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

Niladri Shekar Dey[1], Purnachand Kollapudi[2], M V Narayana[3], I Govardhana Rao[4]
Associate Professor, Department of CSE, B V Raju Institute of Technology, Narsapur, Medak Dist.Telangana State, India[1, 2]
Professor, Department of CSE, Guru Nanak Institutions Technical Campus, Hyderabad, India[3]
Assistant Professor, Department of CSE, Osmania University, Hyderabad, India[4]

***Abstract*—Improvements and acceleration in software development has contributed towards high quality services in all domains and all fields of industry causing increasing demands for high quality software developments. In order to match with the high-quality software development demands, the software development industry is adopting human resources with high skills, advanced methodologies and technologies for accelerating the development life cycle. In the software development life cycle, one of the biggest challenges is the change management between versions of the source codes. The versing of the source code can be caused by various reasons such as change in the requirements or adaptation of functional update or technological upgradations. The change management does not only affect the correctness of the release for the software service, rather also impact the number of test cases. It is often observed that, the development life cycle is delayed due to lack of proper version control and due to the improver version control, the repetitive testing iterations. Hence the demand for better version control driven test case reduction methods cannot be ignored. A number of version control mechanisms are proposed by the parallel research attempts. Nevertheless, most of the version controls are criticized for not contributing towards the test case generation of reduction. Henceforth, this work proposes a novel probabilistic refactoring detection and rule-based test case reduction method in order to simplify the testing and version control mechanism for the software development. The refactoring process is highly adopted by the software developers for making efficient changes such as code structure, functionality or apply change in the requirements. This work demonstrates a very high accuracy for change detection and management. This results into a higher accuracy for test case reductions. The final outcome of this work is to reduce the development time for the software for making the software development industry a better and efficient world.**

***Keywords*—Change detection; pre-requisite detection; feature detection; functionality detection and test case change recommendation***

## I. INTRODUCTION

The improvements in the code development is a must to be performed task for all software development cycles, due to the continuous changing client requirements. The improvements or the changes in the software source code can be done in various ways such as version control or requirement tracking or using third party tools. Nonetheless, the most frequent and highly adopted method is the refactoring method as suggested by M. Fowler et al. [1]. The effect of refactoring on the software source code is highly compatible with the change management process and further with the other phases of software development life cycle. The notable outcome by the work of E. R. Murphy-Hill et al. [2] have listed the standard phases of refactoring of source code, which deeply influences the adaptation of the process. The detailed comparative analysis of other versioning methods with refactoring is performed by N. Tsantalis et al. [3] highlighting the benefits of refactoring over other methods. The challenges of refactoring process for any source code cannot be ignored and can cause higher complexity during versioning in case of improper management as demonstrated by M. Kim et al. [4]. Another study focuses on the software development improvisation by Microsoft, suggesting similar measures as documented by Miryung Kim et al. [5]. Also, the similar study is conducted on another open source tool, GitHub, by D. Silva et al. [6] and the result is same as the previous studies recommending similar measures to be followed 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 uses this method.

Nevertheless, the process of refactoring the code can be helpful for making controlled changes into the code, but these changes results into further changes of testing process and test case management. Hence, the demand for change detection and test case verification without repeating the test cases for the features, which has not changed during the refactoring process, is highly prioritized by the industry practitioners and researchers. Thus, this work attempts to provide a solution to the change detection and test case reductions.

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

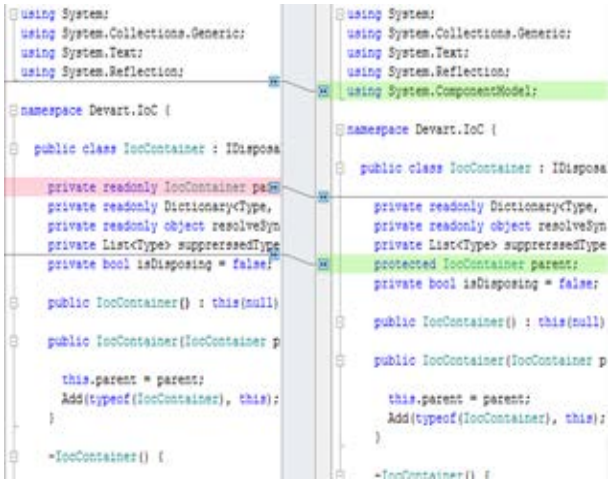

Fig. 1. Source Code Change Detection.

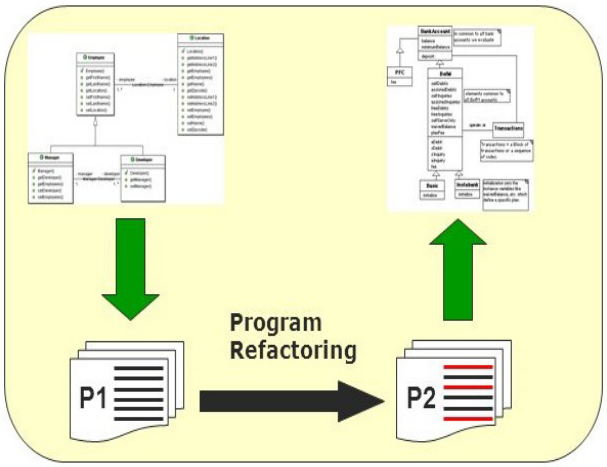


Fig. 2. Refactoring of Source Codes.

## II. Parallel Research Outcomes

The versioning of the source code is performed in order to include changes in the source code. Often the changes are recommended by the customer or the changes are made due to the technical requirements fulfilments. Thus, refactor results into changes in pre-requisites or the feature of the source code or functionality of the source code. Hence, detecting the correct changes are the prime important task.

In order to detect the correct changes after a source code is refactored is the prime task. A number of parallel researches are taken place to accomplish this task. In this section of the work, the parallel research outcomes are analysed.

The first case study produced by E. R. Murphy-Hill et al. [2] have reported a framework that collects the historical data from the source code version control and integrates the changes into popular Eclipse IDE. The advancements of this work are done by S. Negara et al. [7], where the process of using meta data generated by version history is used. Nevertheless, this process is completely dependent 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, the work of J. Ratzinger et al. [8] proposes a framework to generate commit messages during the refactoring process. This feature enables the framework to detect all changes including the minor updates. Regardless to mention, this framework is expected to be deployed from the beginning of the code development life cycles, which makes this framework being criticized among the practitioner's community. The other popular strategies supporting this method were also made. The work of Miryung Kim et al. [5] have finetuned the framework for detecting further detection of changes.

Yet other popular methods for detecting the change are analysing the pattern and behaviours of the source code as demonstrated by G. Soares et al. [9] or analysing the software code metrics as represented by S. Demeyer et al. [10].

In the other hand, detecting refactoring using the static code analysis is also 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] have proposed an alternative method for detecting the source code changes using the templates. The major bottleneck of this process is to separate the workable templates from the templates, which does not defer any functionality. In order to improve this process, M. Kim et al. [13] proposed a logical separation of the templates using querying the construction of the code.

Furthermore, all the bottlenecks of the existing works are summarized and analysed 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. Identification of Scope for Improvements

Furthermore, with the 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 researches, the following short comings are identified:

- Firstly, the general-purpose regression testing is carried out on a complete set of source code which is produced and modified time to time in the software development life cycle. Most of the instances it is been observed that the pre-configured test cases are deployed in the new version of the source code. Regardless to mention that most of the test cases are configured to test the areas where no changes are made. Hence, the optimizations of the test cases are completely ignored.
- Secondly, during the manual generation of the test cases, the identification of the high priority test cases is carried out. Most of the parallel researches depends on the pre-defined functional requirements given by the customer to decide the priority of the functional requirements and based on this available information, the priority of the test cases is decided. It is natural to understand that, due to this often the hidden and critical functionalities are ignored and as well as the test cases to validate these functionalities.

- Third, automation of the test case generation is 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 researches have demonstrated all possible combinations to evolve the optimization of test cases.

This work addresses the first problem mentioned in the work.

Henceforth, in the next section of the work, the proposed change detection algorithm is discussed.

## IV. Proposed Change Detection

The changes made into the source code using refactoring of the codes, must be identified for reducing the test cases or generating 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*

The algorithm is visualized graphically here in Fig. 3.

**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 "package" and "import" statements*

*Step - 5. List the inclusion of Prerequisite statements*

*Step - 6. List the exclusion of Prerequisite statements*

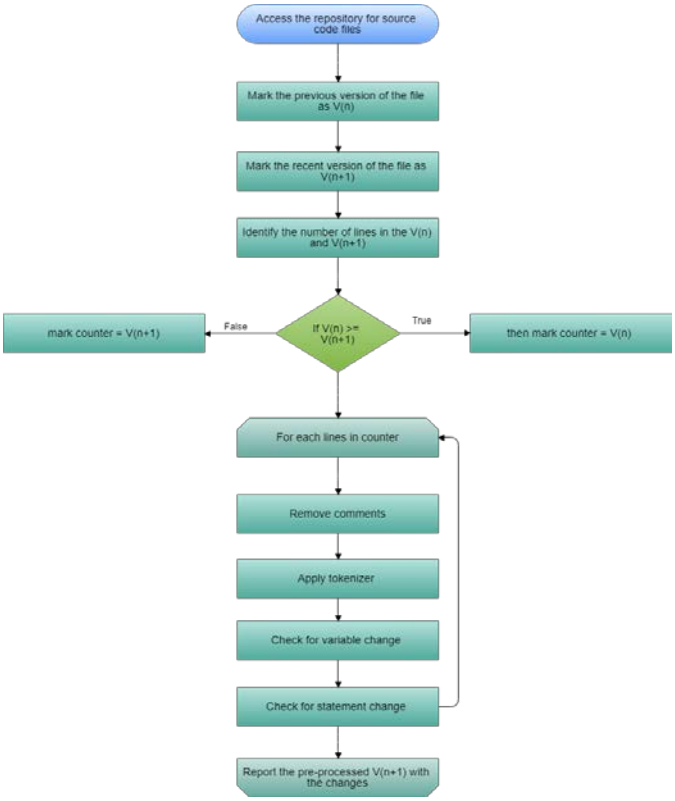


Fig. 3. Process flow of SCPP Algorithm.

The algorithm is visualized graphically here in Fig. 4.

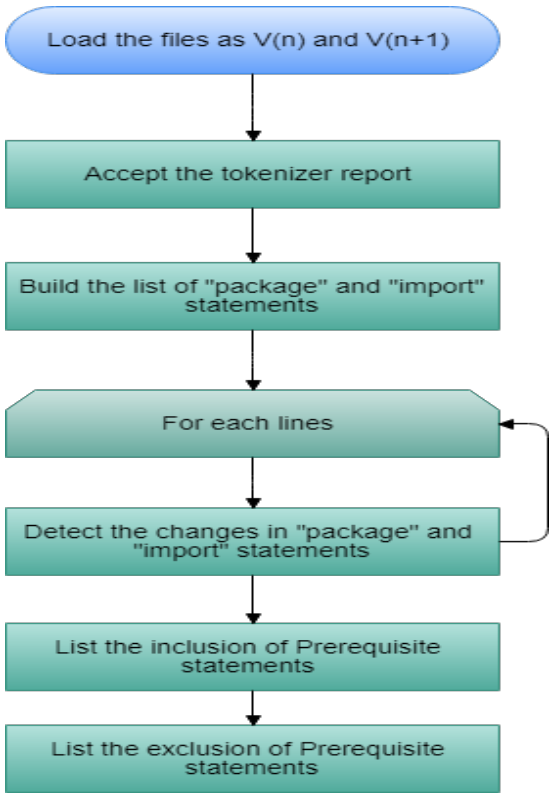


Fig. 4. Process flow of PRCD Algorithm.

**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*

The algorithm is visualized graphically here in Fig. 5.

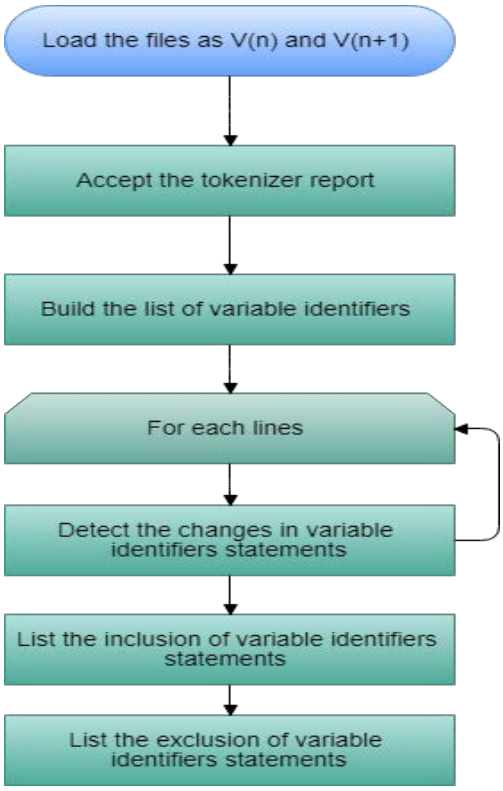


Fig. 5. Process flow of CFCD Algorithm.

**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*

The algorithm is visualized graphically here in Fig. 6.

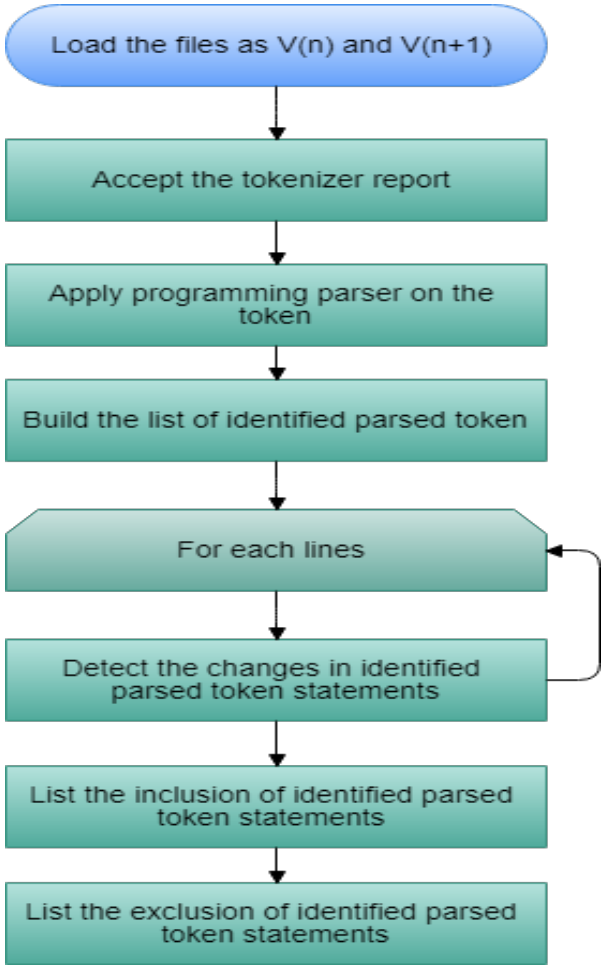


Fig. 6. Process flow of SFCD Algorithm.

Henceforth, with the detailed understanding of the proposed change recommendation algorithm, this work furnishes the test case change identification method in the next section.

## V. Proposed Test Case Change Recommendation

The testing is one of the most important phases in the software development life cycle. With the recent developments in software, the automation in the test cases have grown popularity. Due to the refactoring of the source codes, often the test cases are also 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, in this section of the work, the proposed test case change recommendation algorithm is proposed.

The algorithm is visualized graphically here in Fig. 7.

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

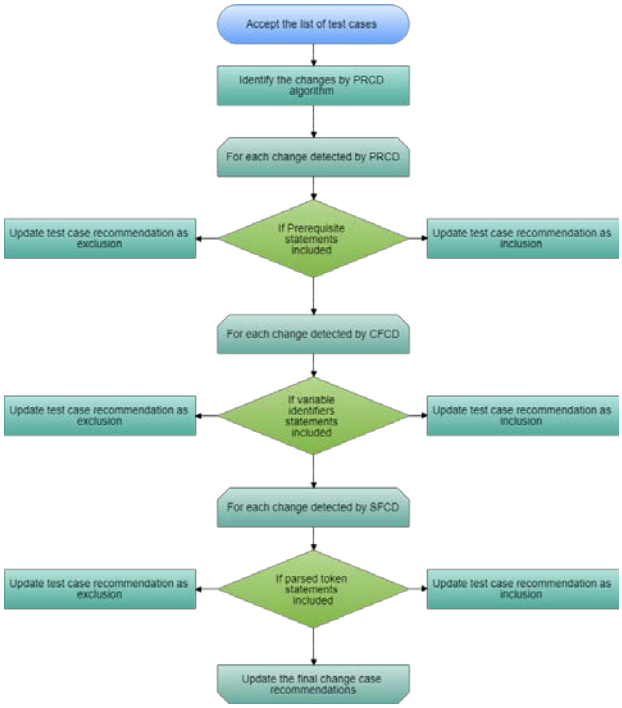


Fig. 7. Process flow of TCCR Algorithm.

**Algorithm - 5**: Test Case Change Recommendation (TCCR)

*Step - 1. Accept the list of test cases*
*Step - 2. Identify the changes by 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*

## VI. Proposed Automated Framework

In this section of the work, the proposed automated test case change recommendation framework is elaborated. The proposed framework demonstrates how different components are collaborated and coupled together for making the complete process automated (Fig. 8).

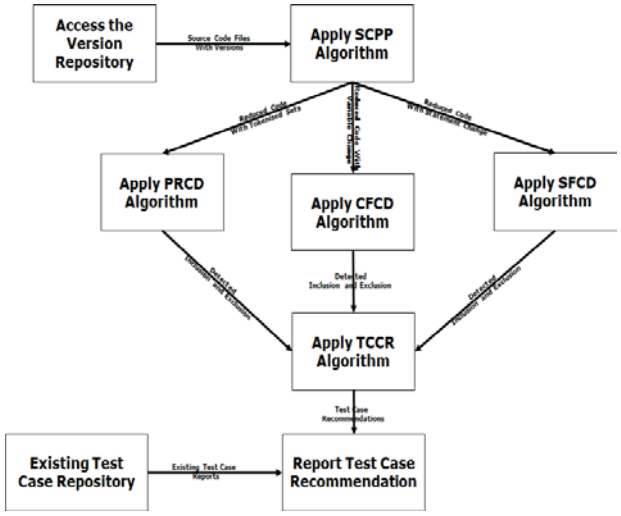


Fig. 8. Proposed Automated Test Case Change Recommendation Framework.

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 access from the location 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 to tokenize the source code files. Once the tokenization is completed, the same source code files are pushed to the proposed PRCD, proposed CFCD and proposed SFCD algorithms. The result from these algorithms are identification of pre-requisite 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.

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

## VII. Results and Discussion

The results obtained from the proposed automated framework is highly satisfactory and are discussed in this section of the work. Due to the highly integrated structure of the framework, the results are discussed under multiple separate factors as Experimental Setup, Pre-processor Output, Change Detection Output, Pre-Requisite 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 the Java's

"diff" utility. Diff Utilities library is an Open Source library for playing out the correlation/diff activities between writings or some sort of 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 in Table I.

TABLE I. EXPERIMENTAL SETUP

| Artefacts | Description |
|---|---|
| Repository Source | GitHub |
| Total Number of Repositories | 5 |
| Version Control Tool Used (Can be integrated with any tool) | Git |
| Syntax Perser | Parse Tree |
| Number of Iteration for Detection in each repository | 10 |

### B. Pre-Processor Output (SCPP Algorithm)

Secondly, the pre-processing outputs are listed here in Table II.

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 in Fig. 9.

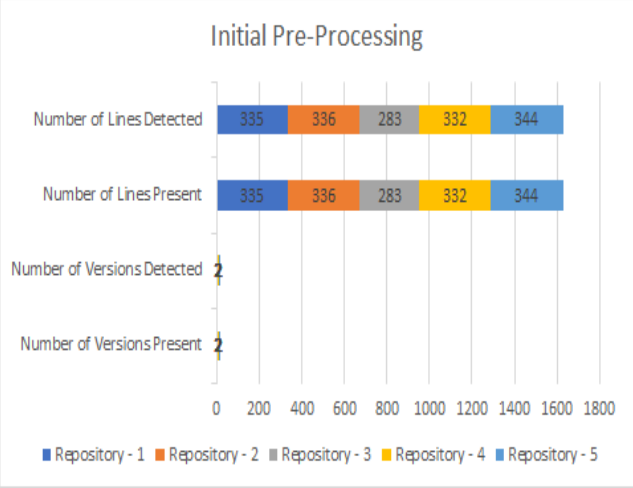


Fig. 9. Initial Pre-Processing Phase Results.

Further, the tokenizer output is discussed in Table III.

The result is visualized graphically here in Fig. 10.

Furthermore, the comment removal phase output is discussed in Table IV.

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 |

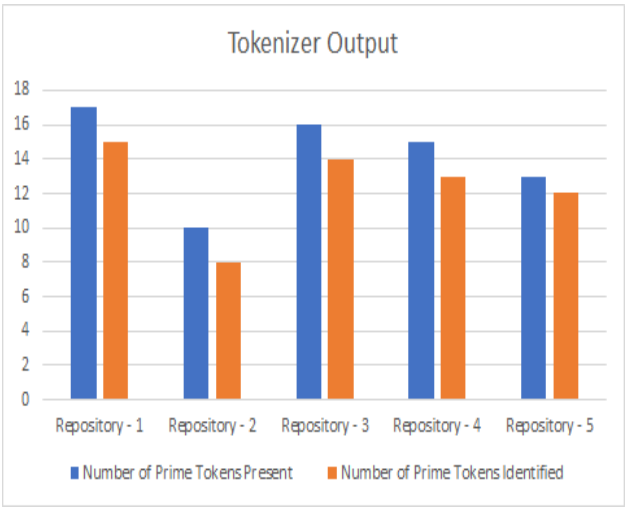


Fig. 10. Tokenizer Phase Results.

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 in Fig. 11.

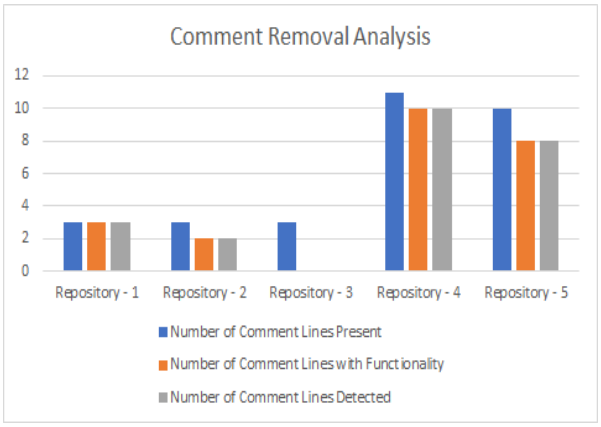


Fig. 11. Comment Line Removal Analysis.

### C. Change Detection Process Output

Thirdly, the change detection process outputs are listed here in Table V.

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 |
| 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 in Table VI.

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 in Fig. 12.

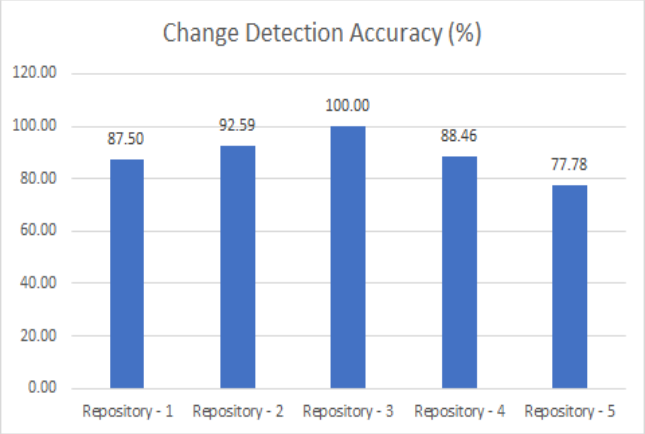


Fig. 12. Change Detection Accuracy Analysis.

### D. Prerequisite Requirement Change Detection Output

Fourthly, the Prerequisite Requirement Change Detection outputs are listed here in Table VII.

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.contentobjects.jnotify. JNotifyListener; |
| Repository - 2 | Removed | java.io.*; |
| Repository - 2 | Removed | java.text.SimpleDateForma; |
| 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 | javax.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.SimpleDateForma; |
| Repository - 3 | Removed | java.util.Calendar; |
| Repository - 3 | Removed | java.awt.Dimension; |
| Repository - 3 | Removed | java.awt.Toolkit; |
| 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.SimpleDateFormat; |
| 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 in Table VIII.

TABLE VIII. PREREQUISITE REQUIREMENT CHANGE DETECTION SUMMERY

| Source Code Repository Name | Number of Prerequisite Added | Number of Prerequisite 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 in Fig. 13.

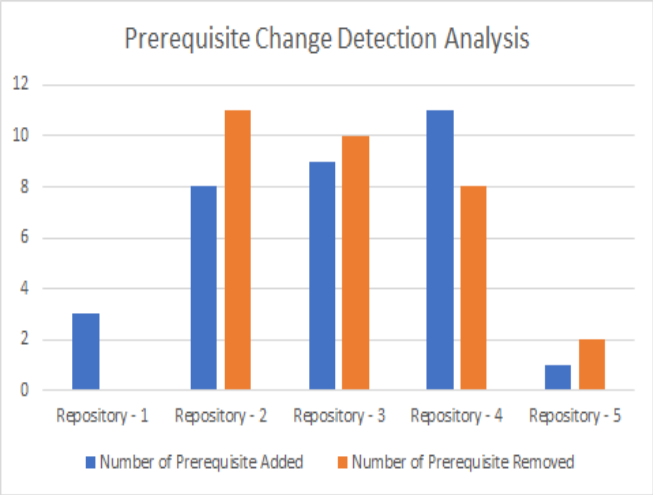


Fig. 13. Pre-Requisite Change Detection Analysis.

## E. Code Feature Change Detection Output

Fifthly, the Code Feature Change Detection outputs are listed here in Table IX.

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 |
| 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 in Table X.

TABLE X. CODE FEATURE CHANGE DETECTION SUMMERY

| Source Code Repository Name | Number of Feature Added | Number of Feature 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 in Fig. 14.

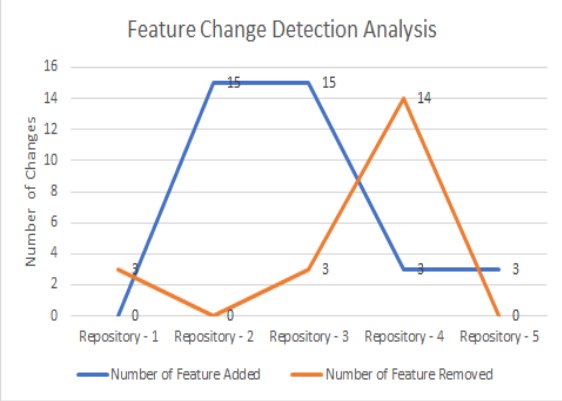


Fig. 14. Code Feature Change Detection Analysis.

## F. Source Functionality Change Detection Output

Sixthly, the Source Functionality Change Detection summary is presented here in Table XI.

TABLE XI. SOURCE FUNCTIONALITY CHANGE DETECTION SUMMERY

| 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 in Fig. 15.

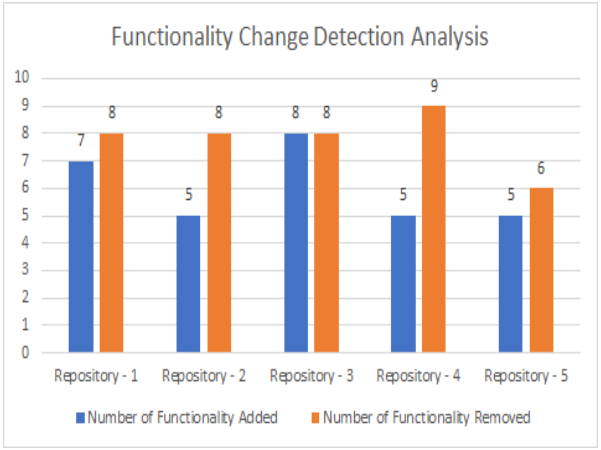


Fig. 15. Source Functionality Change Detection Analysis.

## G. Test Case Change Recommendation Output

Finally, the Test Case Change Recommendation outputs are presented here in Table XII and Table XIII.

TABLE XII. SOURCE FUNCTIONALITY CHANGE DETECTION SUMMARY – 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 SUMMARY – 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 and in order to justify the claim of improvements, it is must to carry out the comparative analysis. Hence in this section of the work, the comparative analysis with the popular existing works are performed on the framed metric for comparison (Table XIV).

TABLE XIV. Comparative Analysis

| System Details | Change Detection Capabilities | Pre-Requisite 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, Pre-Requisite 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. Conclusions

The software development industry completely relies on the accurate change management. The change driven structure or process of any organization makes 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 life cycle specifically the testing. Due to any modification to the source code, the testing operations also has to change. The challenge is to identify the current change and reduce repetition of the testing tasks. Thus, this work provides an automatic framework with Change Detection Capabilities, Pre-Requisite Detection Capabilities, Feature Detection Capabilities, Functionality Detection Capabilities and Test Case Change Recommendation for better test case managements. The major and most unique outcome of this work is to identify and recommend any changes in the test cases for making the world of software development 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.