Priority based Hybrid Automation Testing

Thesis submitted in partial fulfillment of the requirements for the award of degree

of

Masters of Technology

in

Computer Science and Applications

Submitted By
Priyanka
(Roll No. 601103013)

Under the supervision of
Mrs. Sunita Garhwal
Assistant Professor, SMCA

School of Mathematics and Computer Applications
Thapar University
Patiala –147004

July 2013

Certificate

I hereby certify that the work which is presented in the thesis entitled, **"Priority based Hybrid Automation Testing"**, in partial fulfilment of the requirements for the award of degree of the Master of Technology in **Computer Science and Applications,** submitted in School of mathematics and Computer Applications of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of **Mrs. Sunita Garhwal** and refers others researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other university.

Priyanka
801103013

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

15/7/13

Mrs. Sunita Garhwal
Assistant Professor
School of Mathematics and Computer Applications

Countersigned by

Dr. Rajesh Kumar
Head
School of Mathematics and Computer Applications
Thapar University
Patiala

Dr. S.K. Mohapatra
Dean Academic Affairs
Thapar University
Patiala

Acknowledgement

I would like to acknowledge everyone who supported me during my experience at Thapar University and my work on this thesis.

First of all, I would like to thank my family, who stayed with me and supported me all the time while I made this dream come true. I extend my thanks to my thesis supervisor, Mrs. Sunita Garhwal, who helped me to materialize my ideas on this thesis. Her enthusiasm and optimism made this experience both rewarding and enjoyable. Most of the novel ideas and solutions found in this thesis are the result of our numerous stimulating discussions. Her feedback and editorial comments were also valuable for writing this thesis.

My thesis work would be incomplete without thanking my friends who were always there in the hour of need.

Abstract

Software testing is an important means to ensure software quality and to improve software reliability. The rapid development in the software industry has raised new demands and challenges for improving the quality of software testing. The software applications once developed need to be maintained as the frequent updates in the software makes them vulnerable without rigorous testing. As the size and complexity of software applications is continually growing, manual testing becomes infeasible. Automation of the software testing process saves time and provides better utilization of resources and thus, plays a significant role in the testing activity.

The proposed hybrid approach extends support to automatic testing by providing functionality of both GUI and Code based testing. Moreover, it also prioritizes modules under test to achieve a certain degrees of confidence in the testing of applications that are under tight project deadlines. The proposed approach allows to combine the benefits of both GUI and Code based testing while keeping a simple interface, and by treating the two types of tests in a unified fashion.

List of Figures

List of Tables

Table of Contents

Chapter 1

Introduction

This chapter includes basic introduction of software testing and its types.It also consists of brief introduction of the automation testing and its advantages over the manual testing.

1.1 Software Testing

Software Testing is regarded as an integral part of software development process. It is directly related to software quality and reliability. It is the process of analyzing a software item to detect the differences between existing and required conditions (that is, bugs) and to evaluate the features of the software item. The main purpose of software testing is not only to reveal bugs and eliminate them but also to serve as a tool for verification, validation and certification. Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. It also provides an objective, independent view of the software to allow the business to appreciate and understand the risks at implementation of the software [1]. Software systems are getting more and more important for organizations and at the same time they are growing bigger and more complex. Thus, the importance of software quality is also rising.

Quality is one of the most important aspects of software products. If the software does not work, it is not worth a lot. The drawbacks caused by faulty software can be much higher than the advantages gained from using it. In highly competing markets quality may determine which software product is going to be a success and which ones are going to fail. Low quality software products have a negative impact on the firm's reputation and unquestionably also on the sales. For these reasons organizations have to invest in the quality of the software products.

1.2 Essential Elements of Testing

The five essential elements required for successful software testing and to improve the effectiveness and efficiency of testing are as follows [2]:

- *Test strategy:* The purpose of testing is to find defects, not to pass easy tests. A test strategy basically tells which types of testing seem best to do, the order in

which to perform them, the proposed sequence of execution, and the optimum amount of effort to put into each test objective to make testing most effective.

- *Testing plan:* A testing plan is a simply a part of project plan that deals with the testing tasks. It details who will do which tasks, starting when, ending when, taking how much effort, and depending on which other tasks.
- *Test cases:* They are prepared in advance in the form of detailed examples and are used to check whether the software actually meet its requirements or not.
- *Test data:* In addition to execution of test cases, a tester needs to come up with test data to use. This often equals sets of names, addresses, product orders, or whatever other information the system uses. A starting database is also required to test query functions, change functions and delete functions in addition to the examples to input.
- *Test environment:* It is used to carry out testing process. A tester needs a place to perform the testing and the right equipment to use. Unless the software is very simple, one PC will not suffice. Test environments may be scaled-down versions of the real thing, but all the parts need to be there for the system to actually run.

1.3 Test Information Flow

Testing is an activity performed for evaluating software quality and for improving it. Hence, the goal of testing is systematically detection of different classes of errors. Error can be defined as a human action that produces an incorrect result[3]. The objective is to design tests that systematically uncover different classes of errors with a minimum amount of time and effort. Therefore, they have a good chance of finding a yet undiscovered error.

In the following figure (Figure 1.1):

- The software configuration includes software requirement specification, a design specification and source code.
- A test configuration includes the test plan, procedure, test cases and testing tool.
- Evaluation is performed by comparing the test results with the expected results.

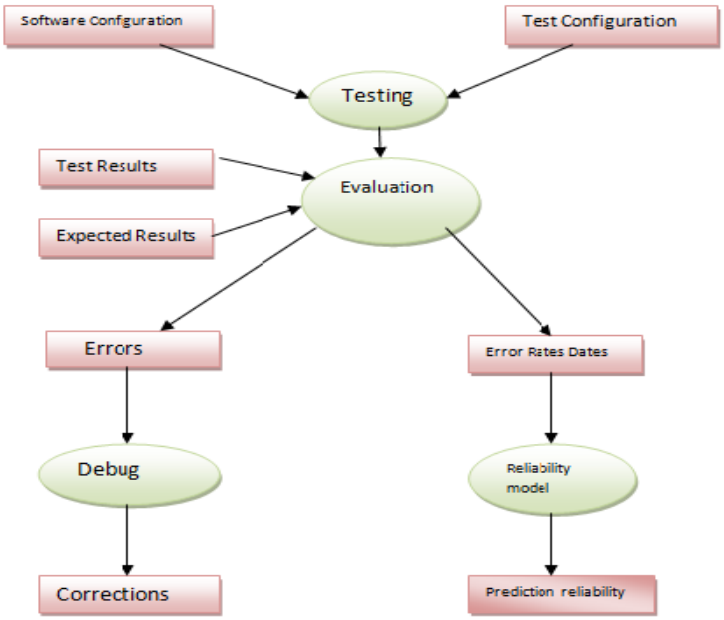

Figure 1.1: Test Flow Diagram[3]

1.4 Software Testing Life Cycle

Software Testing Life Cycle (STLC) identifies the types of test activities that should be carried out to accomplish quality assurance process in a software development project [4]. Software testing has its own life cycle that intersects with every stage of the SDLC either it is Waterfall, Spiral or Agile. However, STLC varies from one to another based on size of project, test team, test in Scope/out of scope, and code release date. So, knowledge about some of the major phase in STLC,quality assurance activities during phases and role of a tester is necessary to accomplish the task with a mark.STLC is a group of testing steps followed to deliver quality product. Testing has become an important phenomenon during and after development of any software project.

The various steps in testing process are as follows:

- Test Planning: In this phase, project managers and software tester has to decide the things that are needed to be tested and to prepare testing time lines. Some Quality Assurance activities during planning phase are:
  i) High level test plan
  ii) Identify review process, metrics
  iii) Bug reporting procedures

iv) Acceptance criteria for quality assurance

v) Schedule

- Test Design**:** Once the test plan is decided up on, next step is to decide what types of testing should be carried out at different stages of SDLC. Secondly, it is also decided whether there is anyneed or plan to automate. If there is a need for automation, appropriate time to automate and type of specific documentation needed for testing is determined. The test design phases are:

  i) Prepare functional test case documents

  ii) Negative value case document

  iii) Prepare validation test document

  iv) Finalized test environment is decided

- Construction and Verification**:** In this stage all testing activities are carried out. Some Quality Assurance activities during analysis phase are as follows:

  i) Develop test case format and validation matrix

  ii) Develop and plan test cycles matrices and time lines

  iii) Begin writing test cases based on functional validation matrix

  iv) Map baseline data to test cases to business requirements

  v) Identify type of testing i.e. automation or manual

  vi) Testing environment, automation system setup

- Testing Cycles***:***In this stage, tester performs all the test cycles until test cases are executed without errors.

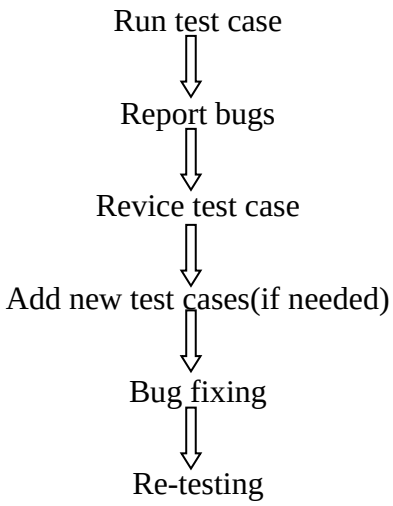

Figure 1.2: Testing Cycle [4]

- Final Testing and Implementation: In this stage, the tester executes all the incomplete test plans and re-testing activities. Following are the steps that tester needs to follow:
  i) Code freezing
  ii) Run test cases for including performing level.
  iii) Communicate defect tracking metrics.
  iv) Uncover new software bugs
  v) Documentation
- Post Implementation: In this stage, line of attack to avoid similar problems in future projects is identified and plans are created to improve the processes. The recording of new errors and enhancements is an on-going process.

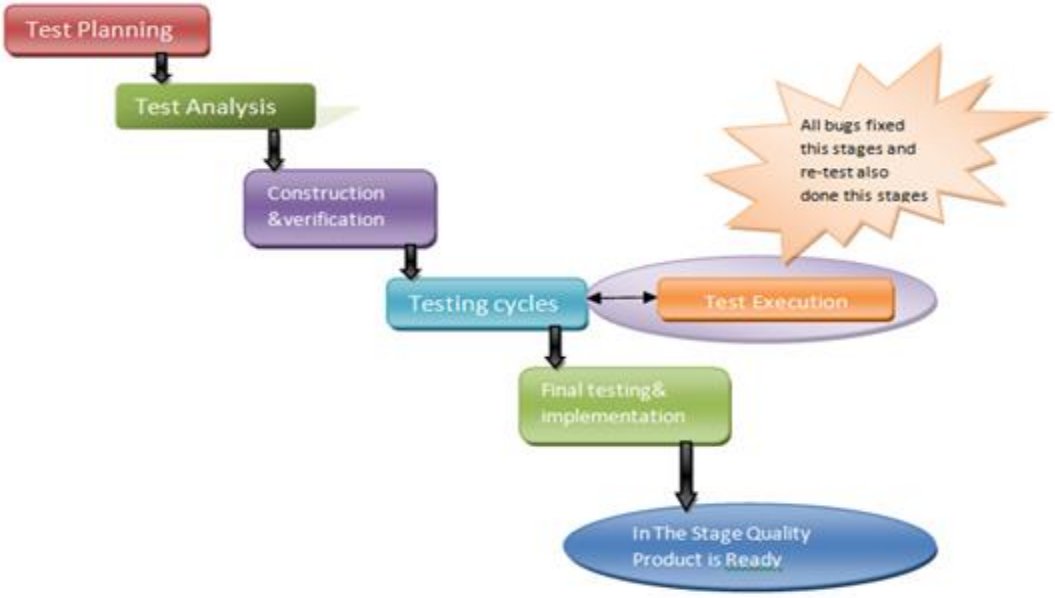

Figure 1.3: Software Testing Life Cycle[4]

1.5 Testing Methods

There exist many different methods and procedures for software testing. Every method focuses on a certain aspect of the system and none of them is able to cover the whole spectrum[5]. They all have their strengths and weaknesses in finding faults. Therefore, a combination of different testing techniques will always be necessary to detect failures with different causes. Some testing techniques are discussed in the following sections.

1.5.1 Black Box Testing

Black box testing, also called functional testing and behavioral testing, focuses on determining whether or not a program does what it is supposed to do based on its functional requirements[4]. Black box testing attempts to find errors in the external behavior of the code categories like incorrect or missing functionality, interface errors, errors in data structures used by interfaces,behavior or performance errors and initialization and termination errors.

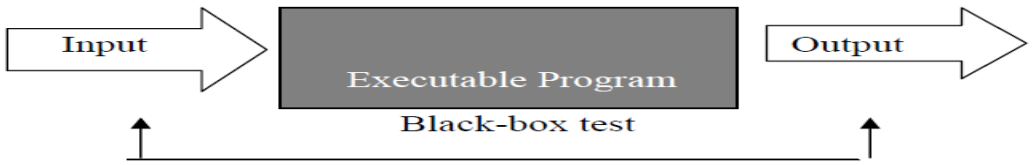

Figure 1.4: Black Box Testing[6]

1.5.2 White Box Testing

White box testing is also known as glass box testing. It relies on the internal knowledge of the system as a method for testing [4].Knowledge of the code is needed during glass box testing in which tester has access to the internal data structures and algorithms including the code that to be implemented. The white box tester (most often the developer of the code) knows what the code looks like and writes test cases by executing methods with certain parameters. In the language of Validation & Verification, black box testing is often used for validation (are we building the right software?) and white box testing is often used for verification (are we building the software right?).

Figure 1.5: White Box Testing [7]

1.5.3 Gray Box Testing

Gray box testing is the combination of black box testing and white box testing. In which a black box tester is unaware of internal structure of the application to be tested, while a white box tester knows the internal structure which includes the access

to internal structures as well as the algorithms for define the test cases[8]. Gray box testing is beneficial because it applies straight forward technique of black box and influences it against the code target systems in white box.

1.6 Levels of Testing

Levels of testing include the different methodologies that can be used while conducting software testing[9]. Following are the levels of software testing:

- Functional Testing.
- Non- functional Testing.

1.6.1 Functional Testing

This is a type of black box testing that is based on the specifications of the software that is to be tested. The application is tested by providing input and then the results areexamined that need to conform the functionality of the software. Functional testing of the software is conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. The parts of functional testing are unit testing, integration testing, system testing, regression testing, acceptance testing. Five steps are involved when testing an application for the functionality.

Step I - The determination of the functionality that the intended application is meant to perform.

Step II - The creation of test data based on the specifications of the application.

Step III - The output based on the test data and the specifications of the application.

Step IV - The writing of test scenarios and the execution of test cases.

Steps V - The comparison of actual and expected results based on the executed test cases.

An effective testing practice will see the above steps applied to the testing policies of every organization and hence it will make sure that the organization maintains the strictest of standards when it comes to software quality.

1.6.2 Non-Functional Testing

The non-functional testing is based upon the testing of the application from its non-functional attributes. Non-functional testing software involves testing from the requirements which are non-functional in nature related but important a well such as performance, security and user interface etc.[5]. Some of the important and commonly

used non-functional testing types are load testing, stress testing, and portability testing.

1.7 Types of Testing

Software testing is the process of identifying defects in developed application. Developed system/application is tested thoroughly in order to provide conformance to functional and non-functionalrequirements[10]. Software is tested either manually or using testing tools.

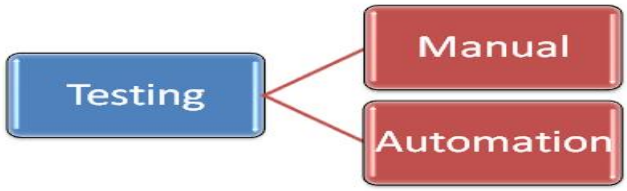

Figure 1.6: Testing Types [10]

1.7.1 Manual Testing

It includes the testing of the software manually i.e. without using any automated tool or any script. In this type, the tester takes over the role of an end user and test the software to identify any un-expected behavior or bug [11]. There are different stages for manual testing like Unit testing, Integration testing, System testing and User acceptance testing.Testers use test plan, test cases or test scenarios to test the software to ensure the completeness of testing. Manual testing also includes exploratory testing as testers explore the software to identify errors in it.

1.7.2 Automation Testing

Automation testing which is also known as test automation is when the tester writes scripts and uses software to test the software. This process involves automation of a manual process[12]. Automation Testing is used to re-run the test scenarios that were performed manually, quickly and repeatedly. Apart from regression testing, automation testing is also used to test the application from load, performance and stress point of view. It increases the test coverage;improve accuracy, saves time and money in comparison to manual testing [13].

1.7.3 Manual versus Automation Testing

Manual and automation are two disciplines of testing with same goal, but different approaches [10]. Each has its own limitations and benefits, so there should be balanced use of both disciplines. Manual testing is conducted by writing test cases and then manually executing them. Manual testing follows series of activities which make test process complete and efficient. Manual testing helps in early testing as testing teams can start writing test cases once requirements are baseline. Manual testing has advantage of human intuition and common sense in form of risk based approach and other techniques like black box, which can't be replaced by automation tools. But executing each test case and logging results manually is time consuming and repetitive process. Consider scenario of regression testing, which requires multiple execution cycles, wherein manual testing often fails to provide required coverage. Due to repetitive tasks, sometimes it becomes mundane and tiresome, so results can be potentially inconsistent and can also affect productivity of testers.

To overcome the limitations of manual testing, automation testing is introduced in testing process, as automation testing provides increased coverage and faster execution of test cycles.

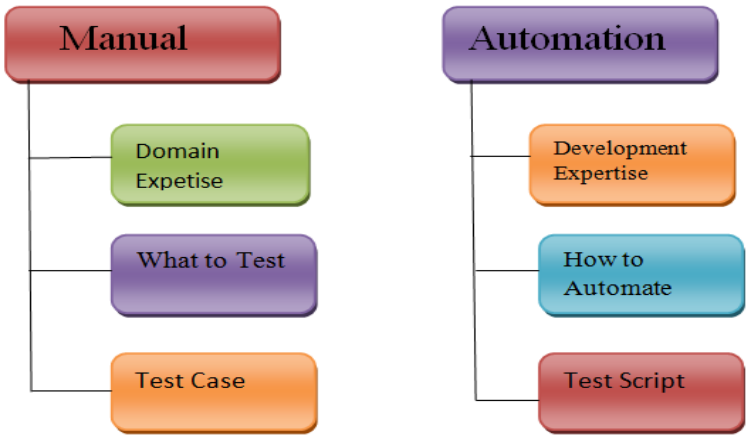

Figure 1.7: Manual Vs Automation Testing[10]

Automation testing refers to the process in which testing tools are used in controlling the execution and implementation of test scripts so that results could be compared to the expected results. Automation testing is fundamentally different from manual testing as it deals with altogether different issues and opportunities. In its most basic function, an automation testing tool is actually used to check if a application under

test is indeed as functional and operational as it should be before it is delivered to customer. Automation testing can be conducted for functional as well as for non-functional (such as security and performance) testing.

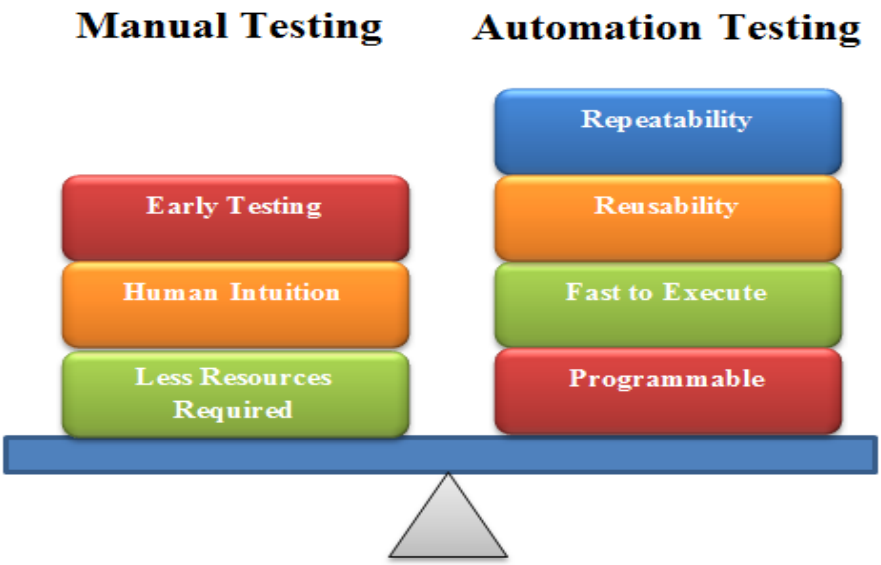

Figure 1.8: Benefits of Automation over Manual Testing[10]

Automation testing offers many benefits. Some of which are listed below:

- Less repetitive work
- Requires comparative less resources
- Consistency and repeatability
- Better coverage
- Faster Execution
- Accuracy in results
- Quick Reporting
- Offers great reusability
- Supports programmability

1.8 Graphical User Interface Testing

Graphical User Interface (GUI) testing is a process to test application's user interface and to detect if application is functionally correct. GUI testing involves carrying set of tasks and comparing the result of same with the expected output and ability to repeat same set of tasks multiple times with different data input and same level of accuracy[14]. GUI Testing includes how the application handles keyboard and mouse

events, how different GUI components like menubars, toolbars, dialogs, buttons, edit fields, list controls, images etc. reacts to user input and whether or not it performs in the desired manner. Implementing GUI testing for an application early in the software development cycle speeds up development improves quality and reduces risks towards the end of the cycle. GUI Testing can be performed both manually with a human tester or could be performed automatically with use of a software program[15].

1.9 Code Based Testing

The principle of structural code based testing is to have each and every statement in the program executed at least once during the test[16]. Based on the premise that one cannot have confidence in a section of code unless it has been exercised by testing, structural code based testing attempts to test all reachable elements in the software under the cost and time constraints. The testing process begins by first identifying areas in the program not being exercised by the current set of test cases, follow by creating additional test cases to increase the coverage. Methods proposed for structural code based testing include statement coverage, decision coverage and condition coverage

Statement coverage intends to have each executable statement of the program tested. However, it is quite difficult to achieve 100% statement coverage, because it is common to have some sections of the code for dealing with error conditions or infrequent events. A primary disadvantage with this method is that it is insensitive to the branch decisions in the program. Hence it is considered to be the weakest form of coverage, since it is only effective for testing computational statements.

1.10 Software Testing Tools

Program testing and fault detection can be aided significantly by testing tools and debugger. The features of testing tools include such as program monitors, permitting full or part monitoring of program code including :

- Instruction set simulator, permitting complete instruction level monitoring and trace
- Facilities program animation,conditional breakpoint and permitting step-by-step execution at source level or in machine code

- Code coverage reports are generated.

There are different types of tools in the software development life cycle in which test management tools can be used in the whole software development life cycle. Test design andinspection tools can be used in requirement specification, in architectural design and in the detailed design phases[17]. The static analysis tools help testing in the coding phase. Execution and comparison tools can be used overall on the right side of V-model. Dynamic analysis tools are usable in functionality, integration and unit testing. They assess the system while the software is running. Coverage tools are designed specifically for unit testing. Acceptance and system tests fall in load and performance tools. GUI test drivers have features of many other tools and are useful in the whole implementation and evaluation area, but they are designed for GUI testing and are distinctly an own group.

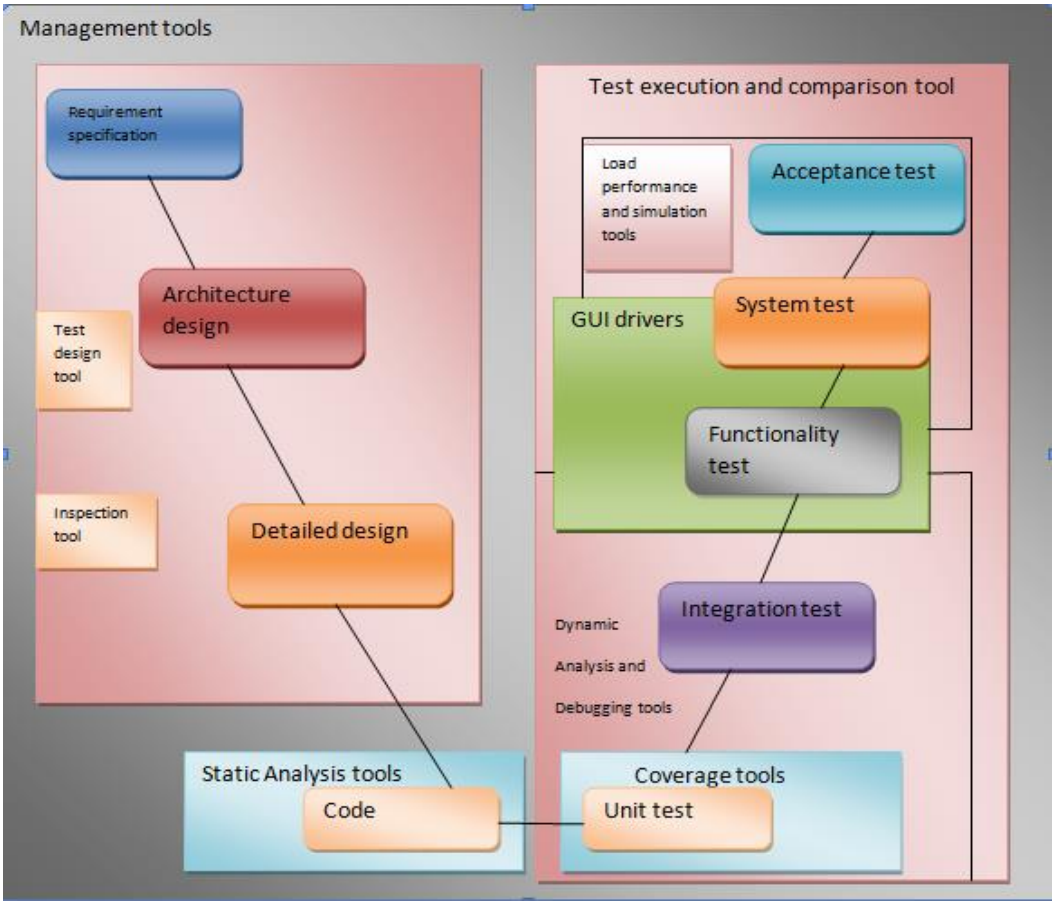

Figure 1.9: Representation of Software TestingTool[17]

Chapter 2

Literature Survey

This chapter aims to review previous work on automation testing and code based testing and also identifies the important issues, methods of automation GUI testing.

2.1Code Based Testing

T. Chen [16] evaluated and compared the performance of adaptive random testing and random testing from another perspective, that of code coverage. As various investigations, higher code coverage not only brings a higher failure-detection capability, but also improves the effectiveness of software reliability estimation. The author conducted a series of experiments based on two categories of code coverage criteria structure-based coverage and fault-based coverage from which author concluded that adaptive random testing is a more effective software testing method than random testing , not only because of its smaller F-measures, but also because of its higher structured-based and fault-based coverage. By achieving higher coverage than random testing with the same number of test cases, adaptive random testing also improves the effectiveness of software reliability estimation, and increases our confidence in the reliability of the program under test even when no failure is detected.

T.Macedo *et al.* [14] presented a research project in which the component-based software on the side of the component provider as well as the on the side of the component user can have all the activities of coding and testing. The purpose is to provide a better management and control of coding and testing activities of component-based software, so that at least the number of retesting activities may be substantially reduced. In order to make this real, it will be necessary to make available a mutual and effective information exchange between component's provider and user. The proposed work contributed to the planning of the component's coding and testing activities, planning of the component maintenance phase, bidirectional information exchange between component provider and component user, automation of the process of management and control of the component's coding and testing activities.

2.2Automation Testing

X. Menganalyzd [19] introduced the characteristics, functions and other aspects of software automation testing. In addition, improvements to the overall process of software automation testing were proposed, providing assistance to the future work of software automation testing development.The author combine features of traditional manual testing and automated testing, so that automated testing strategies and tools work as a weapon in the hands of testers, release them from boring and repetitive work, focusing time and energy on complex work requiring intelligent judgment and other new test cases. Any single technology or operation advancement cannot independently ensure a large scale of improvements to software development efficiency, stability and maintainability in a short time. Automated testing is also an accumulation of experience, gradual process, software testers cannot be expected to automate all the tests in a short time. Successful automated testing requires developing appropriate automated test program, and reasonable automated testing is the first step in the success implementation of the automated testing.

M. Ali *et al.* [20] presented a framework that can help towards the full automation of the software testing process. This framework takes into account the whole gamut, starting from requirement specification to the final deliverables. They presented a complete experimental setup for evaluation of the proposed framework to show the effectiveness of their work and make it more comprehensive.
In future, the proposed framework can be effectively implemented anda full scale empirical study can be carried with over other frameworks available in the market.

A. Leitner *et al.* [21] have shown that the seamless integration between manual and fully automated tests has several advantages, such as the support it provides for regression testing, and the unification of coverage data. They have shown how the interfaces to an automated testing strategy and a manual one can be unified the user provides the set of classes to be tested; these classes directly drive the automated strategy and are input for a selection process of manual unit tests. The already existing features of AutoTest open the way towards continuous testing (or testing in thebackground), based on the integration of AutoTest in an integrated development environment (IDE). Because the tool can test software completely autonomously, it

can be constantly run in the background while the developer is writing the code. As soon as a bug is found, the corresponding routine canbe marked (in an un-intrusive way), to draw the developer's attention that the current implementation of the routine is not correct with respect to its specification. A further advantage of integrating AutoTest into an integrated development environment is that the test scope selection is automated too, since the IDE can keep track of classes that have changed. Moreover, AutoTest can be used in a test-driven development process the manual tests (which are written first in such a process) will be continuously executed by AutoTest in the background as long as they fail, the corresponding routines will be highlighted (similar to syntax and type check errors), and these warning signs will only disappear once the test cases pass.

C. Huang *et al.* [22] proposed two important issues on software reliability modeling and software reliability economics i.e. testing effort and efficiency. The impact of software testing effort and efficiency on the modeling of software reliability, including the reliability measure and the cost for optimal release time is determined. The author proposed a generalized logistic testing-effort function which relates work profile directly to the natural flow of software development. They also described the effort of applying new tools and techniques for increased efficiency of software testing and studied the related optimal software release time problem from the cost-benefit viewpoint. New reliability problems are formulated to incorporate software testing effort and efficiency.

N. Iqbal *et al.* [23] discussed the solution for the key problems of software testing in quality assurance. The existing software practices have some problems such as testing practices, attitude of users and culture of organizations. All these tree problems have some combined problems such as shortcuts in testing, reduction in testing time, poor documentation, shortcuts in testing, reduction in testing, poor planning and co-ordination, lack of user involvement, lack of management support, inadequate knowledge of application environment, improper staffing, poor testability etc. The author has recommended some strategies to provide solution for the existing testing problems.

R. Yin *et al.* [24] analyzed the present status of software testing and then put forward his views on improving the quality of software testing. The author has suggested criteria that can establish the reusable library, enhance the management and also carry out development of software testing standards strengthen the automated testing and the quality of testers to improve the quality of software testing.

2.3 GUI Automation Testing

P. Nagarani *et al.* [25] proposed an approach to automate the software testing process using Coded UI (User Interface) tool. Automation of the test plays a significant role in testing activity, as it saves time and provides better utilization of resources. The test automation has many situations to deal such as mapping of user specifications to identification of the test cases, test case generation,maintenance of test cases and test scripts.

The proposed Coded UI tool in VSTS (Visual Studio Team Server) is useful to create completely automated testsfor the validation of the functionality and behavior of the application user interface. Coded UI tool has the capabilities of efficiency in recording, ability to generate scripts, data driven testing, reporting of test result, script reusability, playback of the scripts. The playback of the scripts can be done manytimes after the recording of the user actions is performed.

The proposed Coded UI tool and the generic framework created for the toolhelps in automating the software applications undertest with complete test coverage and saving the hugeamount of time and resources.Also, the automation in proposed tool helps in maintenance of the scripts in an understandable manner which can be reused any number of times depending on the requirement of the user.

The proposed automation testing tool can be enhanced by the integrating the automated GUI testing with code based testing to design the hybrid approach.

J.Prabhu *et al.* [26] proposed an approach for automatic modeling of Java GUI applications for Application Under testing (AUT) purposes, implemented proof-of-concept tool support for the approach, and combined the implemented GUI tool with an open source AUT tool to form a tool chain for automated modeling and testing of Java GUI applications. The approach aims to reduce the amount of manual effort required to model GUI applications to enable automated testing.

The strengths of the approach in comparison to the automated testing tools include automatically generating human readable graphical models while requiring none or only a little manual effort. The graphical models provides the test engineers, a way to manually elaborate the models, for example inserting valid input values for specific input fields of the GUI application, and allow comparison between testing tools based on the actual implementation and requirements of the system.
The tool chain was used to automatically model and test several open source Java GUI applications, resulting in the breakthrough of several unknown errors and usability problems.

The approach seems hopeful but there are also limitations. As the approach is based on running and observing existing software, the AUT must be an executable Java GUI application, and the models are based on the actual implementation instead of designed and expected features. Also, more work is needed on the GUI Tool, as the proof of concept implementation is not yet mature enough to be used in the software industry. In future GUI Tool can be improved further so that the generated models and reports would inform about the detected usability issues and include information about the changes that happened in the GUI after a specific interaction.

M. Grechanik *et al.* [27] offer a novel and effective approach called REST for maintaining and evolving test scripts so that they can test new versions of their respective gaps. They built a tool to implement the approach, and the results of evaluation show that users find more failures and report fewer false positives in test scripts with our tool than with competitive approaches. The REST can be considered as the first step towards developing different solutions for this big and pervasive problem.

In the GUI Comparison Algorithms, it is important to develop algorithms that compare GUIs with a high degree of automation and precision. The process information about GUIs may enable researchers to create solutions in which these platform will “heal” broken test scripts.A huge potential lies in developing type systems that enable type checking test scripts with respect to GUI objects of GAPs. Few case study participants strongly suggested that REST should be integrated with QTP since the results of the REST static analysis can easily be verified by running test scripts under QTP. In addition, participants with testing experience reported that

they used QTP during experiments with REST to verify some reported failures, and they thought it improved their productivity.

S.Bauersfeld *et al*. [28] introduces a Java library called GUITest, which allows generating fullyautomated GUI robustness tests for complex applications, without the need to manually generate models or input sequences. The author has explained the robustness testing operation and also, presented effective results on its applicability
Despite the simplicity of this test, the results are quite promising and show the applicability of this approach even for large scale GUI applications with complex interfaces.
In the future, more sophisticated action selection can be planned for machine learning techniques and metaheuristics to find sequences which are more likely to crash the system undertest (e.g. sequences that consume a lot of memory or have longexecution times).

P. Aho *et al*. [29] presented an approach for enhancing automatically generated GUI models with valid test data. They have presented their work on the GUI Driver tool and implemented the required new features to support the presented iterative modeling process, where each round of iteration includes an automated modeling phase and a manual phase of providing valid test data.

In comparison to the related work, the strength of this approach is the practical way to combine manually provided test data into the automatically generated GUI models. Because of this the GUI Driver tool is able to reach and model more states of the GUI applications resulting in more accurate GUI models. As this approach is based on running and observing an existing GUI application, it can be applied only on implementations mature enough to be executed. In cases of old applications that are already validated against the requirements, this approach can provide test automation support for regression testing during the maintenance changes.

In future, the efficiency of approach can be evaluated by testing a variety of Java GUI applications. Also, the functionality for detecting usability issue can be extended and improved further.

M. Ali *et al*.[30]proposed a time driven model based approach for automatic test case generation and execution. This approach is best suited for an environment where time

is a scarce resource and programmers and testers have to meet strict deadlines with least compromise on quality of software. The approach directly follows from the observation that screen based flow testing of a typical web application can sufficiently be done by functional testing of business logic of the application. The idea is to extract function signatures to acquire domain knowledge. This domain knowledge will be used to identify equivalence classes. Equivalence classes will be helpful in generating sufficient number of prioritized test cases to be executed. The authors divided testing coverage into different confidence levels achieved by the tester or programmer based on size of test suite and sufficiency of testing technique used. This step wise approach will facilitate the tester to execute high priority test cases very quickly and to achieve certain level of confidence about the correctness of the application.

In future, the proposed approach can be implemented effectively which can prove very helpful for the target industry as well as any growing industry with small sized software organizations working in a fierce market competition.

Chapter 3

Problem Statement

This chapter focuses on the gap analysis and the objectives of the thesis work.

3.1 Gap Analysis

Testing is an important aspect in the software industry. The complexity of modern software packages makes manual testing a tedious task. Testers need to focus on critical software features or more complex cases, leaving repetitive tasks to the test automation system. Automated testing can help to improve efficiency of the testing process in order to identify areas of a program that are prone to failure.

Inadequate and ineffective testing is responsible for many problems regarding software reliability faced by computer users. Some of the gap analyses areas follows:

- There is a need to develop a new approach to automate the software testing process to get larger coverage with same or even smaller number of testing personnel.
- As personnel cost and time limitation are the significant restraints of testing process, an efficient testing tool is required to overcome these limitations.

Hence, the need of tool with more coverage in less time becomes vital.

3.2 Objectives of Thesis

The objectives of thesis are as follows:

- To analyze the role of automation in the software testing and study an existing approach for automated testing of GUI and code based applications.
- To design a hybrid approach which includes combination of manual testing, code based testing and automation testing coupled with prioritybased testing.
- To implement the proposed approach in java language.

Chapter 4

Proposed Work

In order to accomplish an effective automation testing, a priority based hybrid automation approach has been proposed. The existing GUI automation testing is enhanced by adding the functionality of code based testing which is further improved by adding thelayer of priority to the automation testing. The approach is used to create automated tests for the validation of the functionality and behavior of the application. In order to ensure the platform independence in the proposed approach, java has been used for the implementation

4.1 Framework ofProposed Approach

The proposed work follows the hybrid approach by combining the features of GUI testing and the code based testing in single standalone testing approach. The efficiency of the work is further enhanced by prioritizing the modules of the software being tested to achieve a certain degrees of priority in the automated GUI testing. The proposed approach is a combination of following:

- Manual testing
- Automation testing
- Code driven testing
- Priority based testing

4.1.1 Manual Testing

It is the process of manually testing the software for defects/bug. Manual testing requires a tester to perform the manual test operations without the help of any automation software.

A manualtester performs the following steps:

- Select the module to be tested
- Enter a test case manually
- Execute the test.
- Verify the actual result with expected result.
- Record the results as pass or fail.

- Make a summary report of pass and fail test cases.
- Publish the report.

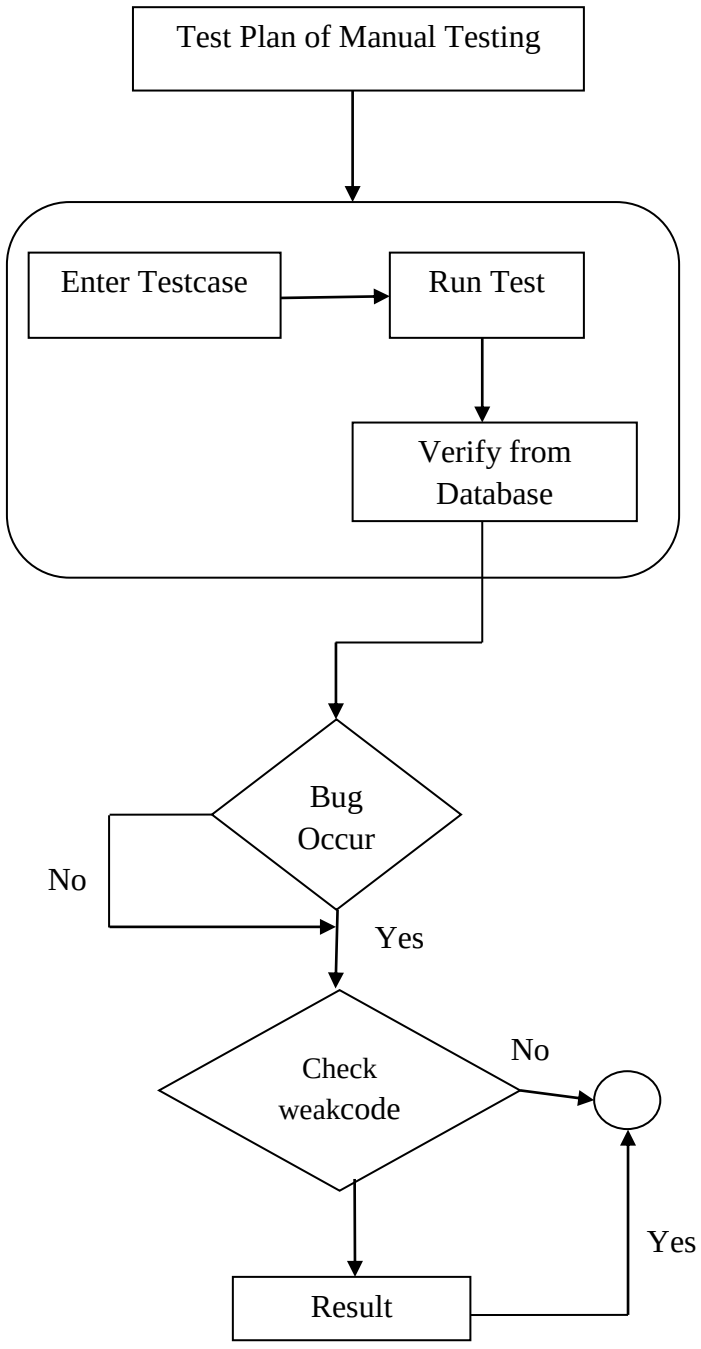

Figure 4.1: Flow Chart of Manual Testing

4.1.2 Automation Testing

Automation testing means using automation approachto execute the test case suites. It involves the use of software to control the execution of the test.

The tester would perform the following steps for automation testing.

- Select the module to be tested.
- Browse the test case suite file.
- Start the testing process.

The automation approach performs the following:

- Enter the test data into the system under test
- Compare the actual outcomes with predicted outcomes
- Record the test case results as pass or fail
- Generate the detailed test report

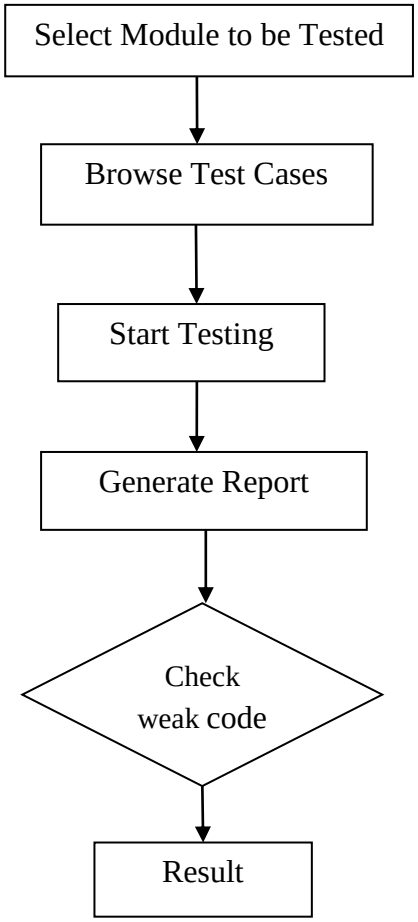

Figure 4.2: Flow Chart of Automation Testing

4.1.3 Code Based Testing

The principal of code based testing is to have each and every event/action in the program executed at least once during the test. On the basis of the code based testing, a tester attempts to determine all the reachable elements in the software under the test. The testing process begins by first creating xmi file using Agro UML tool for the

software under test followed by parsing the xmi to examine the weak code i.e. the code which is not being used during the current execution of the software.

4.1.4 Priority Based Testing

In this phase of testing, tester defines priority for various modules of application under test. The priorities will be used to execute tests over the specified modules in a specific order. High priority modules will be tested prior to the low priority modules. Once the testing process for a particular module is over, the test report is generated and the tester is notified accordingly. The testing process continues till the test report for each module under test is generated successfully.

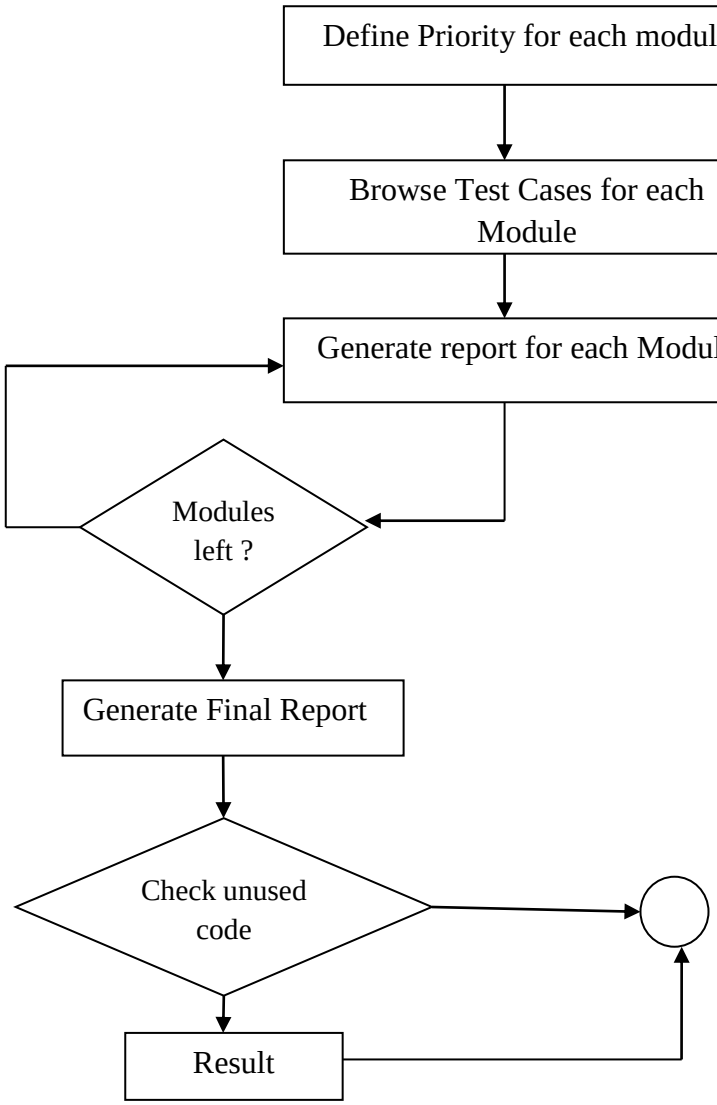

Figure 4.3: Flow Chart of Priority based Testing

4.2 Characteristics of Proposed Approach

The proposed approach has the following capabilities:

- *Efficiency in recording*: It can capture all the events and actions that were performed on the application during the automation and manual testing which ever being performed at that particular instant of time.
- *Reporting of test results*: The test results can be reported passed/failed statement at the time of manual, automation and priority testing.
- *Parsing the code*: It can parse i.e. analyze the source code of any program written in java to examine the various fields, methods, constructors and the interfaces of the java program being examined.
- *On camera testing*: The user has a privilege to have the snapshots of the various events involved during the whole testing process.
- *Prioritizing the testing of modules*: The user can prioritize various modules under test to define the flow of testing in the required manner.
- *Finding the weak code:* The approach can help the user to determine the weak code i.e. code not being used during the testing of the software under observation.

Chapter 5

Results

The proposed work is designed using hybrid approach by combining the features of GUI testing and the Code based testing. The proposed approach has also the functionality of prioritizing the modules of the software being tested to achieve the efficiency in the terms of time consumed by the testing process.

The various phases of designed approach are explained in the below actions:

5.1 Automation Testing Phase

In order to start the automatic testing, 'Automatic' option is selected from the 'GUI Testing' menu.

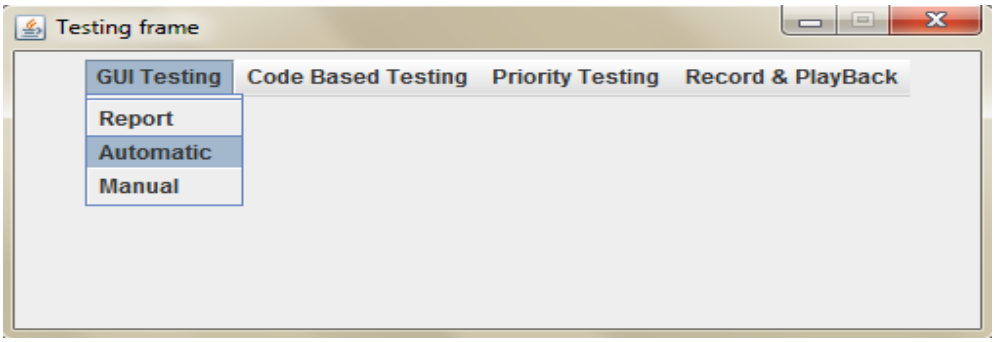

Figure 5.1: Main Frame of Testing

Now, the user has to select module to be tested. After selecting the module, the user needsto browse the corresponding test case file and start the testing.

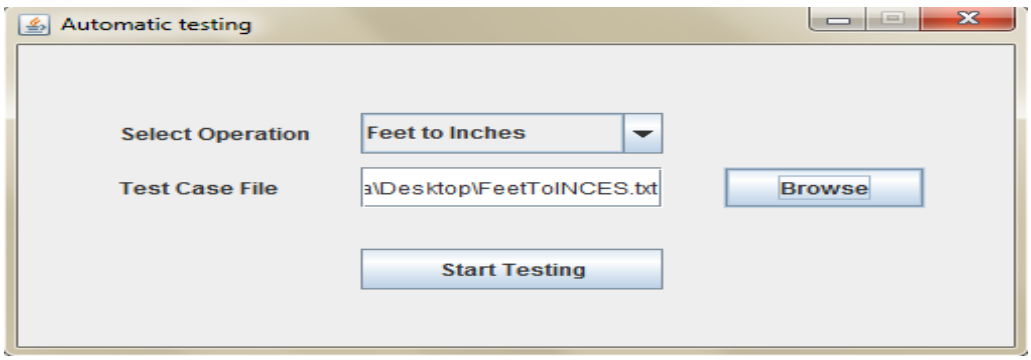

Figure 5.2: Input Frame for Automatic Testing

The testing process not only automates test case execution, but also test result verification without any user intervention.

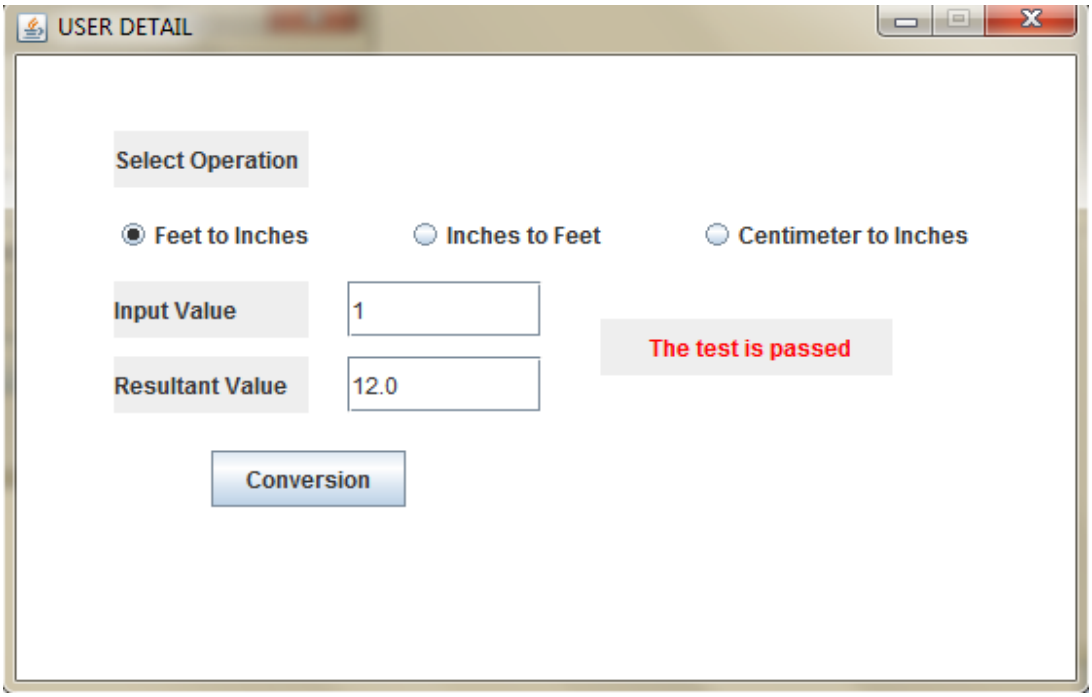

Figure 5.3: Testing Frame for Automatic Testing

After the completion of testing process, corresponding test report is generated by selecting 'Report' option from 'GUI Testing' menu. The test report is used to determine which test case is passed and which test case is failed.

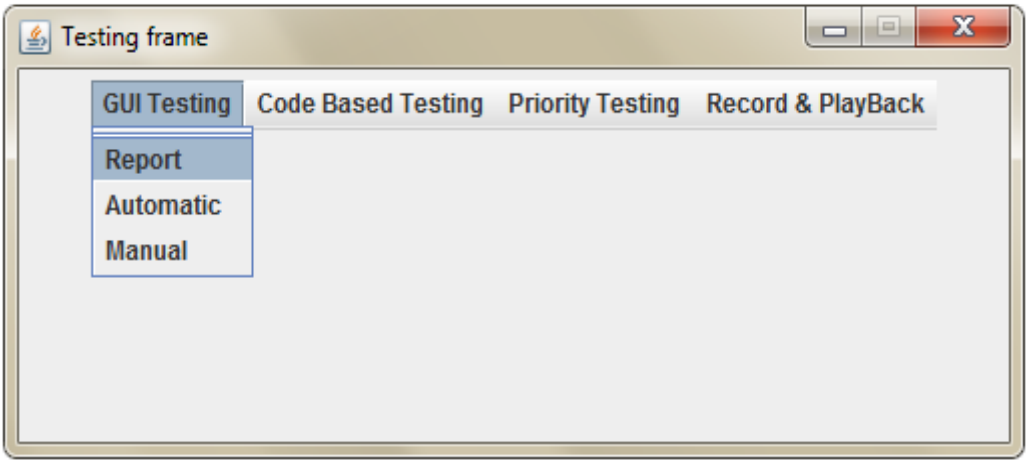

Figure 5.4: Operation Selection for Report Generation

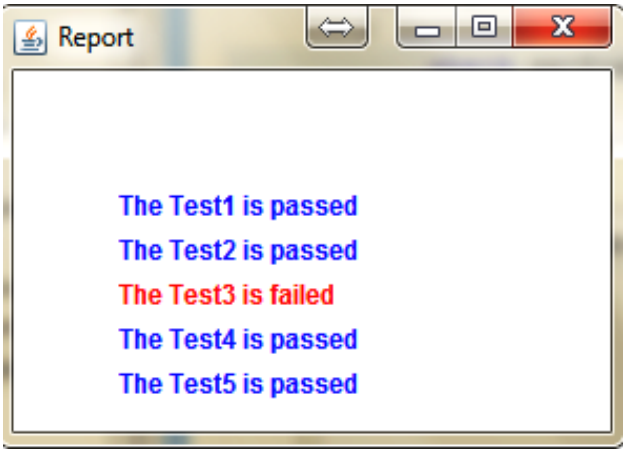

Figure 5.5: Generated Report

After performing the testing, user has an option tocheck the corresponding weak/unused code for the software under test.

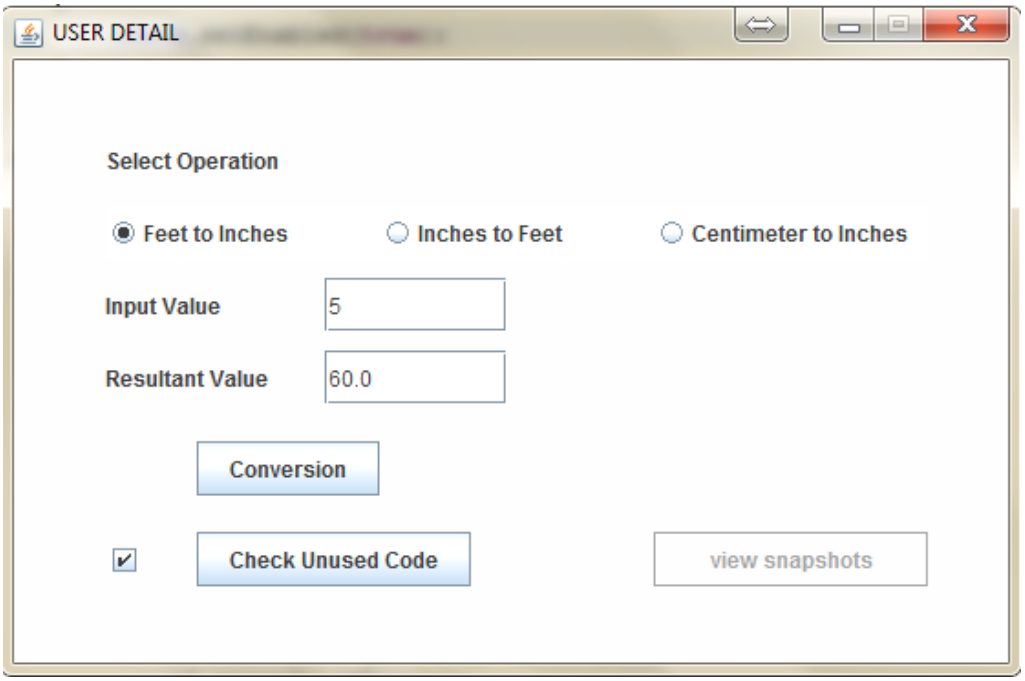

Figure 5.6: Selecting option for Weak Code

In weak/ unused code testing, user can determine the components which are not used being used in that particular testing.

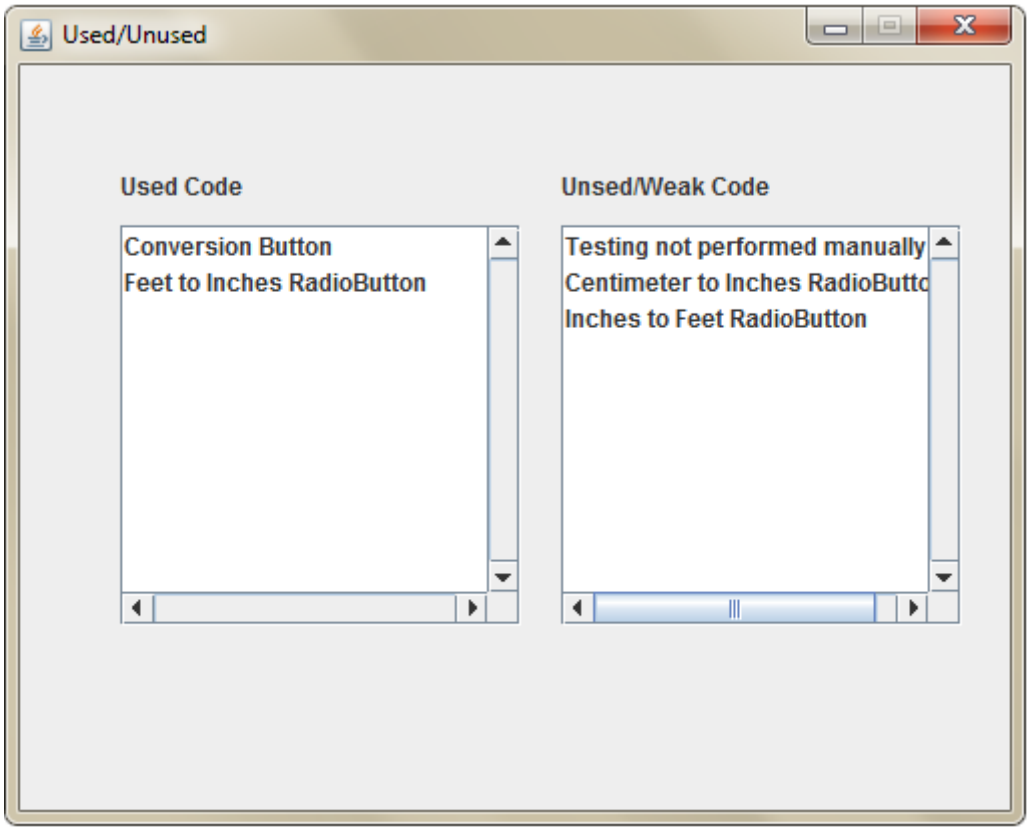

Figure 5.7: Weak Code Frame

The tool also provides the functionality of record and playback for automation as well as manual testing. To start the recording process, the user has to select the 'record' option from 'Record and Playback' menu and select the desired option. After that the corresponding testing is performed. By the end of testing process, user can playback the recording done previously by choosing play option.

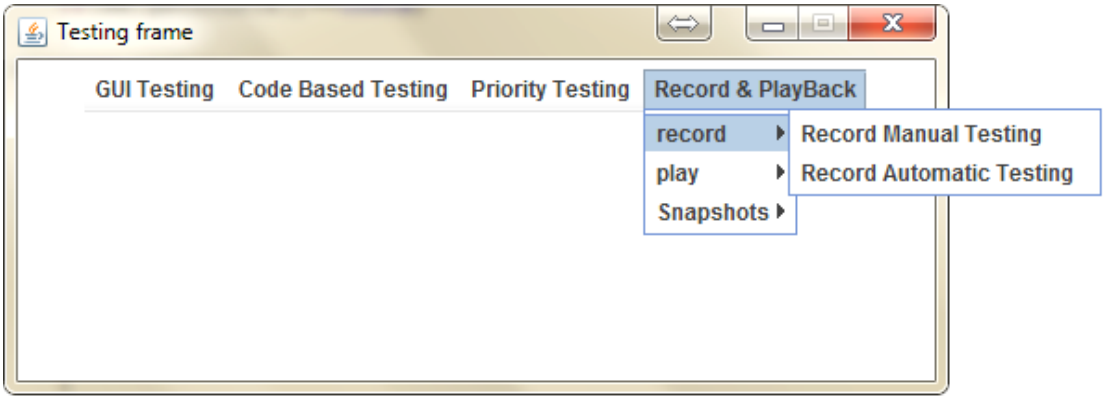

Figure 5.8: Operation Selection for Recording Events

The tool also performs the functionality of ‘on camera’testing for automation as well as manual testing. If the user want to have the snapshot of the particular testing process then the user have to select the snapshot option for the same. The snapshot is generated for any interaction of user/automation tool with the testing process.

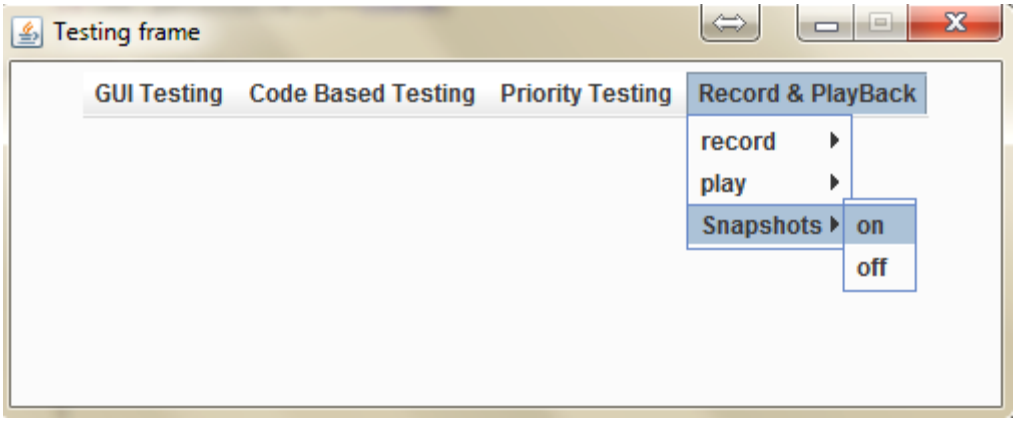

Figure 5.9: Operation Selection for On-Camera Testing

5.2 Manual Testing Phase

In order to perform testing process manually, user selects ‘Manual’ option from the ‘GUI Testing’ menu.

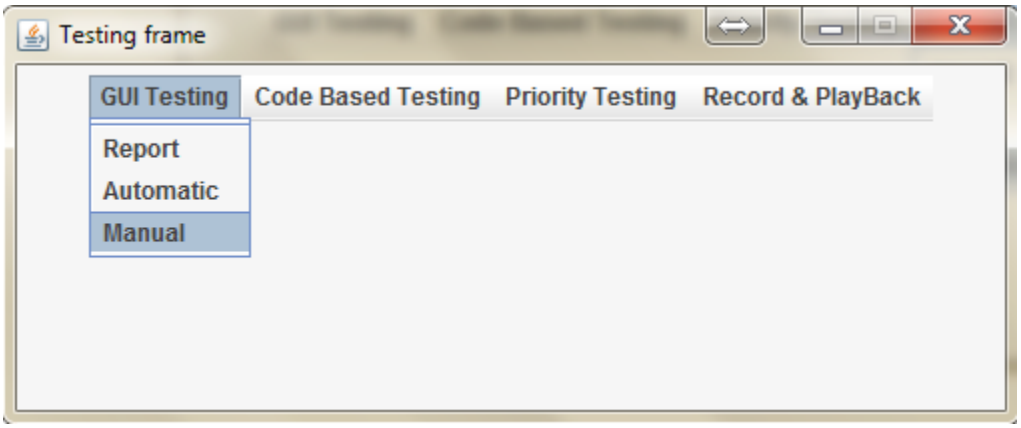

Figure 5.10: Operation Selection for Manual Testing

Next, the user has to select the module to be tested. After selecting the particular module, the user needsto enter the test case manually and start the testing process. Similar to automatic testing, the user can also check the weak code for the software under test.

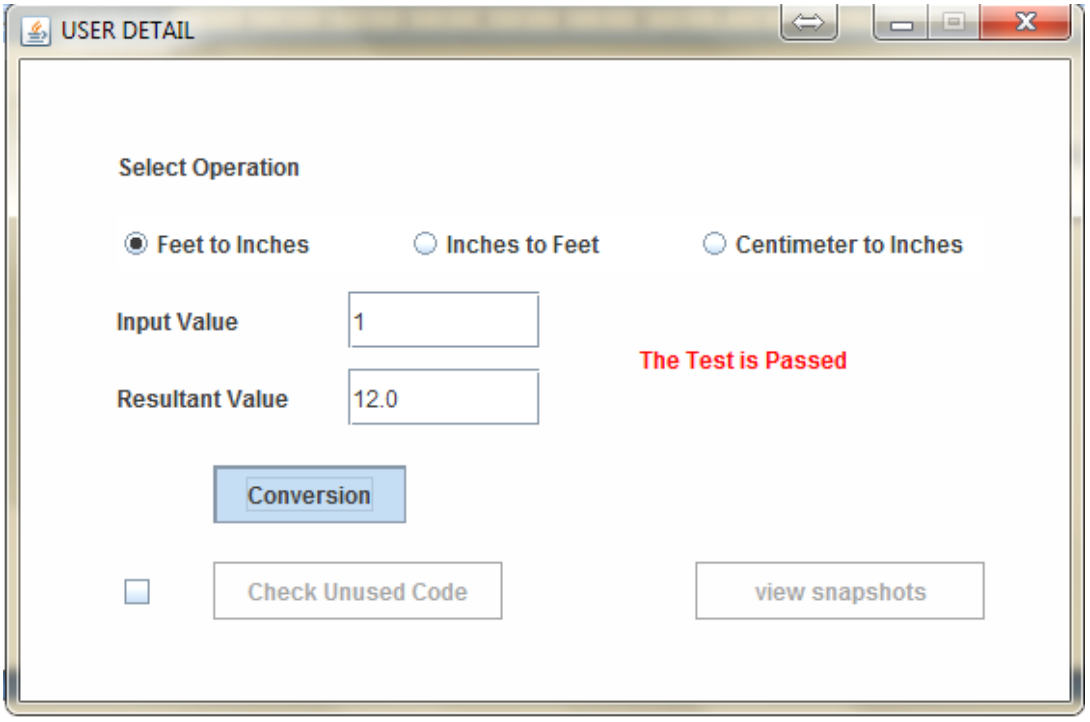

Figure 5.11: Testing Frame for Manual Testing

5.3 Code Parser

The proposed tool also provides an interface for the code parser. The parser can parse any java source file to determine various components i.e. fields, methods, constructors and interfaces associated with that particular java source code file.

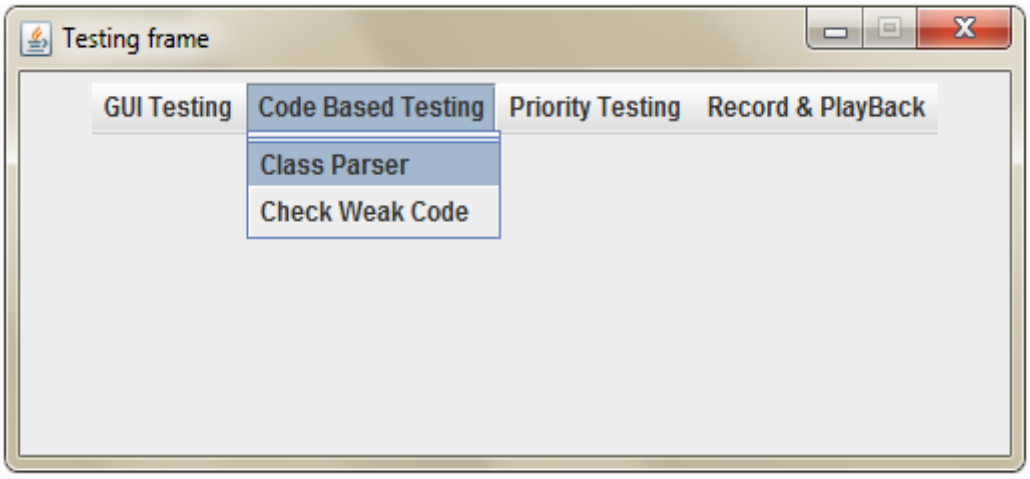

Figure 5.12: Operation Selection for Class Parser

Figure 5.13: Input Frame for Code Parser

On clicking the 'Parse Code', the user can determine the list of fields, functions, constructors and interfaces on the particular java file that parsing

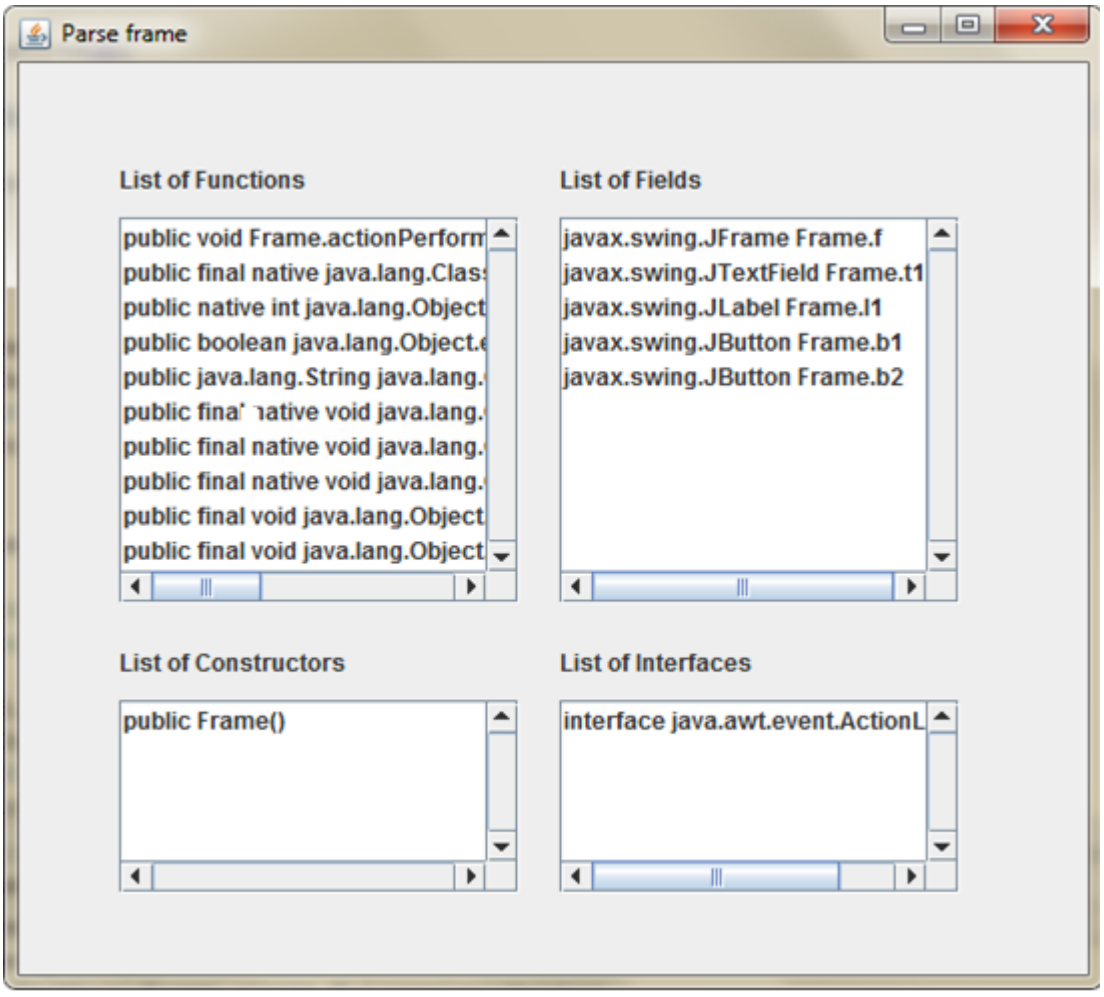

Figure 5.14: Output Window of Code Parser

5.4 Priority Based Testing

In this particular phase of testing, the user can define the priority for each module under test. The testing process begins by clicking 'Priority Testing' menu.

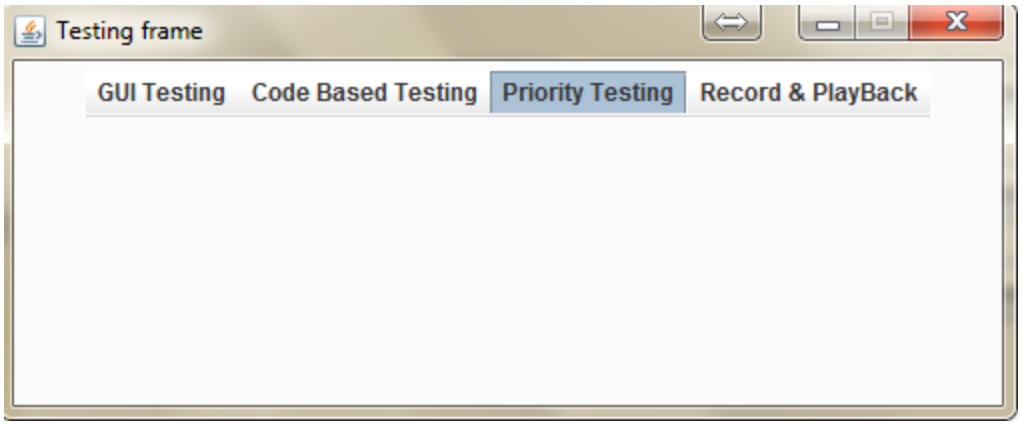

Figure 5.15: Selection Operation for Priority Testing

On the priority testing frame, the user is required to define priority for various modules of application under test and browse the test casesfile for each of the module before starting the testing process.

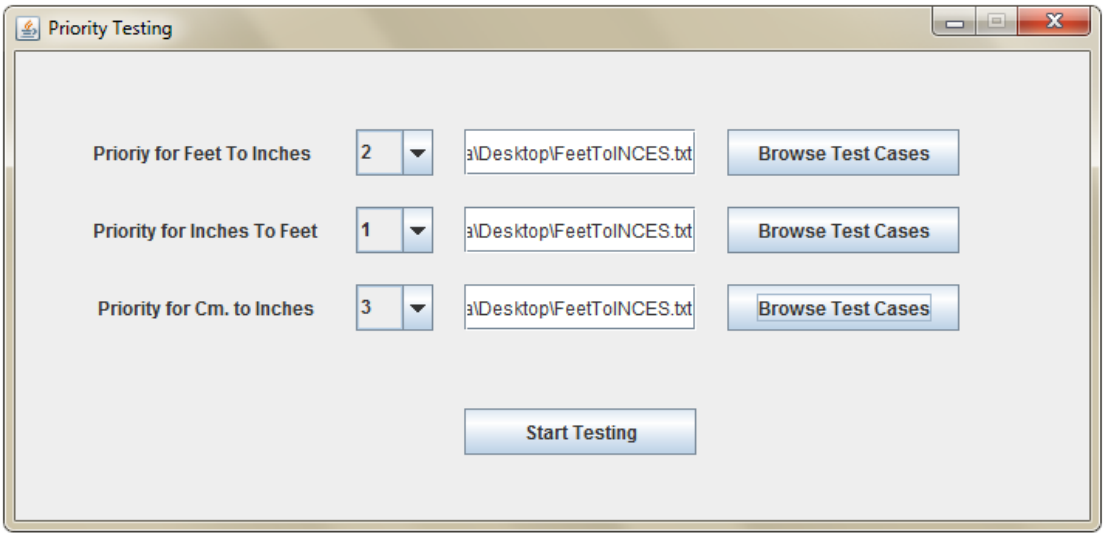

Figure 5.16: Input Frame for Priority based Testing

When a testing process for a particular module is completed successfully, the corresponding test report is generated and the tester is notified accordingly.

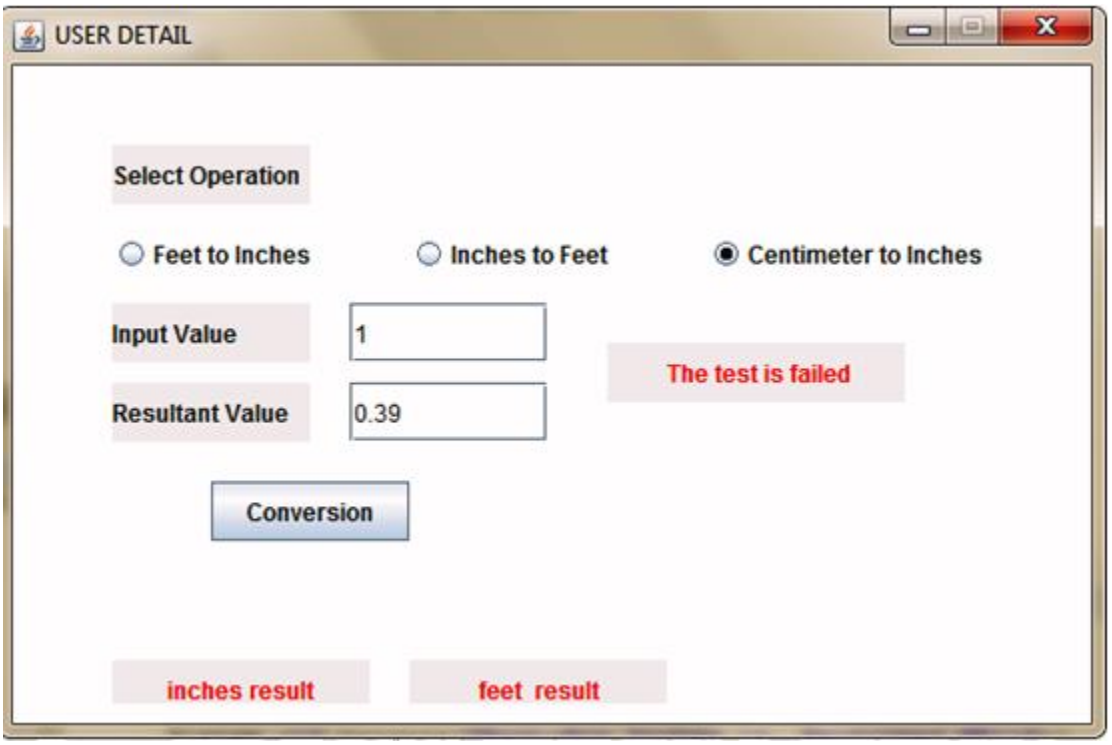

Figure 5.17: Notifications on Testing Window

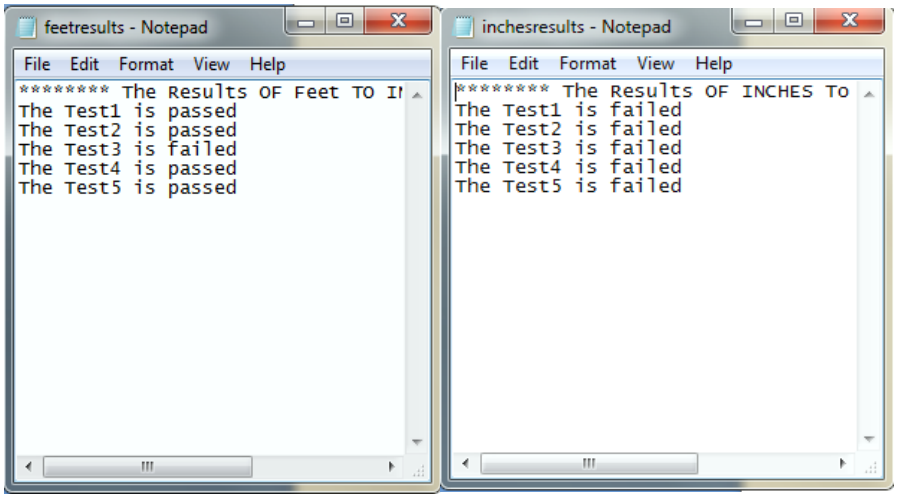

Figure 5.18: Auto-Generated Reports

After testing processes for each of the module under test is completed successfully, the test report for the whole testing process can be generated to check the pass/fail test cases.

Figure 5.19: Report for Priority based Testing

After the completion of testing process, the user canalso find the weak/unused for the testing process. Weak code can be realized by checking the option for checking the unused code on the Testing window.

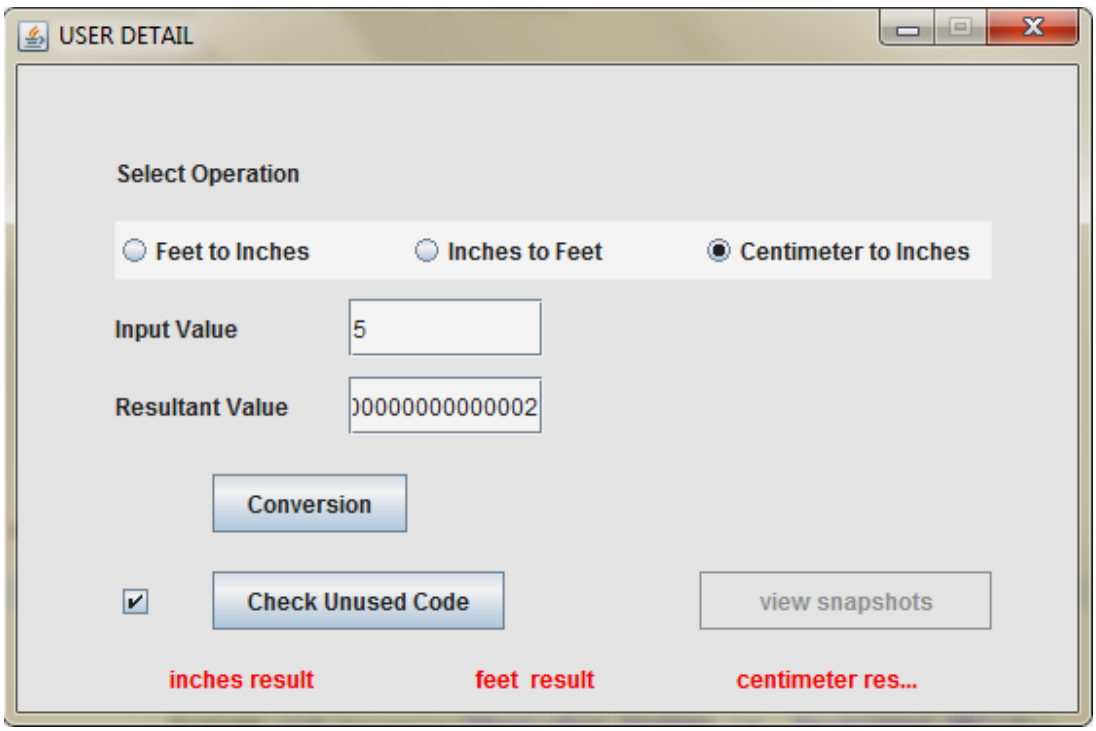

Figure 5.20: Selection Operation for Weak Code

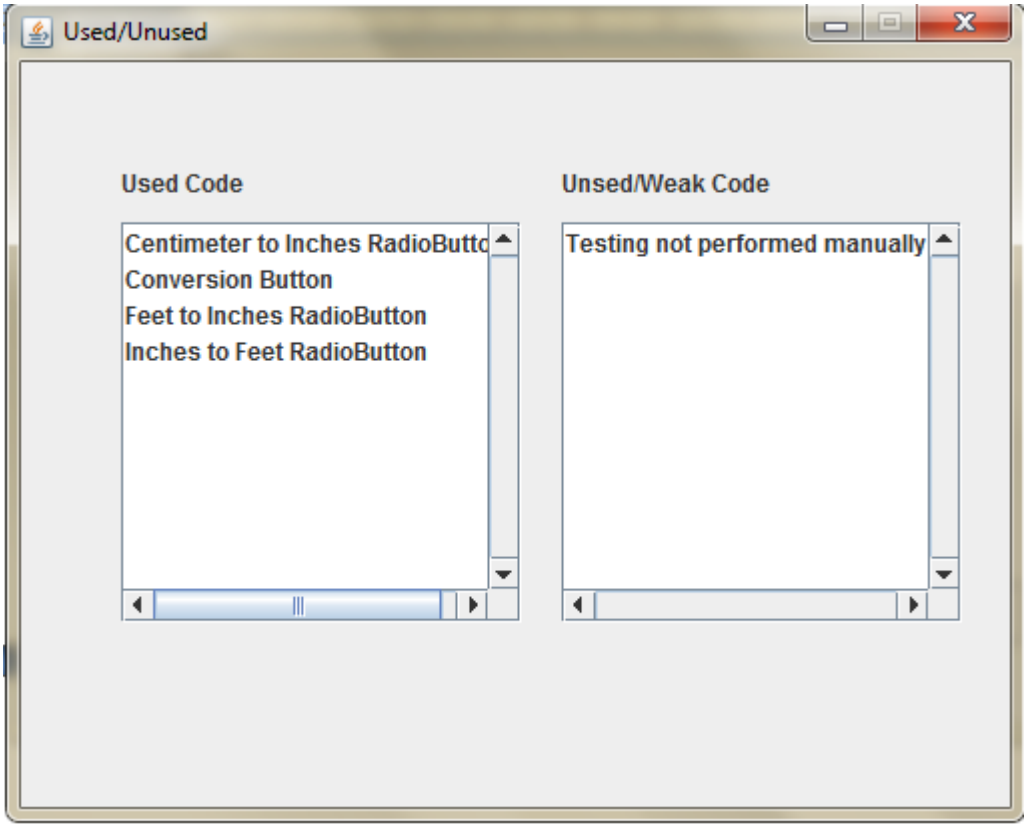

Figure 5.21: Weak Code Window for Priority Testing

5.5 Comparative Analysis of Testing Types

The efficiency of the existing automation approach and proposed priority based hybrid automation approach is determined by testing both of the approaches on the 30 test cases .The approach is tested on three different modules i.e. feet to inches, inches to feet and centimeter to inches. The test cases are chosen as random numbers in the range of 0-100.

The following table (Table 5.1) shows the efficiency of the proposed priority based hybridapproach over the existing automatic approach. From the results, it can be concluded that proposed approach is more efficient in terms of time in comparison to existing approach.

Table 5.1 Time of Execution for Testing Approaches

Testing Approach	**Time of Execution (in milliseconds)**
Existing	242242
Proposed	218394

The test cases for the testing process are shown in the following table (Table 5.2).

Table 5.2 Test Case Values and Expected Results for Various Modules

S. No.	Test Case Value	Automation testing			Priority testing		
		Expected Result (Inches to Feet)	Expected Result (Feet to Inches)	Expected Result (Cm. to Inches)	Expected Result (Inches to Feet)	Expected Result (Feet to Inches	Expected Result (Cm. to Inches)
Test Case 1	10	120	0.83	3.94	120	0.83	3.94
Test Case 2	31	372	2.58	12.21	372	2.58	12.21
Test Case 3	12.7	152.4	1.05	5.0	152.4	1.05	5.0
Test Case 4	53	636	4.41	20.8	636	4.41	20.8
Test Case 5	14	168	1.16	5.5	168	1.16	5.5
Test Case 6	65	780	5.41	25.61	780	5.41	25.61
Test Case 7	16	192	1.33	6.30	192	1.33	6.30
Test Case 8	17.2	206.4	1.43	6.7	206.4	1.43	6.7
Test Case 9	18	216	1.5	7.09	216	1.5	7.09
Test Case 10	19	228	1.58	7.48	228	1.58	7.48
Test Case 11	30	360	2.5	11.82	360	2.5	11.82
Test Case 12	0.25	3	0.02	0.09	3	0.02	0.09
Test Case 13	3.46	41.52	0.28	1.36	41.52	0.28	1.36
Test Case 14	1.75	21	0.14	0.68	21	0.14	0.68
Test Case 15	21	252	1.75	8.2	252	1.75	8.2
Test Case 16	33	396	2.75	13.00	396	2.75	13.00

S. No.	Test Case Value	Automation Testing			Priority Testing		
		Expected Result (Inches to Feet)	Expected Result (Feet to Inches)	Expected Result (Cm. to Inches)	Expected Result (Inches to Feet)	Expected Result (Feet to Inches)	Expected Result (Cm. to Inches)
Test Case 17	7.87	94.44	0.65	3.10	94.44	0.65	3.10
Test Case 18	45	540	3.75	17.73	540	3.75	17.73
Test Case 19	37.5	450	3.12	14.77	450	3.12	14.77
Test Case 20	80	960	6.66	31.52	960	6.66	31.52
Test Case 21	3.2	38.4	0.26	1.26	38.4	0.26	1.26
Test Case 22	9	108	0.75	3.54	108	0.75	3.54
Test Case 23	62	744	5.16	24.42	744	5.16	24.42
Test Case 24	4.7	56.4	0.39	1.85	56.4	0.39	1.85
Test Case 25	51	61	4.25	20.0	61	4.25	20.0
Test Case 26	0.50	6	0.04	0.19	6	0.04	0.19
Test Case 27	73	876	6.08	28.7	876	6.08	28.7
Test Case 28	22	264	1.83	8.66	264	1.83	8.66
Test Case 29	78	936	6.5	30.73	936	6.5	30.73
Test Case 30	11	132	0.91	4.33	132	0.91	4.33

The following graph (Figure 5.22) depicts the efficiency of proposed approach over the existing approach.

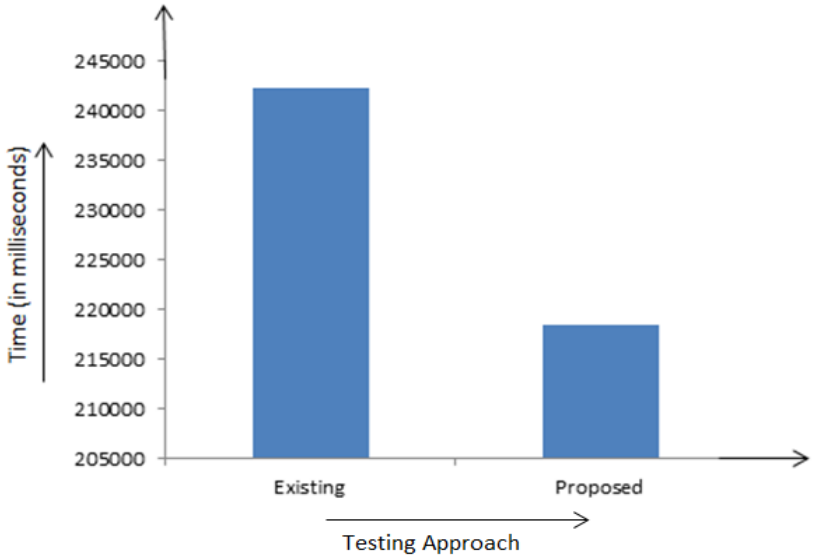

Figure 5.22: Testing Vs. Time

Chapter 6

Conclusion and Future Scope

6.1 Conclusion

Graphical User Interface (GUI) software is often large and complex, and the correctness of GUI's code is essential to the correct execution of the overall software. GUI testing is vital for quality assurance because the GUI tests are performed from the view of the endusers of the application. Many times all thefunctionality of the application can be invoked throughthe GUI and therefore GUI tests can cover the entireapplication.

As the size and complexity of software are growing continuously, manual testing becomes very difficult and tedious task. But on the contrary, automated testing can help to improve efficiency of the testing process in order to identify areas of a program that are prone to failure. Although automated testing is highly efficient but it does not allow the user to specify which module tobe tested first.

The proposed approach allows the user to define priority for each of the modules under test. After each of the modules are tested effectively, the corresponding testing report is generated automatically and is available for the review while other modules are being tested. The priority based automation approach helps to minimize the user intervention to a large extent, which makes it highly efficient and less time consuming.

The code based testing helps the user to identify the structural components which are not being used in that particular phase of the testing process, hence enables user to debug the entire application efficiently. The capability of record and playback and on-camera testing further improves the efficiency of proposed approach.

Hence, priority based hybrid automation approach proves to be highly efficient in terms of quality and time.

6.2 Future Scope

Following are the future direction in which work can be carried out:

- The code parsing feature of the proposed approach is able to parse only the java source code files which can be extended by making the it language independent.

- The design of proposed approach can be generalized to test any GUI java application.
- Code based testing can be further explored so that the approach can serve as a total solution for testing.

Bibliography

[1] Roger S. Pressman, "Software Testing Strategies," *Software engineering,* 5th ed. New York, McGraw-Hill, pp: 478-505, 2001.

[2] *http://www.isixsigma.com/industries/software-it/five-essentials-software-testing.*

[3] *http://testerinyou.blogspot.in/2010/12/software-testing-life-cycle*.

[4] *http://www.buzzle.com/articles/phases-of-testing-life-cycle.html.*

[5] H. Freeman, "Software Testing", *IEEE Instrumentation & Measurement Magazine*, pp: 48-50, 2002.

[6] *http://qainsights.com/black-box-testing.*

[7] *http://muzakstudyzone.blogspot.in/2012/12/white-and-black-box-testing-techniques.html*

[8] *http://searchsoftwarequality.techtarget.com/definition/gray-box.*

[9] J. Hill *et al.,* "Unit Testing Non-functional Concerns of Component-based Distributed Systems", *Proc. IEEE International Conference on Software Testing Verification and Validation*, pp: 406-415, 2009.

[10] *http://automationtesting/using/PitfallsofAutomationTesting.htm.*

[11] M. Auguston *et al*., "Test Automation and Safety Assessment in Rapid Systems Prototyping", *Proc. 16th IEEE International Workshop on Rapid System Prototyping,* 2005.

[12] S. Berner *et al., "*Observations and Lessons Learned from Automated Testing", *Proc. 27th IEEE International Conference,* pp: 571 – 579, 2005.

[13] T. Pajunen *et al., "*Model-Based Testing with a General Purpose Keyword-Driven Test Automation Framework", *Proc. 4th IEEE Fourth International Conference,* pp: 242 – 251, 2011.

[14] T. Chang *et al*., "GUI Testing Using Computer Vision", *Proc.10th SIGCHI Conference on Human Factors in Computing Systems*, pp: 1535-1544, 2010.

[15] P. Aho *et al., "*Enhancing Generated Java GUI Models with Valid Test Data", *Proc. IEEE International Conference,* pp: 310 – 315, 2011.

[16] T. Chen *et al.,* "Code Coverage of Adaptive Random Testing", *Proc. IEEE Conference on Transactions on Reliability,* vol. 62, no. 1, pp: 226-237, 2013.

[17] P. Pohjolainen, *http://www.cs.uku.fi/tutkimus/Teho/SoftwareTestingTools.pdf*, 2002.

[18] T. Macedo and C. Torres, "Management and Control of Coding and Testing of Component-based Software", *Proc. 31st IEEE Annual International Computer Software and Applications Conference*, vol. 1, pp: 664-666, 2007.

[19] X. Meng, "Analysis of Software Automation Test Protocol", *Proc. International Conference on Electronic & Mechanical Engineering and Information Technology*, vol. 8, pp: 4138-4141, 2011.

[20] M. Ali and T. Saha, "A Proposed Framework for Full Automation of Software Testing Process", *Proc. IEEE International Conference on Informatics, Electronics & Vision,* pp: 436-440, 2012.

[21] A. Leitner *et al.,* "Reconciling Manual and Automated Testing: the AutoTest Experience.", *Proc. 40th Hawaii International Conference on System Sciences,* pp: 261-271, 2007.

[22] Chin-Yu Huang *et al.,* "Software Quality Assurance Methodologies and Techniques", *Journal of Advances in Software Engineering*, vol. 1, 2012.

[23] N. Iqbal and M.Qureshi, "Improvement of Key Problems of Software Testing in Quality Assurance", *Journal of Science International*, vol. 21, pp: 25-28, 2009.

[24] R. Yin and X. Ding, "How to Improve the Quality of Software Testing", *Proc. IEEE International Conference on Systems and Informatics*, pp: 2533-2536, 2012.

[25] P. Nagaraniand and R. VenkataRamanaChary, "A Tool Based Approach for Automation of GUI Applications", *Proc. 3rd IEEE International Conference of Computing Communication & Networking Technologies*, pp: 1-6, 2012.

[26] J. Prabhu *et al.*, "A Model for GUI Automated Testing Framework in Software System", *International Journal of Computer Applications,* vol. 64, no.15, 2013.

[27] M. Grechanik *et al.*, "Maintaining and Evolving GUI-Directed Test Scripts", *Proc. International Conference on Software Engineering*, pp: 408-418, 2009.

[28] S. Bauersfeld and J. Vos, "GUI Test: A Java Library for Fully Automated GUI Robustness Testing", *Proc. 27th IEEE International Conference on Automated Software Engineering,* pp: 330-333, 2012.

[29] P. Aho *et al*., "Enhancing Generated Java GUI Models with Valid Test Data", *Proc. IEEE Conference on Open System,* 2011.

[30] M. Al *et al.,* "Priority-Based Automated Testing of Web Application", *Proc. IEEE Conference on Communication and Information Technology,* 2012.

[31] A. Cervantes, “Exploring the Use of a Test Automation Framework”, *Proc*. *IEEE* Conference on Aerospace, 2009.

[32] E. Collins “Strategies for Agile Software Testing Automation: An Industrial Experience”, *Proc*. *IEEE 36th* Conference on Computer Software and Applications Workshops, 2012.

List of Publications

Communicated

1. Priyanka and Garhwal S., "GUI Automation Testing: A Systematic Review", International Journal of Engineering Research and Technology (IJERT), vol. 2, issue. July 2013.

2. Priyanka and Garhwal S., "Priority based Hybrid Automation Testing", International Journal of Computer Applications Technology and Research (IJCATR), vol. 2, July-August 2013.