International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 4s
DOI: https://doi.org/10.17762/ijritcc.v11i4s.6568
***Article Received**: 30 January 2023 **Revised**: 05 March 2023 **Accepted**: 20 March 2023*

An Automated Framework for Detecting Change in the Source Code and Test Case Change Recommendation

[1]U. Sivaji, [2]V. Mahalakshmi, [3]G S Sivakumar, [4]Sasirekha. R, [5]R Venkataramana

[1]Associate Professor, Department of IT, Institute of Aeronautical Engineering, Dundigal, 500043, Telangana, India.
Email: sivaji.u117@gmail.com

[2]Assistant Professor, Department of Computer Science, College of Computer Science & Information Technology, Jazan University, Jazan45412, Saudi Arabia.
Email: mlakshmi@jazanu.edu.sa

[3]Associate Professor,Department of ECE, Pragati Engineering College, Surampalem. Kakinada district, AP.
Email: skgompa@yahoo.com

[4]Assistant Professor, Department of Computer Science and Engineering, Sathyabama Institute of Science and Technology, Chennai.
Email: sasirekharajeshkumar@gmail.com

[5] Assistant Professor, Department of CSE, Sri Venkateswara College Of Engineering, Tirupati, Andhra Pradesh
Email: venkataramana.r@svcolleges.edu.in

Abstract—Improvements and acceleration in software development have contributed towards high-quality services in all domains and all fields of industry, causing increasing demands for high-quality software developments. The industry is adopting human resources with high skills, advanced methodologies, and technologies to match the high-quality software development demands to accelerate the development life cycle. In the software development life cycle, one of the biggest challenges is the change management between the version of the source codes. Various reasons, such as changing the requirements or adapting available updates or technological upgrades, can cause the source code's version. The change management affects the correctness of the software service's release and the number of test cases. It is often observed that the development life cycle is delayed due to a lack of proper version control and due to repetitive testing iterations. Hence the demand for better version control-driven test case reduction methods cannot be ignored. The parallel research attempts propose several version control mechanisms. Nevertheless, most version controls are criticized for not contributing toward the test case generation of reduction. Henceforth, this work proposes a novel probabilistic rule-based test case reduction method to simplify the software development's testing and version control mechanism. Software developers highly adopt the refactoring process for making efficient changes such as code structure and functionality or applying changes in the requirements. This work demonstrates very high accuracy for change detection and management. This results in higher accuracy for test case reductions. The outcome of this work is to reduce the development time for the software to make the software development industry a better and more efficient world.

Keywords- Change Detection, Prerequisite Detection, Feature Detection, Functionality Detection, and Test Case Change

I. INTRODUCTION

The upgrades in the code improvement are an absolute necessity to be performed task for all product advancement cycles because of the nonstop changing customer prerequisites. The enhancements or adjustments in the product source code should be possible in different ways, for example, variant control or prerequisite following or utilizing outsider devices. Regardless, the refactoring technique is the most successive and profoundly received strategy proposed by M. Fowler et al. [1]. The impact of refactoring on the product source code is exceedingly good with the change of the boarding procedure and further with different periods of programming improvement life cycle. The remarkable result crafted by E. R. Murphy-Hill et al. [2] has recorded the standard periods of refactoring of source code, which profoundly impacts the adjustment of the procedure.

The analysis of the similar examination of other forming strategies with refactoring is performed by N. Tsantalis et al. [3], featuring the advantages of refactoring over different techniques. The difficulties of the refactoring process for any source code can't be overlooked. They can cause a higher multifaceted nature during forming if improper management occurs, as shown by M. Kim et al. [4].

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 4s
DOI: https://doi.org/10.17762/ijritcc.v11i4s.6568
***Article Received**: 30 January 2023 **Revised**: 05 March 2023 **Accepted**: 20 March 2023*

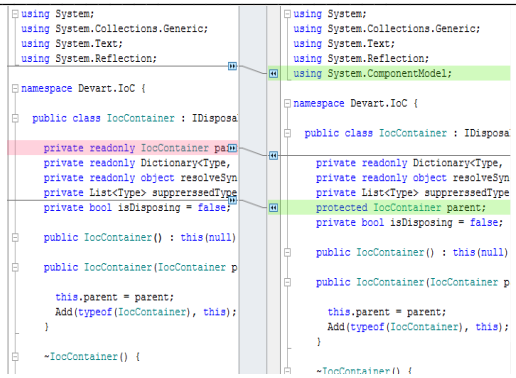

Fig. 1 Source Code Change Detection

Another examination centers around the product improvement act of spontaneity by Microsoft, proposing comparative measures as reported by Miryung Kim et al. [5]. Likewise, the comparative examination is led on another open-source apparatus, GitHub, by D. Silva et al. [6]. The outcome is the same as the past investigations prescribing comparable measures to be pursued for safe refactoring of the source code [Fig – 1]. Therefore, understanding that the refactoring [Fig – 2] of the source code can be highly helpful for source code changing, most of the development practices use this method. Nevertheless, refactoring the code can help make controlled changes to the code, but these changes result in further changes in the testing process and test case management. Hence, industry practitioners and researchers highly prioritize the demand for change detection and test case verification without repeating the test cases for the features which have not changed during the refactoring process. Thus, this work attempts to solve the change detection and test case reductions.

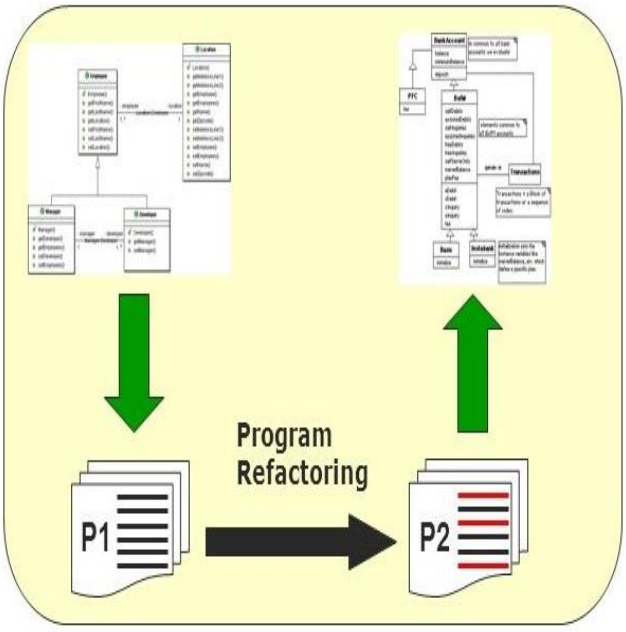

Fig. 2 Refactoring of Source Codes

The rest of the work is furnished as Section – II analyzes the outcomes from the parallel research. In Section – III, the problem definition and the scope for improvements are listed. In Section – IV, the proposed change detection algorithm is discussed. Section – V elaborates on the proposed test case detection and reduction algorithm. In Section – VI, the proposed complete automated framework is furnished. In Section – VII, the results are discussed. In Section VIII, the comparative analysis for understanding the improvements is discussed; in Section IX, this work presents the conclusion.

II. BACKGROUND AND FRAMEWORK

The source code's versioning is performed to include changes in the source code. Often the customer recommends the changes, or the changes are made due to the technical requirements fulfillment. Thus, refactoring results in changes in prerequisites or the feature of the source code or functionality of the source code. Hence, detecting the correct changes is an important prime task. The prime task is to detect the correct changes after a source code is refactored. Several similar types of research are taken place to accomplish this task. In this section of the work, the parallel research outcomes are analyzed.

The first case study by E. R. Murphy-Hill et al. [2] reported a framework that collects historical data from the source code version control and integrates the changes into the popular Eclipse IDE. The advancements of this work are done by S. Negara et al. [7], where metadata generated by version history is used. Nevertheless, this process completely depends on the refactoring trails or the auto-generated information during the refactoring process.

Removing the dependencies on the auto-generated information by the refactoring tools, J. Ratzinger et al. [8] propose a framework to generate commit messages during the refactoring process. This feature enables the framework to detect all changes, including minor updates. Regardless, this framework is expected to be deployed from the beginning of the code development life cycle, which makes this framework criticized among the practitioner's community. Other popular strategies supporting this method were also made. The work of Miryung Kim et al. [5] has finetuned the framework for detecting further detection of changes. Yet other popular methods for detecting the change are analyzing the pattern and behaviors of the source code, as demonstrated by G. Soares et al. [9], or analyzing the software code metrics, as represented by S. Demeyer et al. [10].

On the other hand, detecting refactoring using static code analysis is also a widely accepted method. The work by D. Dig et al. [11] on component-based detection of changes made the process of detection automated and specified. Also, the work by K. Prete et al. [12] proposed an alternative method for detecting source code changes using the templates. The major bottleneck

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 4s
DOI: https://doi.org/10.17762/ijritcc.v11i4s.6568
Article Received: 30 January 2023 Revised: 05 March 2023 Accepted: 20 March 2023

of this process is to separate the workable templates from the templates, which does not defer any functionality. M. Kim et al. [13] proposed a logical separation of the templates by querying the code's construction to improve this process. Furthermore, all the bottlenecks of the current works are summarized and analyzed by P. Weissgerber et al. [14]. This work takes up the recommendations and frames the generic scopes for improvements in the next section of the work

III. EASE OF USE

Furthermore, with a detailed understanding of the refactoring process outcomes by various research attempts and the strong connection with the change detection with test case management, in this section of the work, the research problems are identified. Based on the outcomes of the parallel research, the following shortcomings are identified:

- Firstly, general-purpose regression testing is carried out on a complete set of source code produced and modified from time to time in the software development life cycle. In most of the instances, it has been observed that the pre-configured test cases are deployed in the new version of the source code. Hence, the optimizations of the test cases are completely ignored.
- Secondly, during the manual generation of the test cases, the high-priority test cases are identified. Most of the parallel research depends on the pre-defined functional requirements given by the customer to decide the priority of the functional requirements. Based on this available information, the priority of the test cases is decided. It is natural to understand that the hidden and critical functionalities are often ignored, and the test cases validate these functionalities.
- Third, automation of the test case generation is a demanding area of research for regression testing. Nonetheless, the processes are far from perfection and complete acceptability.
- Finally, defining the priority test cases depends on various factors. None of the parallel research has demonstrated all possible combinations to evolve the optimization of test cases.

This work addresses the first problem mentioned in work. Henceforth, in the next section of the work, the proposed change detection algorithm is discussed

IV. PROPOSED CHANGE DETECTION

The changes made to the source code, using refactoring of the codes, must be identified to reduce the test cases or generate an outline of test cases. The proposed change detection algorithm is developed in total four parts.

Algorithm - 1: Source Code Pre-Processor (SCPP)
Step - 1. Access the repository for source code files
Step - 2. Mark the previous version of the file as V(n)
Step - 3. Mark the recent version of the file as V(n+1)
Step - 4. Identify the number of lines in the V(n) and V(n+1)
Step - 5. If V(n) >= V(n+1), then mark counter = V(n)
Step - 6. Else, mark counter = V(n+1)
Step - 7. For each line in counter a. Remove comments b. Apply tokenizer c. Check for variable change d. Check for statement change
Step - 8. Report the pre-processed V(n+1) with the changes

By using the above Source Code Pre-Processor(SCPP) algorithm, the input, output, and functionality achieved are
Input: a repository of source code files
Functionality: each line in the counter removes comments. applied tokenizer and checked for variable change and also checked for statement change
Output: report the pre-processed V(n+1) with the changes

Algorithm - 2: Prerequisite Requirement Change Detection (PRCD)
Step - 1. Load the files as V(n) and V(n+1)
Step - 2. Accept the tokenizer report
Step - 3. Build the list of "package" and "import" statements
Step - 4. For each line a. Detect the changes in the "package" and "import" statements
Step - 5. List the inclusion of Prerequisite statements
Step - 6. List the exclusion of Prerequisite statements

By using the above Prerequisite Requirement Change Detection(PRCD) algorithm, the input, output, and functionality achieved are

Input: the files V(n) and V(n+1) and the tokenizer report
Functionality: For each line, Detected the changes in the "package" and "import" statements
Output: inclusion of Prerequisite statements & exclusion of Prerequisite statements.

Algorithm - 3: Code Feature Change Detection (CFCD)
Step - 1. Load the files as V(n) and V(n+1)
Step - 2. Accept the tokenizer report
Step - 3. Build the list of variable identifiers
Step - 4. For each line a. Detect the changes in variable identifiers statements
Step - 5. List the inclusion of variable identifiers statements
Step - 6. List the exclusion of variable identifiers statements

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 4s
DOI: https://doi.org/10.17762/ijritcc.v11i4s.6568
Article Received: 30 January 2023 Revised: 05 March 2023 Accepted: 20 March 2023

By using the above Code Feature Change Detection (CFCD) algorithm, the input, outputs, and functionality achieved are

Input: the files V(n) and V(n+1) and the tokenizer report
Functionality: For each line detected the changes in variable identifiers
Output: inclusion of variable identifier statements & exclusion of variable identifier statements.

Algorithm - 4: Source Functionality Change Detection (SFCD)

Step - 1. Load the files as V(n) and V(n+1)
Step - 2. Accept the tokenizer report
Step - 3. Apply programming parser on the token
Step - 4. Build the list of identified parsed token
Step - 5. For each line
  a. Detect the changes in identified parsed token statements
Step - 6. List the inclusion of identified parsed token statements
Step - 7. List the exclusion of identified parsed token statements

By using the above Source Functionality Change Detection (SFCD)algorithm, the input, outputs, and functionality achieved are

Input: the files V(n) and V(n+1) and the tokenizer report
Functionality: For each line detected the changes in identified parsed token statements
Output: inclusion of identified parsed token statements & exclusion of identified parsed token statements

V. PROPOSED TEST CASE CHANGE RECOMMENDATIONS

Testing is one of the most important phases in the software development life cycle. With the recent developments in software, test case automation has grown popular. Due to the refactoring of the source codes, the test cases are often affected. These can cause the following situations:

- Inclusion of the new test cases
- Exclusion of the existing test cases, and
- Removal of the duplicated test cases

Thus, considering these factors, the proposed test case change recommendation algorithm is proposed in this work section.

Algorithm - 5: Test Case Change Recommendation (TCCR)

Step - 1. Accept the list of test cases
Step - 2. Identify the changes by the PRCD algorithm
Step - 3. For each change detected by PRCD
  a. If Prerequisite statements included
    i. Update test case recommendation as inclusion
  b. Else
    i. Update test case recommendation as exclusion
Step - 4. For each change detected by CFCD
  a. If variable identifiers statements included
    i. Update test case recommendation as inclusion
  b. Else
    i. Update test case recommendation as exclusion
Step - 5. For each change detected by SFCD
  a. If parsed token statements included
    i. Update test case recommendation as inclusion
  b. Else
    i. Update test case recommendation as exclusion
Step - 6. Update the final change case recommendations

By using the above Test Case Change Recommendation (TCCR)algorithm, the input, outputs, and functionality achieved are

Input: the list of test cases
Functionality: changes detected by PRCD, CFCD, SFCD
Output: Producing final change case recommendations

Furthermore, with the understanding of the proposed algorithms, in the next section of this work, the proposed automated framework is elaborated

VI. PROPOSED AUTOMATED FRAMEWORK

This work section elaborates on the proposed automated test case change recommendation framework. The proposed framework demonstrates how different components are collaborated and coupled to automate the process [Fig – 8].

The automated framework is designed to reduce the time needed for verifying and reducing or introducing test cases to the existing test case repositories. Firstly, the source code version files are accessed from where all source codes are stored, usually called the source code repository. The source code repository is maintained by the version control tools used by any organization. This proposed framework does not apply any constraints on the version control features. Rather only expects the versioning to be done only on separable source codes. After the source code files are loaded, the pre-processing algorithm is deployed on the source code to reduce the comments and tokenize the source code files. Once the tokenization is completed, the same source code files are pushed to the proposed PRCD, CFCD, and SFCD algorithms. The result from these algorithms is the identification of prerequisite changes, identification of feature or variable changes, and identification of functionality changes, respectively. Finally, the recommendation algorithm, TCCR, generates the final recommendations based on the existing test case repository.

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 4s
DOI: https://doi.org/10.17762/ijritcc.v11i4s.6568
***Article Received**: 30 January 2023 **Revised**: 05 March 2023 **Accepted**: 20 March 2023*

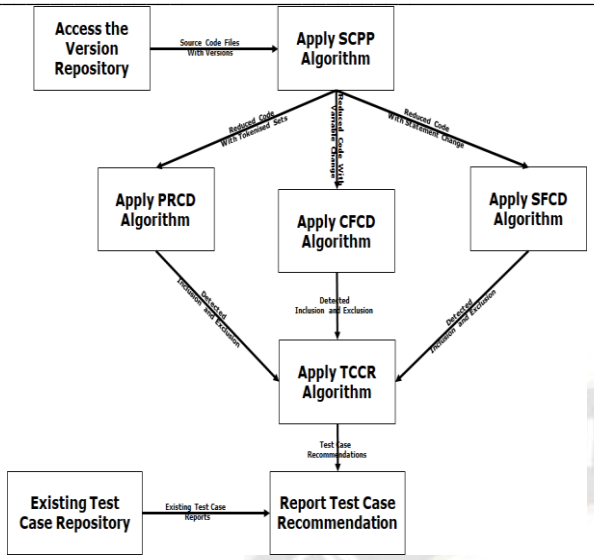

Fig. 3 Proposed Automated Test Case Change Recommendation Framework

Further, with a detailed understanding of the complete framework workflow, the results are discussed in the next section of the work.

VII. RESULTS AND DISCUSSION

The results obtained from the proposed automated framework are highly satisfactory and are discussed in this work section. Due to the highly integrated structure of the framework, the results are discussed under multiple separate factors such as Experimental Setup, Pre-processor Output, Change Detection Output, Prerequisite Test Case Availability, Recommendation Output, Variable Test Case Recommendation Output, and Functionality Test Case Recommendation Output.

A. Experimental Setup

Firstly, the experimental setup is discussed here. The primary component of the experiment relies on Java's "diff" utility. Diff Utils library is an Open Source library for playing out the correlation/diff activities between writings or some information: processing diffs, applying patches, creating bound together diffs or parsing them, producing diff yield for simple future showing (like one next to the other view) et cetera. The other details are discussed here [Table – 1].

TABLE I
EXPERIMENTAL SETUP

Artifacts	Description
Repository Source	GitHub
Total Number of Repositories	5
Version Control Tool Used (Can be integrated with any tool)	Git
Syntax Parser	Parse Tree
Number of Iterations for Detection in each repository	10

B. Pre-processor Output (SCPP Algorithm)

Secondly, the pre-processing outputs are listed here [Table - 2].

TABLE II
SCPP ALGORITHM

Source Code Repository Name	Number of Versions Present	Number of Versions Detected	Number of Lines Present	Number of Lines Detected
Repository - 1	2	2	335	335
Repository - 2	2	2	336	336
Repository - 3	2	2	283	283
Repository - 4	2	2	332	332
Repository - 5	2	2	344	344

The result is visualized graphically here [Fig – 9].

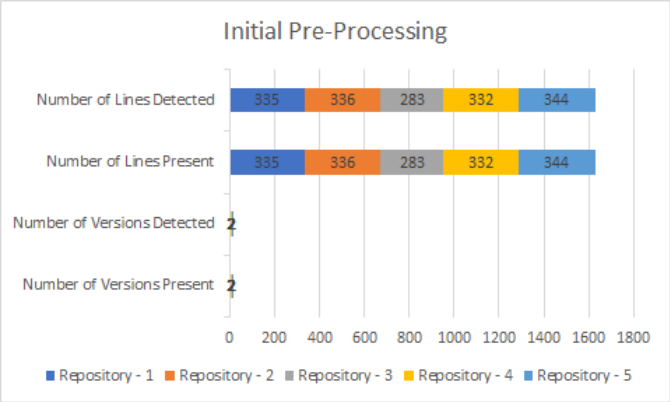

Fig. 4 Initial Pre-Processing Phase Results

Further, the tokenizer output is discussed [Table – 3].

TABLE III
TOKENIZER OUTPUT

Source Code Repository Name	Number of Prime Tokens Present	Number of Prime Tokens Identified
Repository - 1	17	15
Repository - 2	10	8
Repository - 3	16	14
Repository - 4	15	13
Repository - 5	13	12

The result is visualized graphically here [Fig – 10].

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 4s
DOI: https://doi.org/10.17762/ijritcc.v11i4s.6568
***Article Received**: 30 January 2023 **Revised**: 05 March 2023 **Accepted**: 20 March 2023*

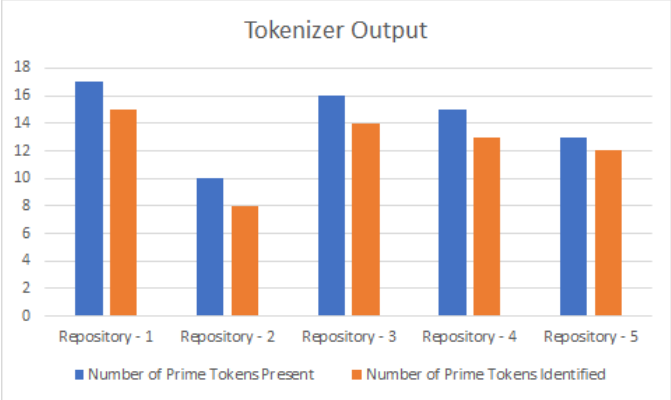

Fig. 5 Tokenizer Phase Results

Furthermore, the comment removal phase output is discussed [Table – 4].

TABLE IV
COMMENT REMOVAL OUTPUT

Source Code Repository Name	Number of Comment Lines Present	Number of Comment Lines with Functionality	Number of Comment Lines Detected
Repository - 1	3	3	3
Repository - 2	3	2	2
Repository - 3	3	0	0
Repository - 4	11	10	10
Repository - 5	10	8	8

The result is visualized graphically here [Fig – 11].

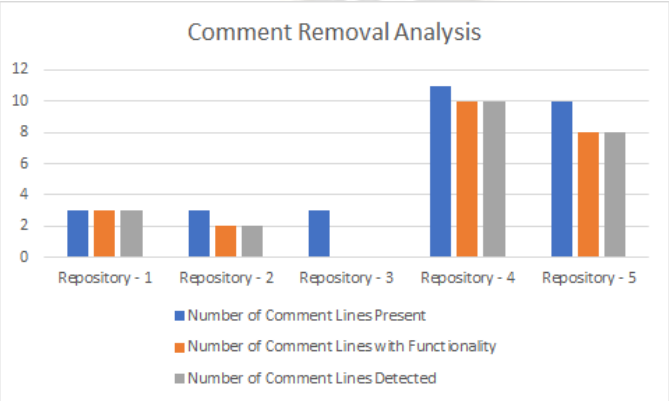

Fig. 6 Comment Line Removal Analysis

C. Change Detection Process Output

Thirdly, the change detection process outputs are listed here [Table - 5].

TABLE V
DETAILED REPORT FOR CHANGE DETECTION

Source Code Repository Name	Change Type	Change Position	Change Size
Repository - 1	Code Removed	34	0
Repository - 1	Code Removed	20	13
Repository - 1	Code Removed	5	0
Repository - 1	Code Removed	0	1
Repository - 1	Code Added	22	2
Repository - 1	Code Added	5	2
Repository - 1	Code Added	0	1
Repository - 2	Code Removed	139	0
Repository - 2	Code Removed	138	0
Repository - 2	Code Removed	134	3
Repository - 2	Code Removed	131	2
Repository - 2	Code Removed	118	12
Repository - 2	Code Removed	77	40
Repository - 2	Code Removed	76	0
Repository - 2	Code Removed	75	0
Repository - 2	Code Removed	71	3
Repository - 2	Code Removed	29	41
Repository - 2	Code Removed	26	2
Repository - 2	Code Removed	7	17
Repository - 2	Code Removed	0	6
Repository - 2	Code Added	164	1
Repository - 2	Code Added	159	4
Repository - 2	Code Added	157	1
Repository - 2	Code Added	136	20
Repository - 2	Code Added	117	18
Repository - 2	Code Added	114	2
Repository - 2	Code Added	111	2
Repository - 2	Code Added	88	22
Repository - 2	Code Added	28	59
Repository - 2	Code Added	19	8
Repository - 2	Code Added	2	15
Repository - 2	Code Added	0	1
Repository - 3	Code Removed	144	0
Repository - 3	Code Removed	143	0
Repository - 3	Code Removed	139	3
Repository - 3	Code Removed	136	2
Repository - 3	Code Removed	123	12
Repository - 3	Code Removed	82	40
Repository - 3	Code Removed	81	0
Repository - 3	Code Removed	80	0
Repository - 3	Code Removed	76	3
Repository - 3	Code Removed	34	41
Repository - 3	Code Removed	33	0
Repository - 3	Code Removed	0	32
Repository - 3	Code Added	164	1
Repository - 3	Code Added	159	4
Repository - 3	Code Added	157	1

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 4s
DOI: https://doi.org/10.17762/ijritcc.v11i4s.6568
***Article Received**: 30 January 2023 **Revised**: 05 March 2023 **Accepted**: 20 March 2023*

Repository - 3	Code Added	136	20
Repository - 3	Code Added	117	18
Repository - 3	Code Added	114	2
Repository - 3	Code Added	111	2
Repository - 3	Code Added	88	22
Repository - 3	Code Added	45	42
Repository - 3	Code Added	43	1
Repository - 3	Code Added	0	42
Repository - 4	Code Removed	166	25
Repository - 4	Code Removed	164	1
Repository - 4	Code Removed	159	4
Repository - 4	Code Removed	157	1
Repository - 4	Code Removed	136	20
Repository - 4	Code Removed	117	18
Repository - 4	Code Removed	114	2
Repository - 4	Code Removed	111	2
Repository - 4	Code Removed	88	22
Repository - 4	Code Removed	45	42
Repository - 4	Code Removed	43	1
Repository - 4	Code Removed	0	42
Repository - 4	Code Added	143	0
Repository - 4	Code Added	139	3
Repository - 4	Code Added	136	2
Repository - 4	Code Added	123	12
Repository - 4	Code Added	82	40
Repository - 4	Code Added	81	0
Repository - 4	Code Added	80	0
Repository - 4	Code Added	76	3
Repository - 4	Code Added	34	41
Repository - 4	Code Added	33	0
Repository - 4	Code Added	0	32
Repository - 5	Code Removed	25	4
Repository - 5	Code Removed	22	2
Repository - 5	Code Removed	5	2
Repository - 5	Code Removed	0	1
Repository - 5	Code Added	20	13
Repository - 5	Code Added	5	0
Repository - 5	Code Added	0	1

Further, the change detection summary is presented here [Table – 6].

TABLE VI
COMMENT REMOVAL OUTPUT

Source Code Repository Name	Actual Number of Changes	Number of Changes Detected	Change Detection Accuracy (%)
Repository - 1	8	7	87.50
Repository - 2	27	25	92.59
Repository - 3	23	23	100.00
Repository - 4	26	23	88.46
Repository - 5	9	7	77.78

The result is visualized graphically here [Fig – 12].

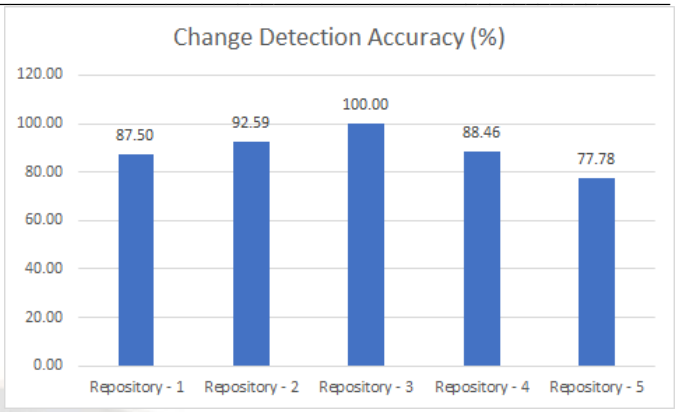

Fig. 7 Change Detection Accuracy Analysis

D. Prerequisite Requirement Change Detection Output

Fourthly, the Prerequisite Requirement Change Detection outputs are listed here [Table – 7].

TABLE VII
DETAILED REPORT FOR PREREQUISITE REQUIREMENT CHANGE DETECTION

Source Code Repository Name	Change Type	Prerequisite Details
Repository - 1	Added	import java.io.*;
Repository - 1	Added	java.util.LinkedList;
Repository - 1	Added	java.util.List;
Repository - 2	Removed	net. Content objects. Notify.JNotifyListener;
Repository - 2	Removed	java.io.*;
Repository - 2	Removed	java.text.SimpleDateFor mat;
Repository - 2	Removed	java.util.Calendar;
Repository - 2	Removed	java.util.LinkedList;
Repository - 2	Added	java.lang.reflect.Array;
Repository - 2	Removed	java. awt.Dimension;
Repository - 2	Removed	java. awt.Toolkit;
Repository - 2	Removed	java. swing.JTextArea;
Repository - 2	Removed	javax. swing.JPanel;
Repository - 2	Removed	javax.swing.JFrame;
Repository - 2	Removed	javax.swing.JScrollPane;
Repository - 2	Added	difflib.ChangeDelta;
Repository - 2	Added	difflib.Chunk;
Repository - 2	Added	difflib.DeleteDelta;
Repository - 2	Added	difflib.Delta;
Repository - 2	Added	difflib.DiffAlgorithm;
Repository - 2	Added	difflib.InsertDelta;
Repository - 2	Added	difflib.Patch;
Repository - 3	Removed	net.contentobjects.jnotify .JNotifyListener;
Repository - 3	Removed	java.io.*;
Repository - 3	Removed	java.text.SimpleDateFor mat;
Repository - 3	Removed	java.util.Calendar;
Repository - 3	Removed	java.awt.Dimension;
Repository - 3	Removed	java.awt.Toolkit;

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 4s
DOI: https://doi.org/10.17762/ijritcc.v11i4s.6568
***Article Received**: 30 January 2023 **Revised**: 05 March 2023 **Accepted**: 20 March 2023*

Repository - 3	Removed	javax.swing.JTextArea;
Repository - 3	Removed	javax.swing.JPanel;
Repository - 3	Removed	javax.swing.JFrame;
Repository - 3	Removed	javax.swing.JScrollPane;
Repository - 3	Added	java.lang.reflect.Array;
Repository - 3	Added	java.util.List;
Repository - 3	Added	difflib.ChangeDelta;
Repository - 3	Added	difflib.Chunk;
Repository - 3	Added	difflib.DeleteDelta;
Repository - 3	Added	difflib.Delta;
Repository - 3	Added	difflib.DiffAlgorithm;
Repository - 3	Added	difflib.InsertDelta;
Repository - 3	Added	difflib.Patch;
Repository - 4	Removed	java.util.List;
Repository - 4	Removed	difflib.ChangeDelta;
Repository - 4	Removed	difflib.Chunk;
Repository - 4	Removed	difflib.DeleteDelta;
Repository - 4	Removed	difflib.Delta;
Repository - 4	Removed	difflib.DiffAlgorithm;
Repository - 4	Removed	difflib.InsertDelta;
Repository - 4	Removed	difflib.Patch;
Repository - 4	Added	net.contentobjects.jnotify .JNotify;
Repository - 4	Added	net.contentobjects.jnotify .JNotifyListener;
Repository - 4	Added	java.io.*;
Repository - 4	Added	java.text.SimpleDateFor mat;
Repository - 4	Added	java.util.Calendar;
Repository - 4	Added	java.awt.Dimension;
Repository - 4	Added	java.awt.Toolkit;
Repository - 4	Added	javax.swing.JTextArea;
Repository - 4	Added	javax.swing.JPanel;
Repository - 4	Added	javax.swing.JFrame;
Repository - 4	Added	javax.swing.JScrollPane;
Repository - 5	Added	net.contentobjects.jnotify .JNotify;
Repository - 5	Removed	java.util.LinkedList;
Repository - 5	Removed	java.util.List;

Further, the *Prerequisite Requirement Change Detection* summary is presented here [Table – 8].

TABLE VIII
PREREQUISITE REQUIREMENT CHANGE DETECTION SUMMARY

Source Code Repository Name	Number of Prerequisites Added	Number of Prerequisites Removed
Repository - 1	3	0
Repository - 2	8	11
Repository - 3	9	10
Repository - 4	11	8
Repository - 5	1	2

The result is visualized graphically here [Fig – 13].

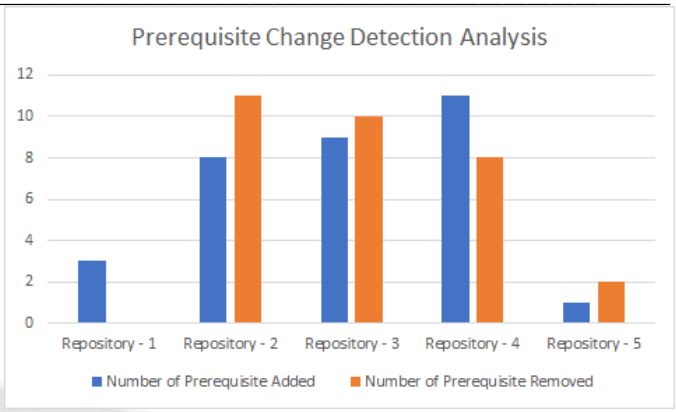

Fig. 8 Prerequisite Change Detection Analysis

E. Code Feature Change Detection Output

Fifthly, the Code Feature Change Detection outputs are listed here [Table – 9].

TABLE IX
DETAILED REPORT FOR CODE FEATURE CHANGE DETECTION

Source Code Repository Name	Change Type	Feature Details
Repository - 1	Remove	watchSubtree
Repository - 1	Remove	watchID
Repository - 1	Remove	res
Repository - 2	Added	N
Repository - 2	Added	M
Repository - 2	Added	MAX
Repository - 2	Added	size
Repository - 2	Added	middle
Repository - 2	Added	kmiddle
Repository - 2	Added	kplus
Repository - 2	Added	kminus
Repository - 2	Added	j
Repository - 2	Added	i
Repository - 2	Added	j
Repository - 2	Added	ianchor
Repository - 2	Added	janchor
Repository - 2	Added	static
Repository - 2	Added	newLength
Repository - 3	Remove	watchSubtree
Repository - 3	Remove	watchID
Repository - 3	Remove	res
Repository - 3	Added	N
Repository - 3	Added	M
Repository - 3	Added	MAX
Repository - 3	Added	size
Repository - 3	Added	middle
Repository - 3	Added	kmiddle
Repository - 3	Added	kplus
Repository - 3	Added	kminus
Repository - 3	Added	j
Repository - 3	Added	i
Repository - 3	Added	j

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 4s
DOI: https://doi.org/10.17762/ijritcc.v11i4s.6568
***Article Received**: 30 January 2023 **Revised**: 05 March 2023 **Accepted**: 20 March 2023*

Repository - 3	Added	ianchor
Repository - 3	Added	janchor
Repository - 3	Added	static
Repository - 3	Added	newLength
Repository - 4	Added	watchSubtree
Repository - 4	Added	watchID
Repository - 4	Added	res
Repository - 4	Remove	N
Repository - 4	Remove	M
Repository - 4	Remove	MAX
Repository - 4	Remove	size
Repository - 4	Remove	middle
Repository - 4	Remove	kmiddle
Repository - 4	Remove	kplus
Repository - 4	Remove	kminus
Repository - 4	Remove	j
Repository - 4	Remove	i
Repository - 4	Remove	j
Repository - 4	Remove	ianchor
Repository - 4	Remove	janchor
Repository - 4	Remove	newLength
Repository - 5	Added	watchSubtree
Repository - 5	Added	watchID
Repository - 5	Added	res

Further, the Code Feature Change Detection *summary is presented here [Table – 10].*

TABLE X
CODE FEATURE CHANGE DETECTION SUMMARY

Source Code Repository Name	Number of Features Added	Number of Features Removed
Repository - 1	0	3
Repository - 2	15	0
Repository - 3	15	3
Repository - 4	3	14
Repository - 5	3	0

The result is visualized graphically here [Fig – 14].

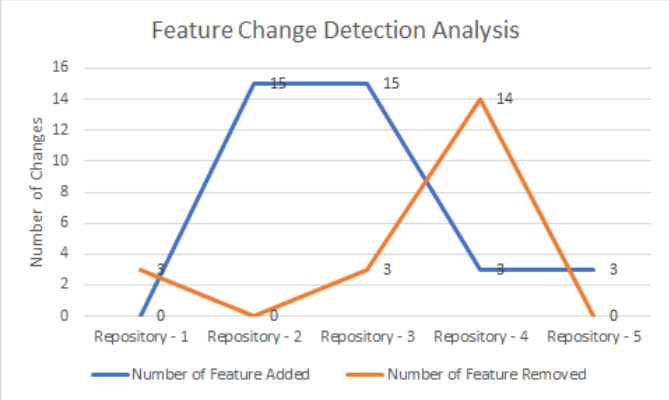

Fig. 9 Code Feature Change Detection Analysis

F. Source Functionality Change Detection Output

The Source Functionality Change Detection *summary is presented here [Table – 11].*

TABLE XI
SOURCE FUNCTIONALITY CHANGE DETECTION SUMMARY

Source Code Repository Name	Number of Functionality Added	Number of Functionality Removed
Repository - 1	7	8
Repository - 2	5	8
Repository - 3	8	8
Repository - 4	5	9
Repository - 5	5	6

The result is visualized graphically here [Fig – 15].

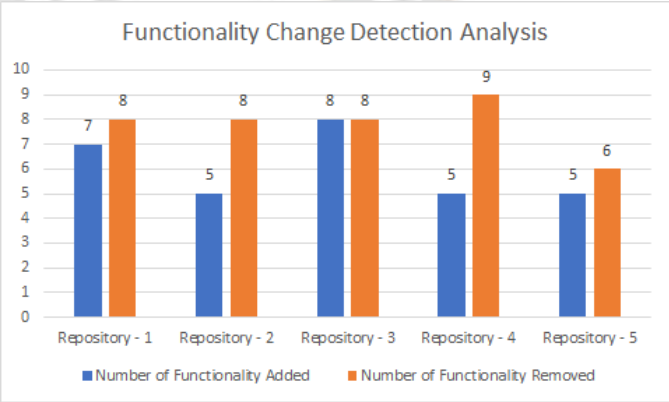

Fig. 10 Source Functionality Change Detection Analysis

G. Test Case Change Recommendation Output

Finally, *the* Test Case Change Recommendation outputs *are presented here [Table – 12] and [Table – 13].*

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 4s
DOI: https://doi.org/10.17762/ijritcc.v11i4s.6568
Article Received: 30 January 2023 Revised: 05 March 2023 Accepted: 20 March 2023

TABLE XII
SOURCE FUNCTIONALITY CHANGE DETECTION SUMMERY – INCLUSIONS

Source Code Repository Name	Prerequisite Added	Feature Added	Functionality Added	Recommendations
Repository - 1	3	1	3	Prerequisite TC Update:3 Feature TC Update:1 Functionality TC Update:3
Repository - 2	8	16	78	Prerequisite TC Update:8 Feature TC Update:16 Functionality TC Update:78
Repository - 3	9	16	78	Prerequisite TC Update:9 Feature TC Update:16 Functionality TC Update:78
Repository - 4	11	3	92	Prerequisite TC Update:11 Feature TC Update:3 Functionality TC Update:92
Repository - 5	1	3	6	Prerequisite TC Update:1 Feature TC Update:3 Functionality TC Update:6

TABLE XIII
SOURCE FUNCTIONALITY CHANGE DETECTION SUMMERY – EXCLUSIONS

Source Code Repository Name	Prerequisite Removed	Feature Removed	Functionality Removed	Recommendations
Repository - 1	0	3	4	Prerequisite TC Update:0 Feature TC Update:3 Functionality TC Update:4
Repository - 2	11	1	58	Prerequisite TC Update:11 Feature TC Update:1 Functionality TC Update:58
Repository - 3	10	3	61	Prerequisite TC Update:10 Feature TC Update:3 Functionality TC Update:61
Repository - 4	8	15	47	Prerequisite TC Update:8 Feature TC Update:15 Functionality TC Update:47
Repository - 5	2	1	1	Prerequisite TC Update:2 Feature TC Update:1 Functionality TC Update:1

Henceforth, with the complete discussions of results, in the next section, this work carries outs the comparative analysis in the next section

VIII. COMPARATIVE ANALYSIS

The improvements over the existing studies are the primary objective of every research. To justify the claim of improvements, it is necessary to carry out a comparative analysis. Hence in this section of the work, the comparative analysis with the popular existing works is performed on the framed metric for comparison [Table – 14].

TABLE XIV
COMPARATIVE ANALYSIS

System Details	Change Detection Capabilities	Prerequisite Detection Capabilities	Feature Detection Capabilities	Functionality Detection Capabilities	Test Case Change Recommendation
M. Fowler et al. [1] 2018	Yes	No	Yes	No	No
Miryung Kim et al. [5] 2016	Yes	No	No	Yes	No
D. Silva et al. [6] 2016	Yes	No	No	Yes	No
M. Kim et al. [13] 2014	Yes	No	Yes	No	No
Proposed Automated Framework 2018	Yes	Yes	Yes	Yes	Yes

It is natural to understand that with the significant improvements and incorporation of Change Detection Capabilities, Prerequisite Detection Capabilities, Feature Detection Capabilities, Functionality Detection Capabilities, and Test Case Change Recommendations, the proposed automated framework have outperformed the other parallel research outcomes.

IX. CONCLUSION

The software development industry completely relies on accurate change management. Any organization's change-driven structure or process puts it ahead of the competition among the other providers. Accommodating the client requests in terms of changes can be highly cost and time ineffective as the changes in the source code can affect the other phases of the

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 4s
DOI: https://doi.org/10.17762/ijritcc.v11i4s.6568
***Article Received**: 30 January 2023 **Revised**: 05 March 2023 **Accepted**: 20 March 2023*

life cycle, specifically the testing. Due to any modification to the source code, the testing operations must also change. The challenge is to identify the current change and reduce the repetition of the testing tasks. Thus, this work provides an automatic framework with Change Detection Capabilities, Prerequisite Detection Capabilities, Feature Detection Capabilities, Functionality Detection Capabilities, and Test Case Change Recommendations for better test case management. This work's major and most unique outcome is to identify and recommend any changes in the test cases to make the software development world faster and economically affordable.

REFERENCES

[1] M. Fowler, Refactoring: Improving the Design of Existing Code. Addison-Wesley, 2018.

[2] E. R. Murphy-Hill, C. Parnin, and A. P. Black, "How we refactor, and how we know it," IEEE Transactions on Software Engineering, vol. 38, no. 1, pp. 5–18, 2012.

[3] N. Tsantalis, V. Guana, E. Stroulia, and A. Hindle, "A multidimensional empirical study on refactoring activity," in Conference of the Centre for Advanced Studies on Collaborative Research (CASCON), 2013, pp. 132–146.

[4] M. Kim, T. Zimmermann, and N. Nagappan, "A field study of refactoring challenges and benefits," in 20th Symposium on the Foundations of Software Engineering (FSE), 2017, pp. 50:1–50:11.

[5] Miryung Kim et al., "An empirical study of refactoring challenges and benefits at Microsoft," IEEE Transactions on Software Engineering, vol. 40, no. 7, July 2016.

[6] D. Silva, N. Tsantalis, and M. T. Valente, "Why we refactor? confessions of GitHub contributors," in 24th Symposium on the Foundations of Software Engineering (FSE), 2016, pp. 858–870.

[7] S. Negara, N. Chen, M. Vakilian, R. E. Johnson, and D. Dig, "A comparative study of manual and automated refactorings," in 27th European Conference on Object-Oriented Programming (ECOOP), 2016, pp. 552–576.

[8] J. Ratzinger, T. Sigmund, and H. C. Gall, "On the relation of refactorings and software defect prediction," in 5th Working Conference on Mining Software Repositories (MSR), 2012, pp. 35–38.

[9] G. Soares, R. Gheyi, D. Serey, and T. Massoni, "Making program refactoring safer," IEEE software, vol. 27, no. 4, pp. 52–57, 2010.

[10] S. Demeyer, S. Ducasse, and O. Nierstrasz, "Finding refactorings via change metrics," in ACM SIGPLAN Notices, vol. 35, no. 10, 2010, pp. 166–177.

[11] D. Dig, C. Comertoglu, D. Marinov, and R. Johnson, "Automated detection of refactorings in evolving components," in 20th European Conference on Object-Oriented Programming (ECOOP), 2006, pp. 404– 428.

[12] K. Prete, N. Rachatasumrit, N. Sudan, and M. Kim, "Template-based reconstruction of complex refactorings," in 26th International Conference on Software Maintenance (ICSM), 2010, pp. 1–10.

[13] M. Kim, M. Gee, A. Loh, and N. Rachatasumrit, "Ref-Finder: A refactoring reconstruction tool based on logic query templates," in 8th Symposium on Foundations of Software Engineering (FSE), 2014, pp. 371–372

[14] P. Weissgerber and S. Diehl, "Identifying refactorings from sourcecode changes," in 21st International Conference on Automated Software Engineering (ASE), 2016, pp. 231–240.

[15] Sudhakara, M., Bhavya, K. R., Kumar, M. R., Badrinath, N., & Rangaswamy, K. (2023). Customer Purchase Prediction and Potential Customer Identification for Digital Marketing Using Machine Learning. In *AI-Driven Intelligent Models for Business Excellence* (pp. 95-111). IGI Global.

[16] Suneel, C. V., Prasanna, K., & Kumar, M. R. (2017). Frequent data partitioning using parallel mining item sets and MapReduce. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, *2*(4).

[17] Rudra Kumar, M., Pathak, R., & Gunjan, V. K. (2022). Diagnosis and Medicine Prediction for COVID-19 Using Machine Learning Approach. In *Computational Intelligence in Machine Learning: Select Proceedings of ICCIML 2021* (pp. 123-133). Singapore: Springer Nature Singapore.

[18] Kalyani, B. J. D., & Rao, K. R. H. (2018, April). Assessment of physical server reliability in a multi-cloud computing system. In *AIP Conference Proceedings* (Vol. 1952, No. 1, p. 020045). AIP Publishing LLC.

[19] Kalyani, B. J. D., Meena, K., Murali, E., Jayakumar, L., & Saravanan, D. (2023). Analysis of MRI brain tumor images using deep learning techniques. *Soft Computing*, 1-8.

[20] Sivaji, U., & Rao, P. S. (2021). WITHDRAWN: Test case minimization for regression testing by analyzing software performance using the novel method.

[21] Sivaji, U., Rao, N. K., Srivani, C., Sree, T., & Singh, M. (2021). A Hybrid Random Forest Linear Model approach to predict heart disease. *Annals of the Romanian Society for Cell Biology*, *25*(6), 7810-7814.