Component Repository with Grey Box Testing Support

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

Master of Engineering
in
Software Engineering

Submitted By
Inderjit Singh
(801231012)

Under the supervision of:
Mrs. Ashima Singh
Assistant Professor

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

June 2014

Certificate

I hereby certify that the work which is being presented in the thesis entitled, "*Component Repository with Grey Box Testing Support*", in partial fulfillment of the requirements for the award of degree of Master of Engineering in *Software Engineering* submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of *Mrs. Ashima Singh* and refers other 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.

Inderjit Sgh

(Inderjit Singh)

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

14/7/14

(Mrs. Ashima Singh)
Assistant
Professor, CSED

Countersigned by

(Dr. Deepak Garg)
Head
Computer Science and Engineering Department
Thapar University
Patiala

14/7

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

Acknowledgement

No volume of words is enough to express my gratitude towards my guide, **Mrs. Ashima Singh**, Assistant Professor, Computer Science and Engineering Department, Thapar University, who have been very concerned and have aided for all the material essential for the preparation of this thesis report. She has helped me to explore this vast topic in an organized manner and provided me with all the ideas on how to work towards a research oriented venture.

I am also thankful to **Dr. Deepak Garg**, Head of Department, CSED, **Dr. Maninder Singh,** Associate Professor, **Ms. Jhilik Bhattacharya**, P.G. Coordinator, for the motivation and inspiration that triggered me for the thesis work.

I would also like to thank the staff members and my colleagues who were always there in the need of the hour and provided with all the help and facilities, which I required for the completion of my thesis.

Most importantly, I would like to thank my **Parents** and the **Almighty** for showing me the right direction out of the blue, to help me stay calm in the oddest of the times and keep moving even at times when there was no hope.

Inderjit Singh
801231012

Abstract

Software components have played an important role in modern software and system development. The main contribution of software components is reuse which helps reduces development cost and time and increase productivity. In software industry, there is progressive increase in reusability of software components in order to increase productivity and to reduce time-to-market. In order to make reusable component available in future projects there must be a system to store software components so that they can be efficiently retrieved as per user requirements.

Software repository is the storage medium used for storing, classifying, maintaining, browsing and retrieving components. Repository must be developed which supports essential features like efficient storage and retrieval. If a specific component is to be searched through keywords or phrase, then the potential keywords need to be stored in software repository. If in any case the keyword or the phrase framed is unable to give desired result, then the software repository is required to be enriched with more efficient retrieval technique. In this study grey testing tool has been implemented with an efficient storage structure which will provide both black box and white box testing support to the reusable components and will store all the test cases generated while performing grey testing of the component for future use.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Component Based Software Engineering

Component-based software engineering is a technique for software development which was emerged in the late 1990s as an approach based on reusing previously developed software components. CBSE is a process that mainly stresses on the design and development of computing systems by making use of reusable software components [1].

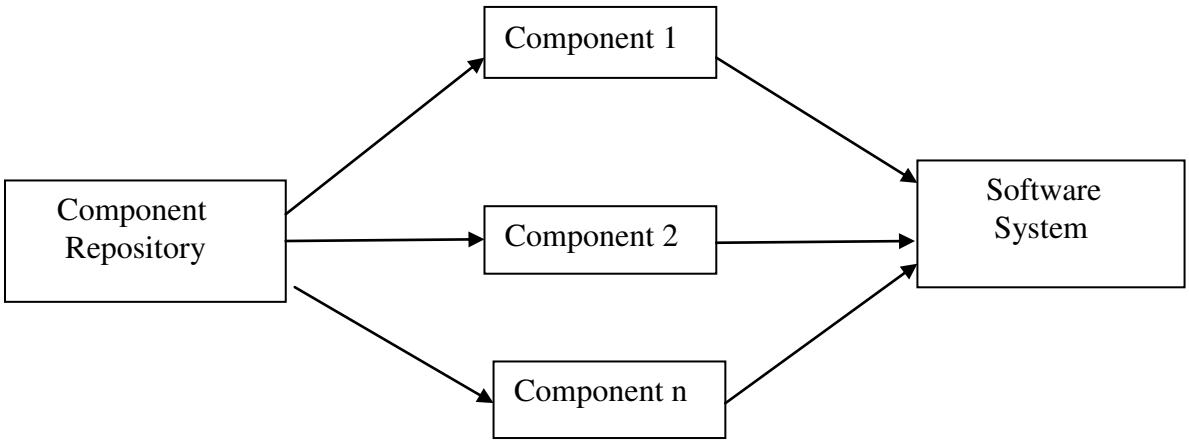

Figure 1.1: Component Based Software Engineering[1]

1.2 Various Issues with Traditional Software Development Models

Followings are some issues with the Traditional Software Models [35]:-

i. Lack of Reusability: In traditional software development models like Waterfall model, Increment Waterfall model development is based on specific requirements provided by the customer. Basically what it means that developers develops the product for specific purpose not for general purpose [9].

ii. Lack of standardized component interface between components: Components interfaces are not designed for a general purpose but for specific project. There is lack of consistent mechanism which will provide support to component interactions.

iii. Lack of Customization: Any kind of changes are not viable in Traditional software development.

iv. Lack of Component Interoperability: Components developed by traditional software development models are not interoperable. Because there is no data exchange mechanism between components and no consistent interaction mechanism between components.

v. The development size and complexity of product increases rapidly on surpassing each stage of the traditional software development model.

vi. Even the cost of development of the product surpasses our predictions in most of the cases.

vii. The product consumes more time and efforts for development, because everything has to be developed from scratch.

viii. In traditional software development models product needs to upgrade mostly after development, because the requirements have to be fixed before the commencement of its development.

1.3 Component-Based Development

All the issues and problems with the traditional developments can be resolved by Component-based development. Basically in component based development the software systems are not developed from scratch but they are developed form pre-existing components stored in the component repository [44]. Some examples of Component Based developments are – building vehicles, furniture etc from existing components, building and storing components which can be reused in different applications. In CBSE maintenance also takes place which is done by replacing of component and introducing new components into system and storing in the repository. It mainly stresses on making use of previously developed components.

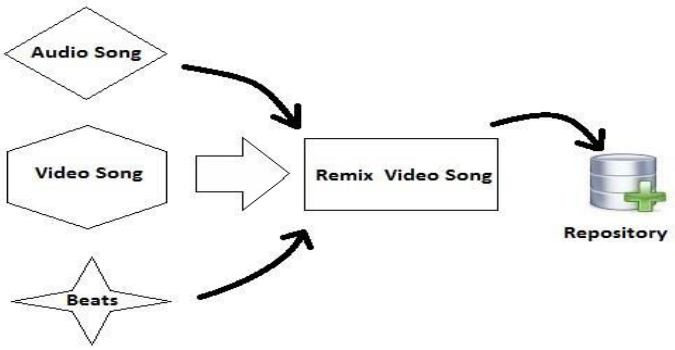

Figure 1.2: Example of CBSE [4]

Figure 2 describes the use of components i.e. there are basically three components like audio songs, video songs and beats. Now these components are mixed in a specified way to get a remixed video song which is stored in a repository for future use.

Component Based Development ensures the increase in richness and quality of the system and the product is developed in a short time. It also provides maintenance by exchanging newly developed assets with other assets. CBD is the process of locating & retrieving reusable assets, qualifying for working and adapting new constraints and conditions in order to develop a new component [41].

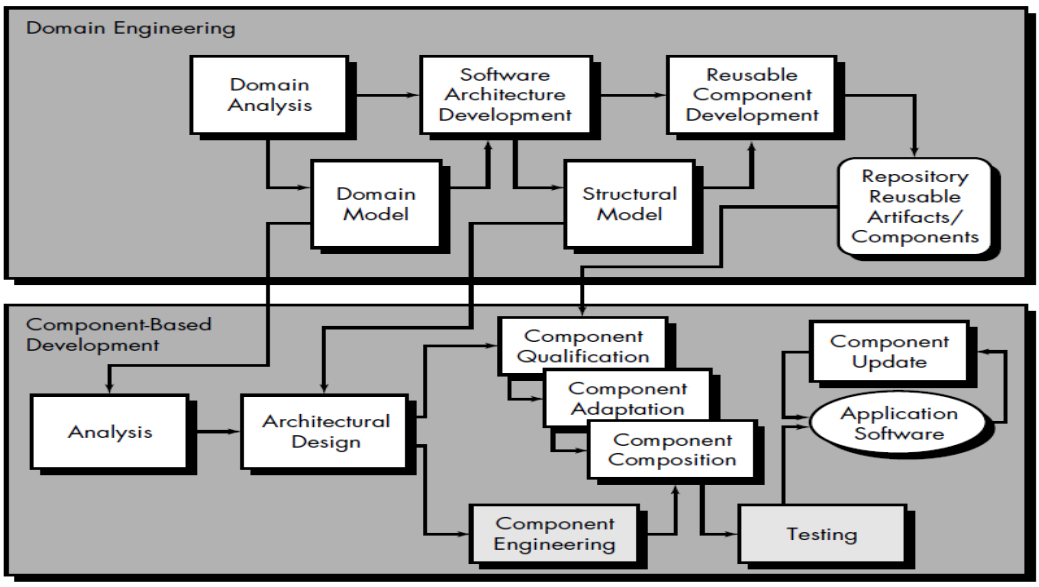

Figure 1.3: Process of CBSE [9]

1.4 Features of CBSE

i. In component based software engineering firstly the architecture of the product is developed then, those components are selected which fits in that particular developed architecture [40].

ii. CBSE technique is less expensive because of reusability of previously developed component i.e. high quality, tested and certified components are reused to form system, which also reduces risk of failure [39].

iii. In CBSE maintenance is easily done by replacing previously developed components either by upgrading stored components or by introducing new component to system.

iv. Time-to-market is less in Component based software development, because product is not developed from scratch.

v. Now the developers are adopting change driven development that is implementing changes in requirement, design and code at any point of time. One of the agile development techniques is SCRUM that supports implementing frequent requirement changes in requirement, design and code [43]. This feature of SCRUM made it the first choice of software developers, But SCRUM approach compromised on software reusability. Whereas CBSE delivers reusable components which basically save the efforts of development for scratch which in turn require repository to store the developed components so that it can be reused in another project.

1.5 Impediments to Reuse

i. A component reusability technique prevails in very small number of organizations and companies.

ii. Though many vendors provide COTS but they are rarely being used by the software developers.

iii. Knowledge and training required to use these components is available in scarcity [34].

iv. Many of the software developer's beliefs that reusability leads to more trouble than it's worth.

v. Large number of Software developers follows developing methodologies which does not encourage the reusability of components.

vi. Only few organizations provide some kind of incentive's while producing reusable components [35].

1.6 Repository

Repository is defined as a collection of software assets or components that are maintained by an organization for storage, browsing and/or retrieval of reusable components so that they can be reused to meet specific requirements which in return result in cost-effective development of the product and will meet productivity goals [37].

1.6.1 Accessing Criteria

Accessing criteria of a repository can be classified in to two categories [16]:

Browsing: Browsing means surveying the content of repository. Browsing can be done at following stages:-

i. It can be done before product design, requirements are not clear that what is needed.
ii. In those circumstances when designer have no clear cut definition of what they need, but do have some idea.

Retrieval: Searching the repository to find the desired components that meets our requirements. Basically performed after product designing and requirement phase.

1.6.2 Reusable Assets

Reusable assets are the types of components which can be stored in repository and can be reused in future projects [9].

i. Executable Code, Source Code
ii. Requirement Specifications
iii. Designs
iv. Test Data
v. Documentation
vi. Architectures

1.6.3 Component Classification

Software components can be classified as following [38]:

i. In-house: Components those are developed inside an organization.
ii. Legacy Systems: Components that are extracted from legacy systems.
iii. COTS: Third party components those are not stored in the repository.

A storage structure is required to store all the above classified components which will provide all time access to the stored components.

1.6.4 Repository Features

Generally repositories have following features:

i. Automation: Repository must have Automated Graphical User Interface support, for easy and effective searching and retrieval of components [33].
ii. Standard component framework: Proper standardized framework should be developed which will describe purpose, functional description, quality level, must give information about the number of time the particular component is used and associated legal restrictions.

iii. Classification scheme: Productive classification scheme for each domain. Effective classification schemes are essential to assist the user in locating and comparing library components, and to speed the process of identifying appropriate components for the task at hand.

iv. Documentation: One of the essential features is that repository must provide complete system and component documentation.

Modern, expert systems, knowledge-based approaches and new technologies for efficient browsing and retrieval are the hot topics of research [9].

The system and component documentation allows the re-user to determine which components have reuse potential with regard to specific requirements, and to fully apprehend the process of obtaining components for reuse in a new application.

1.7 Software Testing

Software's are basically tested in order to measure and enhance software quality. Software testing is a standardized process and is also included as major part of the software maintenance life cycle. Software testing is also considered as a type of validating and verifying process [23].

Verification is the process by which the developers insure that the developed product satisfies all the constraints which were enforced before its development. In other words, the developed project is behaving as intended [30].

Validation is the process by which the developers insure that the developed product satiate the all the requirements at the culmination phase of the development i.e. the product is developed according to customer specifications.

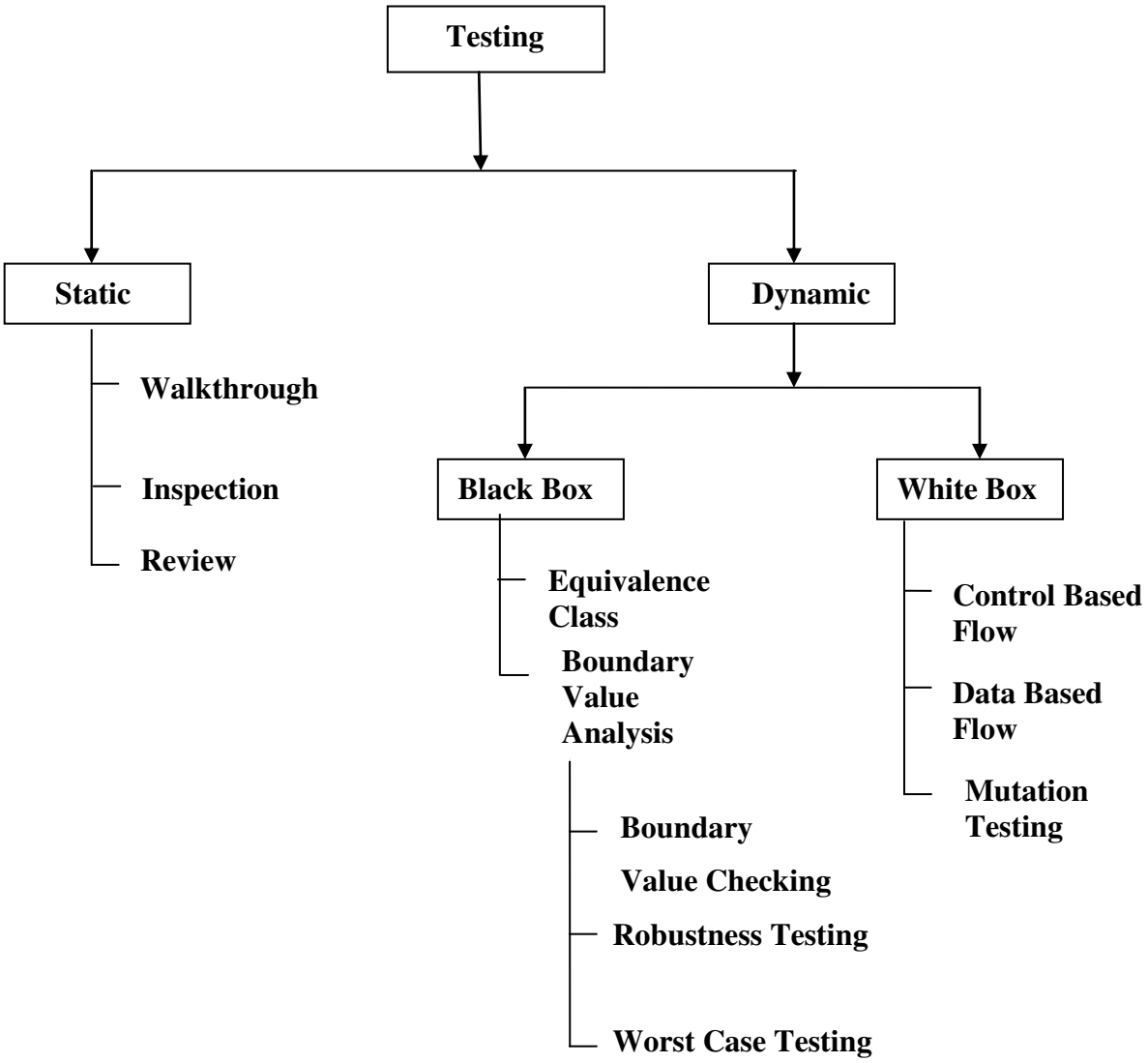

Figure 1.4: Classification of Testing [30]

1.7.1 Static Testing

Static testing is a type of application or program testing technique in which the actual software or we can say the actual code is not executed [5]. This type of testing is totally opposite to dynamic testing. In static testing the program is not explored in detail but basically checks the sanity of the implemented and algorithm and document. The whole code is probed manually for errors. This type of testing is basically performed by the developer who wrote the code. Static testing is generally of three types which are as following:

Walkthrough

Walkthrough in the field of software engineering is a kind of peer to peer review in which the developer of the program leads the whole development team and other

stakeholders. All the members associated with the product in development asks the questions and point out all the errors, lack of compliance with the established standards.

Inspection

An inspection is among the most customary review technique used while project development. The primary aim of inspection is that the entire participant should come to harmony on a specific discussion about the project to be developed. Inspection includes discussion on the software specification, test data and test plans. In this type of review a specific work product is selected at one time and full team is gathered for an inspection meeting in order to verify and validate the work product. Every individual prepares for the inspection meeting by analyzing the work product, then each inspector express their views and notify each error.

Review

In Code review the developed code is probed in a systematic order. The main goal of code review is to locate and fix errors that were being overlooked in the development phase which will also contribute to increase the overall quality of the product to be developed.

Code reviews are used to locate and expunge common unsafe things like race conditions, memory leak, overflow of the buffer etc. Therefore enhancing the overall quality and security of the product being developed.

1.7.2 Dynamic Testing

Dynamic testing is generally of two types which are as followings:

Black Box

Black box testing is a type of software testing in which testers are unaware of the structure of a program; they don't have any knowledge about the internal structure of the program. They are not concerned how the input is processed and the output is generated [26].

In black box testing tester is concerned only about that for a particular input whether the required result is obtained or not. In a black box approach the test cases are designed in such a way so that all the requirements mentioned in the software requirement specifications are verified and validated [23].

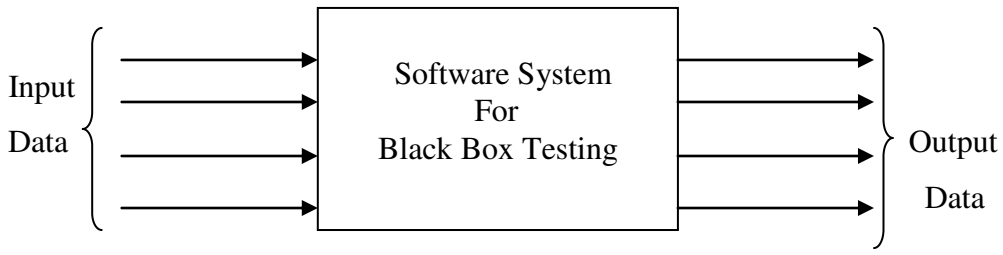

Figure 1.5: Black Box Testing Process

Black Box Testing is basically of three types which are as followings:

i. Equivalence Class Testing

Equivalence Class partitioning is a type of software testing technique in which the input data is divided into classes of equivalent data by using which the test cases can be easily generated. In principle, test cases should be designed in such a way that all the partitions should be covered at least once [23]. Equivalence class partition aims at defining test cases that uncovers error lies in all the cases, thereby minimizing the total count of test cases. The main advantage of this technique is that it consumes very less time in designing the test cases because this technique lowers the count of test cases.

Equivalence partitioning is mostly applied to the input variable of the code to be tested but in some cases it is also applied on the output variable of the code to be tested. In equivalence partitions the classes or division are generally designed from the software requirement specification.

ii. Boundary Value Analysis

Boundary Value Analysis mainly concentrates on all the input variables included in the developed code or function. For better understanding let us consider two variables **α** and **β** which are to be declared and defined in the program. Where the value of **α** lies between the range A and B and the value of **β** lies between C and D.

$$\mathbf{A \leq \alpha \leq B}$$

$$\mathbf{C \leq \beta \leq D}$$

A is the minimum and B is the maximum value for the variable **α**,

Similarly, C is the minimum value and D is the maximum value for the variable **β.** The main aim of boundary value analysis is to probe errors occurring near the minimum and maximum values of the input variables.

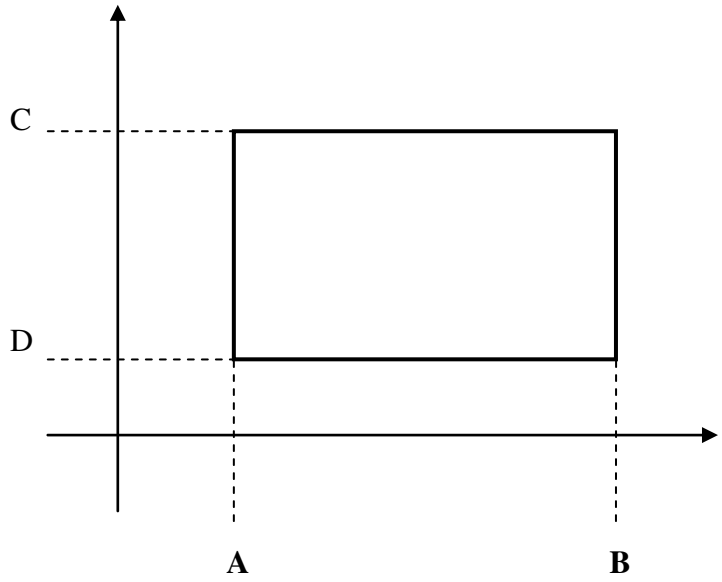

Figure 1.6: Boundary Region for the variable α and β

There are generally three major types of Boundary value analysis techniques which are followings:

a. Boundary Value Checking

Boundary value Checking is the type of testing in which the test cases are designed by fixing one variable at its extreme ends and other input variables are kept at their respective nominal values. The values of the variable at its extreme end can be fixed at followings:

- Can be fixed at its Minimum value (Min)
- Value just above the minimum value (Min+)
- Can be fixed at its Maximum value (Max)
- Value just below the maximum value (Max-)

In boundary value checking method 4n+1 test cases can be designed, where n is the number of variables.

b. Robustness Testing

In Robustness Testing two more cases are included in addition to the cases included in boundary value checking. The additional cases are as followings:-

- A value just above the Maximum value is taken in account (Max+)
- A value just below the Minimum value is also taken in account (Min-)

In this case for n input variables 6n+1 test case are designed.

c. Worst Case Testing

Worst Case Testing is a type of testing in which more than one variable are kept at their extreme end. In worst case testing method for n input variables 5^n test cases are designed.

White Box

It is a type of application testing in which testers are fully aware of the internal structure of the program. Testers have full knowledge regarding the structure of the program. They are concerned how the input is processed and the output is generated. In a white box approach the inner details internal of the application under test are reviewed by the tester [28]. Statement coverage and decision coverage identifies part of the program is accurately tested and also determines the dead code. White Box testing is basically of three types which are as followings:

a. Control Based Flow

Control Flow Graph (CFG) is the important entity of the control based flow testing. It basically depicts the flow of control in the program [25]. It requires the complete awareness of the internal structure of the program that is the reason this type of testing is mostly done by the developer of the program.

The important parts of the CGF are as followings:

- Node
- Edges or Links
- Decision Node
- Junction Node
- Regions

In Control based flow testing all the path's in the program are explored and are tested in order to check whether all the regions in the code are reached or tested.

Some of the Basic Path Testing Terminologies' are as followings:

- Path
- Segment
- Path Segment
- Length of a Path
- Independent Path

- Cyclomatic Complexity: Complexity in general is defined as the degree to which a system or component has a design implementation which is difficult to understand and verify. It is denoted with the symbol V(G).

V(G) = e-n+2P

V(G) = d + P

V(G) = number of Regions in the graph

Where,

e is the number of edges,

n is the number of nodes,

d is the number of decisions and **P=2.**

b. Data Based Flow

Data based flow testing is a type of testing technique in which the actual flow of data is depicted i.e. which line is dependent on which line as the value of the variables changes [29]. It basically probes for improper use of value of the variables due to logical errors. It also examines the various changes in the state of the data values in the control flow graph which in return increases the number of test cases in the test plan by executing branch coverage, all statement coverage testing strategies.

Basic Terminologies used in Data Flow Testing are as followings [28]:

Definition Node: Definition node basically contains followings:

- Loop control statements
- Input statements
- Assignment statements
- Procedure calls.

Usage Node: Usage node contains followings:

- Output statements
- Assignment statements (Right)
- Conditional statements
- Loop control statements

Loop Free Path Segment

Simple Path Segment

Definition-Use Path (du-path): A du-path with respect to a particular variable v is a path from definition node to the usage node of that variable. Usage node can be of p-usage type or c-usage type.

Definition-Clear path(dc-path): A dc-path with respect to a specific variable v is a path from definition node to the usage node with a constraint that there will be no other node along the path is a defining node of variable v.

c. Mutation Testing

Mutation testing is a type of testing in which some erroneous code is embedded in the code segment, then a test set is generated by using which mutated code is tested. After executing the test plan the results are documented that whether the test data has detected the erroneous code or not [23].

Mutation testing helps the developer to prepare test plans and test data by interacting with the code and also strengthen up the quality of test data on performing mutation testing iteratively. While mutation testing many versions of the programs are created by introducing a new fault one at a time. These faulty versions of the programs are called mutants of the original programs and are executed with the specially designed test cases.

Grey box

Grey Box testing stresses at both functional part and structural part of the program so the grey box testing is basically combination of black box testing and white box testing. The scope of this testing is to probe the code for defects if any due to improper structure or usability of the product [24].

A tester who performs black box testing doesn't have knowledge about the internal structure of the program to be tested, while in contrast a tester who performs white-box testing does have the knowledge about the internal structure of the application. A tester who performs gray-box testing has partial knowledge about the internal structure and functional requirements.

1.7.3 Other Miscellaneous Types of testing

Unit Testing

It is a type of testing in which a single unit or group of interrelated units are being tested. It basically comes under the category of white box testing i.e. the tester must be aware of the internal structure of the program to be tested, thereby usually done by the developer of the code.

Integration Testing

It is a type of testing in which a number of programmable assets are combined and tested for their given output. In integration testing the communication between the

software and hardware is being checked in case they have some sort of relation. It comes under the category of both white box and black box testing.

Functional Testing

It is a type of testing which is done to verify whether the developed product fulfils the entire functional requirement or not. It comes under the category of black box testing

System Testing

It is a type of testing technique in which the whole system is tested to verify that it works in different operating system or not. It comes under the category of black box testing..

Stress Testing

It is a type of testing technique in which the system is tested in order to evaluate that how the system performs under varying and unfavourable conditions. In this case the load is given to the system beyond its specified limits. It comes under the category of black box testing.

Performance Testing

It is a type of testing in which the speed and effectiveness of the system is documented. In this technique the tester ensures that the system is generating required outputs within a specified time. It comes under the category of black box testing.

Usability Testing

It is a type of testing which is performed by keeping user's perspective in mind. In this type of testing it is evaluated that the system is user-friendly or not, whether the system is easily understandable or not, easily accessible or not, whether the GUI design is pleasing or not. It comes under the category of black box testing.

Acceptance Testing

It is a type of testing which is usually done by the customer to make sure that the deployed software product meets his/her requirements and works as intended. It comes under the category of black box testing.

Regression Testing

It is the type of testing technique which is performed after making some changes in the system, component to ensure that the changes being made is working correctly and is not damaging or affecting the functionality of other components. It comes under the category of black box testing.

Beta Testing

It is a type of testing which is basically performed by the end user, individual or a group of persons outside other than developers or while releasing demo version of the product. Beta testing mainly focuses on locating undesirable errors. It comes under the category of black box testing.

1.8 Test Case Generation

The test case generation approaches available till can be categorized as followings [20]:-

Path oriented techniques

Path-oriented techniques basically generates the test cases from the control flow of the program and identifies all the set of possible paths that must be covered while testing and also signifies about the dead region in the code. These techniques can further be classified in *static* and *dynamic* ones. Static techniques are often based on symbolic execution [21], whereas dynamic techniques obtain the necessary data by executing the program under test.

Goal oriented

Goal-oriented techniques works by defining test cases which will cover the predefine goal i.e. which decision or branch to required to be covered.

Random techniques

Random techniques are purely based on assumption i.e. hit and trial approach [22].

1.9 Organization of thesis

Chapter 1 is the introductory part which gives brief introduction about component based software engineering, need of repository in component based software engineering and it also provides brief details of various component testing strategies.

Chapter 2 provides brief description of various types of existing component repositories along with the detail study of their searching and retrieval methods.

Chapter 3 gives the problem definition along with the research gap and scope of the thesis.

Chapter 4 provides the implementation of repository with grey box testing support and validation of the implementation.

Chapter 5 gives the conclusion and future scope of the implemented approach.

Chapter 2

Literature Survey

2.1 Various Component Repository

Different type Repositories are one of the essential requirements of different Communities/Organizations due to different their functionalities and requirement. In this section different existing repositories has been described in brief. Basically repositories are classified into two categories which are commercial and Government Repositories.

2.1.1 Commercial Repositories

+1Reuse Repository

+1Software Engineering Co. Developed a repository named "+1 Reuse system" which supports the reuse of architectural designs, system documentation, executable code, preprocessed files, modeling information and test suites with their expected test outcomes. Presently +1 Reuse system is working on Sun Workstation platforms with Solaris as Operating system [4]. It provides other significant features like GUI support and common desktop environment.

Software Asset Library Management System (SALMS)

It is a system for categorizing, defining, and searching reusable assets [20]. It is a software system which follows standardized classification scheme and provides efficient mechanism for both storage and retrieval of the software components for future reuse where similar functionality is required.

Salient features of SALMS:

i. It provides supports to the activity of the management, and it helps in the management of the repository and creation of the technical repository.
ii. It supports distributed computing i.e. all the developers in an organization can access the SALMS system by connecting all PC's over the network.
iii. SALMS provide Web based support to all the users.

Automated Software Reuse repository (ASRR)

It is a software system which assists developers by providing them a repository of components [6]. This tool has two main parts which are as followings:

i. The administration part: This part of the tool plays the role of administration and performs the functionalities like the addition, deletion, or grant /revoke user privileges and their attributes like security levels, and other functionalities like access to add new reusable component, edit and delete the same.
ii. Repository: It basically allows the developers to upload assets for storage in order to reuse the same assets to fulfill the similar kind of future needs and allows efficient extraction of the same.

Salient features of ASRR are as followings:

a. Program Control: Valid ID's and passwords are assigned to all the users which are checked at the time of Login.
b. Protection: The ASRR grants and revoke the user's privileges to access the repository like it can limit the number of assets any particular user can store, access and retrieve.
c. Security: Suppose any user which remains inactive for more than 30minutes, the ASRR system automatically logs out that particular user.
d. Easy Access to Reuse Items: It provides easy access to the storage area hence the searching process becomes very easy.

The Universal Repository

The Unisys develops a Universal repository which forms its basis on the principles of object-oriented approach which can act as a core part of an organization. Essential part of the repository is the Repository Services Model (RSM) - which can accumulate all the necessary and sufficient tools, database management system for storage and management of information, programming languages, syntax and semantics, and data.

Customers can add new features to the Universal Repository by adding their self-generated methods in accordance to the framework delivered in the RSM. The combination of all methods described in a storage structure is known as the information model [7].

Online support for learning and training programs are available to help system users to rapidly adopt this new technology and shared catalogue are also available of all software components.

Reuse Library Toolset (RLT)

In 1994 EVB Software Engineering, Inc. developed the Reuse Library Toolset (RLT) [8]. RLT is a system which supports the reuse of architectural designs, system documentation, source code, header files, and modeling information which in turn is totally independent of language being used, methods to design.

Salient features of RLT are as followings:-

i. It provides very simple graphical user interface thus it's very easy for novice to understand and also facilitate yet multifaceted functionality for professional developers.

ii. It also provides additional features like reuse metrics, client-server architectural support and has potential to communicate repositories useful information among heterogeneous platforms and database.

2.1.2 Government Repositories

Defense Software Repository System (DSRS)

It provides automated support for saving and retaining Reusable Software Assets (RSAs) [10]. There are seven software reuse support centre's (SRSCs) which uses DSRS software for managing inventories of reusable assets. The DSRS serves as a central core location for quality and management. Reusable Software Assets provides the software assets so easily by matching their required functionality with existing software applications which has posses same functionality. Government employees and other contractors having government projects have their own DSRS accounts for accessing DSRS. The DoD software community is trying to change its software engineering model from its current software cycle to domain-specific, process-driven, repository-assisted, architecture-based way of constructing software [10]. In these rapid changing conditions, the DSRS has the heyday potential to become standardized repository because it is the only repository with operational features and is interoperable features.

Library Interoperability Demonstration (LID)

The only prototype library system is Library Interoperability Demonstration (LID) [10]. Basically it is used to illustrate how huge storage structures can break into discrete, functional layers connected by open interfaces, such as those described by Asset Library Open Architecture Framework (ALOAF). A reuse library can be distinguished into three separate layers which are connected with the help of open interfaces. Interoperability is possible at each layer [10].

LID has three layers which are as followings:

i. User Interfaces: In LID there is a GUI based browser and it has one keyword based browser for manual searching.
ii. Asset Catalogs
iii. Asset Storage: It is the actual place where the components are stored which is provided by AFS cell by STARS technology.

2.2 Comparison of Various Repositories

Types of Repository	Web Based support	Security Control Feature	Retrieval Method
+1 Reuse Repository	Not supported	Yes	Browsing method
SALMS	Yes	Yes	Keywords based search
ASRR	Not supported	Yes	Keywords based search
Universal Repository	Not supported	Yes	Both Browsing and Keywords based search
RLT	Not Supported	Yes	Keywords based search
DSRS	Yes	Yes	Keywords based search
LID	Not supported	Yes	Keywords based search

Table 2.1: Comparison of various repositories [10]

As per the survey conducted by Luqi and Jiang Guo on software reuse repository [10], it has been concluded that keyword based search is the most commonly used retrieval methods and most of the repositories are web-based applications as shown in table 1.

2.3Techniques of Storage And Retrieval

Manual Storage of Components

The component can be uploaded by users along with the suitable descriptions of an individual component manually, e.g. .exe, .dll components.

Steps for manual storage are as followings:

i. Upload the Components so that they can be stored and reused later whenever same functionality is required.
ii. Repository must allow user to enter component details manually which will enable developer to find the suitable component according to their requirements.

Auto-Detect-Fragment-Store

This approach firstly detects the inner functional details used during its development and fragments the inner details. In order to reuse the component, software developers explores the repository for function name, input, output interfaces. Steps of the approach are as followings:

i. Upload Source-code component: Firstly upload the source-code component which we want to store in the repository for future purpose.
ii. Auto-Detection of interfaces: For auto detection the algorithm goes through the inner details of the source-code component.
iii. Fragmentation of Source-code: At last it the source-code components are fragmented by Function inputs [2] which are:
    - **Function Signature**:-Function name in the form of attribute can also be used.
    - **Module's Return Type:**- Actual Return type of that function.
    - **Number of arguments of that function**
    - **Data Type of each parameter**
    - **Location where the component is stored**
iv. Efficient Inner detail storage: Fragmented details are stored in the form of attributes in the database which will help the users for easy and efficient retrieval of the components.

Factors which play a significant role for effective retrieval of components from repository are as followings [2]:-

- Component Name
- Component ID
- Component Description
- Component Type (.doc,exe, .dll etc…)
- Language in which the component is developed (C, C++, C#, VB etc…)
- Type of Domain (Desktop, Database or Web-based application)
- Number of inputs required

- Number of outputs provided
- Descriptive Document
- Keyword or Tags (Closely related keywords)

Component Retrieval by Keyword Search

In this technique keywords are inputted to the application and are matched i.e. longest possible sequence of characters are matched. On matching longest sequence of characters the component library displays set of possible assets from where user can choose the required one.

Component Retrieval by Fixed Inputs

Components can also be retrieved by providing fixed inputs. Fixed inputs are those whose values cannot changes:

- Type of the component
- Language in which the component is developed
- Domain of the component i.e. whether the component is console based, web based or window based

Text Based Search Method

In this method, the whole component repository is searched and appropriate matched components are displayed for retrieval.

Signature Based Search Method

The user can also search the component on the basis of any particular signature of the components. This type of search is much more efficient than others as it provides an exact match from the requirement.

Operational Semantic Based Search Method

In operational semantic keyword based search, the user will give an input i.e. component name and its prototype, and then the relevant components are displayed along with its input and output. In this we match the candidate asset against a user query on the basis of the candidate behaviour on sample inputs. The operational semantic based search methods are more efficient and provide the exact retrieval of the component. Therefore the precision and recall ratio is better in this case as compare to the other methods.

Retrieval of Software Components from a Heterogeneous Repository

Software component retrieval methods are active realm of research in the technique of component-based development. A semantic syntactic approach can be used to extract components from a versatile component storage structure which is also known as the software supermarket [12]. In software supermarket a label called as component label is attached to each component which will be used for effective and precise retrieval of component. The label provides description of the component and also have semantic of the assets. In this approach a label known as "complete label" is used which has eleven sections which will be inputted in a query given by user "complete query" which will probe the SoS. In order to make the query less cumbersome, the component label is divided into following parts:-

i. Complete label: It has all the important and required sections represented on the label.
ii. Complete query: There is very little different between the complete label and complete query; the only difference is that difference the information is given by the user, whereas the component label is given by the developer.
iii. Required query: In addition to complete label, a required query consists of information like the type of the component, domain in which the component is useful.

In order to retrieve the software assets the function *matchcomparison()* is defined.

Z specification language is used to write semantic description of the complete label. Formal notations in the Z specification language are represented in two ways i.e. specification statements and by using schema definitions. In this approach we will use specification statements. A specification statement basically consists of a variable list, two predicate upon state which are precondition state and a post condition state of the form:

Variable list: [α, β]

Where α depicts preconditions and β depicts post conditions

Suppose a semantic description, *computeSemantic*, on a component label can be represented as:

Vaiable list: [α, β]

and a query semantic description, *querySemantic*, given by the user:

Variable list1: [α1, β1]

Now user will specify set of input values (α1) and the output value (β1) expected from the component they are requesting. If the value of post condition β and post condition β1 is equivalent, i.e. the repercussion of two specifications are the equivalent given the same input values, and then we can make a conclusion that both the components have same functional behavior.

if α= α1 and β = β1

// (perfect match)

OR

if α= α1 and α && then β1 β then

Variable list: [α,β] ⊆ Variable list1: [α1,β1]

// (weak match)

OR

if α= α1 and α1 && then β1 β then

Variable list1: [pre1, post1] ⊆ Variable list: [pre, post]

// (strong match)

2.4 Precise Search and Exact Retrieval of the Components

The basic criteria for comparing the various storage and retrieval methods are [16]:

i. Precision: The precision of a retrieval method can be defined as the ratio of the relevant retrieved components over the total number of retrieved components. The ratio ranges between 0 and 1. Under the hypothesis if all the stored components are traversed we get perfect precision i.e. equal to 1. When specific data values are not available to quantify precision, in that case we can assign values in a discrete five value rating [very low (VL), low (L), medium (M), high (H), very high (VH)] where, VH implies perfect precision and VL implies poor precision.

ii. Recall: The recall of a retrieval method can be defined as the ratio of the relevant retrieved components over the total number of relevant components in the library. Recall ratio ranges between 0 and 1. Under the hypothesis if all the stored components are traversed we get perfect recall i.e. equal to 1. When specific data values are not available to quantify recall, in that case we can assign values in a

discrete five value rating [very low (VL), low (L), medium (M), high (H), very high (VH)] where, VH implies perfect recall and VL implies poor recall.

The aim of any retrieval method is to amplify precision and recall, so as to provide the effective retrieval.

2.5 Cyclomatic Complexity

Complexity in general is defined as the degree to which a system or component has a design implementation which is difficult to understand and verify. Complexity metrics are used to predict critical information about reliability and maintainability of software systems [14].

McCabe proposed the Cyclomatic Complexity metric in 1976, which is widely accepted till now. The important aspect of Cyclomatic Complexity metric is that, it is independent from both language as well as its format. Control flow graph (CFG) of the program is formulated which is used to compute this metric. The nodes (point) in CFG basically represent the statement or line of the program and nodes are joined with the help of directed edges in the sequential flow of the program.

McCabe [2] proposed a mathematical technique which explains the flow of the control in the program with the help of directed graph in which nodes represent statements and directed edges predicts the flow of the program. McCabe defines that complexity count does not depends on the LOC of the program but depends upon the decision structures in the program.

Following theorem and corollary are used to compute Cyclomatic complexity [14]:

Theorem 1: The Cyclomatic count of graph is symbolically denoted by V (G) as in equation in 1.

$$V(G) = e - n + 2 \qquad (1)$$

Where e denotes number of edges and n denotes number of nodes in the graph

Corollary 1: The Cyclomatic count V (G) of a graph G can also be computed as in equation 2 by making the use of the predicate node d (No decision structures in the program)

$$V(G) = d + 1 \qquad (2)$$

Corollary 2: The Cyclomatic count also depends upon the number of regions r in the program as in equation 3.

$$V(G) = r + 1 \qquad (3)$$

According to McCabe complexity, count increases by increase structures and the complexity count can be reduced by removing unnecessary branches. The contribution of each control structure has been listed in the Table 1. [10]

Control Structures	**Complexity Count**
If	+1
ElseIf	+1
For	+1
While	+1
Do-While	+1
Else	0
Switch	Number of Cases +1

Table 2.2: Complexity Count of various Control Structures

Chapter 3

Problem Definition

Instead of reusability, the common approach that is being used by software developers is developing once and uses it once. Huge amount of development efforts, cost incurred and time goes in vain if the software is developed from scratch every time without making use of previously developed components. These efforts can only be saved if these components must be reused whenever similar functionality is required while developing other projects. In order to increase the availability and easy accessibility of reusable components, we need an effective storage structure to store the reusable components. Software repository is a set of software assets that are maintained by organization for possible browsing and/or retrieval [36]. Application of software repository is software reuse. Software reuse can be defined as the process by which new systems or software products are developed by using previously developed components.

Repository must be developed which supports essential features like efficient storage, retrieval, provide inner details of the stored components and proper documentation etc.

3.1 Research Gap

Earlier repositories were developed which posses mainly features like efficient storage and retrieval of components like keyword based, browsing, web support and security control [10]. Later on the repository were developed which posses other feature like automated fragmentation of code or component which will fragments the inner details of the code, automatically detects the interfaces, stores the location of the component and provide complete documentation about the stored components [3].

3.2 Objectives of Study

i. To study the Component Based Software Engineering.

ii. To study the different types of existing Component Repositories and testing strategies.

iii. To develop component repository for component storage and retrieval.

iv. Implementation of Grey Box Testing Support to the developed Component repository i.e. the repository will provide both Black box and White box testing support while storing the component.

3.3 Scope

Repositories are the important part of every organization whether it's a large scale organization or a small scale organization, because some kind of storage structure is required to store In-house developed component in order to support reusability of the developed components.

The study finds its usefulness in the software organization which requires efficient repositories for storage and retrieval. Now a repository is developed which has grey box testing support with an efficient storage structure i.e. it will provide both black box and white box testing support to the reusable components and will store all the test cases generated while performing grey box testing on the component for future use.

Chapter 4

Implementation and Validation

4.1 Implementation

The Thesis entitled "Component Repository with Grey Testing Support" has two sections which are as followings:-

Selective storage

In this section of the project selective storage of the component takes place i.e. only selected components are stored which ensures which future reusability. A gray test tool has been developed by using .Net framework with C#.

Grey-box testing is a combination of white-box testing and black-box testing. The aim of this testing is to search for the defects if any due to improper structure or improper usage of applications. By using this tool we can generate test set by using black box test generation approach or by path coverage approach.

Suppose we have all the specifications for a particular developed product or code then we can use black box test case generation approach to generate appropriate test cases. In case we do not have all the specifications but we do have the working code of the application or program then in that case we can use path coverage test generation approach. In this section both the approaches have been discussed in details with snapshots which are discussed below.

Followings are the steps to work with the repository:-

i. Choose "**Select item**" in order to retrieve and "**Add item"** in order to store new component to the repository.

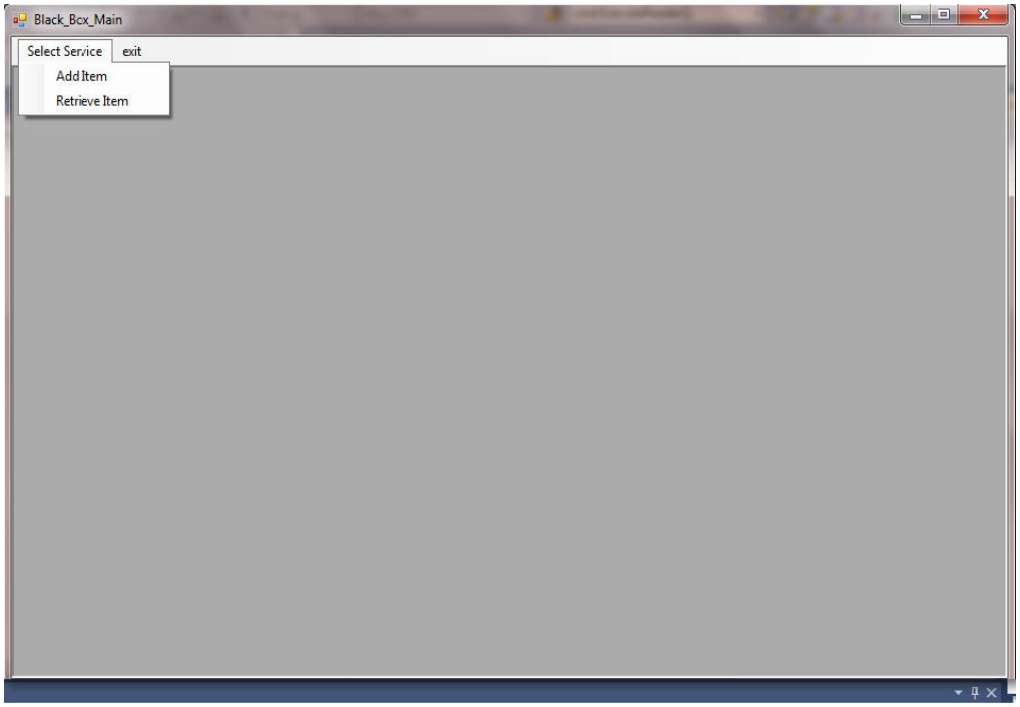

Figure 4.1: Action Selective Window

ii. In order to add item upload the code of the component and Press "**Generate Test Case**" Button.

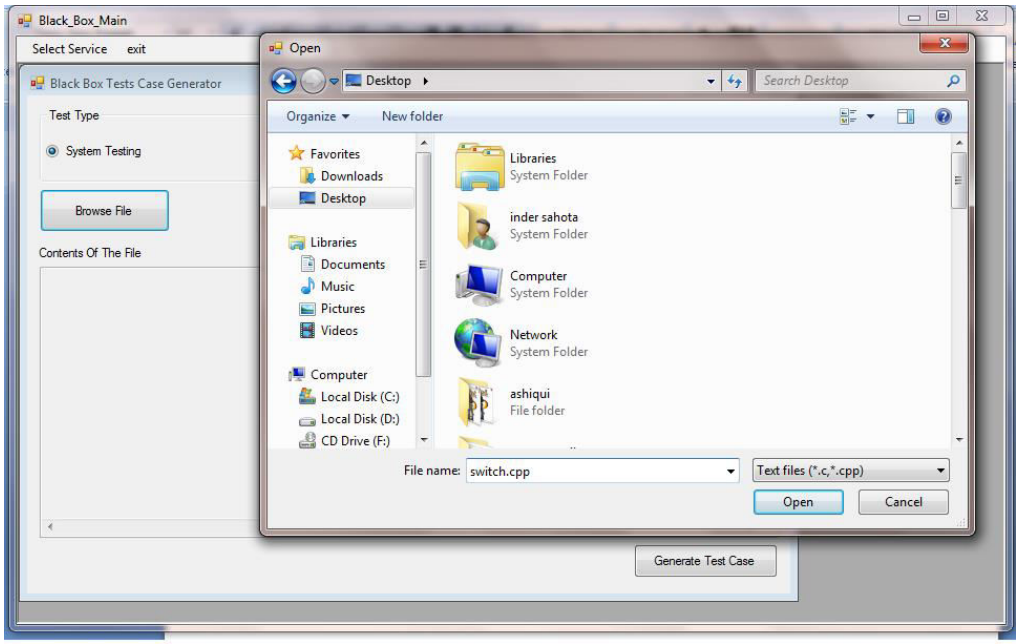

Figure 4.2: Upload Component to be Reused

iii. As per Software Requirement Specification, it is mentioned the maximum and minimum required value. Therefore, enter the maximum and minimum value for both input and output variable and press "**Generate Test Case**" button.

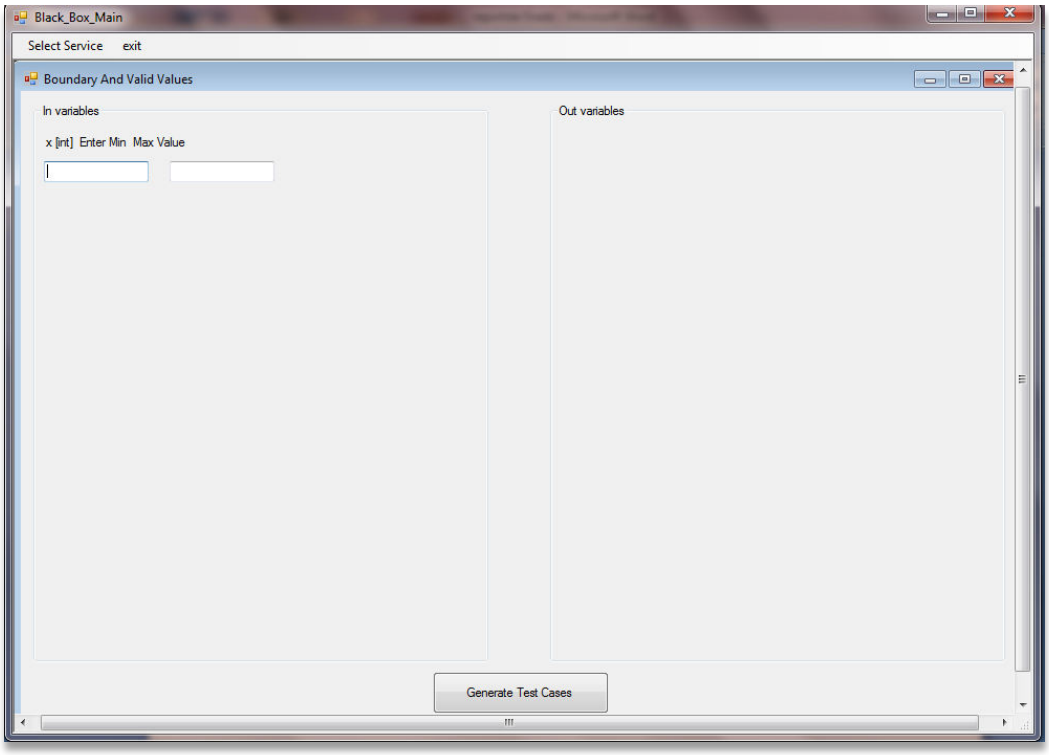

Figure 4.3: Enter the Range of all the Variables

iv. Now the test cases according to the Boundary value checking and Robustness testing are generated.

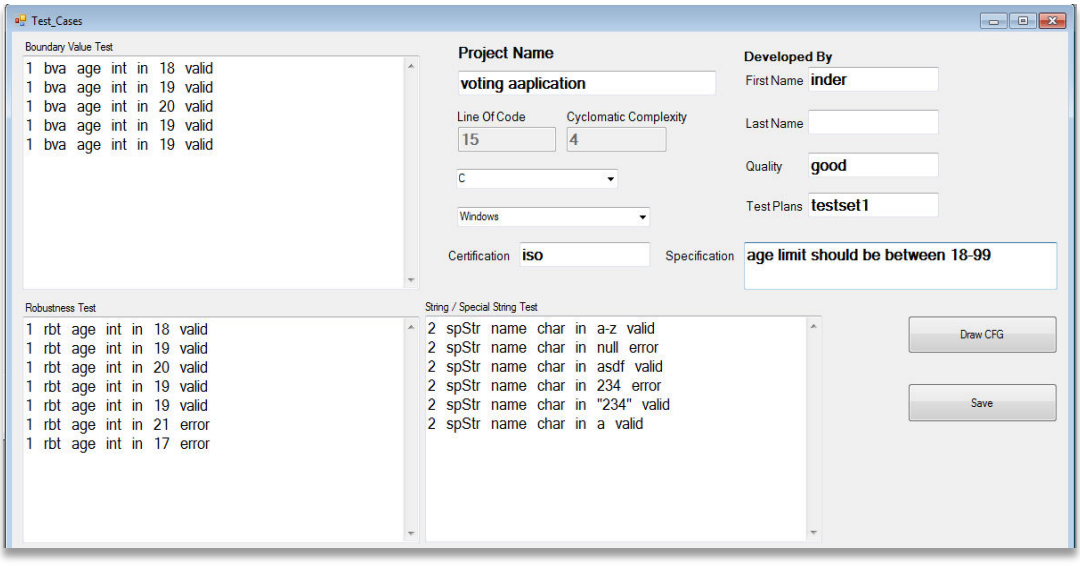

Figure 4.4: Test Set for a particular Program

v. Now, Firstly press "**Save**" button in order to save the generated test case and secondly press "**Draw CFG"** button in order to perform path coverage testing on the particular component.

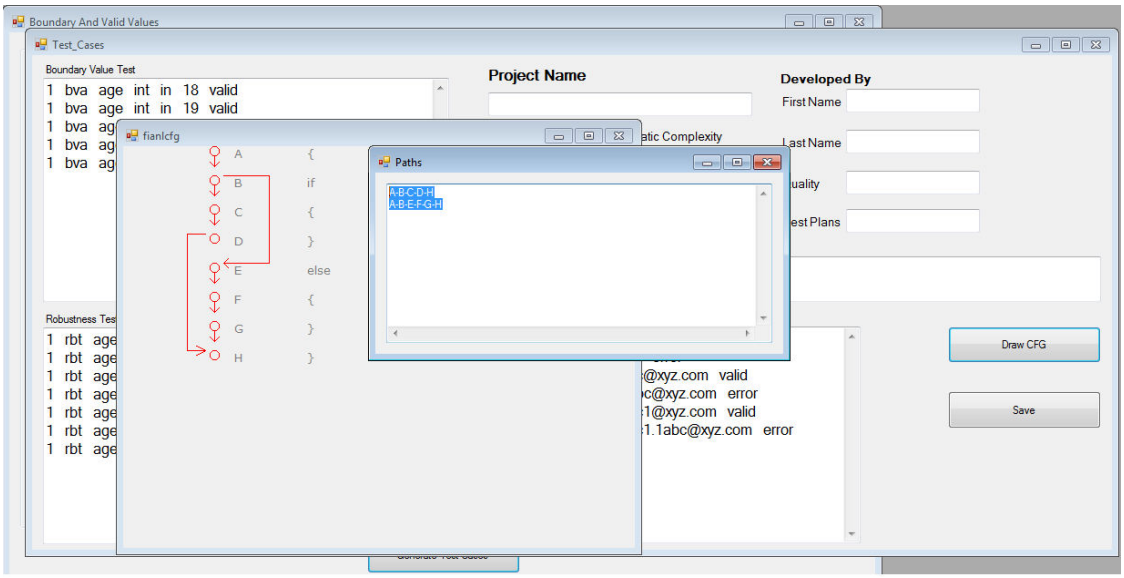

Figure 4.5: Control Flow Graph with all the Paths that are to be covered

At the same time matrix representation of the Control Flow Graph is also generated and displayed.

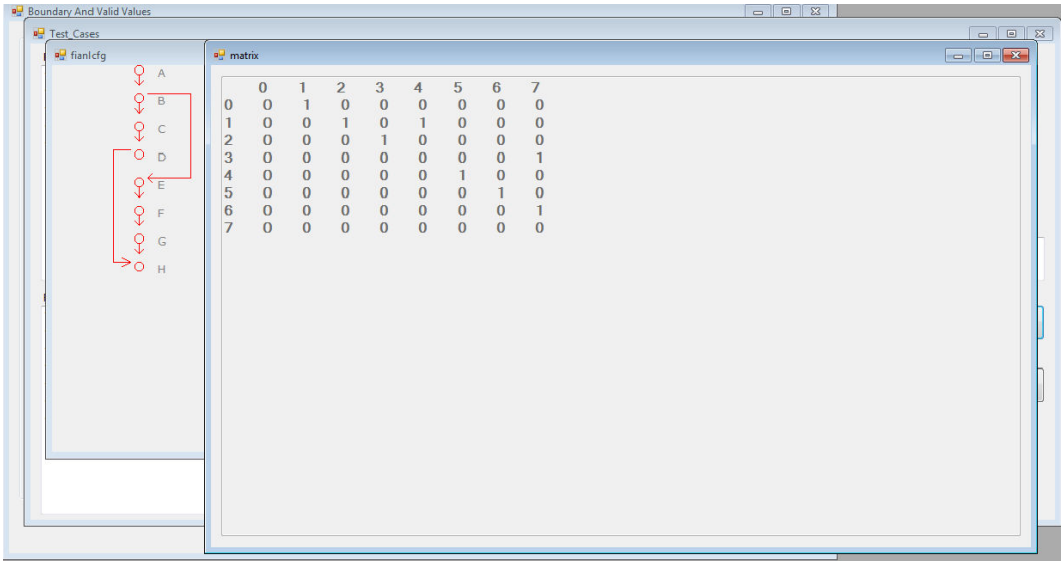

Figure 4.6: Matrix Representation of the Control Flow Graph

Component Retrieval

In the second section the components are retrieved on the basis of function name. In this tool keyword based approach is used for efficient retrieval. In this technique keywords are inputted to the application and are matched i.e. longest possible

sequence of characters are matched. On matching longest sequence of characters the component library displays set of possible assets from where user can choose the required one.

i. Enter the keyword in order to search the project in the repository.

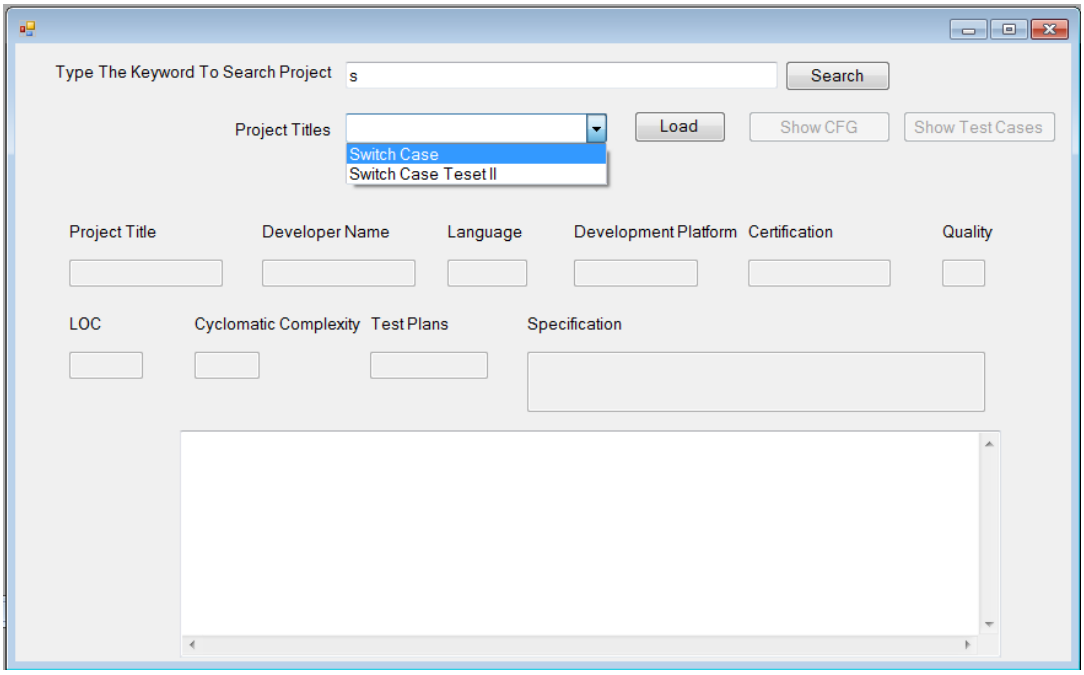

Figure 4.7: Keyword bassed Searching from Database

Like in figure 1 just "s" is entered and then all the project whose title includes keyword "s" are shown in the dropdown list menu.

ii. Choose the project entitled from the list of retrieved product on matching particular keyword and press Load.

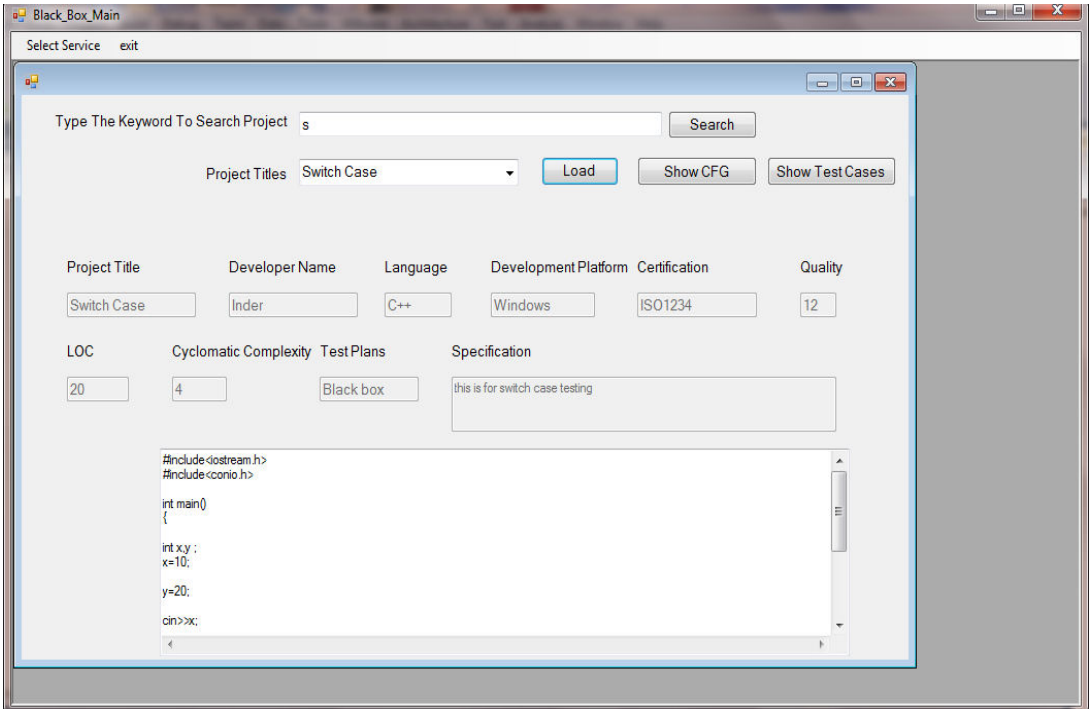

Figure 4.8: Chosen Project is loaded with Corresponding Details

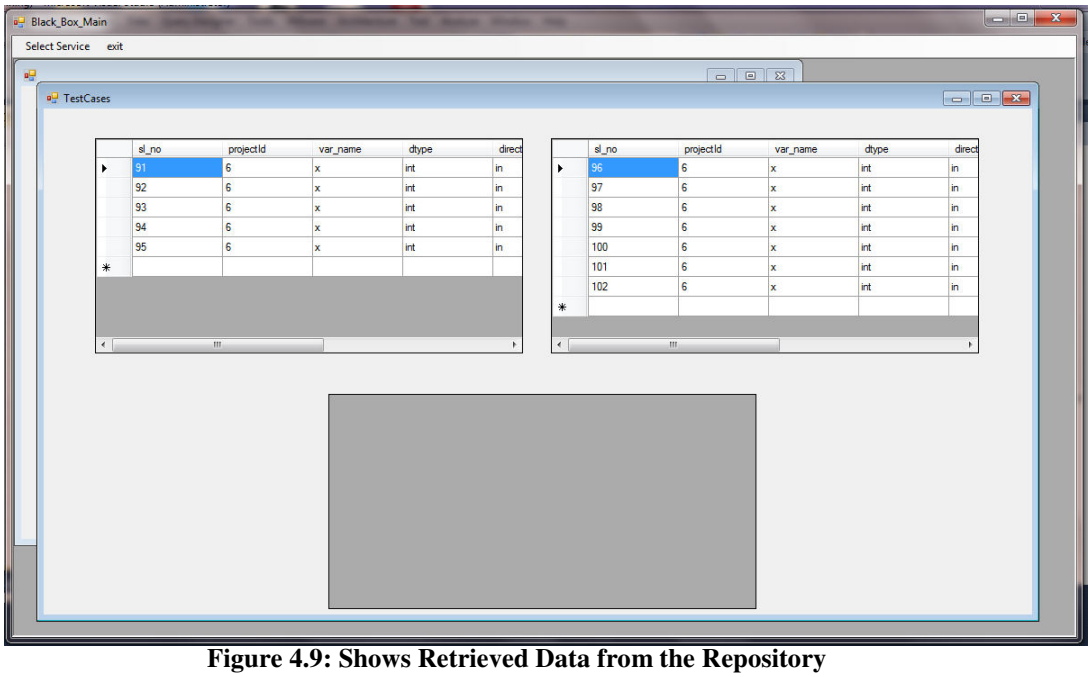

Figure 4.9: Shows Retrieved Data from the Repository

Tables at the Backend

There are basically two tables in the backend which are as followings:-

i. **Table to store the general Description of the component to be stored.**

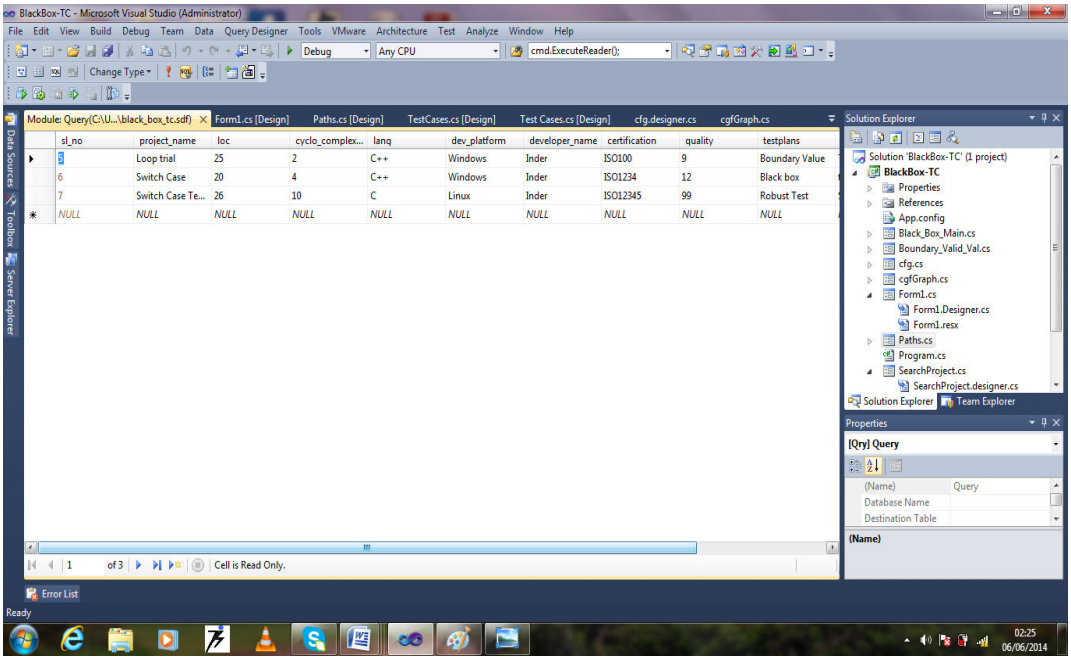

Figure 4.10: Table which stores basic Information about the Component

ii. **Table to store all the test cases second table is maintained.**

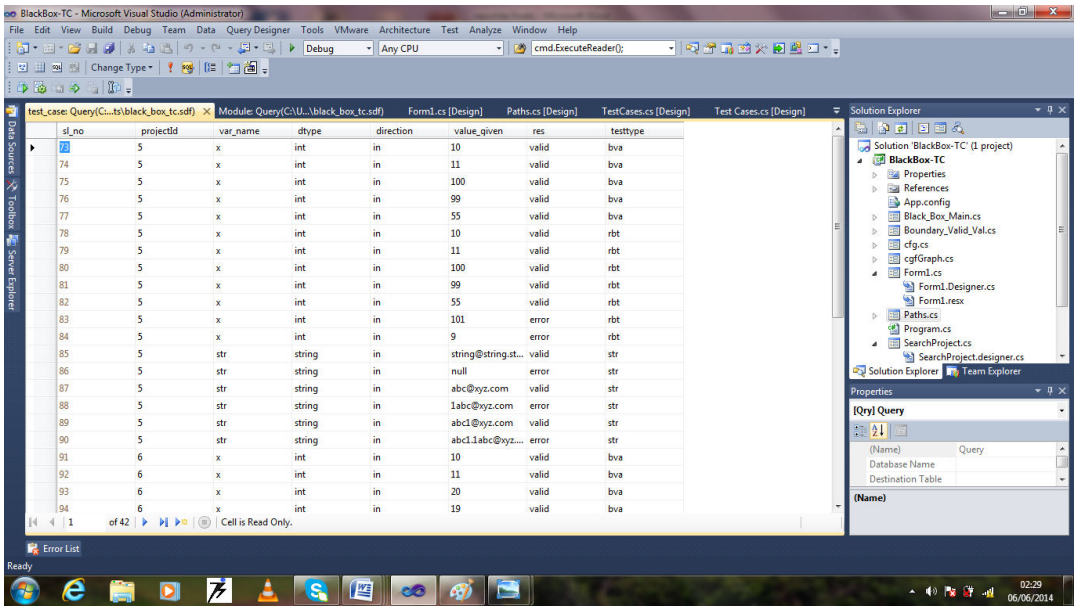

Figure 4.11: Various Test Cases are Stored in this Table

4.2 Validation

Repositories are the important part of every organization whether it's a large scale organization or a small scale organization, because some kind of storage structure is

required to store In-house developed component in order to support reusability of the developed components.

Earlier repositories were developed which posses mainly features like efficient storage and retrieval of components [10]. Later on the repository was developed which posses other feature like automated fragmentation of code or component to be stored [3]. In this study the repository has been developed for small scale organization which not only provides key features of storage and retrieval of component but also provides grey testing supports i.e. repository provides both white box and black box testing support to the components being stored for the future purpose.

Black box testing is a type of testing in which testers are unaware of the internal structure of the program. In black box testing the test suite is prepared on the basis of requirements presented by the client. In order to provide black box testing support Boundary Value analysis has been implemented and test coverage technique along with Cyclomatic complexity metric are considered as best deliverable to provide white box support to the components[14,24].

Chapter 5

Conclusions & Future Scope

5.1 Conclusions

Reusable software component may include source-code, algorithm, executable code, test cases, test plans and documentation. Developing a software component from scratch takes lot of efforts and development time as well; these efforts can be saved by reusing the pre-existing components. Now a repository has been developed which provides grey testing support for the components to be stored for future perspectives i.e. both functional requirement and structural requirement will be checked and respective test cases will be generated and stored.

5.2 Future Scope

This work opens up on a number of avenues for future research. The work can be extended in several directions and some are summarized below:

It can be further can be extended on automatic storage of different types of components like test cases, test plans, UML diagrams as it is only deployed on Source-code.

Enhancements can be made in retrieval methods which will provide more accurate and effective results.

References

.

1. R.S Pressman, *Software Engineering-"A Practitioner's Approach*," New York: McGrawHill International Ltd., 2010.
2. P. Vohra, A. Singh, "A Tool incorporating different Techniques for Effective Component Storage and Retrieval," *In International J. of Scientific & Engineering Research,* Vol. 4 , pp. 1134-1137, 2013
3. P. Vohra, A. Singh, "Automatic Fragmentation and Storage of Code in Component Repository with respect to their Input and Output Interfaces," *A Tool, International J. of Innovative Technology and Exploring Engineering*, Vol. 2, pp.235-238, 2013.
4. "+1 Software Engineeering Corporate Mission", Available at: http://www.plus-one.com/company .html[18Dec, 2013].
5. J.C Huang, "An Approach to program testing," *ACM Computing Surveys,* Vol. 7, No.3, pp. 113-128, 1975.
6. "Project Management Tool Suite System (Automated Software Reuse Repository)", Available at: http://wv.ewa.com-/srr_overview.html[19Dec, 2013].
7. "Universal Repository", Available at: http://www.marketplace.unisys.com/urep [19 Dec, 2013].
8. "EVB Software", Available at: http://gopher.metronet.com:70/0/newprod/by-vendor/E-/evb_software_e/941208.01 [18 Dec, 2013].
9. H. Mili, A. Mili, S. Yacoub and A. Edward, "Reuse-Based Software Engineering," *Wiley-Interscience NewYork, John Wiley & Sons*, 1998, Available at: http://dl.acm.org/citation.cfm?id=504675 [20 Dec, 2013].
10. Luqi and Jiang Guo, "A Survey of Software Reuse Repositories", *Research supported by ARO(38690-MA) and DARPA(99-F759) IEEE,* pp. 92-100, 2000.
11. R. Pyne and E. Mugisa, "Essential Elements of A Component-Based Development Environment For The Software Supermarket," *In Proceedings of IEEE SouthEastern Conference*, Vol. 2 pp 223-229, 2004.
12. R. Pyne and E. Mugisa, "Retrieving Software Components from a Heterogeneous Repository: a Semantic Syntactic Approach, "*In Proceedings of the 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems,* Vol. 3, pp 442-449, 2005.

13. J.M Voas, "The Challenges of Using COTS Software in Component-Based Development*," IEEE Computer*, Vol. 31, No. 6, pp. 44-45, 1998.
14. R.S. Chidamber & C.F. Kemerer, "A Metric suite for Object-oriented Design," *IEEE Transactions on Software Engineering*, Vol. 20, pp. 157-165, June1994.
15. G. Kotonya and A. Rashid , "A Strategy for Managing Risk in Component-Based Systems," *In proceedings of IEEE 26th Euromicro Conference,* pp 12-21, 2001.
16. A. Mili, R. Mili and R.T. Mittermeir "A survey of software reuses libraries", *Annals of Software Engineering*, Vol. 5, pp. 349-414, 1998.
17. R. Hennicker, H. Hussmann, M. Bidoit,"On the Precise Meaning of OCL Constraints," *Advances in Object Modelling with the OCL, Springer LNCS,* Vol. 2263, pp. 70-85, 2001.
18. W.B. Frakes and B.A. Nejmeh, "An Information System for Software Reuse," I*n Software Reuse: Emerging Technology, IEEE Computer Society Press,* pp 142-151, 1990.
19. M. Kessis, Y.Ledru, G. Vandome. "Experiences in coverage testing of a java middleware," *5th international workshop on Software engineering and middleware*, pp. 39-45, 2005.
20. E. Morandin, "SALMS v5.1: A System for Classifying, Describing, and Querying about Reusable Software Assets", *The Proceedings of 5th International Conference on Software Reuse* (ICSR '98) Available at:
21. R.P.Pargas, M.J. Harrold, and R.R.Peck. "Test-data generation using genetic algorithms," *Software Testing, Verification and Reliability*, Vol. 9, No.4, pp. 263-282, 1999.
22. C.Ramamoorthy, S. Ho, and W. Chen "On the Automated Generation of Program Test Data," *IEEE Transaction on Software Engineering,* Vol. 2, No. 4, pp. 293-300, 1976.
23. L. Luo "Software Testing Techniques Technology Maturation and Research Strategy," *Institute for Software Research International Carnegie Mellon University*, Available at: http://mcahelpline.com/tutorials/testing/testing.pdf [15 April, 2014].
24. F. Omar, S.Ibrahim"Designing Test Coverage for Grey Box Analysis," *10th International Conference on Software Quality,* pp. 353-356, 2010.

25. S. Liu and Y. Chen, “A Relation-based Method Combining Functional and Structural Testing for Test Case Generation,” *The Journal of Systems and Software, Elseveir Science*, Vol. 81, No. 2, pp. 234-248, 2008.
26. K. Seo and E. Choi, “Comparison of five black-box testing method for object-oriented software,” *In the Proceedings of 4th International Conference on Software Engineering Research , Management and Application*, Vol. 9, No.11, pp. 213-220, 2006.
27. A. Egyed,” A Scenario-Driven Approach to Trace Dependency Analysis,” *IEEE Transactions on Software Engineering*, Vol. 29, No. 2, 2003.
28. “Estimating Test Development and Code Coverage” Available at http://coderanch.com/t/431616/Agile-Other-Processes/Estimating-Test-Development-Test-Coverage [15March, 2014].
29. J, Horgan and S. London, “Data flow Coverage and the C language,” *In the proceedings of the Symposium on Testing, Analysis*”, pp. 87-97, 1991.
30. BS7925-2, Standard for Software Component Testing,” *British Computer Society SIGIST*”, 1997.
31. “Test Coverage”, Available at: http://en.wikipedia.org/wiki/Code_coverage [18 March, 2014].
32. N. Devos, C. Ponsard, J. Deprez, R. Bauvin, B. Moriau, G. Anckaerts, “Efficient Reuse of Domain-Specific Test Knowledge: An Industrial Case in the Smart Card Domain,” *In the proceedings of 34th International Conference on Software Engineering*, pp. 1123-1132, 2012.
33. Evered M., Menger G., Schmolitzky A., Keedy J. L., "Software Reuse in an Object Oriented Framework: Distinguishing types from Implementation and Objects from Attributes", *6th International Conference on Software Reuse*, pp 420-435, 2000.
34. R. Díaz, "Software Reuse: Issues and Experiences", *American Programmer*, Vol. 6, No. 8, pp. 10-18, April 1993.
35. *Sommerville Ian*, “Software Engineering”, 9th edition.
36. Scott N. Woodfield, David W. Embley, and Del T. Scott, "Can Programmers Reuse Software," *IEEE Software*, pp 152-59, July 1987.
37. H.Mili, O. Marcotte, and A. Kabbaj, "Intelligent Component Retrieval for Software Reuse," *In proceedings of the Third Maghrebian Conference on Artificial Intelligence and Software Engineering*, pp. 258-265, April, 1994.

38. J.M. Conrad, M. Baldwin, S. Curran, and L. Martin, “Using a New Software Product Development Process for a Code Reuse Project,” *In proceedings of the 1999 Engineering of Computer-Based Systems Conference*, pp. 34-40, 1999.
39. T. Ravichandran and A. Rothenberger, "Software Reuse Strategies and Component Markets", *ACM Transactions on Software Engineering*, Vol.46 No. 8, pp. 109-114, August 2003.
40. G.Fischer, "Cognitive View of Reuse and Design." *IEEE Software*, pp. 60-72, July, 1987.
41. G. Caldiera and V.R. Basili, "Identifying and Qualifying Reusable Software Components", *IEEE Computer*, Vol. 2, No. 2, pp. 61-70, February 1991.
42. Jones T. C., "Reusability in Programming: A Survey of the State of the Art," *IEEE Transactions on Software Engineering*, Vol. 10, No. 5, pp 488-494, September 1984.
43. D.M. Balda and D.A. Gustafson, "Cost-estimation Models for the Reuse and Prototype Software Development Lifecycles", *ACM SIGSOFT*, pp. 42-50, July 1990.
44. R.G Fichman and C.F Kemerer ., "Incentive Compatibility and Systematic Software Reuse", *Journal of Systems and Software,* Vol. 57, No. 1, pp 45-60, 2002.
45. *M. Sharma and R. Mall, “*Automatic Generation of Test Specifications for Coverage of System State Transitions*,” Information and SoftwareTechnology,* Vol. 5, pp. 418 – 432, 2009

List of Publications

.

COMMUNICATED

1. I. Singh and A. Singh," A Survey on various Component Repositories with Detail Study of different Methods and Extraction of Components", Paper ID: 1569973823 (IC3 2014).
2. I. Singh and A. Singh, "Grey Testing Support in Software Repository with Keyword based Extraction", Paper ID: 1569982467 (ICACCI Industry Track)