I use for effective testing?
- How much of black box, white box?
- What type of issues should I look for?
- Which is best discovered by testing?
- Which is best discovered via inspection?
- How do I execute the tests? Manual/Automated?
- What do I automate?
- What tool should I consider?
- How do I know that I am doing a good job?
- What metrics should I collect and analyse?

4.5.1 Contents of test strategy

- ***Features to focus on :***
  - List down the major features of the product.
  - Rate importance of each features (Importance = Usage frequency * failure criticality).
- ***Potential issue to uncover:***
  - Identify potential faults.
  - Identify potential incorrect inputs that can result in failure.
  - State the type of issues that you will aim to uncover.
  - Identify what types of issues will be detected at each level of testing.
- ***Types of test to be done*:**
  - State the various tests that need to be done to uncover the above potential issues.
  - Identify the test techniques that may be used for designing effective test cases.
- ***Execution approach:***
  - Continue what test will be done: manual/automated.
  - Outline tools that may be used for automated testing.
- ***Test metrics to collect and analyse***
  - Identify measurements that help analyse if the strategy is working effectively.

4.6 Test Planning

Test plan details out the operational aspects to executing the test strategy. Test plan will be derived from the product description, software requirement document, use case documents etc. It may be prepared by a test lead or test manager. A test plan outlines the following

- Effort / time needed
- Resources needed
- Schedules
- Team composition
- Anticipated risk and contingency plan
- Process to be followed for efficient execution
- Roles of various team members and their work

As per the IEEE 829 format, following are the contents of the test plan

1. Test Plan Identifier : Unique company generated number to identify this test plan
2. References : List all documents that support this test plan
3. Introduction : A short introduction to the software under test
4. Test Items : Things you intend to test within the scope of this test plan
5. Software Risk Issues : Identify what software is to be tested and what the critical areas are
6. Features to be Tested: This is a listing of what is to be tested from the users viewpoint of what the system does
7. Features not to be Tested: Listing of what is not to be tested from both the Users viewpoint of what the system does and a configuration management/version control view.
8. Approach : This is your overall test strategy for this test plan
9. Item Pass/Fail Criteria: What are the Completion criteria for this plan? The goal is to identify whether or not a test item has passed the test process
10. Suspension Criteria and Resumption Requirements : Know when to pause in a series of tests or possibly terminate a set of tests. Once testing is suspended how is it resumed and what are the potential impacts
11. Test Deliverables

12. Remaining Test Tasks: There should be tasks identified for each test deliverable. Include all inter-task dependencies, skill levels, etc. These tasks should also have corresponding tasks and milestones in the overall project tracking process
13. Environmental Needs : Are there any special requirements for this test plan
14. Staffing and Training Needs : State the staffing learning/training needs to be done to execute the test plan
15. Responsibilities: Who is in charge? There should be a responsible person for each aspect of the testing and the test process. Each test task identified should also have a responsible person assigned
16. Schedule : Detail the work schedule as Gantt chart
17. Planning Risks and Contingencies : State the top five (or more) anticipated risks and mitigation plan
18. Approvals : Who can approve the process as complete and allow the project to proceed to the next level

4.7 Test cycle

Test cycle is the point of time wherein the build is validated and it takes multiple test cycles to validate a product. Each test cycle should have a clear scope like what features will be tested and what test will be done. Figure 4.4 below shows the test development life cycle. Normally we used to run four rounds of the test cycle. In this period will be catching around 80% of the errors. With the majority of these errors fixed, standard and/or frequently used actions will be tested to prove individual elements and total system processing in cycle 3. Regression testing of outstanding errors will be performed on an ongoing basis. When all major errors are fixed, an additional set of test cases are processed in cycle 4 to ensure the system works in an integrated manner. It is intended that cycle 4 be the final proving of the system as a single application. There should be no Sev1 or Sev2 class errors outstanding prior to the start of cycle 4 testing. Figure 4.5 shows the 4 different cycles (release) of testing that normally follows in software development.

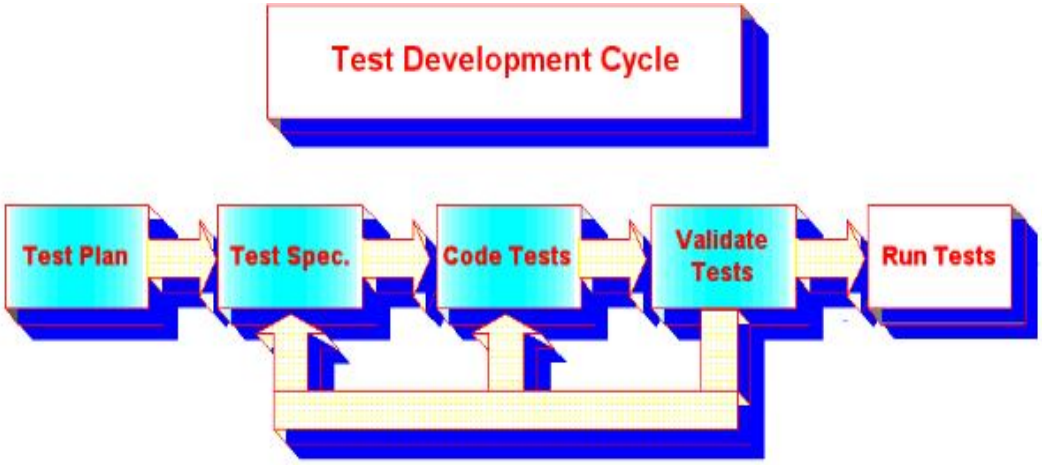


Figure 4.4

	Acceptance 1
Release v0.1	Functional 1
	User Acceptance
	Acceptance 2
Release v0.2	Functional 2
	Regression 1
	Acceptance 3
	Functional 3
Release v0.3	Performance 1
	Bash & Multi-User Testing
	Regression 1
	Regression 2
	Integration 1
	Technical 1
Release v0.4	Regression 1
	Regression 2
	Regression 3
	Installation Test
Contingency	*Per Bug Fix Test Only*

Figure 4.5

4.8 Test Estimation

Test Estimation is the estimation of the testing size, testing effort, testing cost and testing schedule for a specified software testing project in a specified environment using defined

methods, tools and techniques. Effort estimation can consider as a science of guessing. Some of the terms commonly used in test estimation are

Testing Size – the amount (quantity) of testing that needs to be carried out. Some times this may not be estimated especially in Embedded Testing (that is, testing is embedded in the software development activity itself) and in cases where it is not necessary

Testing Effort – the amount of effort in either person days or person hours necessary for conducting the tests

Testing Cost – the expenses necessary for testing, including the expense towards human effort

Testing Schedule – the duration in calendar days or months that is necessary for conducting the tests

To do a proper estimation we need to consider the following areas

- Features to focus
- Types of test to do
- Development of automated scripts
- Number of test cycles
- Effort to design , document test plan, scenarios/cases
- Effort need to document defects
- Take expert opinion
- Use the previous similar projects as inputs
- Breaking down the big work of testing to smaller pieces of work and then estimation (Work break down structure)
- Use empirical estimation models

4.9 Test reports

There are multiple number of test reports are using in various kinds of testing. Some of the commonly used test reports in industry are mentioned below.

4.9.1 Weekly status report

Weekly status report gives an idea about the works completed in a specific week against the plan of actual execution. Companies have their own standard template for reporting this status.

4.9.2 Test cycle report

Product testing has multiple cycles. Management will expect the correct status of each cycle for tracking the project. Test team is responsible for giving report on accomplishments in the cycle and potential testing related risks in a standard template approved by the company.

4.9.3 Quality report

Quality report will give an idea about objectives and subjective assessment of quality of a product on a specific date. A product quality depends on factors like scope, cost and time. Quality lead will consider all these 3 factors before reporting the status in the standard template.

4.9.4 Defect report

A defect report will give a detailed description of defects. This is one of the important deliverables in STLC. An effective defect report will reduce the number of returned defects. A good defect report will reflect the credibility of the tester and also will help for speeding up the defect fixes.

4.9.5 Final test report

This is the report that summarizes the test happened in various levels and cycles. Based on this report the stake holder can assess the release quality of the product.

5. Test Automation Process

In software industry, test automation becomes an increasingly critical and strategic necessity. Assuming the level of testing in the past was sufficient (which is rarely the case), how do we possibly keep up with this new explosive pace of project deployment while retaining satisfactory test coverage and reducing risk? The answer is either more people for manual testing, or a greater level of test automation. After all, a reduction in project cycle time generally correlates to a reduction of time for test. In this chapter we are discussing abut the test automation processes that we employed in our case study for improving the test efficiency.

5.1 Automation Framework Overview

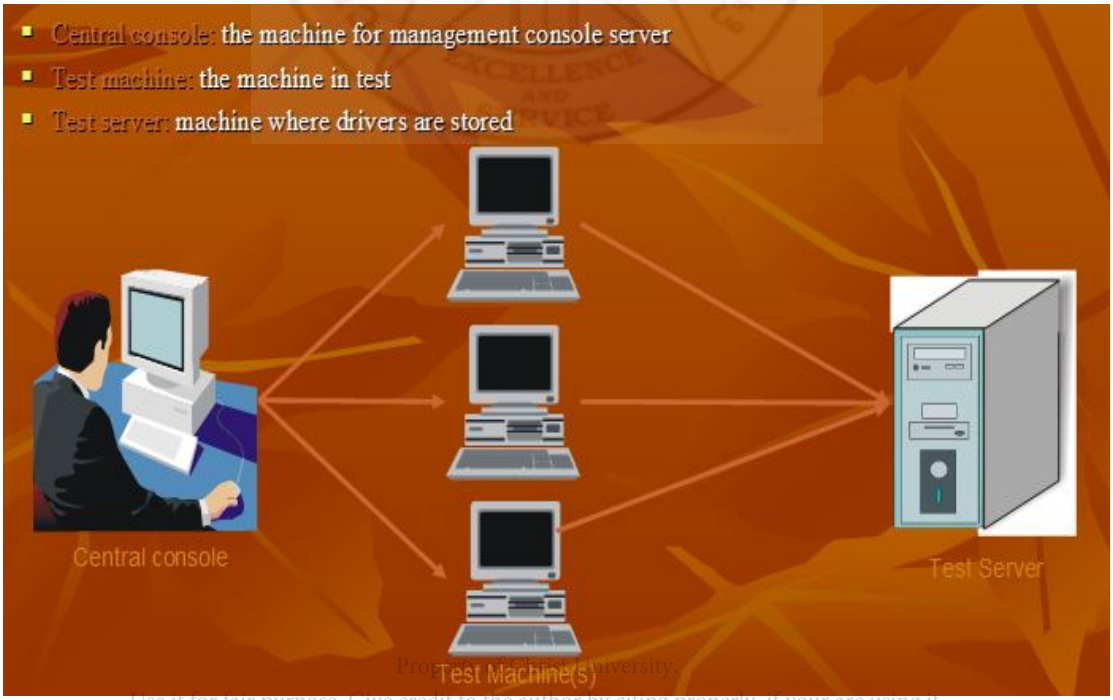


Figure 5.1

Figure 5.1 shows the pictorial representation of the test environment used for the case study. This set up consist of central test server that act as a repository used for storing test drivers and other documents used for testing. Test machines with various operating systems like Windows (32 bit and 64 bit), Linux (32 bit and 64 bit), AIX, Solaris and HP etc act as client machines. The third important component is the test automation frame work (Central console - A server with Rational build forge tool). IBM Rational Build Forge (Figure 5.2- Build forge automation blocks) automates and accelerates build, test and release processes to enable iterative development, high-performance builds and streamlined software delivery. Through an adaptive framework, it helps the teams to standardize and automate repetitive tasks, optimize hardware resources and connect development tools to increase staff productivity, compress development cycles and deliver high quality software, quickly. Following are some of the advantages:

- Leverage of current assets (tools/scripts).
- Integration with other tools (Adapters and IDE plug-in).
- Effective Monitoring through single window (web console).
- Effective environment maintenance.
- Support availability.

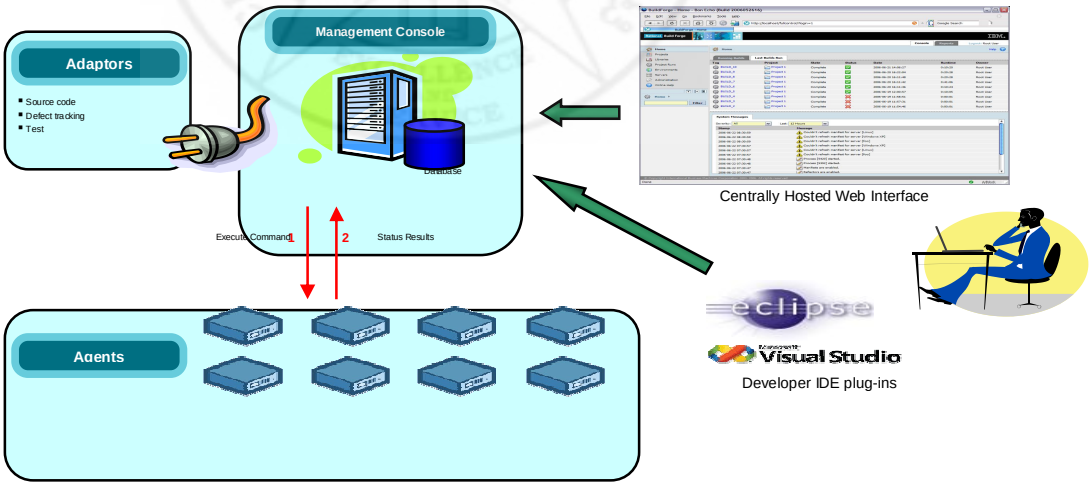


Figure 5.2

5.2 Challenges in software test automation

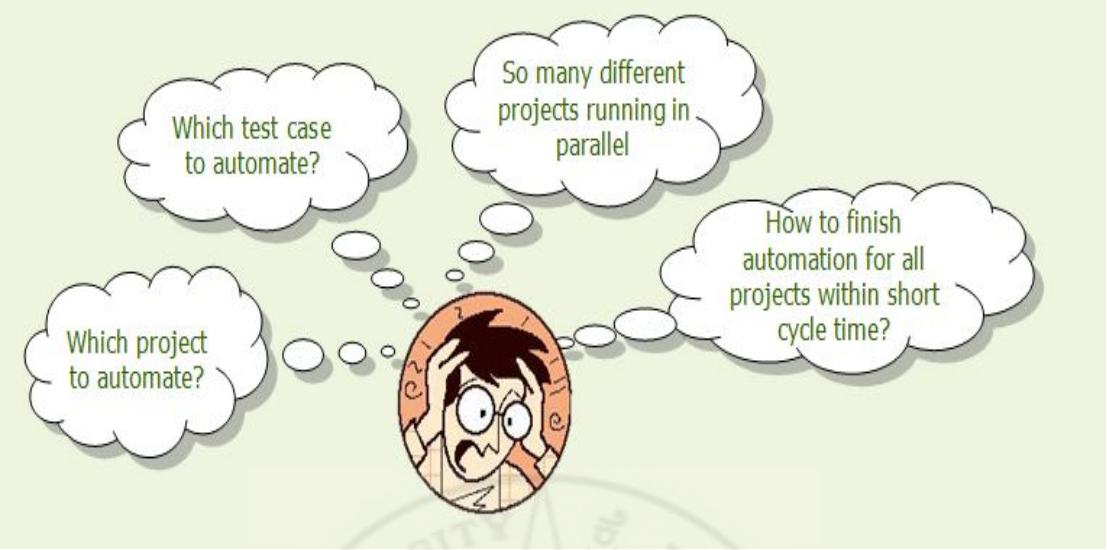


Figure 5.3

Figure 5.3 shows the situation of a test manager who is not following proper test process. It is quite common that many automation testers are being thrown to the automation job without having proper guidelines on the automation test process. They just "dive in" and begin automating test cases without a thought towards any process or strategy. Normally this kind of approach will reach a situation that says "We've invested lot in automation and number of testers allocated for fulltime in automation testing. After Y months, we still do not see any improvement in our overall testing cycle times. In fact, testing seems to be taking longer!" To avoid such a situation we should have a clear understanding about our project and automation process that we are going to implement. Also we should have a proper test strategy in selecting the framework and execution. Some of the factors that need to consider while defining test strategy are:

- Test automation is a fulltime effort, not a sideline.
- The test design and the test framework are totally separate entities.
- The test framework should be application-independent.
- The test framework must be easy to expand, maintain, and perpetuate.
- The test strategy/design vocabulary should be framework independent.
- The test strategy/design should remove most testers from the complexities of the test framework.

5.3 Test Automation

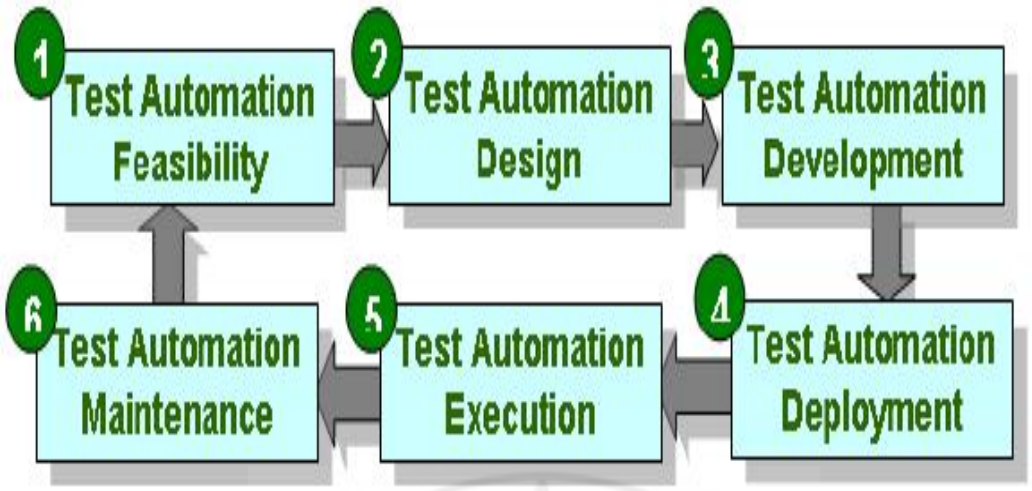


Figure 5.4

Figure 5.4 shows various stages in the test automation process and figure 5.5 shows the test tool selection process in a typical automation project

5.3.1 Roles

Automation lead/manager will be responsible for selection of a tool, development of tools and maintenance activities of the frame work. Test engineer will be responsible for script generation, deployment and execution.

5.3.2 Feasibility study

This stage will decide whether we need to automate the project or not. Failure in this stage may have a larger impact on the project execution. Following are some of the factors considered in feasibility stage:

- Project impact: Automation priority according to project prioritization and delivery.
- Test case selection :
  1. Repetitive test that needs to be run on multiple build.
  2. Frequently used functions.
  3. Tests that run on several different platforms.
  4. Tests that take a lot of effort and time when manual testing.

- Framework availability: Exploring various existing tools and evaluating whether we can use the same in our setup through customization.

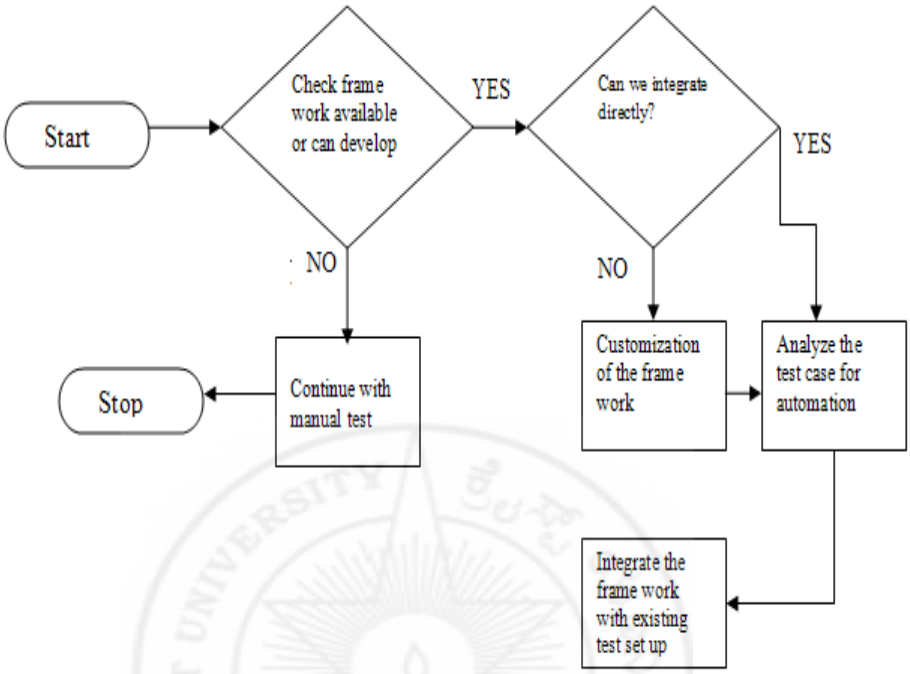


Figure 5.5

Figure 5.6 depicts how we consider the test patterns and test automation in various stages of software testing.

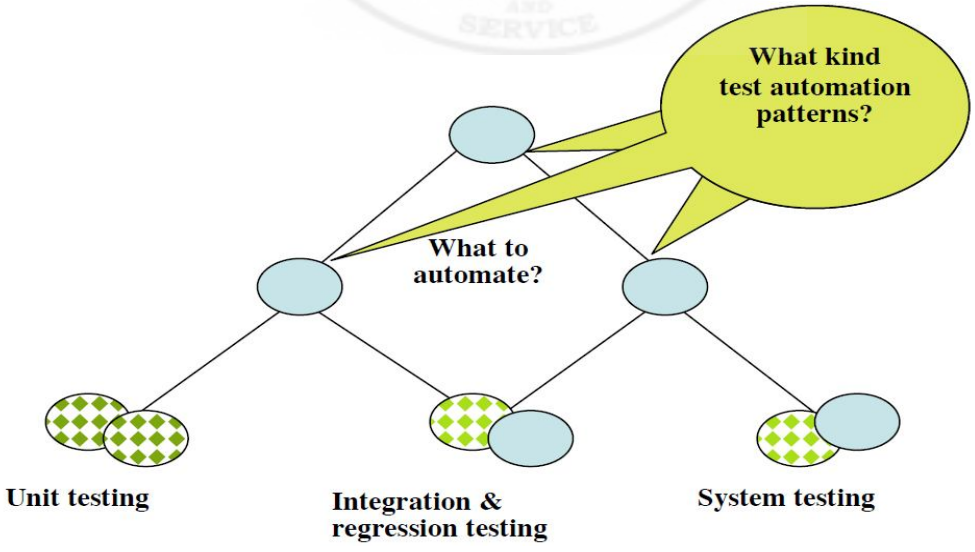


Figure 5.6

5.3.3 Automation design

A good automation test design will tell how a particular function or feature will be tested. A test designer will consider the following facts:

1. What is being tested and how is the test set up?
2. What are the inputs used and from where the inputs are coming?
3. What is being checked and where are the expected results?
4. What are the things need to print?
5. How do you know the test is pass or fail?
6. Keep the output simple and well formatted.

5.3.4 Automation development and deployment

Mainly we need to develop two types of items during the development stage

1. Scripts for execution.
2. Frame work code (or customization of the existing code).

Some projects, whole modules may not be available. In such case we may need to develop stubs and drivers for simulating the module. In our frame work, test developer is responsible for creating the test suites (test scripts), customization of the rational build forge tool and the environment setting for the test bed. Also after developing the script and code we need to regularly check in the same in Clear case (or any other tool) and need to create the proper build for execution. Testers have access permission for taking the same for testing.

5.3.5 Automation execution and maintenance

In testing phase we will execute the test cases either manual or automated fashion. For the automation, selection of the test cases is done using the automation strategy. Normally all the regression test cases will be moving to automated environment for avoiding the repeated manual execution. In our automated environment the tester can select/deselect the test cases that he wants to execute on a particular platform. While executing we can give two different options. In the first option, that is, halt on failure, the test execution will stop if any one of the selected test fails. But in second option, continue on failure, will allow executing all the selected suites, even if one suite fails. At the end of the execution we can see the results and logs of each suite separately. This will give a detailed status of the execution. In our frame work we can execute test cases on any number of platforms simultaneously. Figure 5.7 shows the sample screen shot of the build forge console.

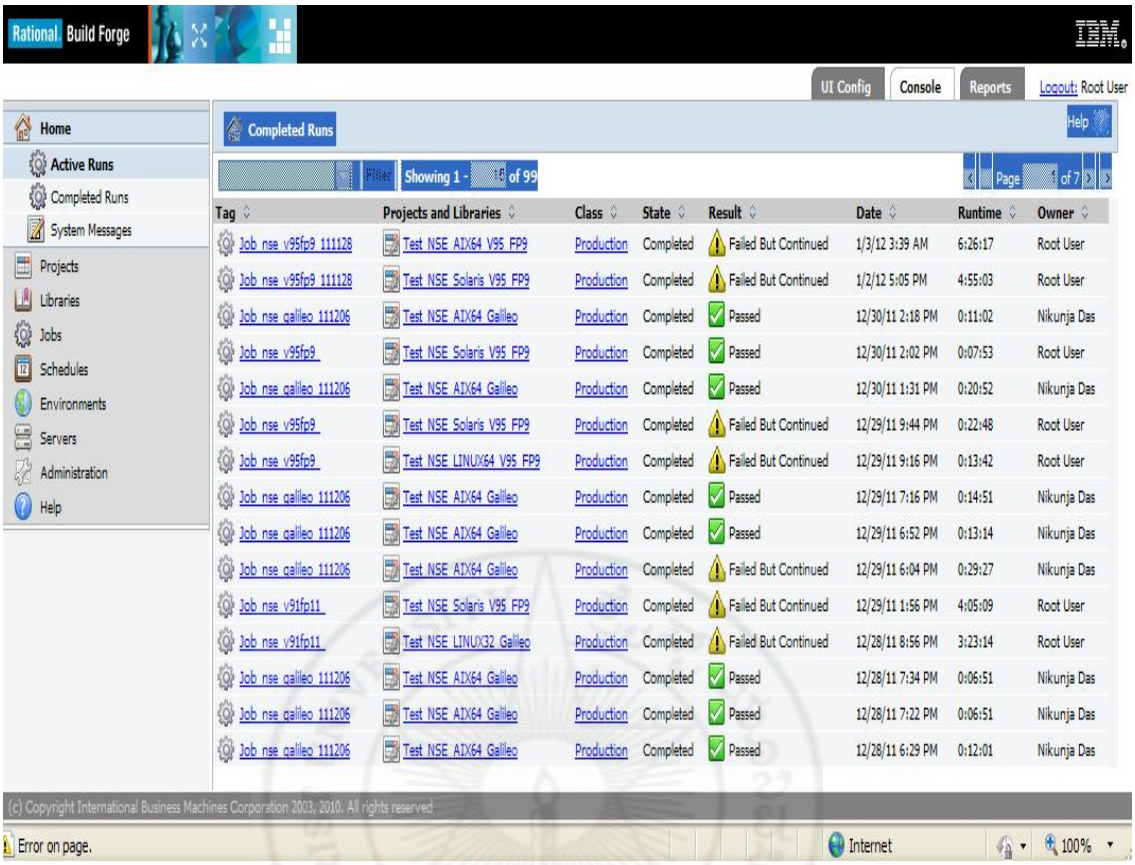


Figure 5.7

5.3.6 Benefits of automation

Following are some of the benefits of test automation

- High coverage for regression testing
- Improve the speed of product to market by reducing the elapsed time for testing
- Improve the productivity
- Generate detailed test logs
- Run the scripts across multiple platforms
- Fast, reliable, comprehensive and reusable
- Cost effectiveness

5.3.7 Test automation success factors

Success of test automation depends on direct and indirect factors of the test organisation.

Direct success factors:

- Test process

- Test management
- Test object delimitation
- Test case determination
- Test data and test data definition
- Test infrastructure and environment
- Test tools selected
- Employees productivity

Indirect success factors:

- Configuration management
- Change management
- Defect management
- Release management
- Requirements management

6. Identification of improvement candidates

This chapter is talking about the areas that need to give more attention for improving the test efficiency of the existing test environment.

6.1 System test automation

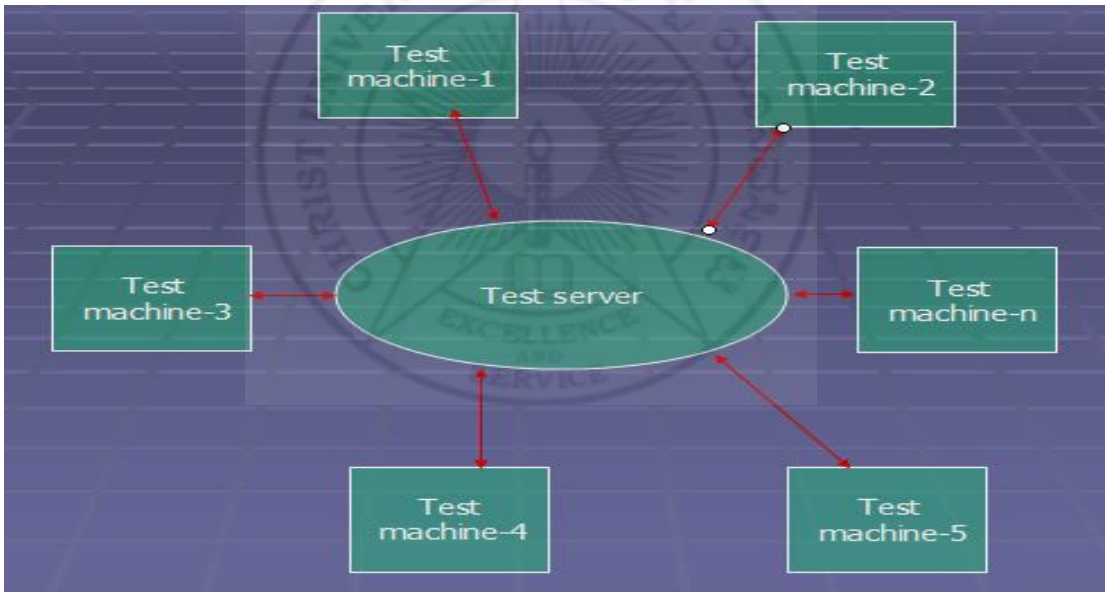


Figure 6.1

Figure 6.1 shows the initial test setup of the database indexing tool. The test server is the place where we store the test documents, test drivers and execution logs of the test runs. Test machines are loaded with software under test. The above setup needs a lot of manual intervention for doing proper test execution. After doing deep analysis on the test setup, we identified this as one of the main bottle necks.

6.2 Install test automation

Indexing tool has multiple builds in each service pack release. Test team needs to ensure that the drivers are working fine before proceeding with a system test. Manual download and installation of database and indexing tool drivers are identified as the problem area for improving the test efficiency. During the analysis stage, team identified the following items for avoiding manual errors, redundancy and thus to improve the test efficiency.

- FTP download of selected levels of drivers to centralized location (Test server)
- Copy the specified driver to the test machine
- Unzip the compressed drivers and extract the same
- Run silent install using response file generated
- DB installation on multiple machines
- Index tool installation on multiple machines
- Un installation of installed drivers after sample verification (if needed)
- Generate reports that summarizes results of the job
- Sample verification
- Scheduled download of various builds

6.3 Lack of risk based testing

Following are some of the points that need to be considered for creating security test plan:

- Creating security abuse/misuse cases
- Listing standard security requirements
- Product architecture risk
- Building risk-based security test plans
- Wielding static analysis tools
- Performing security tests
- Performing penetration testing in the final environment
- Cleaning up after security breaches

In our study we identified that we can have some improvements in security by following proper security test process.

6.4 Parallel execution of test scenarios

In our case study analysis, we spotted some areas where we can implement parallel processing concepts and improve the test efficiency. Following are some of the key points selected for improvements:

- Identifying serially executing independent steps.
- Under utilisation of hardware resources.
- Lack of Modularisation.
- Code redundancy.
- Syntactic optimization of the codes.

6.5 Lack of scheduled execution of the scenarios

In the existing test setup, test team was executing the system test by manually during the office hours. This was one bottleneck for maximum utilisation of the test hardware. Most of the time test machines were free during night time and weekends. For improving the hardware utilisation of the team, we suggested for an automatic scheduled execution of the test scenarios. As per this suggestion, tester can schedule the test execution based on machine availability. Once we schedule the execution, the tool will automatically start the execution without manual intervention.

6.6 Test Management issues

Test team was involved in multiple projects. Due to this multiple activities test manager was facing many issues in proper work allocation and tracking of the allocated work. Many times multiple activities created issues to testers for meeting the dead line. After the case study analysis we suggested a 'test point method' of tracking for ensuring that the work is allocated in a balanced way.

7. Implementation and outputs of the research

This chapter is talking about the implementation work that we carried out in the existing test setup and the major outputs of the research work.

7.1 Regression automation

Figure 7.1 shows the setup that we implemented in our case study. To address the identified problems of system test execution environment, team evaluated many tools and processes and finally shortlisted the build forge tool. IBM Rational Build Forge is an adaptive process execution framework that automates, orchestrates, manages, and tracks all the processes between each handoff within the assembly line of software development, creating an automated software factory. Rational Build Forge integrates into your current environment and supports major development languages, scripts, tools, and platforms; allowing you to continue to use your existing investments while adding valuable capabilities around process automation, acceleration, notification, and scheduling.

New regression environment implementation done after a deep analysis of the current test frame work. Following issues are addressed using the new regression setup:

- Parallel execution of test scenarios
- Scheduling of different jobs
- Log verification from a central console
- Monitoring of long running scenarios from central console
- Report generation

With the new regression setup we are able to move many system test cases to automated regression environment and thus reduced the system test cycle time. Test team was taking around

100 man days of testing effort for each release. After the implementation of the framework we are able to complete the task in 50-70 man days of effort. This product has minimum 5-7 releases per year. With this case study we are able to save 1 to 1.5 man years of testing effort.

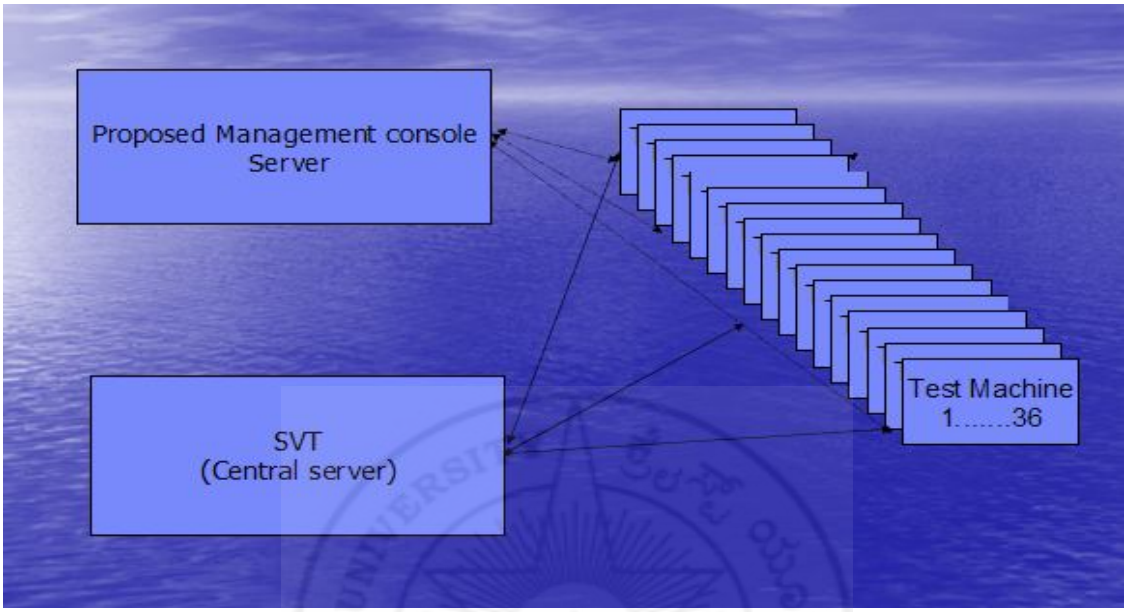


Figure 7.1

7.2 Install test automation

Manual download and installation of DB and index tool drivers are one of the painful issues we faced during system and installation testing. We addressed this issue with the development of new tool named Test Install Automation Tool (TIAT). Figure 7.2 shows the TIAT concepts that we implemented and Figure 7.3 shows the typical work flow of the TIAT tool. With this tool we are able to solve almost all the issues related with driver download and installation (Refer 6.2)

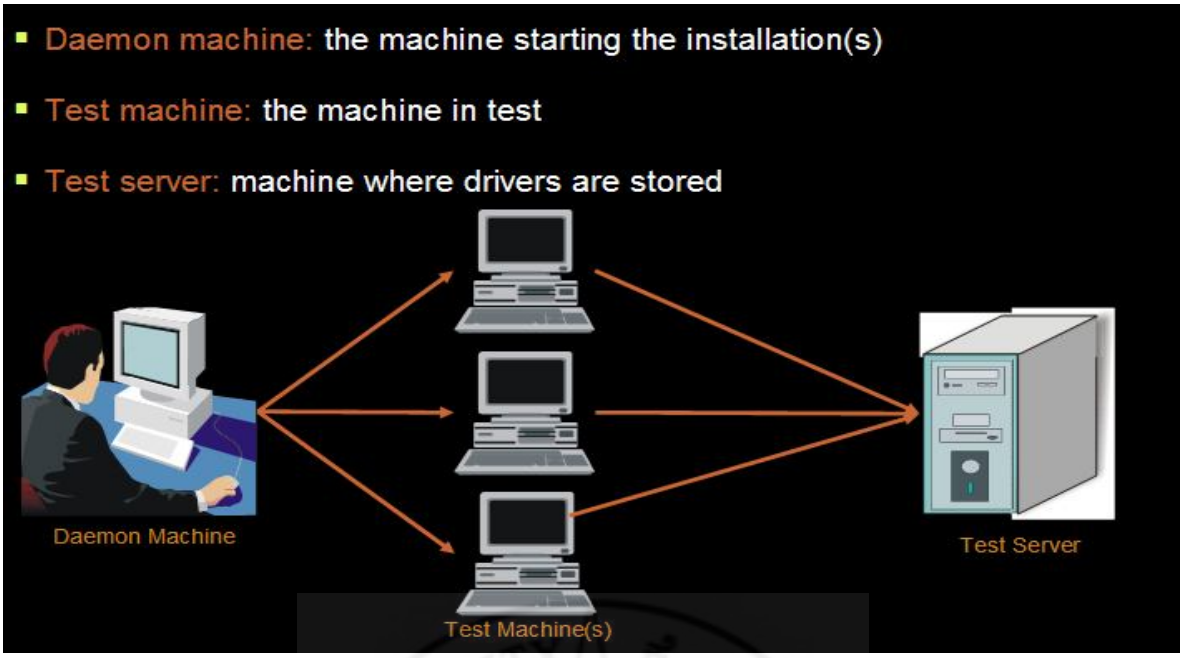


Figure 7.2

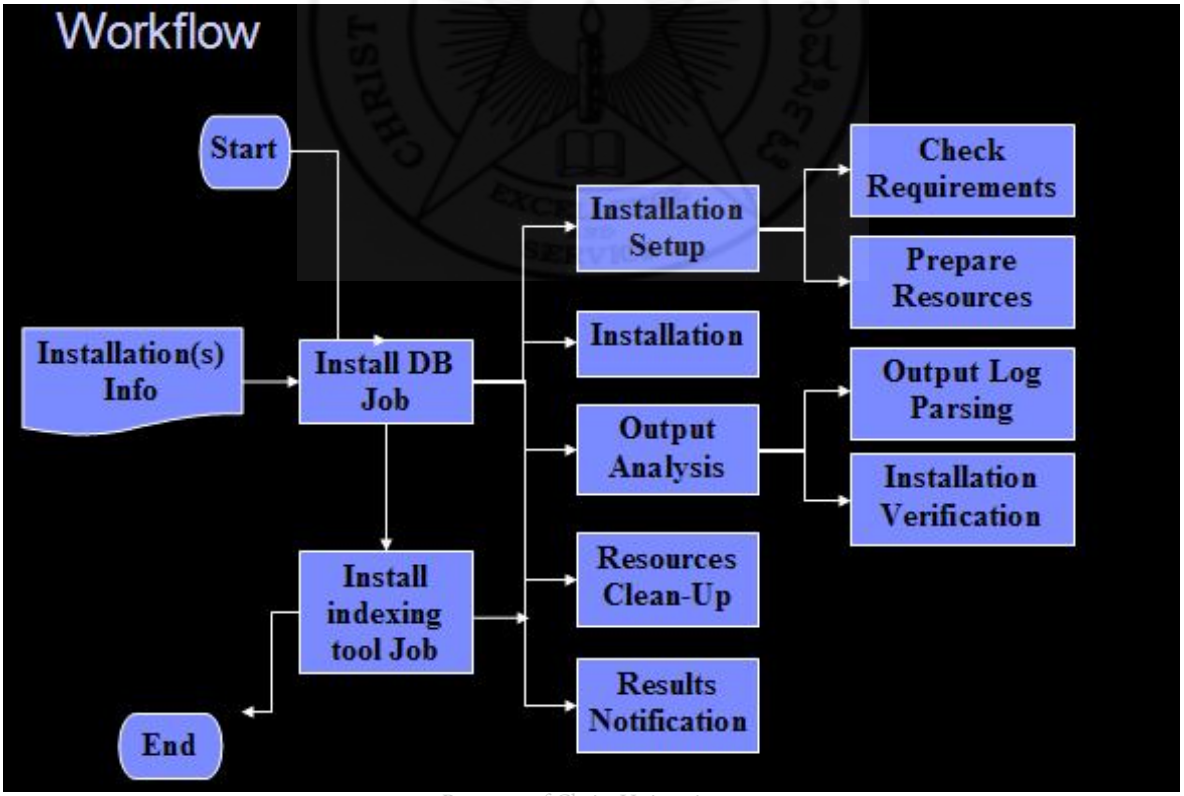


Figure 7.3

7.3 Risk based testing

Risk is the possibility of a negative or undesirable outcome, so a risk could negatively affect customer, user, or stakeholder satisfaction. Through testing, we can reduce the overall level of quality risk. Analytical risk-based testing uses an analysis of quality risks to prioritize tests and allocate testing effort. By introducing this approach we are able to achieve the following things.

- Find out the important bugs earlier in test execution, that reduces the risk of schedule delay.
- Finding important bugs than unimportant bugs, reducing the time spent chasing trivialities.
- Provide the option of reducing the test execution period in the event of a schedule crunch without accepting unduly high risks.

7.4 Parallel processing

Parallel processing is used for completing long tasks in a short duration of time. Large tasks will be divided in to multiple numbers of smaller tasks and run concurrently on several nodes /instances. Figure 7.4 shows the sequential processing of a large task and Figure 7.5 shows the execution of divided tasks in parallel.

Figure 7.4

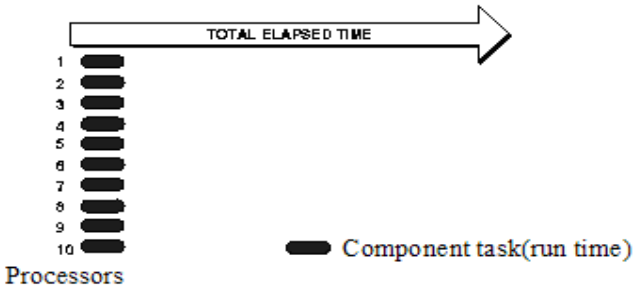


Figure 7.5

In our case study, for increasing the speed of query search on different languages and environments we applied the parallel processing techniques for completing the task in fast mode. In the initial setup, searching of the documents on different languages was happening in a sequential order. Using the new method we are able to increase the search speed of the query by splitting the sequential scripts in to multiple tasks by considering the language and environment. Figure 7.6 shows the initial setup of the search process and Figure 7.7 shows the new processing mode using parallel execution. While splitting the tasks, two types of issues we need to address.

- Structuring the task
- Preserving the sequence of tasks that need to be executed serially

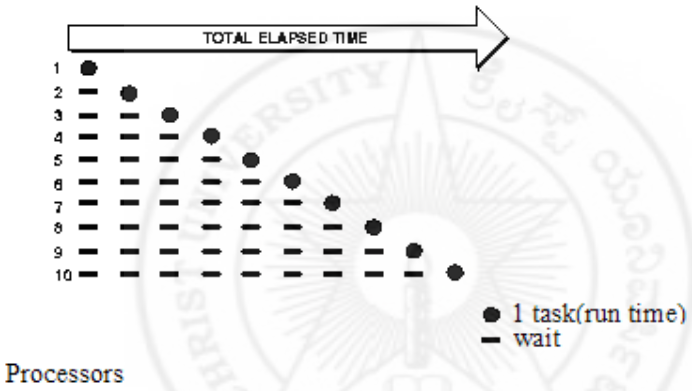


Figure 7.6

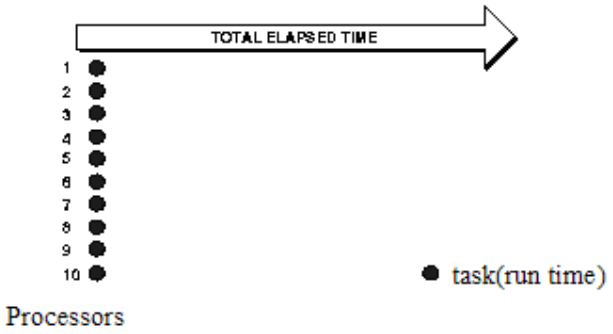


Figure 7.7

7.4.1 Database environment for parallel execution

Figure 7.8 shows the database environment and hierarchical relationship between systems, instances and databases. For running different tasks on different instances we need to set the OS environments variables and data base environment and registry variables. Following are the some of the characteristics of our parallel execution environment:

- Each language has its own instance
- Code page needs be to set for each language
- All the instances on the same server behave like separate installations of database
- All instances are sharing same database manager program files
- Each instance can run task concurrently
- Task synchronization
- To be run on using single or multiple CPUs
- Problem broken in to discrete parts that can be solved concurrently
- Each part is further broken down in to a series of instructions
- Instruction from different parts execute simultaneously on different instances /CPUs

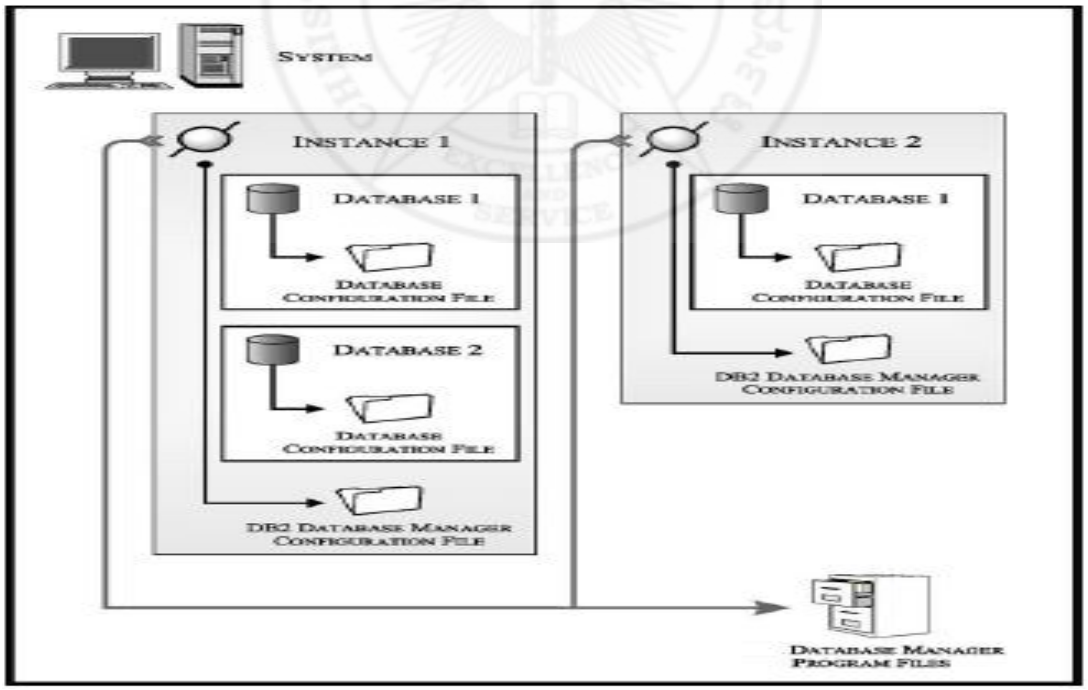


Figure 7.8

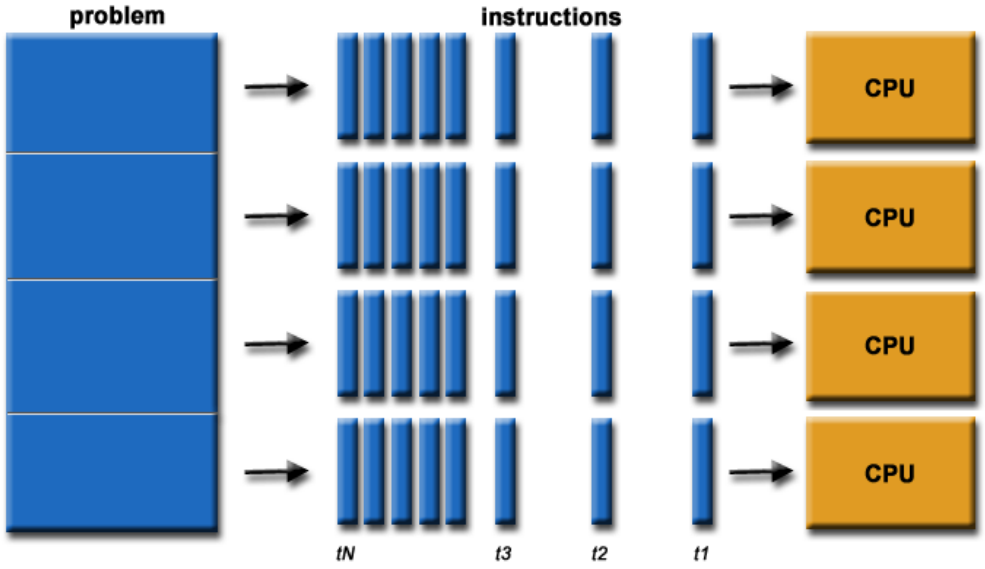


Figure 7.9

Figure 7.9 shows the pictorial representation of multiple tasks runs on different CPUs. By introducing the parallel processing in test execution, we were able to achieve the following things

- Save time: By splitting the sequential task in to multiple parallel tasks we are able to finish the search related testing in a $1/5^{th}$ of the original time
- Save money : We are able to save tester's time and machine time
- Solve larger problems : For testing the creation of larger indexes and index updates, processing of documents on multi core machine is very helpful
- Provide concurrency in execution

7.4.2 Parallel queries

This approach is introduced in our case study for analysing the reliability of the indexing tool while running multiple queries in parallel. In this setup the database will be residing on one machine and various kinds of documents (pdf, html, xml etc.) will be regularly added to the database. The index update will be happening in a regular time interval. While doing index update, parallel search queries will be sent to the database. With this scenario, we are trying to simulate a real time customer production environment. Here we use a multi threading concept. From the remote machine, we will execute the search queries as multiple threads. Below are the steps that we need to follow for running the parallel queries.

- Setup the data base machine and remote client
- Export the server and load the test setup
- Set the DB2 communication using TCP/IP
- Catalog the test server nodes and database on client machines
- Create proper directories on server machines for creating the database and indices
- Run the test suite on test server
- Run the parallel queries from the client machines after connecting to the server database
- Monitor the test execution and verify the logs for errors
- Clean up the machine after test execution

Refer Appendix I for scripts and queries

7.5 Scheduled execution

Figure 7.10 shows the screen shot of the scheduling screen in build forge execution framework. We implemented the scheduling option in our framework for maximum utilisation of the machine. With this option the test team is able to schedule the execution of regression scenarios in advance based on the availability of the machine.

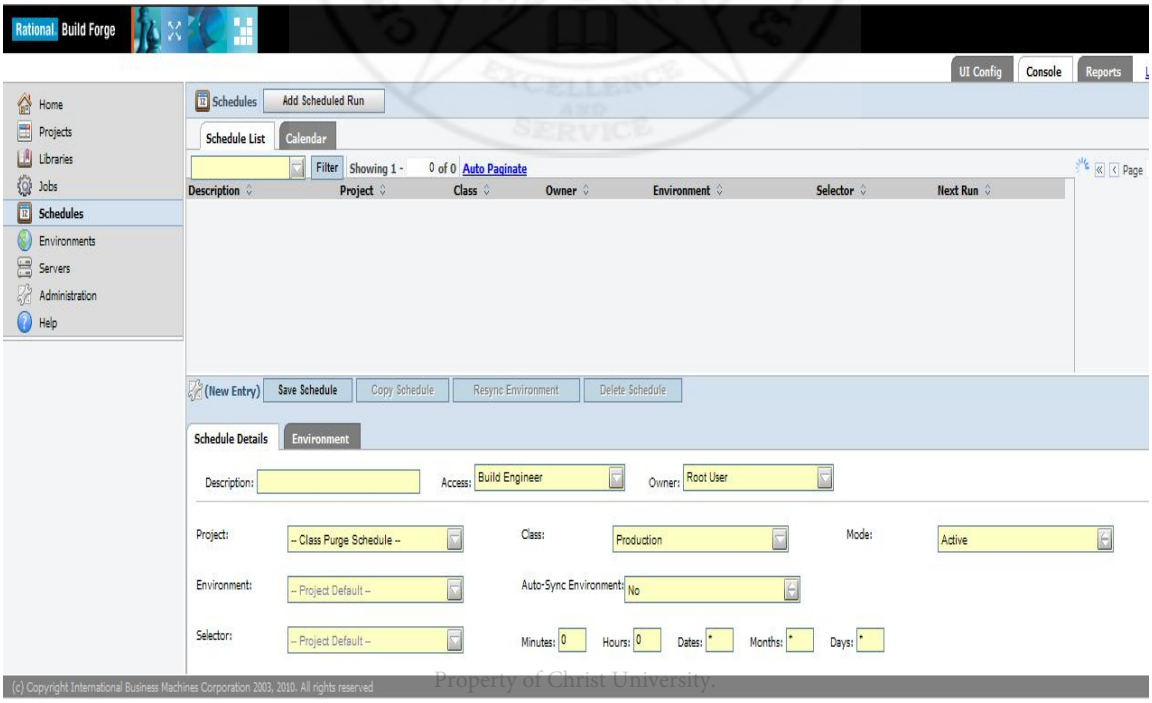


Figure 7.10

7.6 Test Management using test point method

Proper tracking of the test execution and test management was one of the improvement area we noticed in project execution. By introducing a new tool called Test Tracking Tool (TTT) and a new method named as test point method, we are able to address most of the issues we faced in this area. TTT, an IBM test tracking tool helped us to track the test status of each tester separately. TTT has multiple options to customize the tracking view based on our requirements. We used to follow a test point method, where we assigned test points to the scenarios based on the importance and duration of execution. This helped us to easily predict the duration of the test cycle. Using this method, test manager will get a clear idea, how the testing is progressing. Based on the test points completion manager can take early decisions like, whether the testing will meet the project dead line, whether the team is overloaded, is there any extra resource needed, etc. And he can make adjustments in manpower utilization based on the test point's completion. Initially we were facing problems for reporting daily progress of the testing due to the incompletion of long running scenarios. Tracking the progress on each platform (product used to test on 20 + platforms) and getting the correct report from each tester was also another painful task. This caused a lot of confusion in test management for rotating resources to other work and for re allocation of scenario to different testers. After implementing the test point method and graphical report using test points, we are able to solve the test management issues

Below example shows, how a test manager is planning the testing using test point method

Assume that 10 test points (TP) = 1 man day

After analysing the selected scenarios of the test phase, manager got total test points of 560 TP. This means he need total 56 man days of execution. Based on test start date and end date manager can easily decide the number of testers need to allocate for this test cycle. For example the manager want to finish the execution in 28 days, he can allocate 2 people for this test phase

Time allotted for test completion = 28

Total test points in test cycle = 560

Number of test points need to cover in one day = 560/28 = 20 TP

Total tester needed = 20/10 = 2 person

Based on this calculation, the manager can easily monitor the test progress and if there any shortage in execution he can easily adjust the resources with below calculations

Planned number of test points completion on N^{th} day of execution = N * 20

Actual number of test points executed on N^{th} day = M

Difference in expectation = N*20 –M

Eg: -After 10th day as per the plan we need to complete 10* 20 TP s. But actual number executed is only 140.

Difference in plan is 200-140 = 60 TP s

That means as per the plan, testing is lagging behind by 3 days.

Here test manager can change his plan by adjusting the days/resources etc.

This early planning will help the manager to avoid missing dead lines of test execution. Also with this approach he can prepare pictorial representation of test progress. See the below sample chart of execution. Using the below graph (Figure-7.11), manager will get a clear picture of the execution progress. Also he will get an idea of total defects found in test phase. With this approach the manager can handle any number of releases without any management issues.

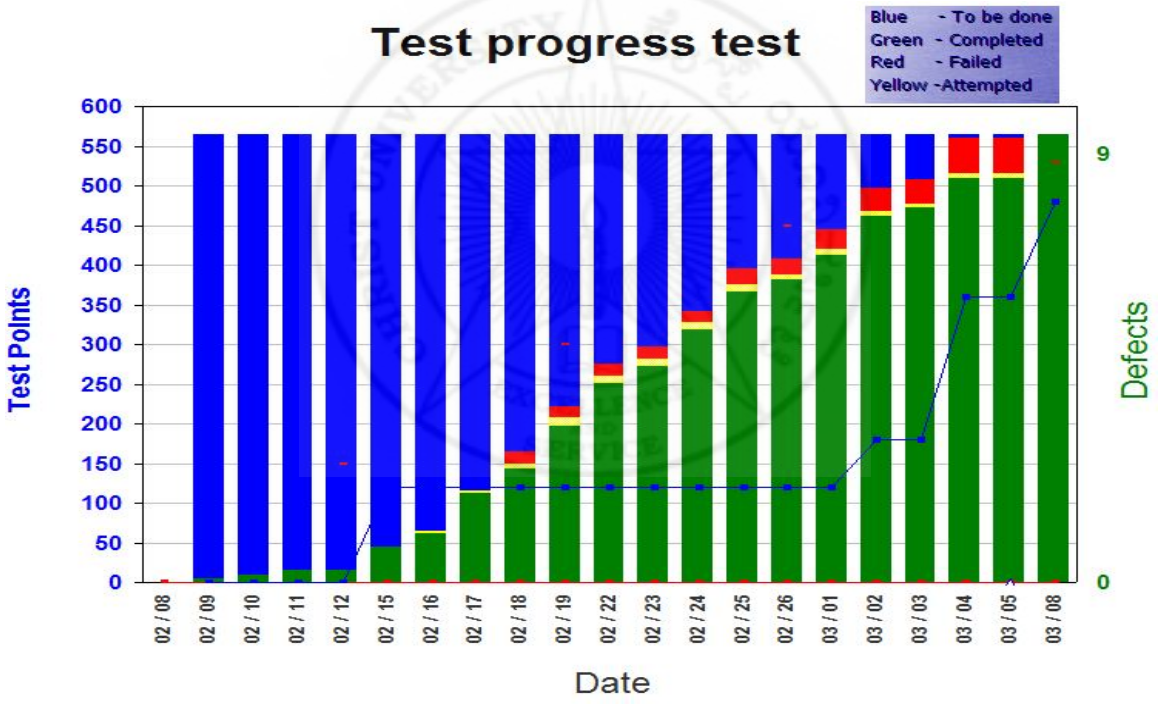


Figure 7.11

7.7 Measurable results after implementation

Based on the implementation proposal and the action taken to improve the test execution and test management of indexing tool, team was able to address the challenges specified and actually benefited in terms of test effectiveness and productivity improvement. Overall, it improved the test management. Below are some of the significant benefits and improvements achieved.

- Implemented new regression frame work and thus are able to increase the testing coverage.
- Reduced manual intervention for the system test and thus saved 30-50% system test execution time (Approximate saving of one person year).
- Introduced parallel execution of the scenarios on test machines and that helps to improve CPU utilisation.
- Report generation
- Automated the DB, index tool driver download and installation process
- Scheduled execution of test scenarios
- Introduced new method called - Test Point method - for tracking the test execution progress
- Introduced new database tool -TTT - for tracking the individual test status
- Improved hardware utilisation using VM ware and through machine sharing option
- New GUI interface
- Central console for analysing the test logs and reports

7.8 Recommendations

Based on the work done by the team in the test execution and management area, below are some of the recommendations to testing engineers. This can apply to any software testing project where there is a scope of automation and efficiency improvement.

- Know your efficiency to know what to improve
- Institute risk based testing for catching defect in early test cycle
- Setup regression frame work as early as possible and move repeatedly executing system test scenarios to regression
- Introduce light weight test automation
- Use IBM Rational Build Forge, that you can easily integrate into your current environment and support major development languages, scripts, tools, and platforms
- Try to exploit the maximum existing investments while adding valuable capabilities around process automation, acceleration, notification, and scheduling.
- Automate your install verification test (IVT) so that you can run the IVT for each and every build without manual intervention

- Introduce proper test tracking system using easily understandable graphical approach

8. Summary and Conclusions

This thesis followed a case study approach in a database environment. Throughout the research, team focused on identifying the limitation of the existing environment and suggested how we can improve the test effectiveness by using various tools, theories, standards etc. The investigations gave results showing possibilities for increasing the test efficiency. At the end of implementation stage, team got an appreciation certificate for saving a minimum of one person year of effort (Refer appendix – I).

8.1 Outputs of the research

Following are the major outputs of the research work:

- Introduced a new simple approach called test point method for easy tracking of the test execution progress
- Implemented a multi threading algorithm for simulating the customer environment for test execution
- Introduced a new algorithmic solution for solving the issues in system management with a title of "Generating index for data stored in a conf file: A new way of System management"
- Reduced manual intervention for the system test and thus saved approximate one personal year of test execution time
- Introduced new tool called TIAT for install and download

8.2 Publications

Following are the publications prepared during the thesis work:

- Effective testing: An investigative approach for improving test efficiency (International Journal of Scientific & Engineering Research, Volume 3, Issue 2, February-2012-ISSN 2229-5518)
- Software Test Automation Process –STAP – (Presented a paper in eit-12- National conference)
- Effective testing: A customized hand book for testing professionals and students- has been accepted for publication in IJSER**,** which will be published in IJSER Volume 3, Issue 5, May 2012
- Generating index for data stored in a conf file: A new way of System management: This algorithm Published in IBM developer works article
- Effective testing: A combinatorial approach for improving test efficiency – Wrote a paper and waiting for submitting to an international conference on testing

8.3 Validation of hypothesis

From the project results we validate the hypothesis as follows

- Manual intervention for running the test suites: After evaluation of the existing test setup we came to know that the reason for the less test efficiency is due to manual execution of the test suites and for reducing the same we decided to integrate all the test suites in an automation frame work and selected the Rational build forge for the same.
- Manual download and installation of DB and indexing tool drivers: Install verification of each build for the DB and indexing tool was very repetitive and time consuming task. The team was spending around 2-3 person months for install test. We decided to automate the same and introduced a new tool TIAT for the same.
- Parallel execution of different scenarios on different machines from central point: We investigated the existing lab setup of the test environment and came to know that the CPU utilisations of the various machines are very less. For improving the test efficiency we introduced **instance level and processor level parallelism**.
- Scheduled execution of multiple scenarios: Due to manual execution of the test suites, most of the time, during nights and weekends the machines were free. We implemented a scheduled execution for addressing the issue. With this mechanism we are able to improve the hardware utilisation during nights and other non office hours.
- Tracking the test status for multiple service packs: Introduced a new tool called Test Tracking Tool for point wise tracking of the each service packs.

- Tracking the test progress for each service pack: Introduced a test point method for proper tracking of the test progress using graphical method.
- Tracking the test status of the individual test team members: Test tracking tool has an option for individual tracking of the tester.

8.4 Future work

Following are the areas that we planned for further investigations related to efficiency improvements of the testing:

- Effective testing using combinatorial testing approaches
- Reliability analysis of real time system
- Issues in test management for outsourced testing projects
- Issues in Knowledge transfer

Appendix I: Test script sample

This portion contain the scripts that are used for setting up the customer environment (Refer 7.4.2)

[Environment setup: Common to database machine and remote client]

Export Server= <sever machine>

. ttcsetup nse svt R2-3

[DB2 communication]

db2set DB2COMM=TCPIP

export SvceName=DB2_"$DB2INSTANCE"

db2 Update DBM CFG Using SvceName $SvceName

db2stop

db2start

[On remote clients]

db2 Catalog TCPIP Node $Server Remote $Server Server $Port Number

Eg: - db2 Catalog tcpip node node3 remote sunspool.in.ibm.com server 60000

db2 Catalog Database A819UPD At Node $Server

db2 Catalog db A819UPD at node node3

db2 Terminate

db2 List Node Directory

db2 List DB Directory

[On database machine]

mkdir -m 777 -p "$TTC_DATAROOT"/nsesvt/indices/"$USER"/A819UPD

mkdir -m 777 -p "$TTC_DATAROOT"/nsesvt/work/"$USER"/A819UPD

testall -noabortonfail -suite: index_update_1000x

[On remote clients: Run queries and check output]

export Parallel Queries=25

db2 Connect To A819UPD User nseqlwe Using test@123

rexx /svttest/ttcR1/nsesvtR2-3/tools/lnb_test_tools/longLastQuery_aix_special.rex $Parallel Queries $Prefix

[On database machine: Check for locks while queries are running]

Open a new server session and do the ttcset up

Check for locks periodically:

rexx /svttest/ttcR1/nsesvtR2-3/tools/lnb_test_tools/lockManagerCheckPlusPlatform.rex <number of loops> <sleep time interval> <platform (Unix or windows)>

Eg:- rexx $TTC_ROOT/tools/lnb_test_tools/lockManagerCheckPlusPlatform.rex 1000 10 Unix

[On database machine: Check for locks after the test is done]

rexx /svttest/ttcR1/nsesvtR2-3/tools/lnb_test_tools/lockManagerCheckPlusPlatform.rex 1 0 Unix

[On remote clients: Check output files for NSE errors]

grep -in CTE21 /ttcdata/"$Prefix"_mass_search_special* > /ttcdata/"$Prefix"_CTE21.txt

grep -in CTE0116 /ttcdata/"$Prefix"_mass_search_special* > /ttcdata/"$Prefix"_CTE0116.txt

grep -in CTE0119 /ttcdata/"$Prefix"_mass_search_special* > /ttcdata/"$Prefix"_CTE0119.txt

grep -in CTE0157 /ttcdata/"$Prefix"_mass_search_special* > /ttcdata/"$Prefix"_CTE0157.txt

[Cleanup activities]

[On remote clients]

db2 Uncatalog Database A819UPD

db2 Uncatalog Node $Server

db2 Terminate

[On database machine]

export DB2DBDFT=A819UPD

db2 Disconnect All

db2text Drop Index Admin.Index For Text

db2text Disable Database For Text

db2text Stop

db2 Deactivate Database $DB2DBDFT

db2 Drop Database $DB2DBDFT

Script to create one long lasting process to run queries

```
/*********************************************************************/
/* Script to create one long lasting process to run queries          */
/* Done by Abdul  Rauf 31/01/2012                                    */
/* Syntax: longLastQuery_aix_special.rex                             */
/*********************************************************************/
parse arg queryLoops myId.
QueryThreads = queryLoops
if queryThreads = '' | myId = ''then do
```

```
        say "Please enter number of threads"
        say "Syntax is longLastQuery_aix_special.rex <number of threads> <my Userid>"
end

/* trace 'ia' */
i = queryThreads
USER = myId
do while i>0
/*    "`sh    -c    'db2    -tvf    /svttest/ttcR1/nsesvtR2-3/db2_search/mass_search_extended.tvf
1>/ttcdata/"||USER||"_mass_search_extended.out"||i
"2>/ttcdata/"||USER||"_mass_search_extended_err.out"||i "`  &" */
          "`sh    -c    'db2    -tvf    /svttest/ttcR1/nsesvtR2-3/db2_search/mass_search_special.tvf
1>/ttcdata/"||USER||"_mass_search_special.out"||i
"2>/ttcdata/"||USER||"_mass_search_special_err.out"||i "`  &"
        say "This is the:" i "run"
        i=i-1
end
exit
```

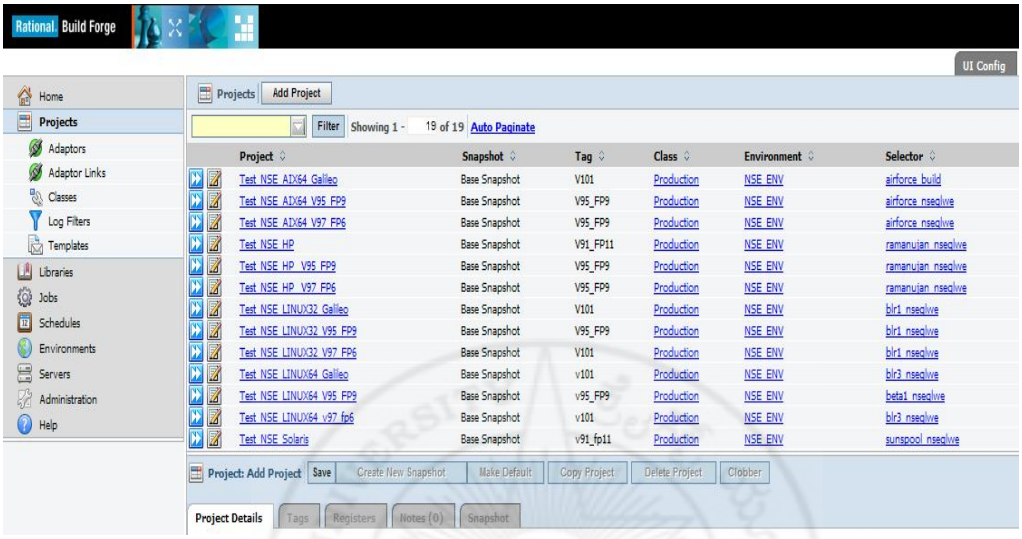
Sample content of the mass_search_special.tvf

```
**************************************************************************
-- DB2 Net Search Extender Test
-- Version:      9.1
-- Usage: Call from the operating system command line:
-- db2 -tvf search
-- For Windows NT: first enter the db2 command line environment
-- with db2cmd command
**************************************************************************
connect to A819UPD user db2admin using test@123;

-- 1. Query

   SELECT count (FILENAME)
      FROM AXML.MASS
      WHERE CONTAINS (CLOB_DOC, '"Tippettstudio"') = 1;

-- 2. Query
   SELECT count (FILENAME),
      NUMBEROFMATCHES (CLOB_DOC, '"Tippettstudio"')
      FROM AXML.MASS;

-- 3. Query
   WITH TEMPTABLE (DocSeqNo,score)
      AS (SELECT DocSeqNo,
            SCORE (CLOB_DOC,'"Tippettstudio"')
         FROM AXML.MASS)
```

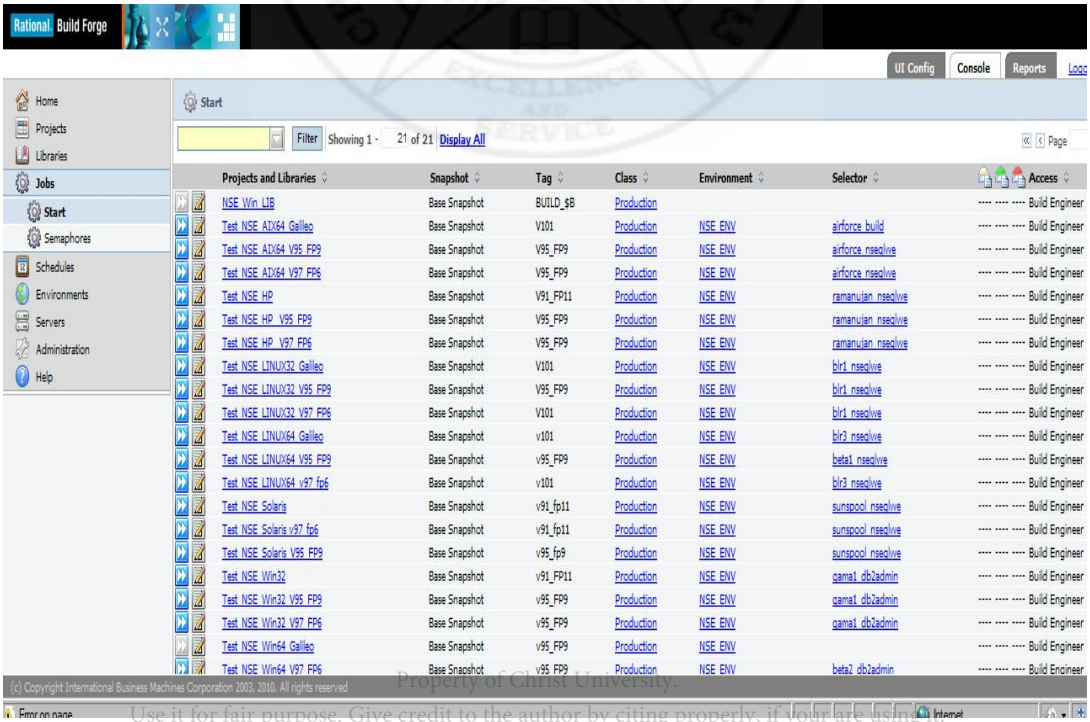
Appendix II: Screen shots

This section will give an idea about the various screens of the regression test environment

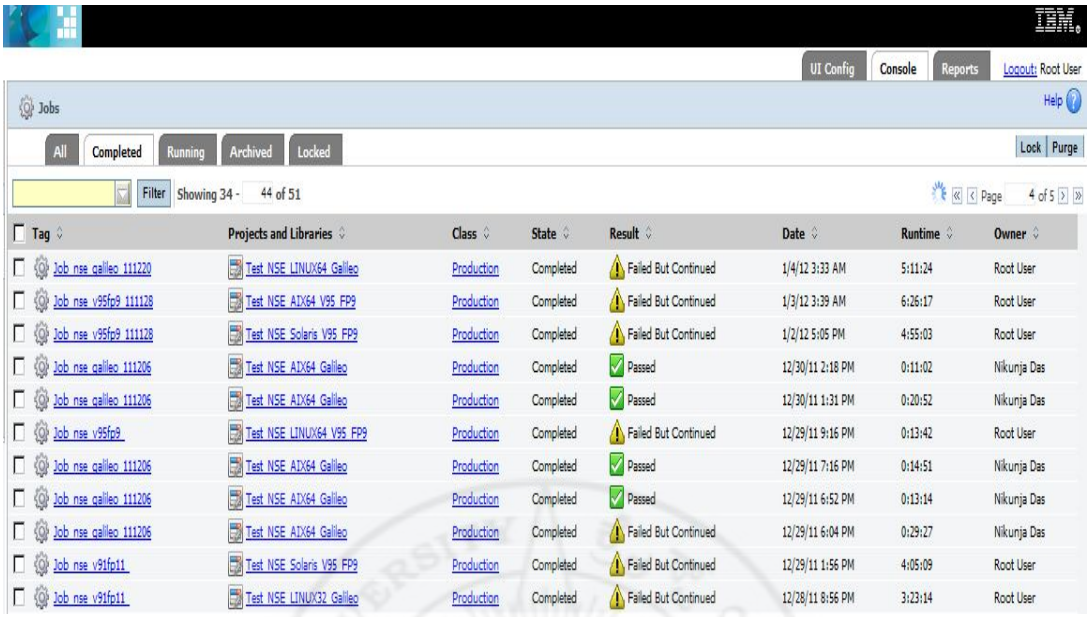
Screen 1- Build forge project screen



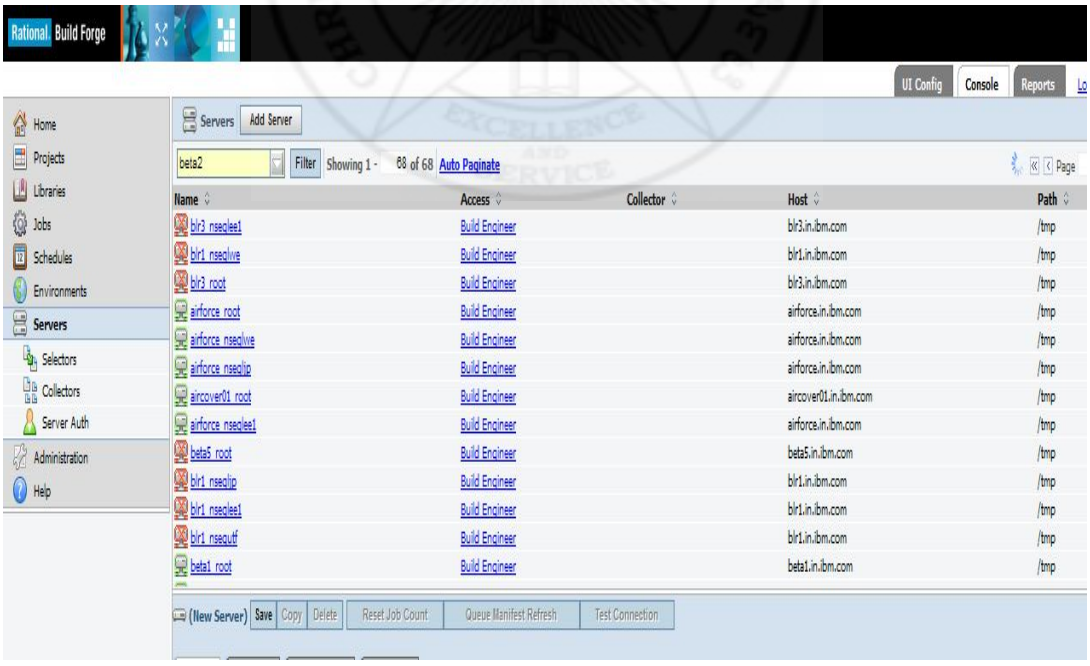
Screen 2- Build forge project execution screen



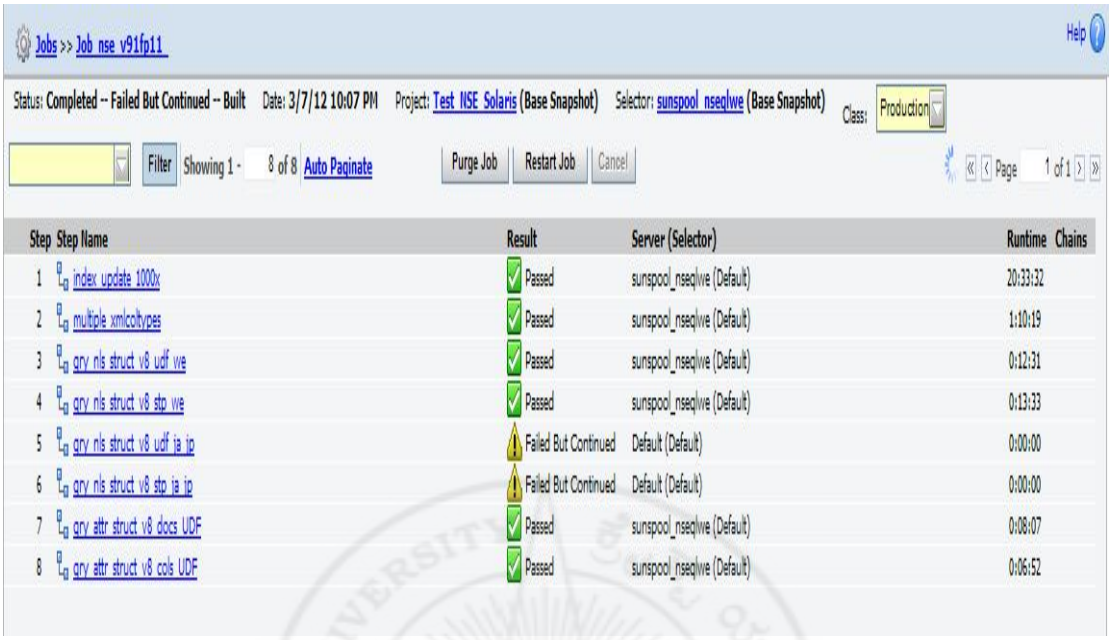
Screen 3- Executed job status screen



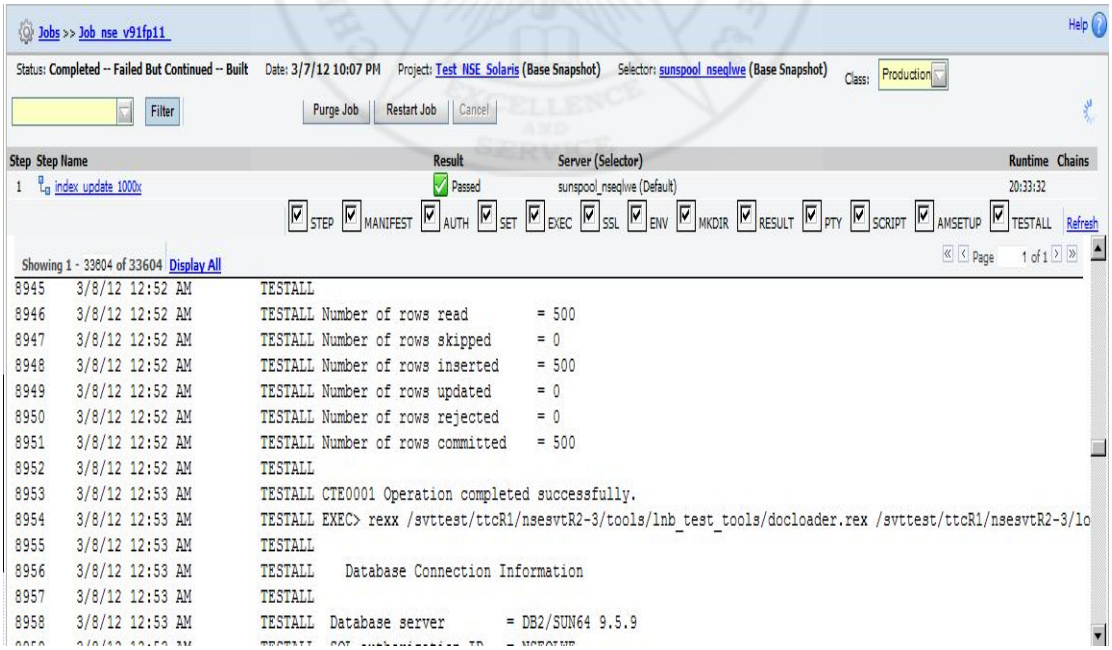
Screen 4- Server added to regression environment



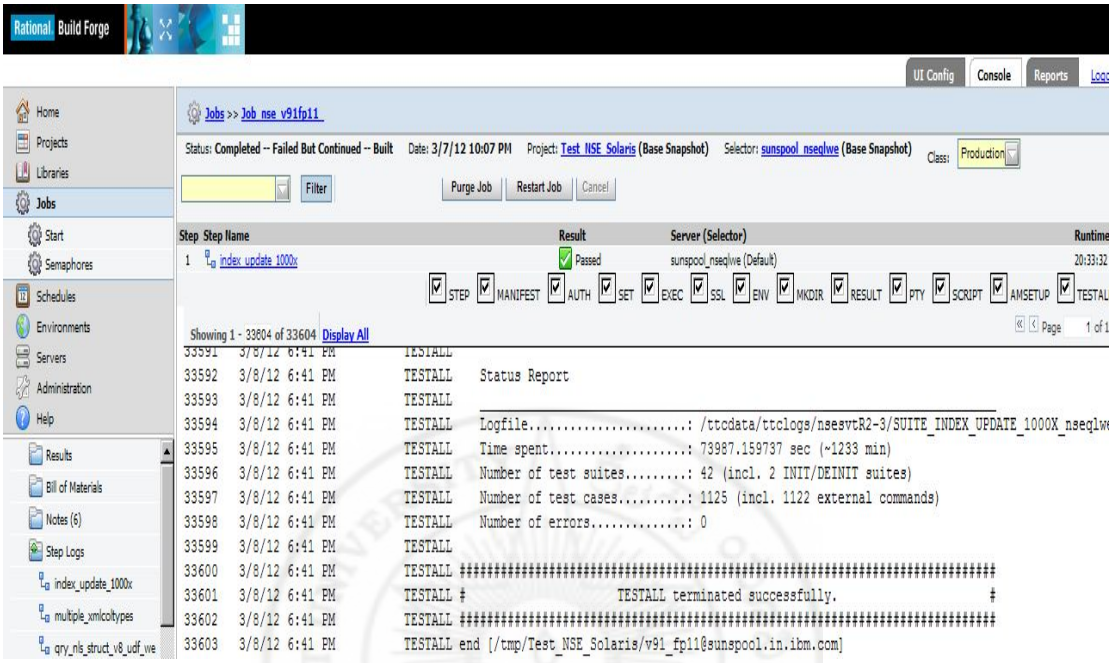
Screen 5- Suite wise execution status



Screen 6- Log display screen

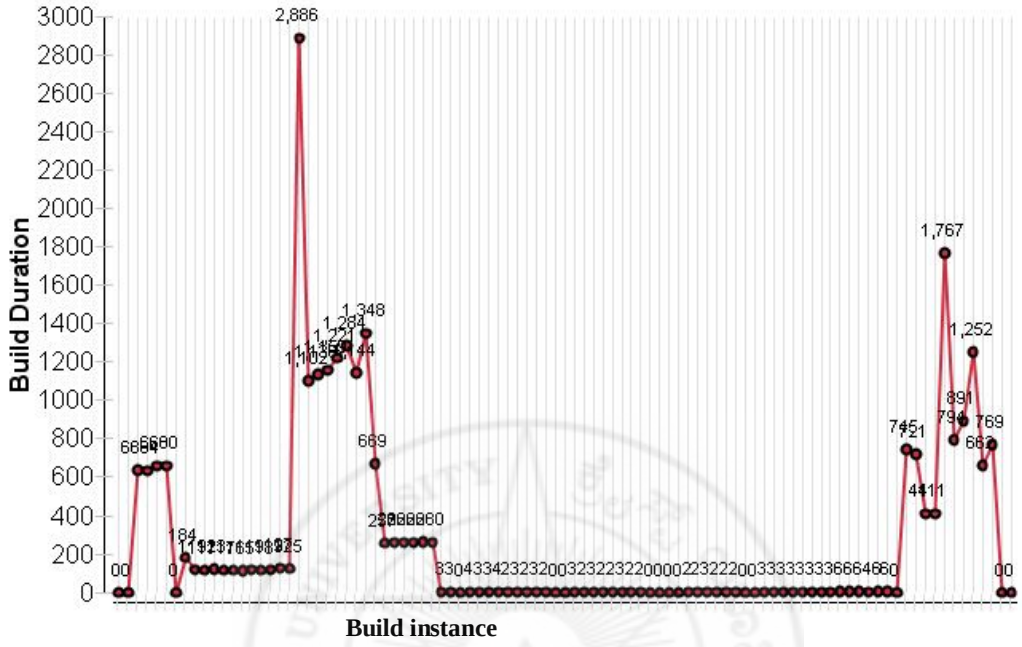


Screen 7- Log summary

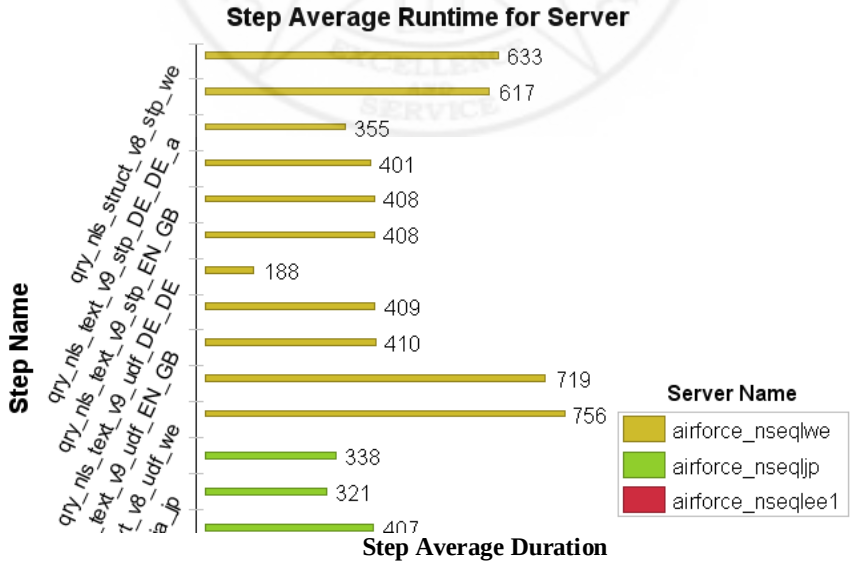


Screen 8- Project wise execution summary

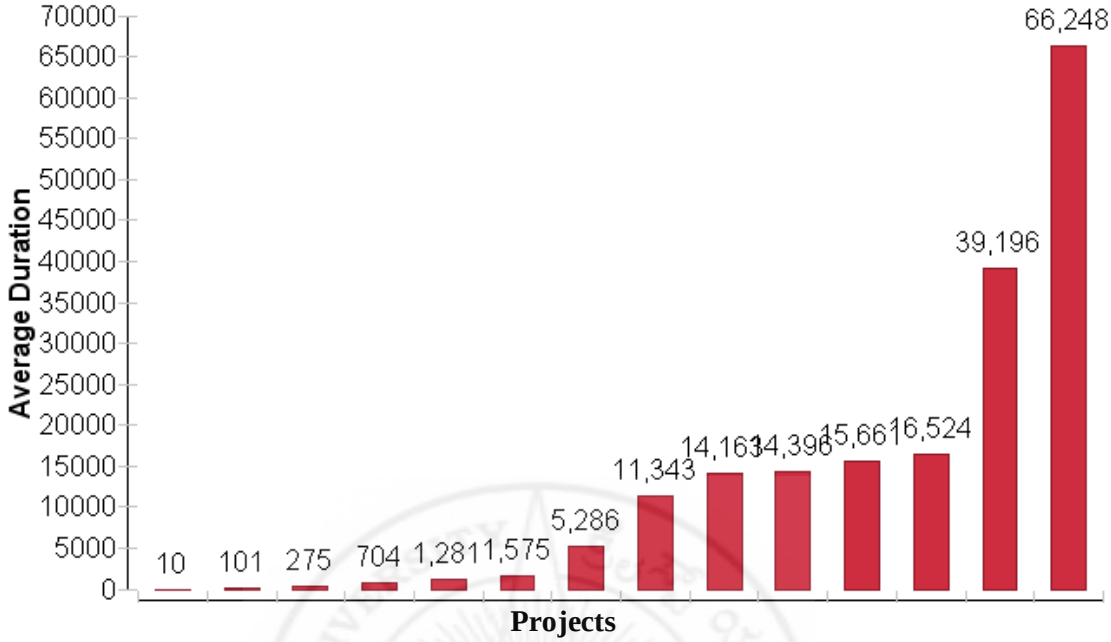
Screen 9 - Build duration Vs Build instance graph



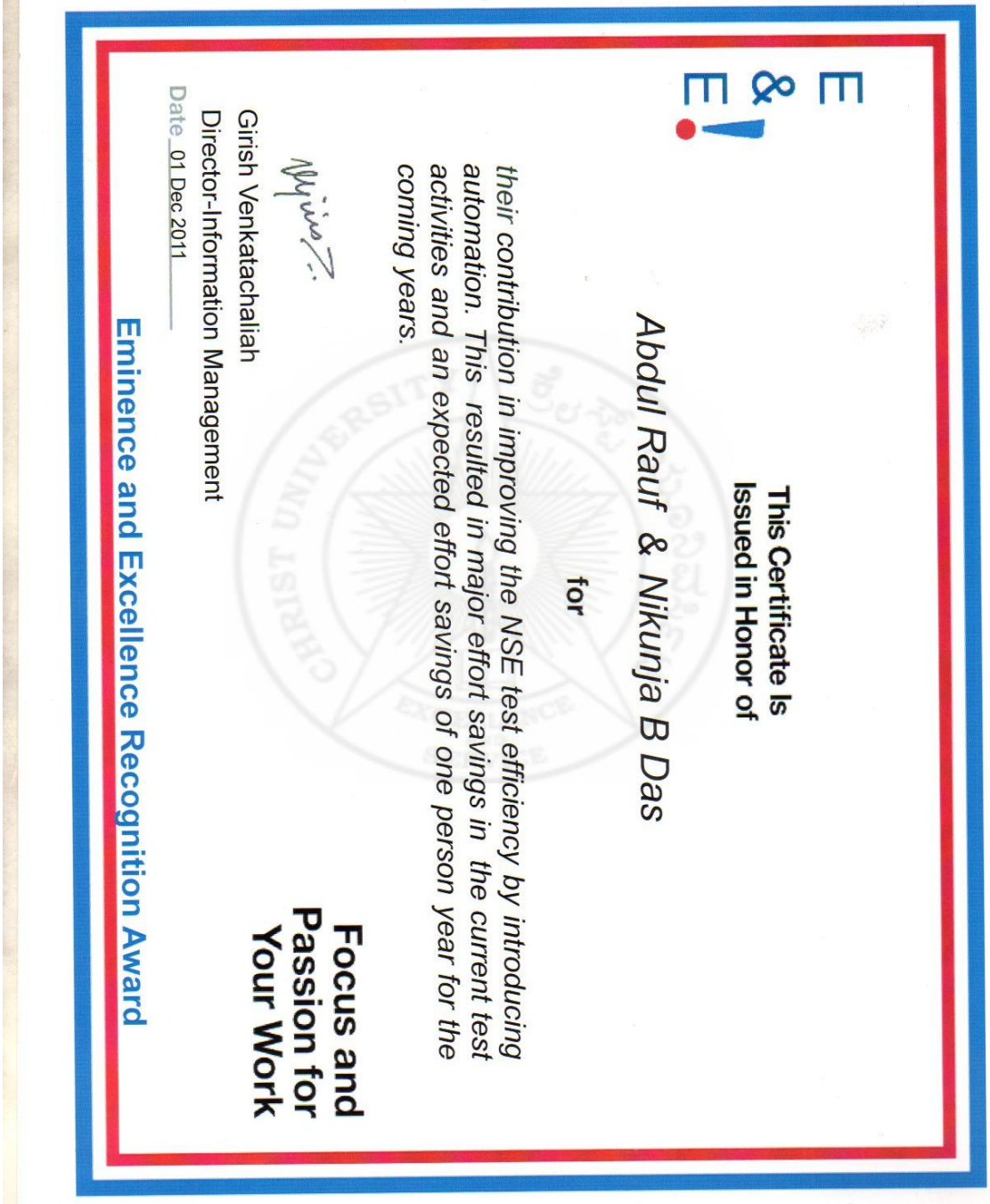
Screen 10 - Scenario runtime graph



Screen 11 – Sample capacity report



Appendix III: Recognition certificate



E & E

This Certificate Is
Issued in Honor of

Abdul Rauf & Nikunja B Das

for

their contribution in improving the NSE test efficiency by introducing automation. This resulted in major effort savings in the current test activities and an expected effort savings of one person year for the coming years.

Focus and Passion for Your Work

Girish Venkatachaliah
Director-Information Management

Date 01 Dec 2011

Eminence and Excellence Recognition Award

Appendix IV: Paper presentation certificate

CHRIST
UNIVERSITY
Bangalore, India
Declared as Deemed to be University under Section 3 of UGC Act 1956

Certificate

This is to Certify that

Mr. ABDUL RAUF E.M.

authored/~~co-authored~~/presented a paper titled

Software Test Automation Process [STAP]

in the **Third National Conference on Emerging Trends in IT eit12** organised by Department of Computer Science and Faculty of Engineering, Christ University, Bangalore, held on 2 March 2012

Dr (Fr) Thomas C. Mathew
Vice Chancellor

Prof. Joy Paulose
Organizing Chair

References

[1] Cem Kaner, Jack Falk, Hung Quoc Nguyen, *'Testing Computer Software'* 2nd Edition, 2001, ISBN: 81-7722-015-2.

[2] Boris Beizer, *'Software Testing Techniques'*, 1st Reprint Edition, 2002, ISBN: 81-7722-260-0.

[3] Booz Allen Hamilton, Gary McGraw, *'Software Security Testing'*, IEEE SECURITY & PRIVACY, 2004, PP 1540-7993.

[4] *'Test Plan Template'* (IEEE 829-1998 Format), 2001, Software Quality Engineering - Version7.0.

[5] Toshiaki Kurokawa, Masato Shinagawa, *'Technical Trends and Challenges of Software Testing'*, Science & Technology Trends, 2008 – Quarterly review no.29.

[6] IBM Rational build forge V 7.13 – Information Center document.

[7] Viraj Kumbhakarna, *'A Practical Approach to Process Improvement Using Parallel Processing'*, PharmaSUG2011 - Paper PO03.

[8] Lars-Ola Damm, *'Evaluating and Improving Test Efficiency'*, Master Thesis, Software Engineering, June 2002, Thesis no: MSE-2002-15.

[9] K. Burr and W. Young, *'Combinatorial Test Techniques: Table-Based Automation, Test Generation, and Test Coverage'*, Proc. Int'l Conf. Software Testing, Analysis, and Review (STAR), 1998; http://aetgweb.argreenhouse.com/papers/1998-star.pdf .

[10] Rick Kuhn, Yu Lei and Raghu Kacker, *'Practical Combinatorial testing: Beyond Pair wise', 2010* \http://csrc.nist.gov/groups/SNS/acts/itpro-final.pdf .

[11] D. Richard Kuhn, Raghu N. Kacker, Yu Lei, *'Practical Combinatorial testing'*, NIST Special Publication 800-142.

[12] Elfriede Dustin, http://www.combinatorialtesting.com, December 2011.

[13] Ambler S.W., *'Introduction to Test Driven Development, 2006'*. http://www.agiledata.org/essays/tdd.html, December 2011.

[14] Binder R., *'Testing Object-oriented Systems'*, Addison-Wesley, 1999.

[15] Marciniak, J., *'Encyclopedia of Software Engineering'*, John Wiley & Sons Inc, 1994, ISBN 0-471-54004-8.

[16] IEEE Std. 610.12-1990, *'Standard Glossary of Software Engineering Terminology'*, 1990.

[17] A Nagappan , *'Linux Desktop Testing Project – LDTP tutorial'* http://ldtp.freedesktop.org, August 2011.

[18] IBM , *'Federated Integration Test (FIT)'* http://salwiki.rtp.raleigh.ibm.com./confluence/display/fit, August 2011.

[19] SQS Software Quality Systems AG, *'Software test automation –White paper'*, August 2010.

[20] Rex Black, *'Advanced Software Testing vol.1'*, Fifth Indian Reprint ,2011, ISBN: 13-978-81-8404-698-4.

Acknowledgements

Many people contributed to this dissertation in innumerable ways, and I am grateful to all of them. First and foremost, I would like to express my sincere gratitude to my advisor V Balaji for the continuous support of my Mphil study and research, for her patience, motivation, enthusiasm, and immense knowledge. Her guidance helped me in all the time of research and writing of this thesis.

I would like to thank Mohammed Shaffi - Project Manager - IBM India, who has given an opportunity for conducting the case study in IBM Lab.

I wish to express my sincere thanks to Nikunja B Das -Test Engineer - IBM India, who helped in automation works for making the regression test environment available for this work

My sincere thanks to Saleema. J.S for her valuable advice and friendly help. Her extensive discussions around my work was very helpful for this study

During this work I have collaborated with many colleagues for whom I have great regard, and I wish to extend my warmest thanks to all those who have helped me with my work in the Department of Computer Science, Christ University – Bangalore – India

I owe my loving thanks to my parents, Ahammed Kutty and Fathima, who have been a constant source of support

My wife Sajna P.V and my son Muhammad Razi, has been, always, my pillars, my joy and my guiding light, and I thank them.

Last but not the least, I would like to thank God Almighty for giving me the chance, the strength and patience for doing this research work.