Sudan University of Science and Technology

College of graduate studies

كلية الدراسات العليا

A proposed Framework for Versions Control using Abstract Syntax Tree Analysis

اطار مقترح للتحكم في الاصدارات باستخدام تحليل شجرة البنية المجردة

A Thesis Submitted in Partial Fulfillment of the Requirements of M.Sc. In Computer Science

Prepared by:

Salma Ali Zain Elabden

Supervisor:

Dr. Ahmed Mohamed Elsawi

November 2019

DEDICATION

To the meaning of love, affection and dedication

To my parents (Ali - Afaf)

To the husband Muhannad who helped me and encouraged me to achieve my dreams

To my lovely kids who suffered with me (Reem - Ahmed - Adham)

Thank you all

ACKNOWLEDGEMENTS

Praise is to Allah, Lord of the Worlds, the Merciful, thanks in the first and last to Allah.

I would like to express my deep appreciation and thanks to Dr. Ahmed Mohamed El Sawy, who supervised this research and assisted me with his knowledge and valuable observations and his encouragement to me.

I also extend my sincere thanks and gratitude to my husband, Muhannad Abdul Rahman, for his support and encouragement and assistance.

I would also like to thank

Mr. Ahmed Bashir.

ABSTRACT

As a result of increased competition and customer expectations many companies such as Google and Facebook have embraced rapid development. In rapid development methodologies such as scrum, several releases are launched before the final delivery of the product. Each particular time period is called Sprint. When the developer or the tester needs to revert to a particular version, they find it difficult to do so. This research provides a proposed framework with the aim of analyzing the different versions and finding the difference between them in terms of classes, functions and variables. This framework designed this framework using the Abstract Syntax Tree Analysis to analyze the versions, then compare and extract a report to help the developer determine the required version. The results show that the a proposed framework takes a times less than one hundredth and four tenth Millisecond to compare two files and find the difference between them, and less than 5870 Millisecond to compare two versions. The framework performance is evaluated by comparison with tools (JAPICC, CodeCompre). The performance evaluation was conducted using three small projects (as a case study). When comparing two files, the results showed that the CodeCompre is better than the framework. The CodeCompre compares the entire file, compared to the framework, which compares only the names of functions and variables. The main conclusion is that the framework showed good results in finding the difference between versions in terms of classes, functions and variables compared to the JAPICC tool. The contribution and importance of this study is that the framework can be used by companies that adopt Scrum methodology to compare versions.

المستخلص

نتيجة لزيادة المنافسة وتوقعات العملاء تبنت كثير من الشركات مثل جوجل وفيس بوك التطوير السريع. في منهجيات التطوير مثل سكروم يتم اطلاق العديد من الاصدارات قبل التسليم النهائى للمنتج . كل فترة زمنية معينة تسمي سبرنت. عندما يحتاج المطور أو المختبر العودة إلى إصدار معينـة ، يجد صـعوبة فـي القيـام بـذلك. يقدم هذا البحث اطار مقترح بهدف تحليل الاصـدارات المختلفة وايجاد الفرق بينها من حيث الفئات والدوال والمتغيرات. صمم هذا الاطار باستخدام شجرة تحليل البنيـة المجردة لتحليل الاصدارات ثم مقارنتها واستخراج تقرير لمساعدة المطور في تحديد الاصدارة المطلوبة. تبين النتائج ان الاطار المقترح ياخذ زمن اقل من مئة واربعة عشر ملي ثانية في مقارنة ملفين وايجاد الفرق بينها واقل من 5870 ملي ثانية لمقارنة اصدارين. يتم تقييم اداء الاطـار بمقارنتـه مـع الادوات (JAPICC, CodeCompare). اجريت عمليـة تقيـيم الاداء باستخدام ثـلاث مشـاريع صـغيرة (كدراسـة حالة).عنـد مقارنـة ملفـين اظهـرت النتـائج ان الاداة CodeCompare افضل من الاطار. الاداة تقارن كامل الملف مقارنـة بـالاطار الذي يقارن فقط اسماء الدوال والمتغيرات .الاطـار المقترح افضـل عند مقارنـة اصـدارين يمكنـه مقارنـة كل الملفات داخل الاصـدارات مـن حيث الـدوال والمتغيرات بينمـا تقـارن الاداة JAPICC الدوال والفئـات فقط. الاستنتاج الرئيسى ان الإطار اظهر نتائج جيدة في ايجاد الفرق بين الاصدارات من حيث الفئات والدوال والمتغيرات عند مقارنـة بـالاداة JAPICC. مسـاهمة واهميـة هذة الدراسـة ان الاطار يمكن ان يستخدم من قبل الشركات التي تتبع منهجية سكروم لمقارنة الاصدارات .

TABLE OF CONTENTS

List of Tables

List of Figure

Abbreviation

Abbreviation	Meaning
RAD	Rapid Applications Development
CVS	Concurrent Versioning System
SVN	Subversion
AST	Abstract Syntax Tree
GT	GumTree
JAPICC	Java API Compliance Checker
XML	Extensible Markup Language
JOSN	JavaScript Object Notation
RTED	Robust Tree Edit Distance
JDT	Java development tools
JAVAC	Java programming language compiler

Chapter One

1.1Introduction:-

With the evolution of the software industry, several methodologies have emerged to rapid release with shorter delivery times such as agile and scrum [1], so new features can be launched within weeks or days instead of several months [2].

There are several reasons that led software companies to adopt rapid development in their product development such as rapid response to frequent changes in technology and the labor market [3], increase competition and customer expectations, meet changing market demands [4], providing good and cheap products to customers [5] and traditional methods are no longer suitable for the development of modern systems requiring customer participation [6].

Large companies such as Google and Facebook used rapid development, enabling them to get feedback from the customer faster but reduced testing time[2]. Increasing competitive pressures on companies makes them undertake projects that are necessary for their survival [3]. Firefox has adopted rapid release to be able to compete with Chrome already adopted it [1].

Rapid Applications Development is a life cycle to accelerate software development, improve quality and reduce cost and they consist of several techniques including Scrum, Flex, Extreme Programming and others [8].

Scrum is a framework for dealing with changing requirements and aims to deliver products faster [7], and is based on the participation of customers and enables them to change their requirements during development [8].

The change of requirements can be during any stage of rapid development. The change occurs as a result of four reasons to meet the needs of outside the company, adding new features, as a result of the new technical demands, or as a result of learning by individuals or groups [3].

Scrum is consists of three stages is the pre-game, development and post-game, in the development phase the system is being developed in sprints [6].

Sprint is a set of repetitive cycles where new features are added to the features included in the previous design [8]. It’s a fixed period of time, ranging from 1-4 weeks [6].

Scrums encourage team work and communicate actively [9], the team has been working on several versions in the same sprint [10].

The versions are managed by versions control system such as concurrent versioning system (CVS) and Subversion (SVN) [12].

Svn tortoise is an open source system that is used to store files in a repository that allows scanning and restoring your old versions and managing them with source code. It’s systems to facilitate collaborative work and code sharing and provide information about the development process [12], these repository are often very large [8].

In scrum it is difficult for beginner developers, testers and new members to complete their tasks when they cannot completely understand the project [13], so the developers who use rapid development tend to include more comments in the source code in order to explanation [13].

The testing is the hardest part in Scrum [14], and because of the extra pressure to finish software development on time, and that project managers are likely to reduce their testing activities [19].

The testers have no knowledge of the source code and do not care about taking part in the development process since the collection of requirements [16].

Static test is designed to examine the source code in the early stages and minimize errors [17].It’s the software is tested without executing the code. Like Reviews, check syntax, Static analysis, etc. [18]

Static Analysis is the process of examining the software without the actual executing the programs, [12] and there are several techniques for static analysis from the matching of the strings until the matching of the graphic. [23]

The **matching techniques** that can be used for software version merging, program difference, regression testing, and multi-version program analyses.[12]

Abstract syntax tree is a tree representation of the abstract syntactic structure of source code. [20]

Neamtiu built an algorithm that tracks simple changes to variables, types and functions based on abstract syntax tree representation. [20] Abstract syntax trees are also used in analyzing source code.

1.1.Research Problem Statement :

In rapid applications development many versions are released before delivery of the product when completed. When the developer or tester needs to get back to a specific version, the tester finds it difficult to do so because the search manually is expensive. So we need a framework to test the different versions and explain the differences between them in terms of classes, functions, variables and get a detailed report on their contents. All the studies that have been reviewed find the difference between two files and not two versions of the same project, so this framework was developed [21] [22].

1.2. Hypotheses of Research:

This framework enables us to test different versions and determine the differences between them in terms of classes, functions and variables and extract a report on their contents.

This framework can facilitate the developer's decision to return to the appropriate version.

1.3. Importance of the Research:

Through this framework, the versions identified by the tester are tested to determine the differences in the programming codes, in term classes and details of each class in the project to make the appropriate decision to copy the appropriate version and then continue to test it.

1.4. Objectives of the Research:

The study aims to develop a framework that:

- Tests the different versions of the software repositories.
- Extract a report of each program's information and program details.
- Facilitates the decision of the tester, to determine the version to be referred to at the time, during rapid development

1.6 Research Methodology:

This research contains two phases. Firstly determine the problem, gathering data related to the research (data, tools, Algorithms, etc.) and clarify the objectives of the research. Secondly design the framework to achieve the objectives of research, determine the characteristics and advantages of the research. Provide recommendations.

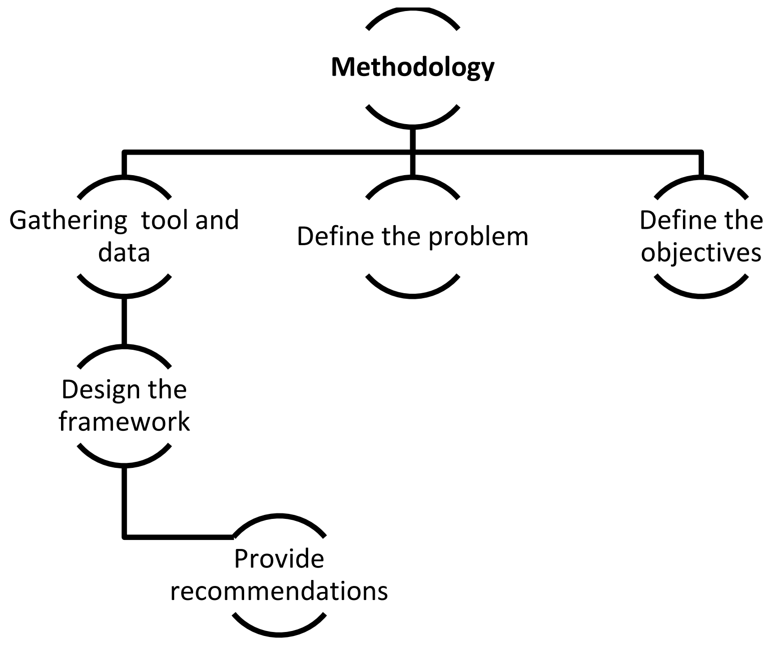


Figure 1.1: Research Methodology

1.7. Scope of the Research:

The approach we follow can be treated as static testing (static analysis).The static analysis by using abstract syntax tree to test rapid application development

1.8.Contributions of the Research:

This research provides one major contribution as well as other related contribution: First, a framework to testing the different versions and extract a report of each program's information to find appropriate version for tester. Second, save effort in accessing the right version.

1.9. Thesis Organization:

This research has organized into Seven Chapters as following:-

Chapter one:

Research problem background, research problem, research importance, research objectives, and research hypotheses, research methodology, research scope, contributions of research, thesis organization, summary chapter one.

Chapter two:

Introduction, software testing, software testing techniques, static testing, static analysis, abstract syntax tree, Rapid Applications Development, scrum, sprint algorithms and tools used in this research.

Chapter three:

Methodology

Chapter four:

Validation and Evaluation

Chapter five:-

Discussion of the results

Chapter six:-

Conclusion and future work.

1.10 Summary:

In the rapid development methodologies such as the framework of the scrum contains many different versions of the source code related to the project. The release of each period of time called sprint and managed by a special repository to control and facilitate the collective programming.

When you need to go back to a particular version, the developer finds it difficult to do so. This framework has been developed to help the developer or tester find the required version

Chapter Two

Literature Review

2.1 Introduction:

That many organizations spend 40% of their resources on software testing [19], the testing software used to check for errors to prevent software failure [24].

That it includes many techniques such as black box testing, white box testing box and others. The testing process is validation and verification [25]. It is a way to ensure the quality of the software [9].

That Scrum helps to improve the practice of testing by detecting any obstacles or impediments [26]. Rapid test does not just mean the test is fast but is done according to plan [27]. The test is done during development not end delivery of the product at an early stage, ensures the early detection of errors and defects [9].

2.2 Software Testing:

Software testing is an important method for evaluating software quality and achieving a higher level of reliability [24]. It is a broad term to include a wide range of different activities, from testing a small piece of software instructions (unit testing) to checking a large information system [25].

It is to monitor the software system to verify its validity, whether acting as intended [28]. It as a process to implement a program or application for the purpose of finding errors during implementation [29].

The test is included at each stage of the software development cycle [25], and the test may take a long time but helps to ensure quality and meet user expectations [9].

In some rapid development methodologies, developers may be uncomfortable to write tests cases before actually writing the program [11].

2.3 Software testing techniques:

2.3.1 Software Testing Strategies

The Software Testing Strategies integrates several software test case design technique into a well-planned series of steps that result in successful testing of software [31].

The Software Testing Strategies involves four main testes as shown in Figure 2.1.

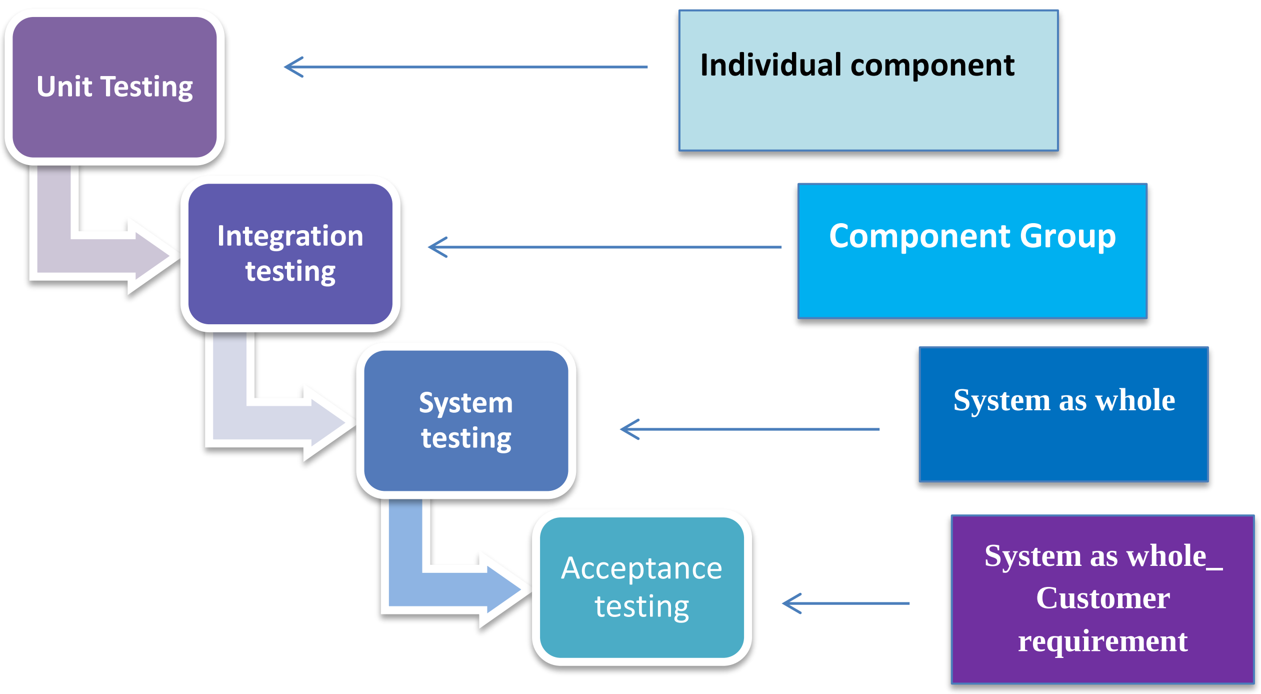


Figure 2.1: Types the Software Testing Strategies [31].

Unit Testing:

Unit testing is the testing of the smallest unit in the program, such as procedures, interfaces, functions, and classes [29]. It's a test at the lower levels and requires less effort to detect errors [24].

Unit testing is the test that is done at the lowest level of the basic components of the program [31].

Integration testing:

Integration testing is the test that is done when two or more modules are combined is often done on the interfaces between the components, to make sure that the work is done correctly [31].

Integration testing is the integration of modules and tests these together [25]. It as the test between all component interfaces and the larger structure of quality assessment [28].

System testing:

System testing, it's a series of tests to make sure that the system works properly and is capable of performing the required functions [29].

System test is the total quality test of the system and the functional and non-functional specifications such as safety, reliability and maintainability [31].

Acceptance testing:

Acceptance testing is the test that is done when the product is delivered to customers by the developer to make sure that the system works [31]. The purpose is to give confidence that the system works to find errors [25].

2.3.2 Software Testing Methodologies:

There are following methodologies for software testing as shown in Figure 2.2

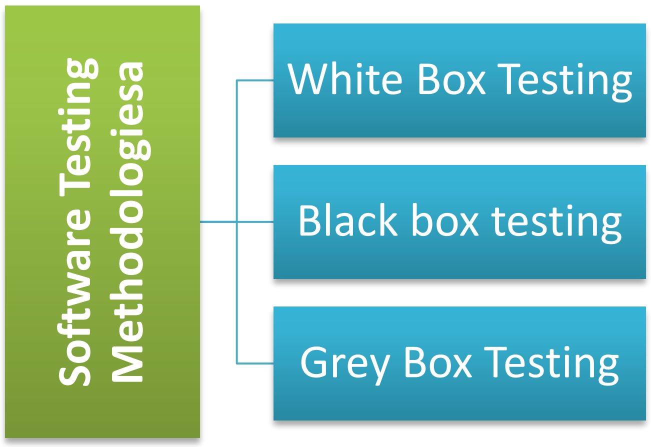


Figure 2.2: Software Testing Methodologies [31]

White box testing:

White box testing is a test technique based on the internal components of the system [29]. The one who tests must have a good knowledge of programming languages and techniques like desk checking, code review, formal inspections, loop testing, control flow testing and others as shown in Figure 2.3 [25].

The test on the internal structure of the code and the programmer must have full knowledge of the structure of the program [31].

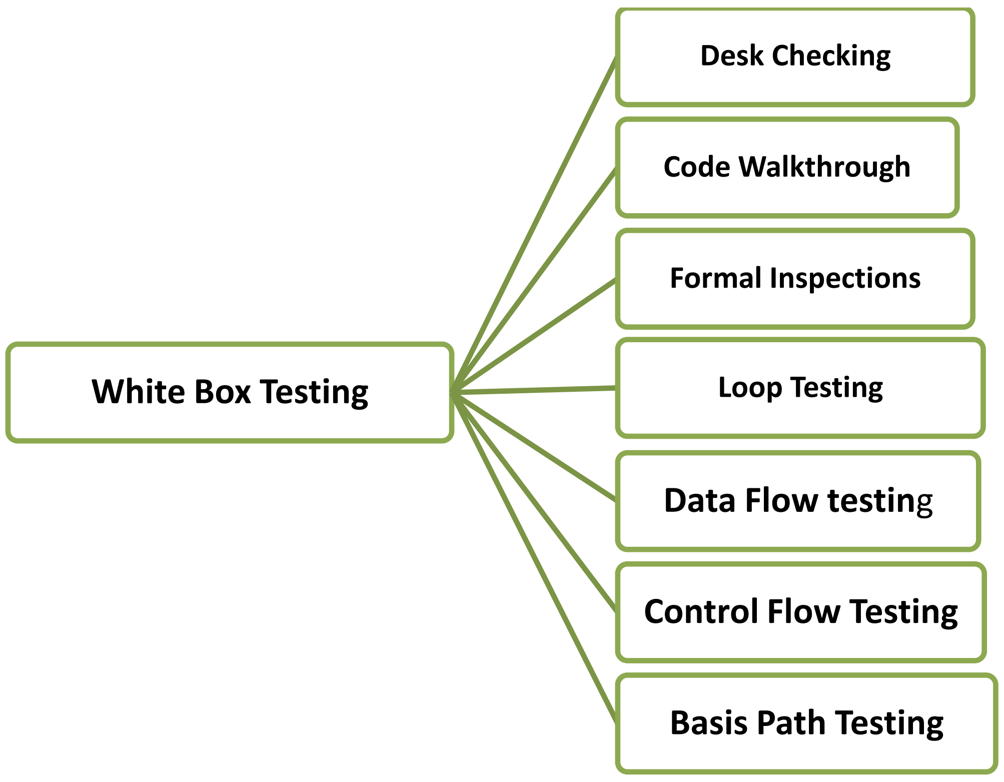


Figure 2.3 Types of white box testing [25]

Desk Checking:

Desk Checking is the test that is performed on the source code and the programmer must have knowledge of the programming language [32], and is a manual method checking programs and usually using a paper and pen to record the result of the program [29].

Formal Inspections:

Is the method of finding errors of instructions and detecting faults and violations [29], and is a formal and economical way to find code and design errors [32].

Control flow testing:

Control flow testing is a test technology that is applied to programming to show how many programs have been tested [29], it is a basic technology that is applied to all programs [32].

Control flow testing is a structural test that uses the control flow in the program's form [32].

Basis Path Testing:

Measurement to illustrate a basic set of implementation paths [29]. It’s used to test each path [32].

Loop Testing:

The test focuses on the validity of loop construction, and knows if the loop could end successfully [32].

Data Flow testing:

Is a test to know how to define or use program variables [32]. Its test uses control flow graph technology [29].

Black box testing:

Black Box Testing is a test technique without looking at its internal components [25]. It’s a technique to ensure that all the information that the system needs is acceptable and provides the right output [29].

Black Box Testing is a test that is based on a set of expected input and output [32]. Figure 2.4 shows the main types of black box testing such as all pair testing, fuzzing, boundary value analysis, and others.

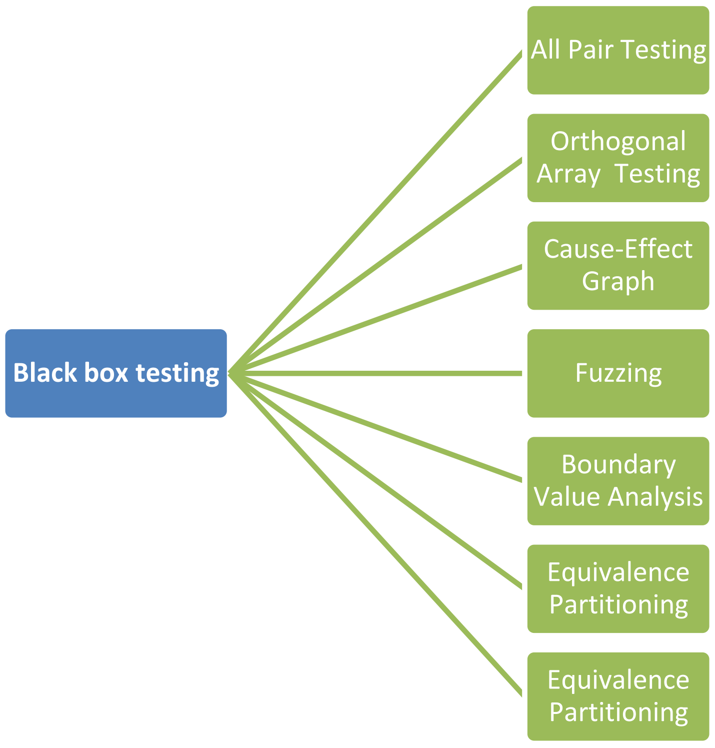


Figure 2.4 Types of black box testing [25]

Equivalence Partitioning:

This test divides the field of entry into the equivalence partitioning from which to derive the test cases [32]. This method helps to reduce the number of test cases [29].

Boundary Value Analysis:

Is the test that focuses on the maximum and minimum limits, error values and model values [32], or the limits where there is a possibility that the system will fail[29].

Fuzzing:

It depends on feeding the random input of the application using incorrect or damaged data [32]. A technology developed by Barton miller in 1989 [29].

Orthogonal Array Testing:

The test that is done when the input field is very small [32], and helps to reduce test groups [29].

Cause-Effect Graph:

Is a technique that creates a graph of input [29]. A relationship is created between the effect and its causes [32].

Gray Box Testing:

Gray box testing is a test that combines the benefits of black and white [32]. It's a test with limited knowledge of the inner workings of the application as well as the fundamental aspects of the system. It uses algorithms and internal data structures less than white [29].

Static Testing:

Static testing is often implied to check the source code or translators or Check syntax, data flow [18]. It is a set of methods used to determine software quality without reference to actual implementation, such as code inspection, analysis symbolic, and static program analysis [24]. It is an analysis of the source code to detect possible defects [33]

Dynamic testing:

Dynamic testing is the use of a variety of methods to ensure the quality of software during the actual implementation of the program with real data [24]. It started before the program was 100% completed in order to test separate jobs or modules [18].

2.4 Static Analysis:

Static analysis and the process of checking the source code without executing the actual program [9]. It is an introduction to the compiler optimization [34].

Static analysis is a test of the source code and knowledge of all possible causes of behavior that may arise at the time of execution. It is an accurate description of the behavior of the program regardless of input or operational environment [35]. Static analysis is used to identify problems in code, including potential errors and unnecessary complexity [23]. Static analysis works by building the program's status model and then determining how the program interacts with this situation [35].

Table 2.1: A comparison between Software Testing Techniques:

software testing techniques	Strengths	Weaknesses
White box testing	• The test can be started before GUI work. • Accurate testing, help improve the code work. • Help the tester to remove the extra instruction lines. • Help the tester to know the type of test input because he has knowledge of the internal coding.	• The programmer needs very good programming languages. • Requires resources and knowledge of the internal structure. • Is a costly test. • It is difficult to test large systems. • Generate test cases in the programming languages .
Black box testing	➢ Reproducible test. ➢ It is used in the operating environment. ➢ Suitable for large systems. ➢ Does not require the tester to have knowledge of programming languages. ➢ The user is involved in the development of tests. ➢ The tests are used several times. Examines the functions without knowledge of the internal implementation. ➢ The generation of cases depends on requirements and design specifications.	➢ If the requirements are unclear it may be difficult to generate test cases. ➢ Repeat tests may occur ➢ Some tests are not performed. ➢ The system failures are unlikely to be found.
Gray Box Testing	• Combines the benefits of black and white. • Experimental scenarios designed by it; needs partial knowledge of internal work.	• Test coverage is limited because there is no access to the source code; • There may be over-testing cases and many paths are not yet tested.
Unit testing	➢ Executed by the application developer. ➢ a little cost ➢ Part of the system is tested without waiting for other parts. ➢ Simple test due to small units.	➢ Time consumes. ➢ The difficulty of generating good test cases.

Integration testing	• All interfaces are tested between all units. • Tested on pre-tested modules.	• It is late after merging units with each other. • Difficulty planning errors.
System testing	• Does not require knowledge of interior design or coding. • The whole system is tested. • Ensures that functional specifications are met.	• It consumes a lot of time because the whole system is tested. • Its inputs are all software components.
Static testing	• Implementation of the test does not depend on the actual implementation. • The test is implicit. • Includes the hard test verification. • Helps to improve quality. • Uses mutation testing to detect errors. • Can be done before the product is completed 100%.	• Tools may need to check the source code. • Need to know the programming languages. • Sometimes the static test cannot be separated from the dynamic test.

2.5Rapid application development:

Rapid application development is a life cycle used for software development, providing us with fast and high-quality development. It is a process that accelerates the software development cycle and includes four phases, requirements planning phase, user design phase, Construction phase, cutover phases as shown in Figure 2.5. That the objective of rapid application development is to engage customers in system development. [8]

As the Ed Yourdon says, Information technology is consumer goods, because developers also have to adopt modern methods to meet user requests such as rapid application development, that the rapid development methodologies focus on code work such as implementation and testing, adding that the test is done in a repetitive manner, it is cheap and easy [8].

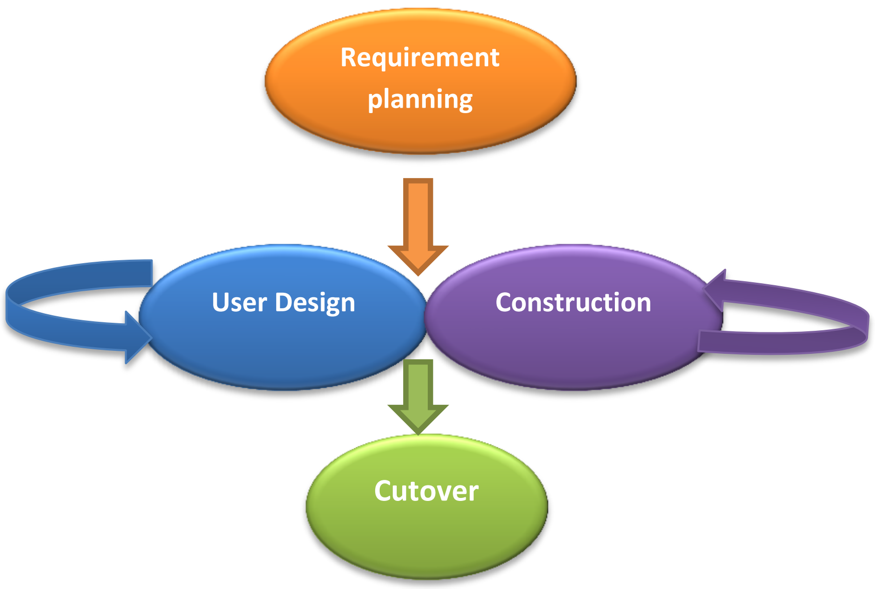


Figure 2.5: Phases in the James Martin approach to RAD[8]

2.6Agile Software Development:

The Agile software development embodies several methodologies including Extreme Programming, Scrum, Kanban, Lean, FDD (Feature-Driven Development), Crystal, DSDM (Dynamic Systems Development Method).

Agile is software development methodology that works to deliver the product repeatedly and gradually to the customer based on collaborative efforts between the team and customer, enhance the team spirit, produces releases at frequent intervals from one to six weeks and encourages rapid and flexible response to change. Figure 2.6 illustrates the benefits of agile methodology [39] [40] [43].

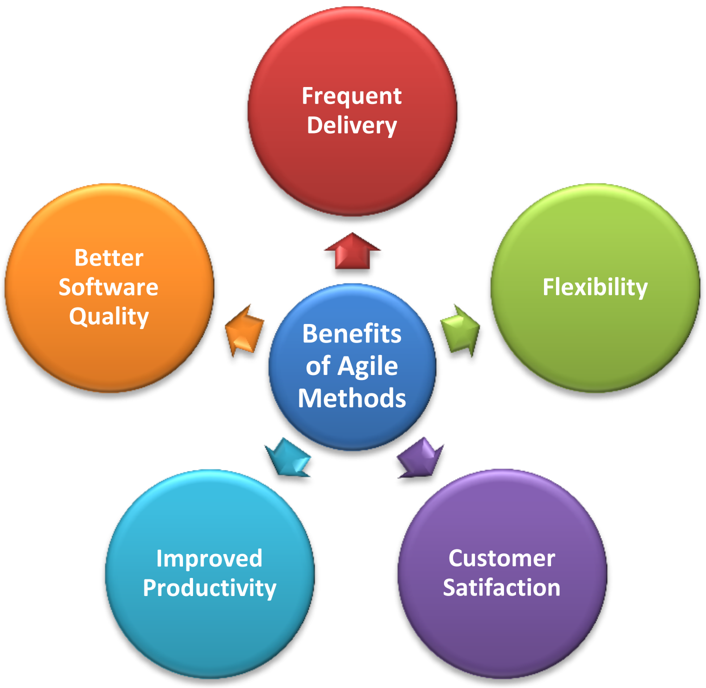


Figure 2.6 Benefits of agile methods [40]

2.7Scrum:

Scrum is a modern and comprehensive product development methodology [8], the customers are included in product development and that users can change their requirements from Scrum [37], It consists of three stages is the pre-game, development and post-game as shown in figure 2.7[6].

This approach involves the participation of many customers; it is suitable for small and medium enterprises and the term Scrum is derived from rugby game [8].

The Scrum helps in improving engineering practices, because it contains administrative activities aimed at identifying deficiencies or impediments to the development process [15].

Scrum includes early customer participation and products are issued repetitively way, so the development process is flexible and responsive to the changing market situation [19]. That requirements gathered in the form of stories and the most important way is to communicate face to face with the client. Frequent meetings with the client help to define requirements more clearly and accurately [11].

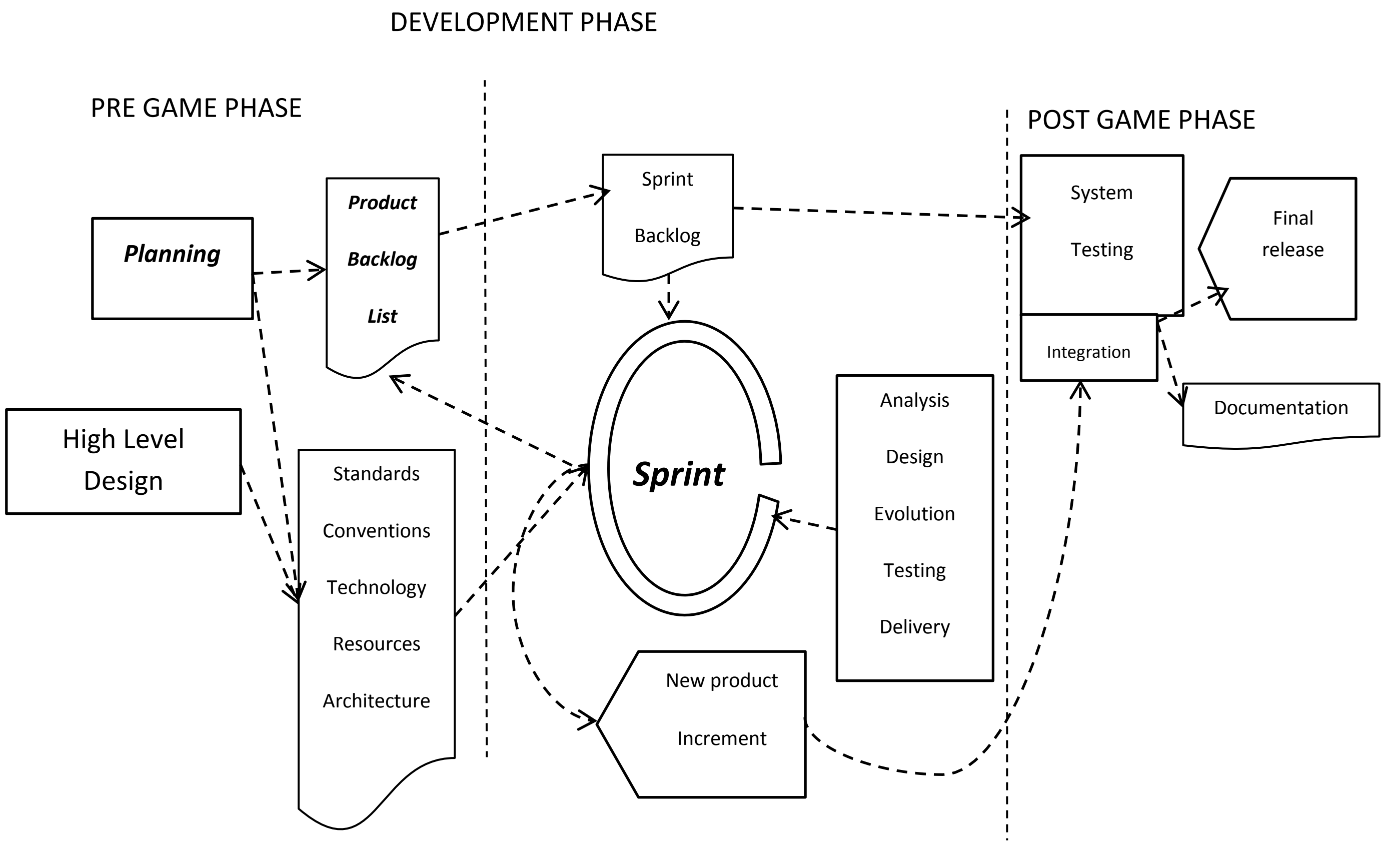


Figure 2.7 Scrum Processes [6].

Scrum Roles:

Product Owner:

The product owner is responsible for both functional and non-functional requirements and determines their priorities in the product backlog [38].

The product owner is responsible for determining which features to perform in each sprint priority is given to the implementation of properties in the product backlog [11].

Scrum Master:

The Scrum Master is an administrative role, who is responsible for implementing scrum as planned [44]. He is responsible for guiding and helping the scrum team in correct understanding the using scrum [6].

The Scrum Master ensures scrum practices until the end of the project [11].

Team Scrum:

A team that has the power to make decisions and organize itself to achieve the goals of each sprint [11]. Responsible for the creation of publications in a gradual way and identify tasks related to the product backlog [18].

Customers:

Customers involved in tasks related to product backlog of the system to be developed. And identify the requirements and objectives [9].

2.8Sprint :

The sprint is a set of development activities conducted over a period of time, usually from one to four weeks as shown in Figure 2.8. [8].

The sprint is an iterative course, where a job is being developed or improved with new additions [6].

The sprint is a small sessions, where new features are added, ranging from 3 to 4 weeks [19].

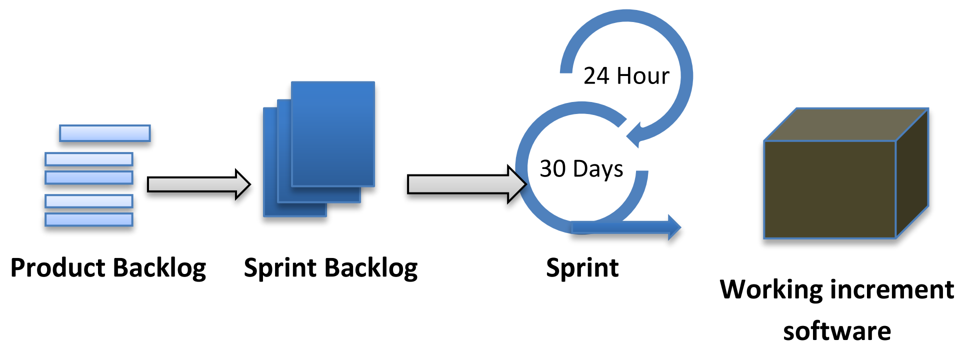


Figure 2.8 sprint activities [13]

Product Backlog:

An ordered list of work to be done to create the product and run by the product owner [11]. The list contains all the requirements that arise by the customer and sales department, and this list is constantly updated [47].

Sprint Review:

A period of time in which the scrum team and stakeholders review the product produced by each sprint. Evaluation of product increasing and planning for future activities [6].

2.9 Extreme programming:

Extreme programming is one of agile methods of software development, which aims to interact with customer during development and improve software quality.

It consists of the six phases, shown in Figure 2.9. It takes into consideration software development in the sense of writing all programs first [39] [40].

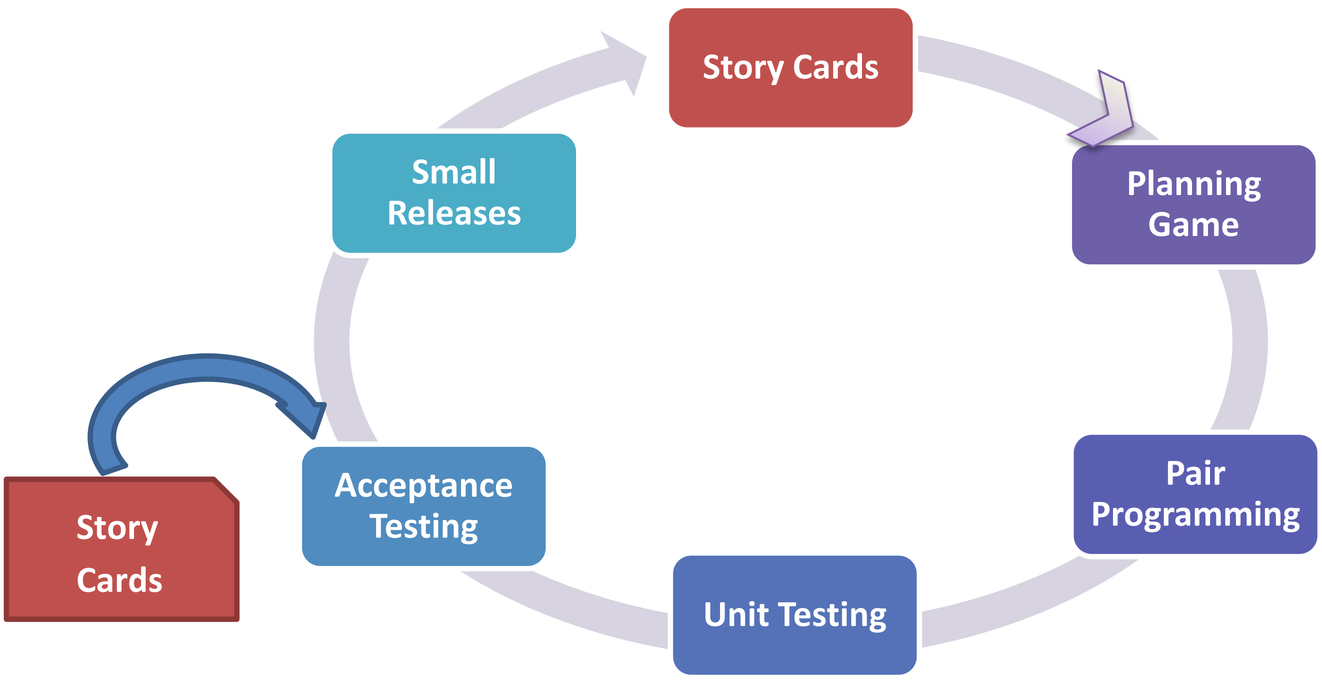


Figure 2.9: shows the development phases of Extreme programming [40]

2.10 Kanban:

Kanban is methodology for visualizing work progress visually with Kanban board. It based on work scheduling to facilitate the delivery of the product on the time. Kanban methodology is characteristics (simplicity, visualization, adaptability, Kanban board Figure 2.10 shows these characteristics [42].

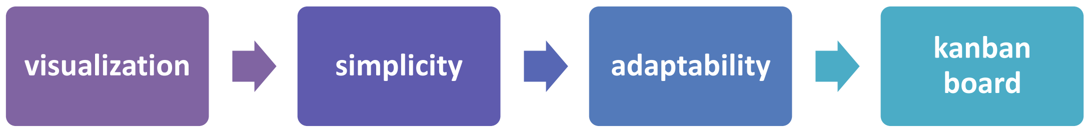


Figure 2.10: Kanban characteristics [42]

2.11 Waterfall model:

The waterfall is sequential development model; progress is made in the form of successive phases. The transition to the next stages does not take place before the completion of the previous stage, meaning if the requirements are not clear, the transition does not take place to the design stage [30]. Figure 2.10 shows the development stages in waterfall model.

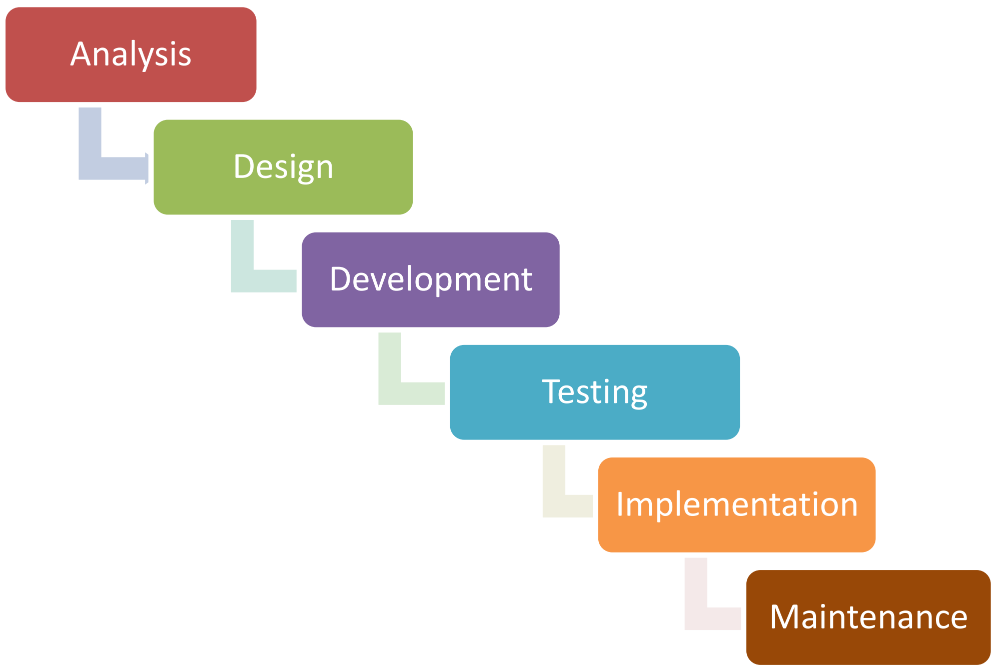


Figure 2.11: shows the development stages of Waterfall model [30]

Table 2.2: A comparison between Agile and Waterfall model:

	Agile	**Waterfall model**
Advantages	• The change is welcomed at any time • Simplicity • Adaptable • Keeping pace with change • Excellent features and good design • Encourages cooperation between team and members.	• Easy to use and understand. • Documents availability. • Preferably for small projects and clear requirement. • The steps are being completed sequentially.
Dis Advantages	• It uses simple documentation. • Not suitable for developing larges system (suitable for devolving small and medium system) • The commitment may be insufficient for developers. • The team should be knowledgeable.	• The change in requirements during the development process affects the development of the product. • Any new addition to the requirements increases the cost of project. • Not all problems that are identified at specific stage are solved at the same stage.

Table 2.3: A comparison between development methodologies:

	AGILE	**SCRUM**	**XP**	**KANBAN**	**RAD**	**WATERFALL**
Time period for one iteration	1-6 weeks	2-4 weeks	1-6 weeks	Time taken From the start of work until Bug fix is live.	2-4 weeks	>four weeks
Project Size		Small and medium. Projects	Small Projects.	Small Projects.	All types of projects.	All types of projects.
Approach of Development	Iterative and incremental	Iterative and incremental	Iterative and incremental	Incremental		Time schedules and sequential
Team Sizes	**7+/-2**	**7+/-2**	**<20**	**-**	**7+/-2**	**>15**
Project Coordinator	Team Master	Scrum Master	XP Coach	Team Work	Team Work	owner
Product Delivery	Continuous Delivery	Delivery as per Time boxed Sprints	Continuous Delivery	Continuous Delivery	Continuous Delivery	Implementation of entire system on the time.
Team Communication	Informal	Informal face to face	Informal	Informal face to face	Informal	Formal Based on documentation

2.12 Rapid Application Development & Software Testing:

Rapid application development testing activities are not considered a separate stage they are an essential part of coding. The test depends on the collaboration of the developer team, the tester and the client to establish test cases [9].

In Test-driven Development, the tests are written before the code is written and the testers may be unfamiliar with it [7]. In extreme programming, the test is a separate stage and the test cases are written before the actual execution of the jobs, and are done automatically. The tests are written by customers. The test is performed continuously after each issue is delivered [44].

In the scrum the test is performed during the creation of the product at the end of each sprint, all previous and subsequent tests are run when a new job is added, the acceptance and regression test is also performed [7].

Agile development delivers tested software at frequent intervals, increasing customer confidence. In addition, the use of test-based development helps to find and repair defects, reuse the code better and improve quality. The practice of testing throughout the life of the product in agile requires testers throughout the project, which contributes to increasing its cost [44].

Table 2.4: The relation between the testing and the rapid development Methodology:-

Methodology	**Advantages of relation**	**Disadvantages of relation**
Scrum	• Defects are identified at an early stage (development stage). • Client participation in developing test scenarios. • Scrum is used for automatic testing because the steps are repeated in the event of any change in the system (every sprint). • Ability to repeat old test sequences on new software releases.	• Because Sprint is a short period of time that does not have sufficient time for testing. • It is a challenge for testers who are used traditional approach. • When stories change; it is a problem for the testers • The testing process is expensive.
Extreme programming	• In the frequency the test is run (unit test) • Functional tests and many tests are performed before product delivery • Client participation in test design. • Test quality for the existence of pair programming. • Versions are subject to a comprehensive test	• The testing process is expensive because it is done repetitively • The test requires knowledge of programming skills.

	• The program is tested continuously.	
Dynamic Systems Development Method	• The test is mainly done in the interation phase • Testing is not a separate stage.	• When the requirements change, the test conditions change as well • Time consumes.
Feature Driven Development	• When a set of features is completed, it is tested (unit test) • The client, developer and tester involved in the development of tests.	• Is challenge for the testers if the requirements are not properly understood.
KANBAN	• Testing is performed continuously upon completion of development by experts • Errors are corrected in each repeat and get a clean code. • Direct tests are performed by all team members.	• The test should be easy and reusable • It should be understood by all team members.

2.13 Related studies:

2.13.1 Abstract Syntax Tree:

Abstract syntax tree picks up the basic components with the deletion of unnecessary grammatical input such as whitespace or comments, to make the program easy to read [46].

Abstract syntax tree is data structures are derived from source code that breaks a syntactic construct into a tree; each node represents a correct grammatical structure, the piece of code [21].

Abstract syntax tree is a tree classified and arranged from the root and each node has a string value [22].

2.13.2 Gum Tree

It’s complete framework to calculate the difference between two abstract syntax trees. It includes possibilities such as:

- Converting a source code into a language-agnostic tree
- Export the produced trees in various formats
- Compute the differences between the abstract syntax trees
- Export the differences between the trees in various formats
- Visualize the differences between the trees graphically

Before going into it must be defined some terms:

T is a tree made up of a set of nodes and T has a root symbolized by the root (T) symbol. Each node t € T has sequence of children (children (t)).

Each node has a value, value (t) = v and a label, label (t) =l.

The Gum tree aims to calculate a series of edit action such as (adds, delete, update, move). Gum tree to compute the mappings between two abstract syntax trees is composed of two successive phases:

Top-down Phase:

Search for the largest number of similar sub-trees between T1 and T2 and then add them to an index list called height-indexed priority list and contain a series of nodes by reducing the height and the list is handled by a set of data structure such as (pop, push, peekMax, open) [22].

To handle the previous list, we use the dice function (measure the ratio of common descendants between two nodes given a set of mappings) to sort the larger mapping from minHeight and place it in the mapping set [22].

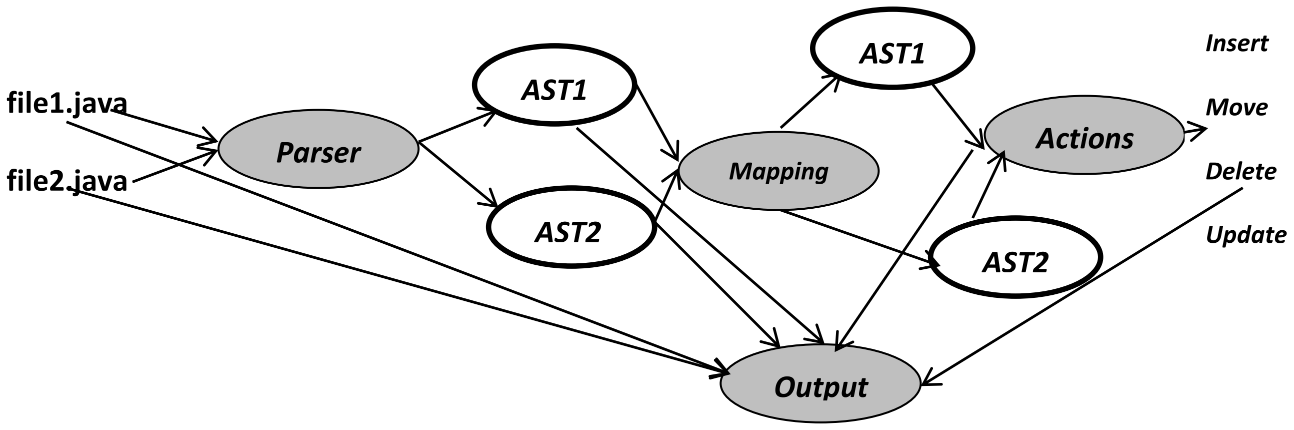


Figure 2.12: Gum Tree Mapping [22].

Algorithm 1: The algorithm of the top-down phase

Data: A source tree T1 and a destination tree T2, a minimum height minHeight, two empty height-indexed priority lists L1 and L2, an empty list A of candidate mappings, and an empty set of mappings M

Result: The set of mappings M

```
1.  push(root(T1), L1);
2.  push(root(T2), L2);
3.  while min(peekMax(L1), peek Max(L2)) > minHeight do
4.      if peekMax(L1) 6 = peekMax(L2) then
5.          if peekMax(L1) > peekMax(L2) then
6.              foreach t ∈ pop(L1) do open(t, L1);
7.          else
8.              foreach t ∈ pop(L2) do open(t, L2);
9.      else
10.         H1 ← pop(L1);
11.         H2 ← pop(L2);
12.         foreach (t1, t2) ∈ H1 × H2 do
13.             if isomorphic(t1, t2) then
14.                 if ∃tx ∈ T2 | isomorphic(t1, tx) ∧ tx 6 = t2
                    or ∃tx ∈ T1 | isomorphic(tx, t2) ∧ tx 6 = t1
                    then
15.                     add(A, (t1, t2));
16.                 else
```

```
17.                     add all pairs of isomorphic nodes of s(t1)
                        and s(t2) to M;
18.             foreach t1 ∈ H1 | (t1, tx) 6∈ A ∪ M do open(t1, L1);
19.             foreach t2 ∈ H2 | (tx, t2) 6∈ A ∪ M do open(t2, L2);
20. sort (t1, t2) ∈ A using dice(parent(t1), parent(t2), M);
21. while size(A) > 0 do
22.     (t1, t2) ← remove(A, 0);
23.     add all pairs of isomorphic nodes of s(t1) and s(t2) to M;
24.     A ← A \ {(t1, tx) ∈ A};
25.     A ← A \ {(tx, t2) ∈ A}
```

Bottom-up Phase:

We take the mapping from the previous stage (top-down phase) as inputs. Then we look for container mapping that is created when the two nodes have the largest number of matched descendants. In container mapping, we look for recovery mapping between strings that are still not identical to the mapping nodes.

For each non-leaf node T1 we extract the candidate list in T2; the node c belongs to T2 if it is not identical, but it has some matching descendants and then we apply the function to search for extra assignments between T1 and T2 and we remove their matching descendants [22].

Algorithm 2: The algorithm of the bottom-up phase.

Data: Two trees T1 and T2, a set M of mappings (resulting from the top-down phase), a threshold minDice and a maximum tree size maxSize
Result: The set of mappings M.

```
a   foreach t1 ∈ T1 | t1 is not matched ∧ t1 has matched
        children, in post-order do
b     t2 ← candidate(t1, M);
c     if  t2≠ null and dice(t1, t2, M) > minDice then
d
e         M ← M ∪ (t1, t2);
f         if max(|s(t1)|, |s(t2)|) < maxSize then
g             R ← opt(t1, t2);
h             foreach (ta, tb) ∈ R do
i             if ta, tb
                  not already mapped and
                  label(ta) = label(tb) then
                    M ← M ∪ (ta, tb);
```

The Gum Tree output is in the form of an XML(Extensible Markup Language) file or a JOSN (JavaScript Object Notation) file, which is the difference between the two files in terms of addition or deletion or modification of the nodes, but cannot find the difference between them in terms of nodes move and to evaluate the performance of the algorithm used manual evaluation, and the performance of the Gum Tree algorithm compared with the diff tool, and applied to real data, using 144 different scenarios for manual performance and 12792 for automatic evaluation. They found that the algorithm is more accurate than the related works but does not work on the moving nodes [22].

2.13.3 Spoon Library:

Spoon is an open source tool that enables you to convert and analyze source code and access all elements of the program (class, variable, method, annotation, and enum declarations) whether for reading or modifying [45]. It breaks code up into a meta-model consisting of three parts: structural part, code part, and references part.

Figure 2.13 shows the structural part [45], The CT mean compile time.

The spoon tool focuses on the model, which was designed on sun JAVAC, and provides complete instructions for software, quick fix errors. Provides developers with a way to query the code in one line of code and also a graphical interface to see the abstract syntax tree a spoon of the program under analysis [45].

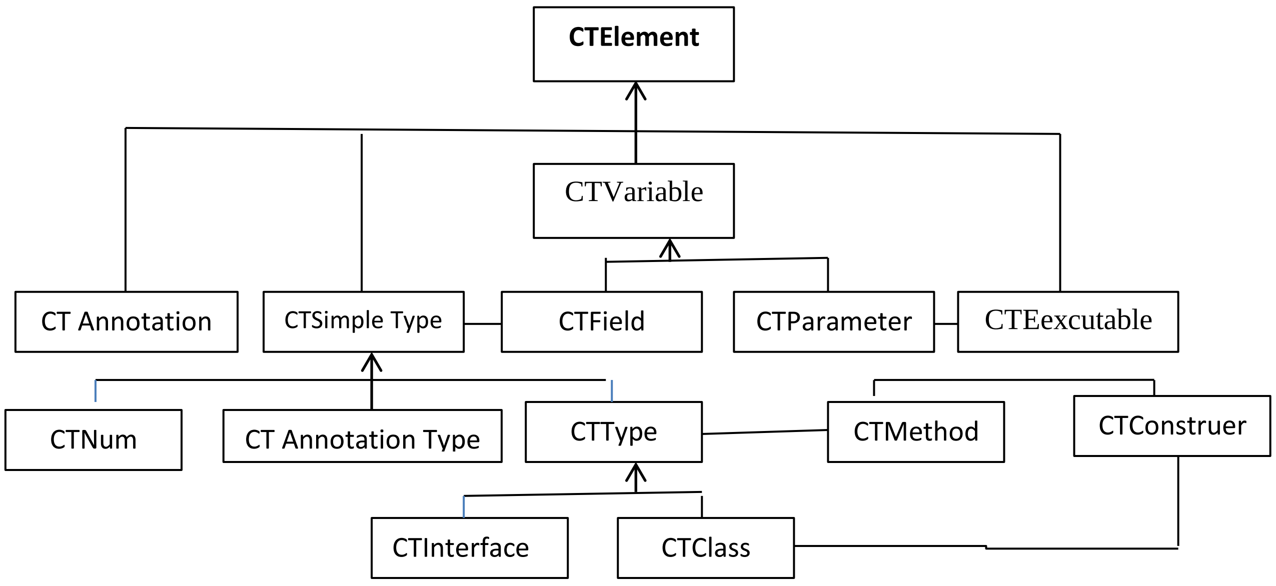


Figure 2.13 Spoons a Metamodel structural part [45]

2.13.4 TypeV Tool:-

A tool for analyzing and visualizing source code by tracking the differences between abstract syntax trees and using the spoon tool to represent the source code to abstract syntax tree the differences were calculated by gum tree algorithm.

TypeV enables you to track developer contributions in the repository and also displays a timeline that is a graph showing the project log. All previous studies find the difference between two files and not the difference between two versions of the same project [21].

2.13.5 Java API Compliance Checker (JAPICC):

A tool is open source for checking backward binary/source compatibility of a Java library API. The tool checks classes declarations of previous and later versions and analyzes changes that may break compatibility: removed methods, removed class, added methods, etc. The tool is intended for developers of software libraries and Linux maintainers who are interested in ensuring backward compatibility [48].

2.13.6 Code Compare Tool:-

Code Compare is a free tool to compare and merge differing files and folders. It can integrate with all popular source control systems such as SVN, Git, and others. Code Compare is shipped both as a standalone file diff tool and a Visual Studio extension [49].

2.14 WinMerge:

WinMerge is open source tool for compare both folders and files, presenting differences in a visual text format that is easy to understand and handle [50].

2.14.1 Static Analysis Framework:

The static analysis framework is framework to analyze and evaluate students programs and measure their quality based on the standards of software engineering, but can only analyze small programs written in Java [23].

Table 2.5: A comparison between tools:

	GumTree	**Spoon tool**	**CodeCompare tool**	**JAPICC tool**	**TypeV**
Conversion to abstract syntax trees	Yes by using the Eclipse JDT parser to java, Mozilla Rhino parser to JavaScript, Fast R parser to R and Coccinelle parser to C.	Yes by using the Eclipse JDT parser	No	-	Yes by using Spoon tool
Comparison two versions	No	No	No	Yes	No
Comparison two files	Yes	Yes	Yes	No	Yes
Open Source	Yes	Yes	Yes	Yes	Yes
Languages supported	Java, JavaScript, R and C.	Java	Java, JavaScript, php, and C.	Java, C	Java

2.15 Summary:

There are several test methods, including white box testing and black box testing; the first is to test the internal components and the second to test the functions of the system.

The test levels are the unit test, which is test part of the system, for example a new function. Integration testing when merging units with each other. The test that is for the system as a whole is the system test.

The static test is performed without the actual implementation of the system and is used in this research.

There are many methodologies that may be used when testing software such as rapid application development and scrum.

This chapter also talked about several tools and algorithms such as spoon, CodeCompre and Gum Tree etc. [45][22]

Chapter Three

3. Research Methodology

3.1 Introduction:

A review of previous literature has been conducted. We have found that rapid release such as scrum is a framework for software development and many releases are released before the final product is delivered within a certain period called sprint[8][19].

The scrum allows changing requirements during system development as a result of customer feedback [7]. When a developer or tester needs to return to a specific version, it is difficult to do so because there are many previous versions in the repository.

This framework can extract a report that is the difference between two versions of the same project in terms of classes, functions, variables that exist in the present version and not in the previous version, or a report about a specific version.

3.2The Proposed Framework:

By collecting data from previous studies, we found to resolve the research problem, it must follow these phases:

Table 3.1: represent proposed framework

Numbers of Phases	The Phase	Describe the phase
1	**Choose language to write research:**	Analyzing source code is popular research topic and the Java language was chosen for study because of its popularity and availability of tools [21] [20] [22]. Development Framework by using: • Net Beans is open-source integrated development environment for developing with JAVA, C ++, etc. • Java Development Kit (JDK) is a software development environment used for developing Java applications.
2	**Selected Versions**	Versions are selected from the repositories by the developer
3	**Convert The Source Code to abstract syntax tree**	Versions are represented, to abstract syntax tree before change and after change (using the Eclipse JDT parser, spoon tool) [45] [21].
4	**Calculate difference between two abstract syntax trees**	Gum Tree is an algorithm to calculate differences between abstract syntax trees established by spoon.
5	**Validation and Evaluation.**	The selected tools for evaluation:- • Code Compare tool • JAPICC tool

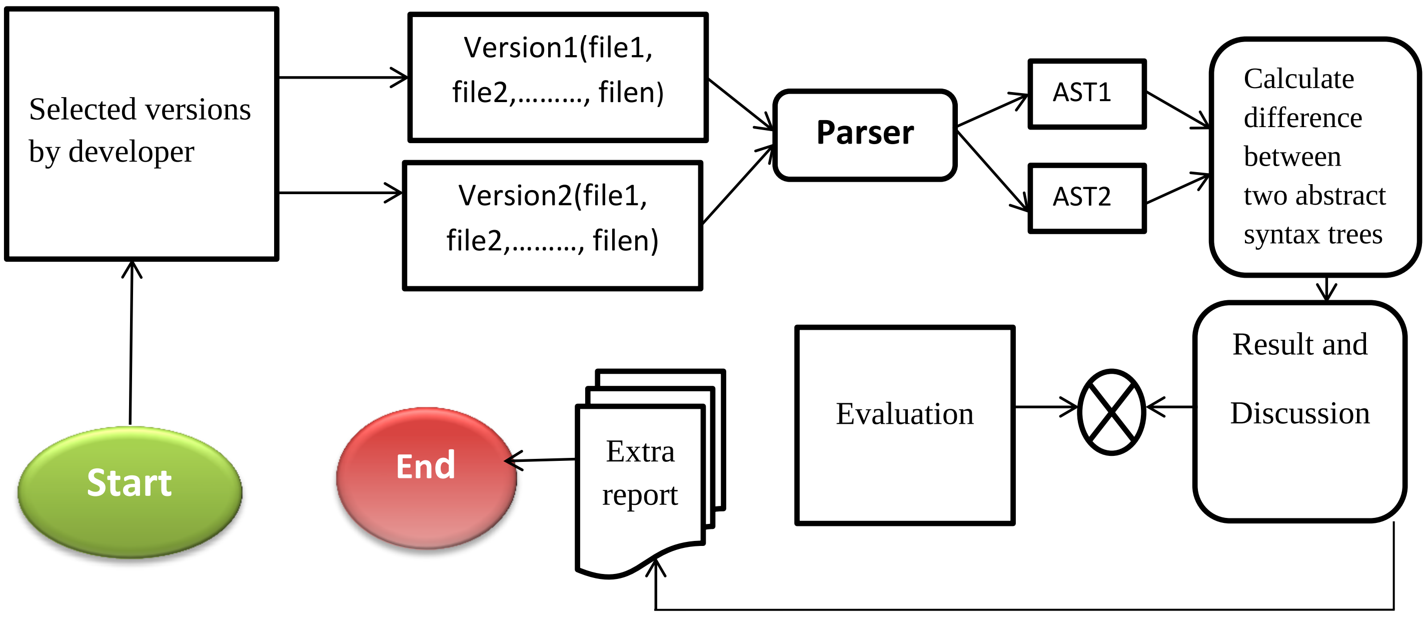

Selected versions by developer
Version1(file1, file2,………, filen)
Version2(file1, file2,………, filen)
Parser
AST1
AST2
Calculate difference between two abstract syntax trees
Result and Discussion
Evaluation
Extra report
End
Start

Figure 3.1 represent the proposed framework

Chapter Four
Validation and Evaluation

Validation:

This section presents the results obtained by using a framework with a case study (versions of gum tree). This was done by comparing the:-

- files
- versions

The process of finding the difference between the versions involves four main phases shown in Figure 4.1. The table 4.1 explains the four steps:

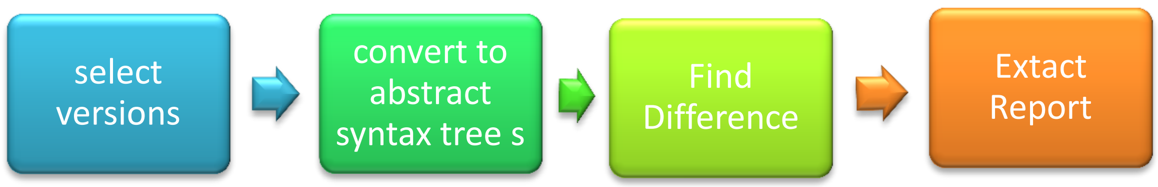


Figure 4.1: Steps to calculate the differences between versions

Table 4.1: Phases for finding the difference between versions:-

Firstly: The two versions will be selected from the repository.
Secondly: Convert each version to abstract syntax trees.
Thirdly: Compare abstract syntax trees
Fourthly: The results are presented through the framework.

4.1 Case Study Gum Tree:

We are selected versions of the Gum Tree as case study because is open source, an availability of versions and documents, written in java and represent real data.

Table 4.2: Description of the datasets:

Name of Version	Number of Version	Number Files	Size of Version
GumTree 1.0.0	V.1.0.0	193	2, 86 MB
GumTree2.0.0	V.2.0.0	184	1,12 MB
GumTree-develop	-	264	70,1 MB
GumTree.2.1.1	V.2.1.1	244	70,4 MB
Total		**885**	**144.8**

a. Compare Two Files:

A comparison was made between four randomly selected samples from the Gum Tree versions in terms of classes, functions and variables names within the program file. Each sample is a pair of files (a file from the previous version with another file from the latest version).

The following table shows the number of differences between a file from the previous version (number of deletions from functions or variables) and a file from the latest version (the number of added from functions or variables).

Table 4.3: the number of deletions and addition in the file versions:-

No	File name	Previous Version		Latest Version	
		Functions	**Variables**	**Functions**	**Variables**
1	GumTree2.0.0\AbstractTree.java GumTree2.1.1\AbstractTree.java	***10***	***2***	***4***	***0***
2	GumTree2.0.0\TreeUtils.java GumTree2.1.1\TreeUtils.java	***3***	***2***	***0***	***0***
3	GumTree2.0.0\MapTree.java GumTree2.1.1\MapTree.java	***0***	***0***	***3***	***0***

From the previous table we note the new functions have not been added to some of the files, where the number of additions is zero and another a large number of functions has been removed from them, such as the number 10.

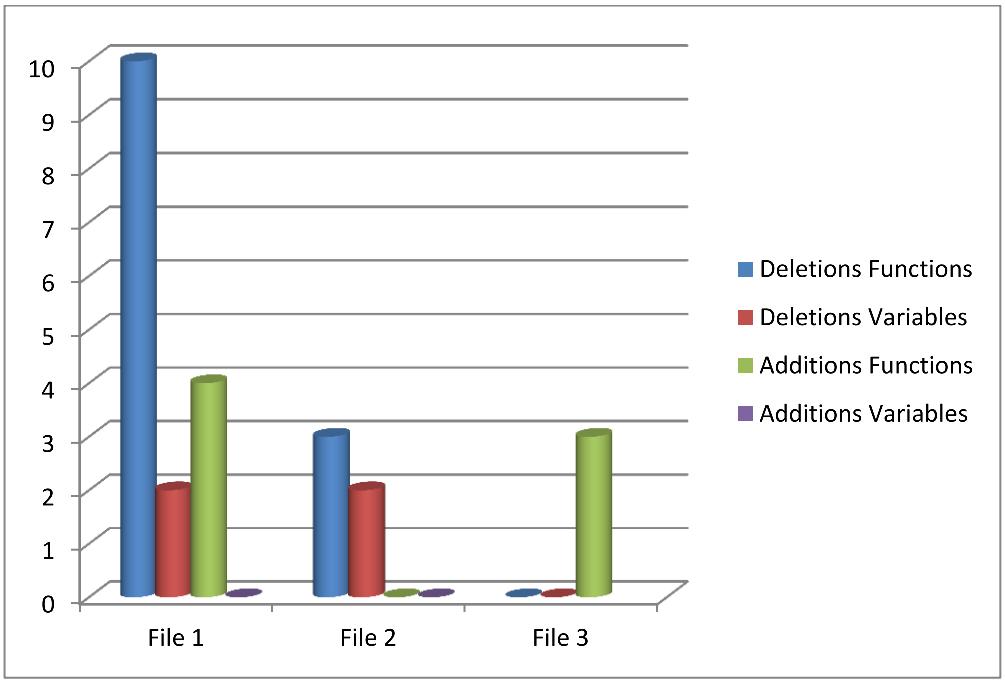


Figure 4.2: The number of deletions and addition in the file versions

The following table shows the time required by the files to convert to an abstract syntax tree and the time to compare the abstract syntax trees to find the difference between them in Milli Second.

Table 4.4: Shows the parsing time and comparison time in Milli Second.

No	**File Name**	**Parse Time of Previous File**	**Parse Time of Latest File**	**Comparison Time**
1	GumTree2.0.0\AbstractTree.java GumTree2.1.1\AbstractTree.java	*235*	*328*	*114*
2	GumTree2.0.0\MapTree.java GumTree2.1.1\MapTree.java	*94*	*124*	*62*
3	GumTree2.0.0\TreeUtils.java GumTree2.1.1\ TreeUtils.java	*504*	*335*	*70*

From the table we find that the time required by the framework to convert the file to abstract syntax tree is less than 504 Milli second and also to compare the Abstract syntax trees is less than 114 Milli second.

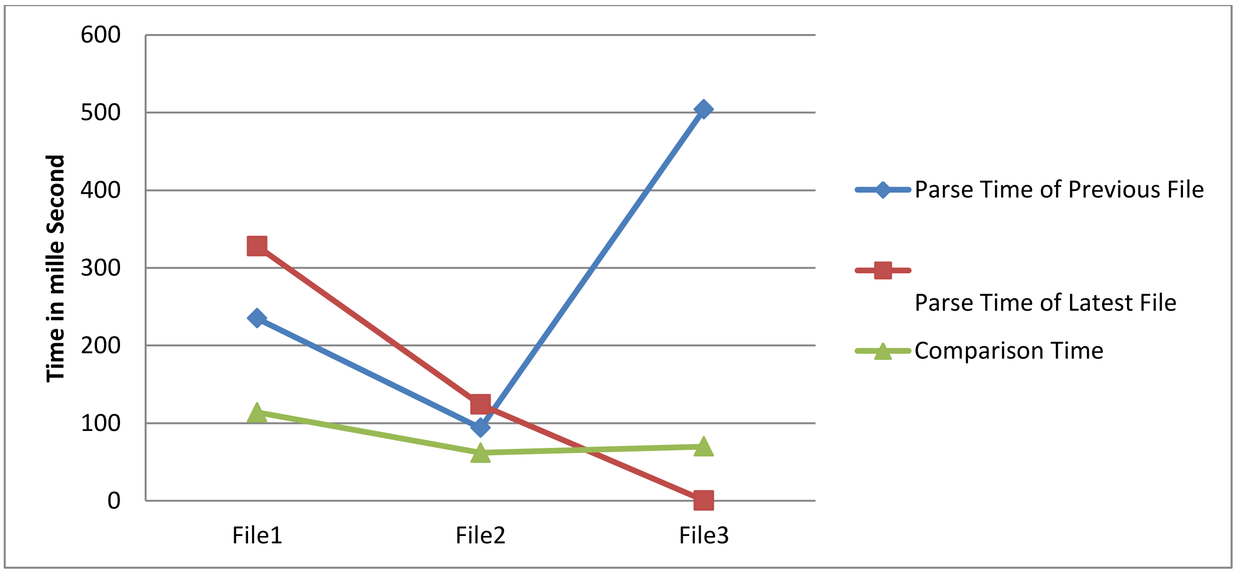
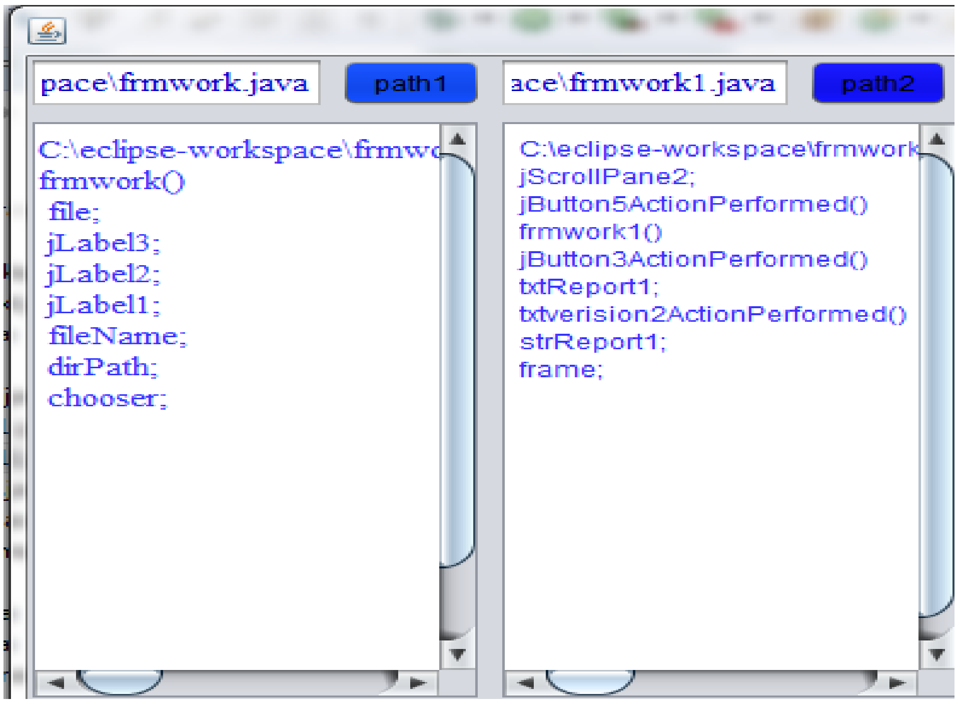


Figure 4.3: Shows the parsing time and Comparison time

b. Compare Versions:

The parsing of all the files within each version to abstract syntax trees and compare with the extracted abstract syntax trees from the latest version files, and the outputs is two reports:

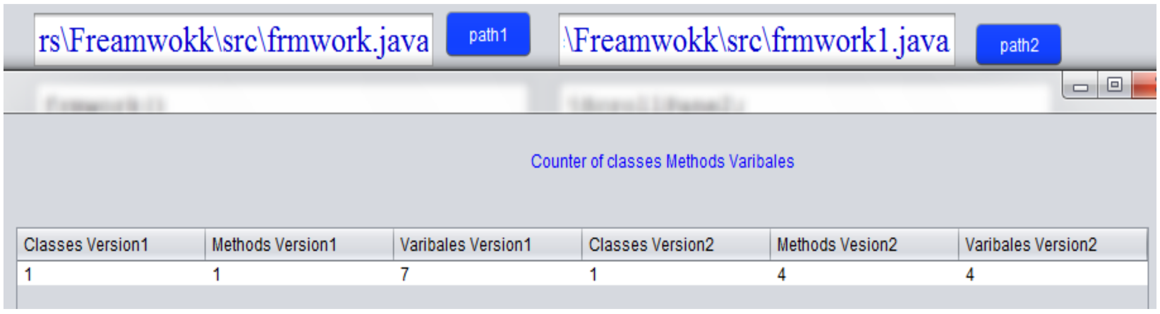
- The first report shows the classes, methods and variables that were removed from the previous version and are not included in the latest version.
- The second report shows the classes, methods and variables that were added in the latest version and are not included in the previous version.



```
C:\eclipse-workspace\frmwo
frmwork()
 file;
 jLabel3;
 jLabel2;
 jLabel1;
 fileName;
 dirPath;
 chooser;
```

```
C:\eclipse-workspace\frmwork
jScrollPane2;
jButton5ActionPerformed()
frmwork1()
jButton3ActionPerformed()
txtReport1;
txtverision2ActionPerformed()
strReport1;
frame;
```

Figure 4.4: The Framework Report



Counter of classes Methods Varibales

Classes Version1	Methods Version1	Varibales Version1	Classes Version2	Methods Vesion2	Varibales Version2
1	1	7	1	4	4

Figure 4.5: The Framework Report(counter of classes, methods and variables)

The following table shows the adjustments that were made by comparing every two versions of GumTree, whether additions in the latest version or deletion from the previous version.

Table 4.5: shows the number of deletions and addition in the versions

No	**Name of Version**	**Previous Version**			**Latest Version**		
		Deletions Classes	**Deletions Functions**	**Deletions Variables**	**Addition Classes**	**Addition Functions**	**Addition Variables**
1	*GumTree 1.0.0, GumTree2.0.0*	***0***	***0***	***15***	***15***	***22***	***40***
2	*GumTree2.0.0, GumTree.2.1.1*	***7***	***31***	***41***	***34***	***117***	***66***
3	*GumTree.2.1.1, GumTree-develop*	***71***	***1626***	***1316***	***45***	***245***	***110***

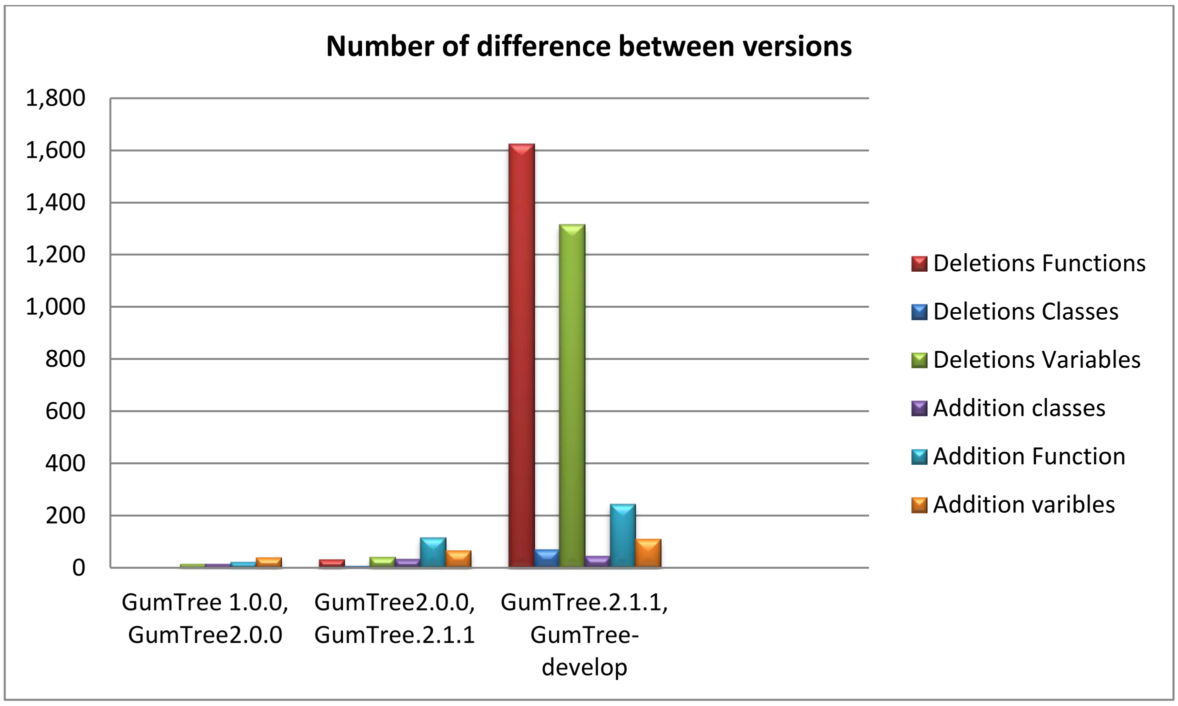


Figure 4.6: Shows the number of deletions and addition in the versions

Table 4.6: Shows the parsing time and comparison time

Name of Versions	Parse Time of Previous Version	Parse Time of Latest Version	Comparison Time
GumTree 1.0.0, GumTree2.0.0	**12029**	**2846**	**5870**
GumTree2.0.0, GumTree.2.1.1	**7078**	**6450**	**670**
GumTree.2.1.1, GumTree-develop	**7415**	**6348**	**390**

From the above table we find that the time required by the framework to convert the version to abstract syntax tree is ***(12029, 2846, 7078, 6450, 7415, 6348)*** Milli second and also to compare the abstract syntax trees is ***(5870, 670, 390)*** Milli second.

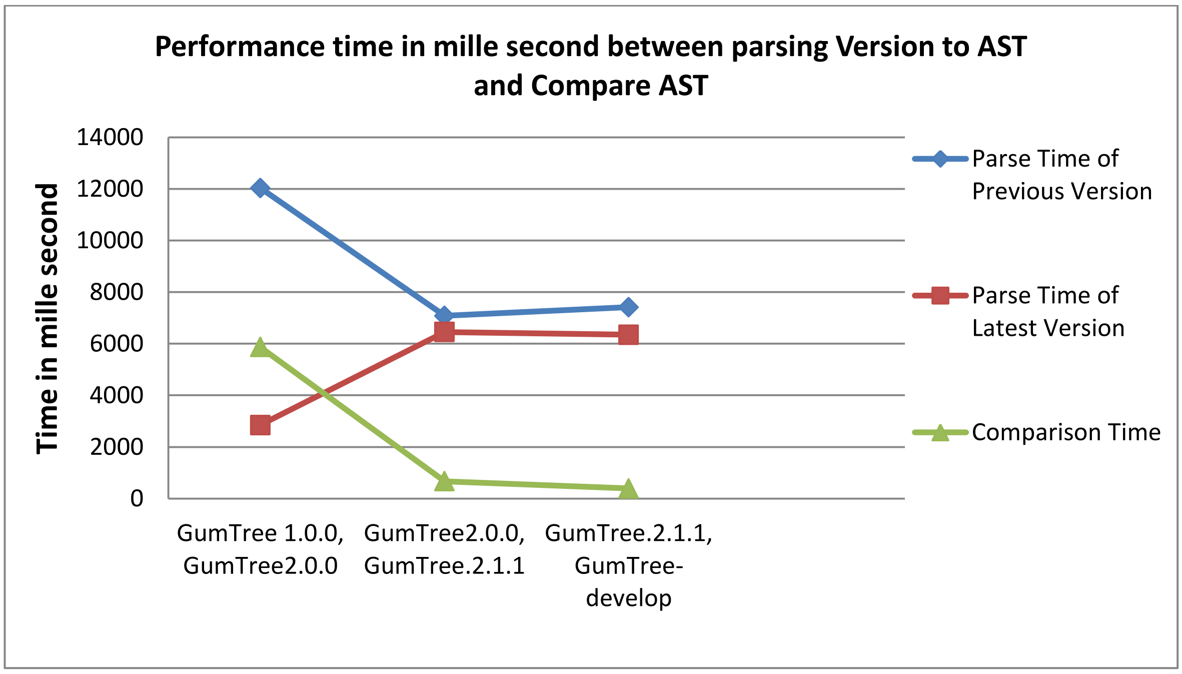


Figure 4.7: Shows the parsing time and Comparison

Evaluation:

There are many tools to find the difference between two files as described in chapter two such as CodeCompre tool. In the evaluation process, two tools (CodeCompre, JAPICC) were selected to test their performance and compare with the performance of the framework.

Three open source projects were used in the evaluation as a case study and are shown in Table 5.1. These projects were selected because they represent real data

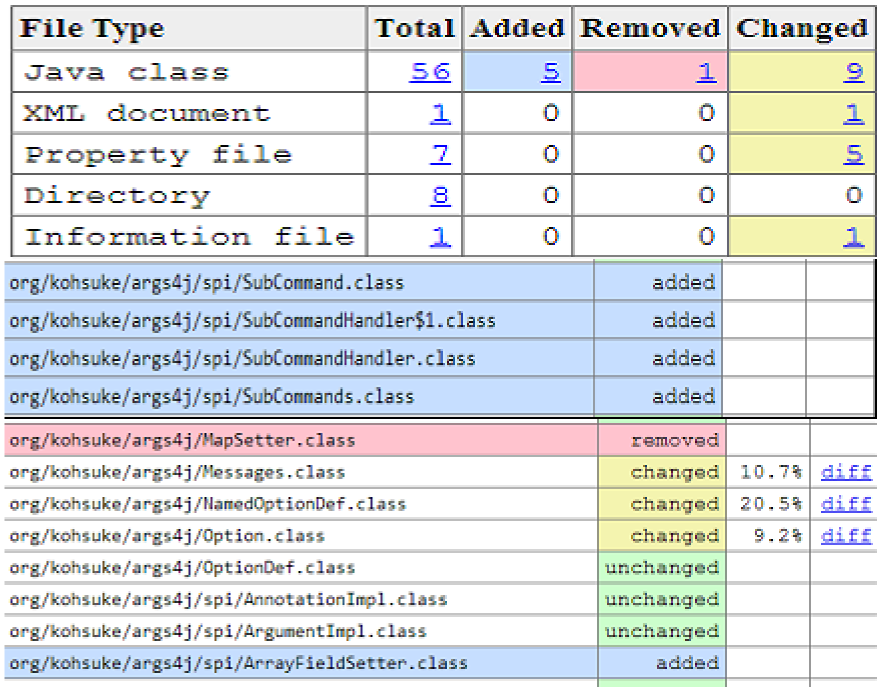
The evaluation is done in LAPTOP: MSI, CPU: Intel (R) Atom (TM) CPU N455 @ 1.66 - 1.67GHz, RAM: 2.00 GB, OS type: 32-bit and OS: Windows.

Table 4.7: the dataset and tools used in the experiment

Comparison type	**Tools**	**Data**
Compare Two Files	Code Compare	Versions of GumTree (V.1.0.0,V.2.0.0,V.2.1.1)
Compare Two Versions	JAPICC	args4j (V.2.0.16, V.2.0.22, V.2.0.23)
		Versions of akka (V2.5.18 , V2.5.19 , V2.5.20 ,V2.5.22)

4.2 Case Study(args4j V.2.0.22, V.2.0.23):

In this paragraph we would like to test the performance of the framework with the JAPICC tool. We choose the args4j library for manual evaluation of document availability. It is an open source library and its small size represents real data. We compared the two versions (args4j-2.0.22, args4j-2.0.23) using a framework and JAPICC tool and obtained the results shown in the figures below:

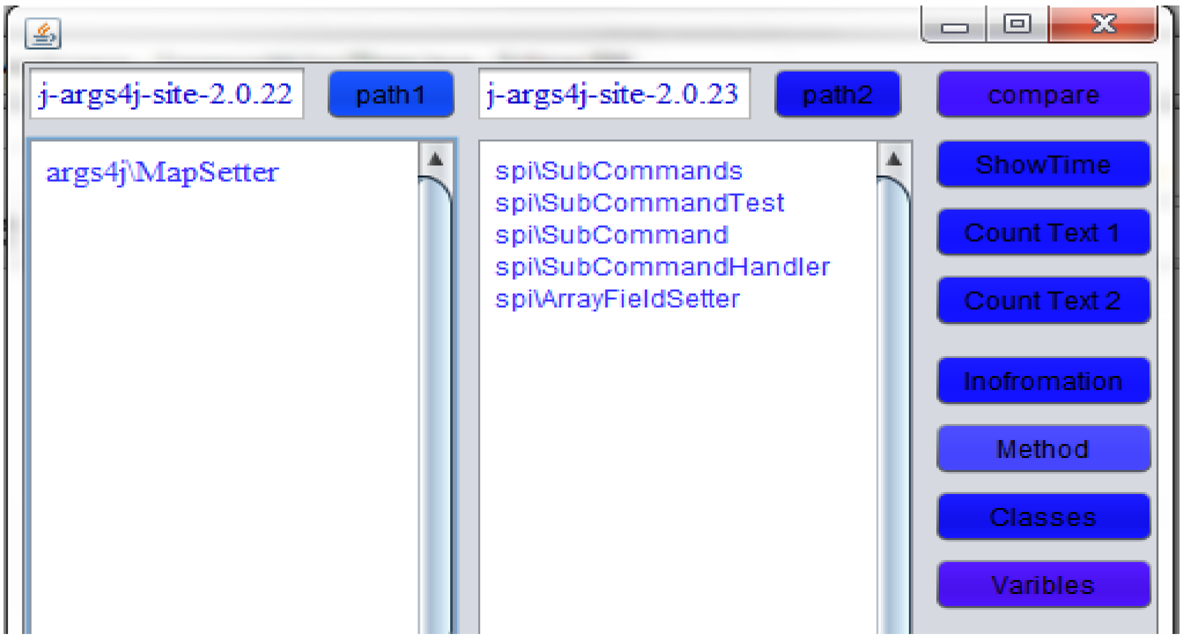


File Type	Total	Added	Removed	Changed
Java class	56	5	1	9
XML document	1	0	0	1
Property file	7	0	0	5
Directory	8	0	0	0
Information file	1	0	0	1

org/kohsuke/args4j/spi/SubCommand.class	added		
org/kohsuke/args4j/spi/SubCommandHandler$1.class	added		
org/kohsuke/args4j/spi/SubCommandHandler.class	added		
org/kohsuke/args4j/spi/SubCommands.class	added		

org/kohsuke/args4j/MapSetter.class	removed		
org/kohsuke/args4j/Messages.class	changed	10.7%	diff
org/kohsuke/args4j/NamedOptionDef.class	changed	20.5%	diff
org/kohsuke/args4j/Option.class	changed	9.2%	diff
org/kohsuke/args4j/OptionDef.class	unchanged		
org/kohsuke/args4j/spi/AnnotationImpl.class	unchanged		
org/kohsuke/args4j/spi/ArgumentImpl.class	unchanged		
org/kohsuke/args4j/spi/ArrayFieldSetter.class	added		

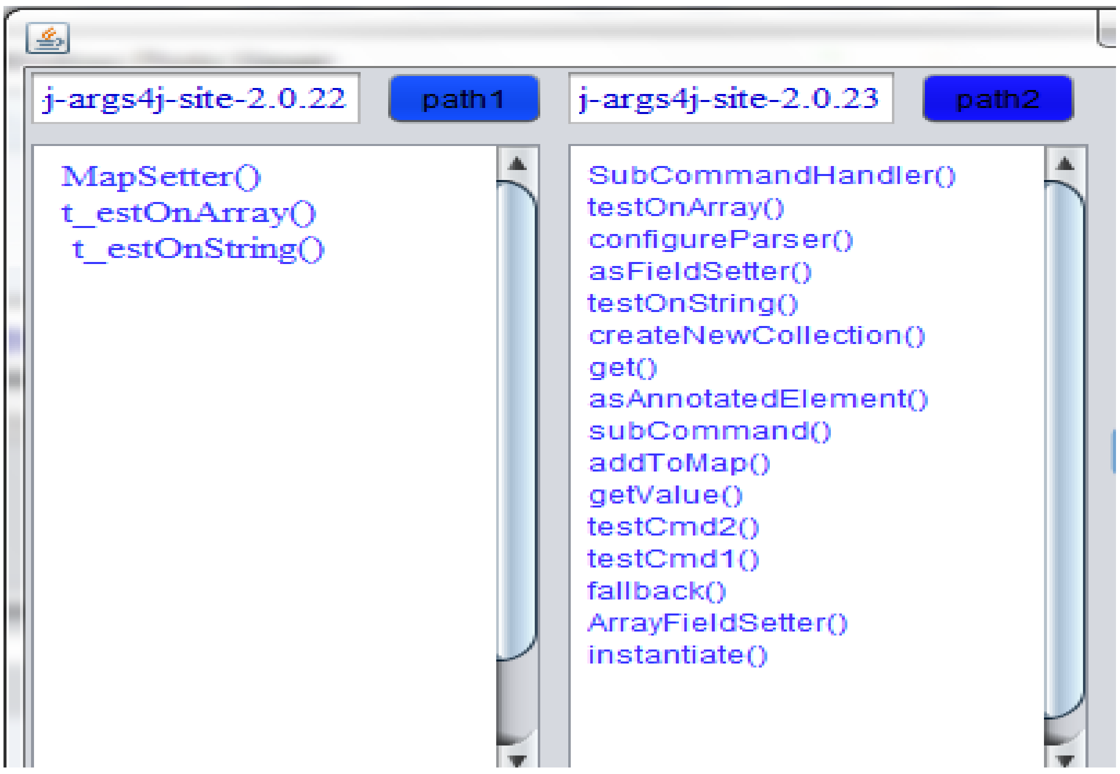
Figure 4.8: The result obtained by JACCI tool



args4j\MapSetter

spi\SubCommands
spi\SubCommandTest
spi\SubCommand
spi\SubCommandHandler
spi\ArrayFieldSetter

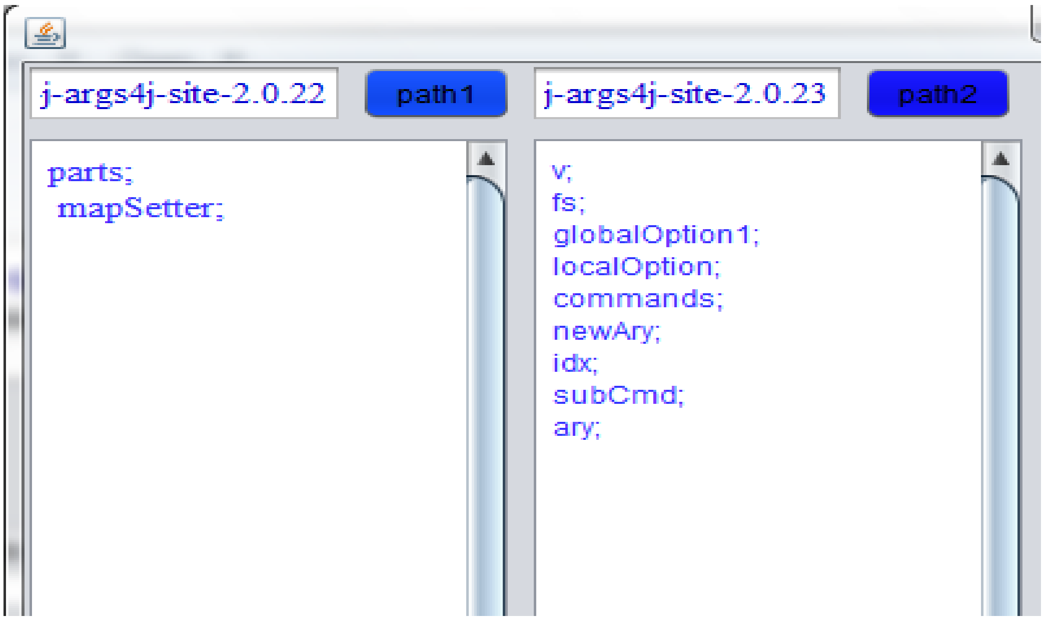
Figure 4.9: The result obtained by framework(names of classes)



MapSetter()
t_estOnArray()
t_estOnString()

SubCommandHandler()
testOnArray()
configureParser()
asFieldSetter()
testOnString()
createNewCollection()
get()
asAnnotatedElement()
subCommand()
addToMap()
getValue()
testCmd2()
testCmd1()
fallback()
ArrayFieldSetter()
instantiate()

Figure 4.10: The result obtained by framework (names of functions)


j-args4j-site-2.0.22

path1

j-args4j-site-2.0.23

path2

```
parts;
 mapSetter;
```

```
v;
fs;
globalOption1;
localOption;
commands;
newAry;
idx;
subCmd;
ary;
```

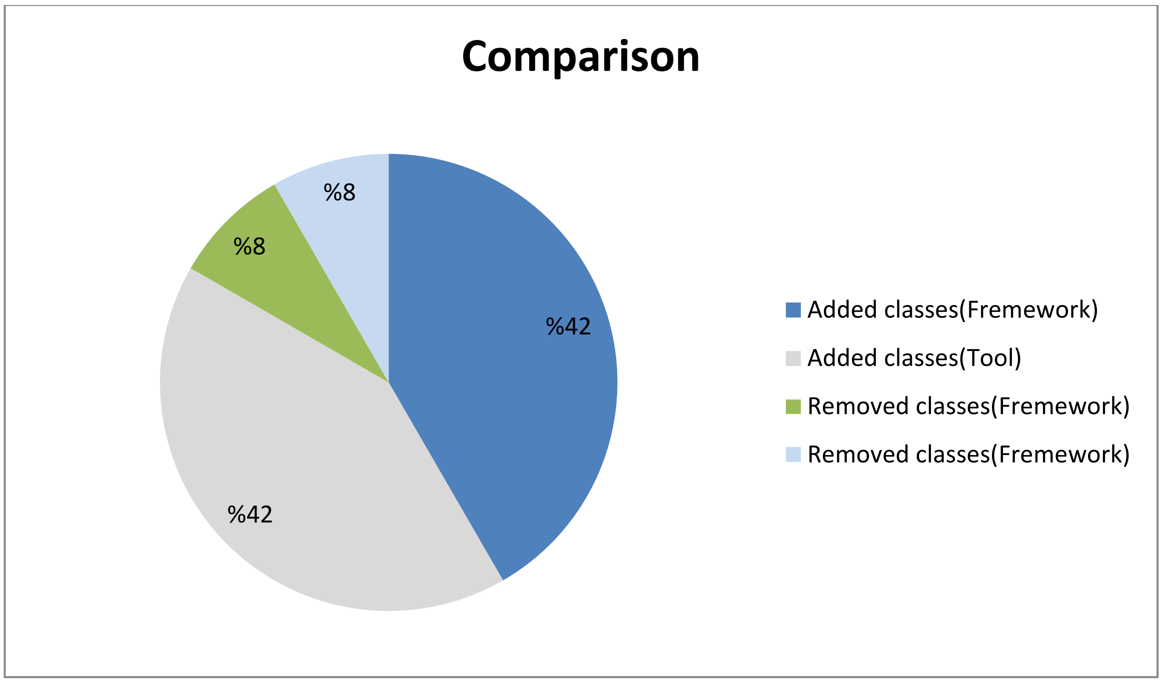
Figure 4.11:The result obtained by framework(names of variables)

We note from the results extracted by the framework and the JAPICC tool shown in figures 4.8 - 4.9 – 4.10 that one file was deleted from version args4j-2.0.22 which is **Map Setter** and five files were added in version args4j-2.0.23 which is **ArrayField Setter, Sub Comm and Handler, Sub Comm and Test, Sub Command and Sub Commands**. For function deleted from version args4j-2.0.22, the framework has identified **three** deleted functions. The JAPICC tool has **five** deleted functions. As for the added functions in version (args4j-2.0.23) **27** functions have been identified in the framework, and **21** functions are in the JAPICC tool.

The framework can find deleted and added variables as shown in Figure 4.11

Table 4.8: The results obtained by comparing the performance of the tool with the framework

Number of Version	**Tool**				**Framework**			
	Added Classes	**Removed Method**	**Removed Classes**	**Added method**	**Added Classes**	**Removed method**	**Removed Classes**	**Added Method**
args4j-2.0.22 ، args4j-2.0.23	5	5	1	21	5	3	1	27



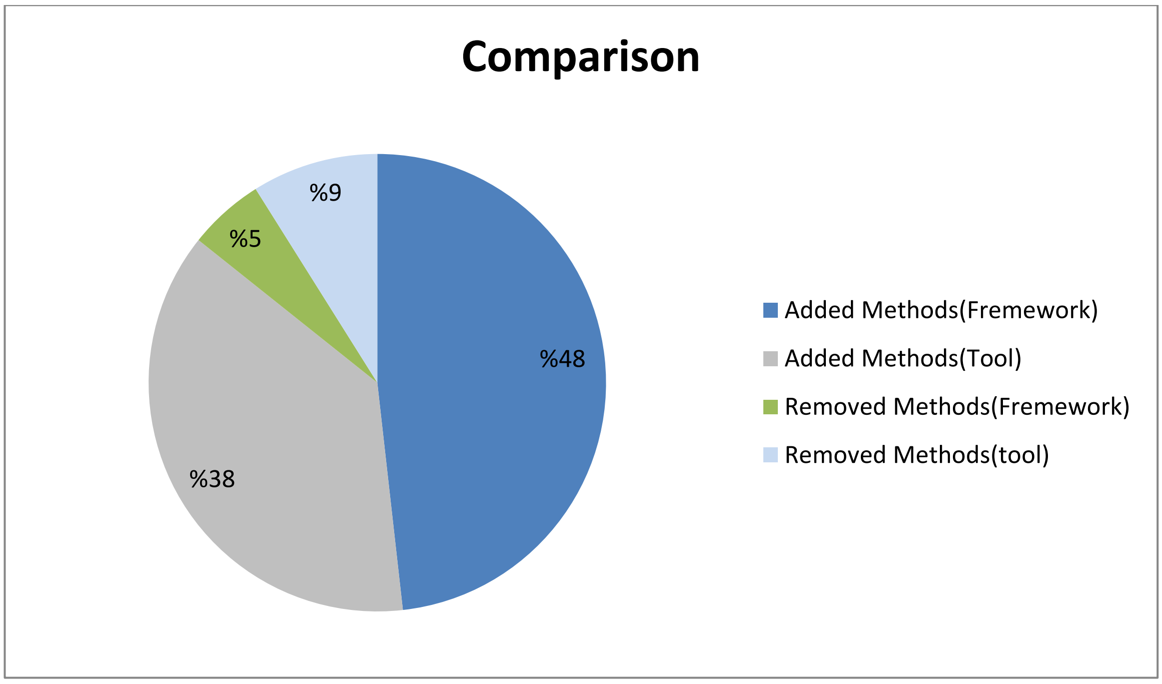


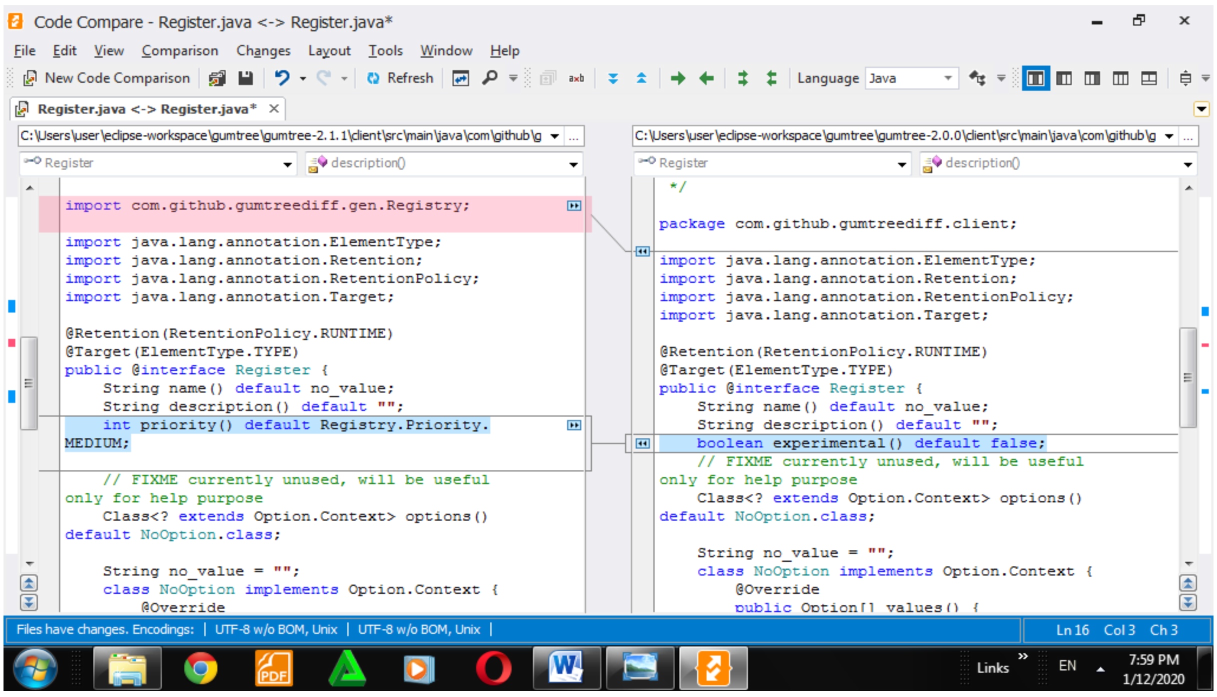
Figure 4.12: Comparison the results of the JAPICC tool with the framework results.

4.3 Case Study (Gum Tree):

The Code Compare was selected because it can compare two files. The experiment is based on a manual evaluation to calculate the differences between two files by the framework and Code Compare.

In the experiment, we used three versions of gum tree as a dataset for a real open source project.

Both of the Code Compare and the framework highlight the deleted lines from the left file, or those added to the right file as it is shown in Figure 5.2.



C:\Users\user\eclipse-workspace\gumtree\gumtree-2.1.1\client\src\main\java\com\github\g

```
import com.github.gumtreediff.gen.Registry;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface Register {
    String name() default no_value;
    String description() default "";
    int priority() default Registry.Priority.
MEDIUM;

    // FIXME currently unused, will be useful
only for help purpose
    Class<? extends Option.Context> options()
default NoOption.class;

    String no_value = "";
    class NoOption implements Option.Context {
        @Override
```

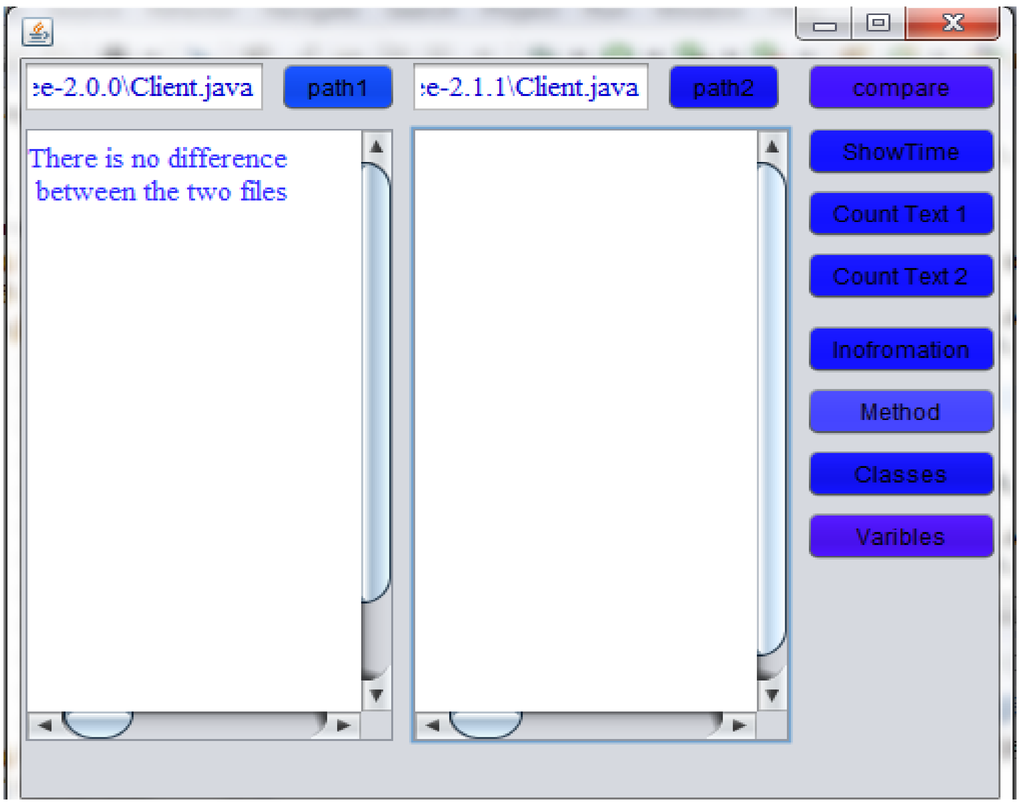
C:\Users\user\eclipse-workspace\gumtree\gumtree-2.0.0\client\src\main\java\com\github\g

```
 */

package com.github.gumtreediff.client;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface Register {
    String name() default no_value;
    String description() default "";
    boolean experimental() default false;
    // FIXME currently unused, will be useful
only for help purpose
    Class<? extends Option.Context> options()
default NoOption.class;

    String no_value = "";
    class NoOption implements Option.Context {
        @Override
        public Option[] values() {
```

Figure 4.13: The main Screen of CodeCompare Tool

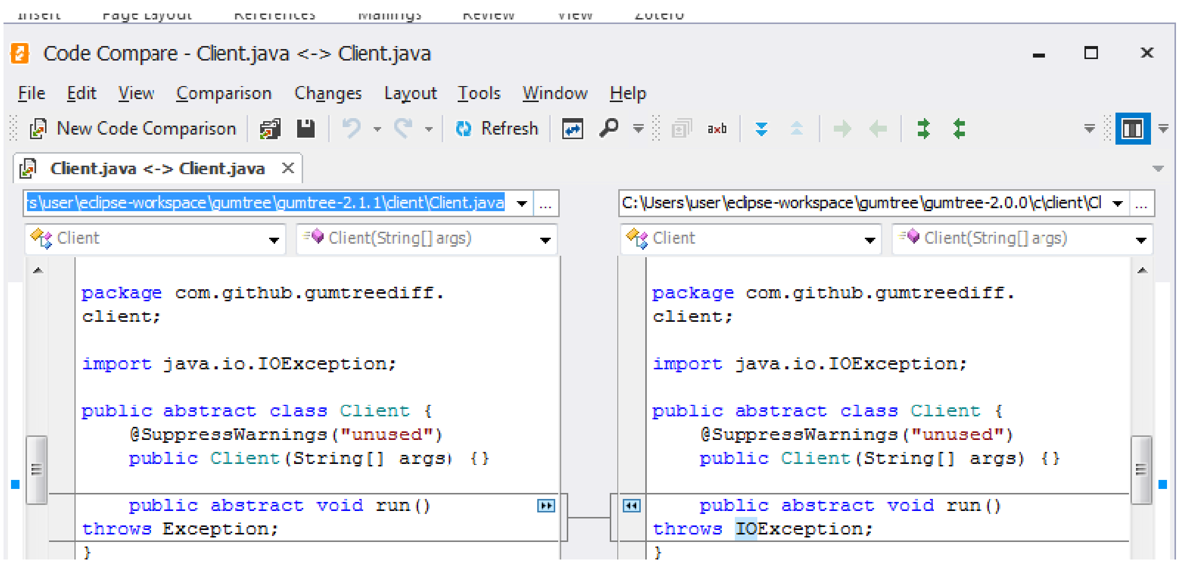
To compare the results, we extracted the changes by comparing previous and the latest versions .We found each file contains at least five changes.

Firstly: Two files were selected from two different versions(gumtree-2.0.0 , gumtree-2.1.1 randomly have the same name, it is the (client) and we did not find a difference between the two files by using the framework or CodeCompre Tool.



There is no difference between the two files

Figure 4.14: The result obtained by framework


Code Compare - Client.java <-> Client.java
File Edit View Comparison Changes Layout Tools Window Help
New Code Comparison
Refresh
Client.java <-> Client.java
s\user\eclipse-workspace\gumtree\gumtree-2.1.1\client\Client.java
C:\Users\user\eclipse-workspace\gumtree\gumtree-2.0.0\c\client\Cl
Client
Client(String[] args)

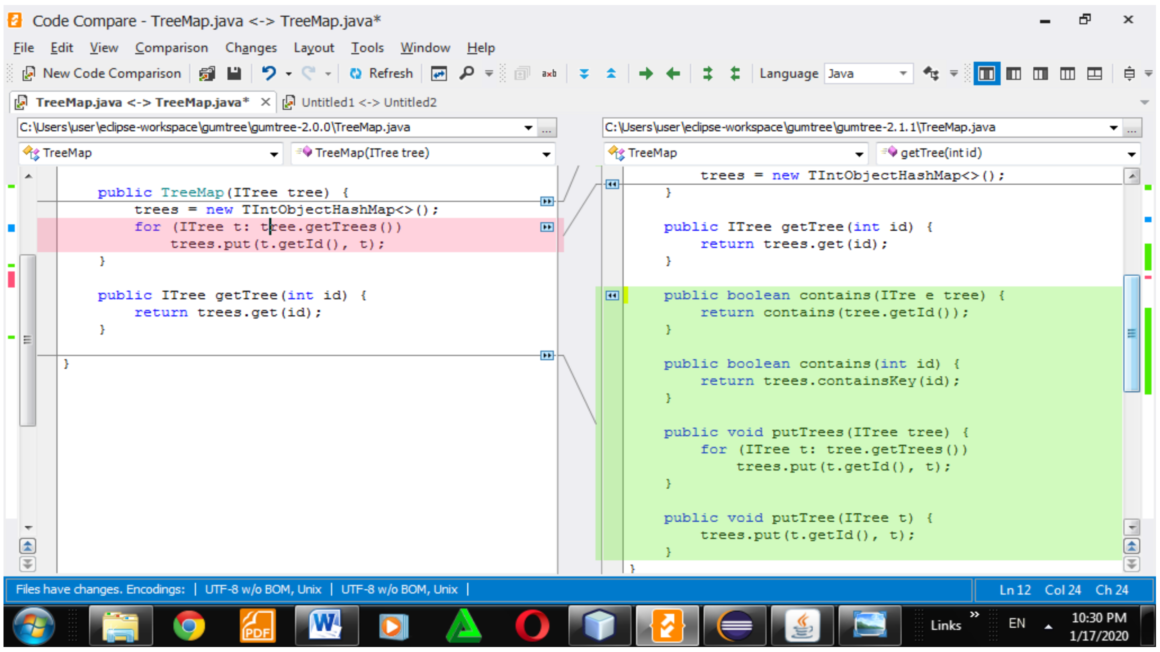
```
package com.github.gumtreediff.
client;

import java.io.IOException;

public abstract class Client {
    @SuppressWarnings("unused")
    public Client(String[] args) {}

    public abstract void run()
throws Exception;
}
```

```
package com.github.gumtreediff.
client;

import java.io.IOException;

public abstract class Client {
    @SuppressWarnings("unused")
    public Client(String[] args) {}

    public abstract void run()
throws IOException;
}
```

Figure 4.15: The result obtained by CodeCompare Tool

Secondly: A comparison between **two samples** selected randomly from the versions of Gum Tree in terms of the names of classes, functions and variables by CodeCompare tool and framework. Each sample is a pair of files one file from the previous version(gumtree-2.0.0) with another file from the latest version(gumtree-2.1.1).

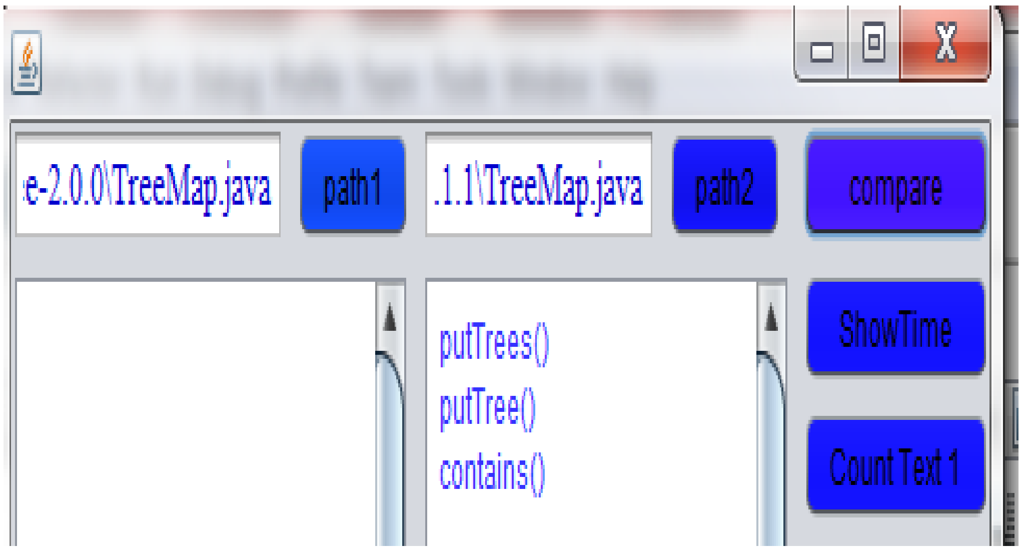
a **The first sample**: the **MapTree** file was chosen from two versions(gumtree-2.0.0, gumtree-2.1.1),Figure 4.16 shows the differences that were found in the CodeCompare tool, and Figure 4.17 shows the differences that were found in the framework, namely the functions(**putTrees(), putTree(), contains()**).


Code Compare - TreeMap.java <-> TreeMap.java*
File Edit View Comparison Changes Layout Tools Window Help
New Code Comparison
Refresh
Language Java
TreeMap.java <-> TreeMap.java*
Untitled1 <-> Untitled2
C:\Users\user\eclipse-workspace\gumtree\gumtree-2.0.0\TreeMap.java
TreeMap
TreeMap(ITree tree)
C:\Users\user\eclipse-workspace\gumtree\gumtree-2.1.1\TreeMap.java
TreeMap
getTree(int id)
Files have changes. Encodings: | UTF-8 w/o BOM, Unix | UTF-8 w/o BOM, Unix |
Ln 12 Col 24 Ch 24
Links
EN
10:30 PM
1/17/2020

```java
    public TreeMap(ITree tree) {
        trees = new TIntObjectHashMap<>();
        for (ITree t: tree.getTrees())
            trees.put(t.getId(), t);
    }

    public ITree getTree(int id) {
        return trees.get(id);
    }

}
```

```java
        trees = new TIntObjectHashMap<>();
    }

    public ITree getTree(int id) {
        return trees.get(id);
    }

    public boolean contains(ITre e tree) {
        return contains(tree.getId());
    }

    public boolean contains(int id) {
        return trees.containsKey(id);
    }

    public void putTrees(ITree tree) {
        for (ITree t: tree.getTrees())
            trees.put(t.getId(), t);
    }

    public void putTree(ITree t) {
        trees.put(t.getId(), t);
    }
}
```

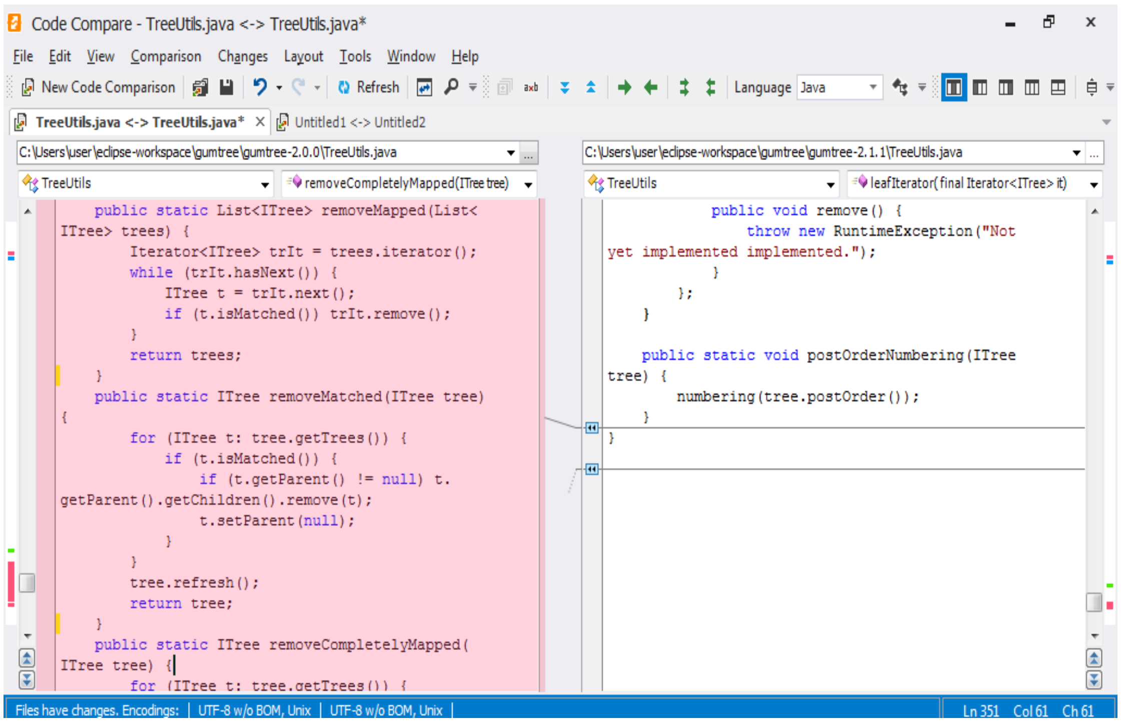
Figure 4.16: The result obtained by CodeCompare Tool



putTrees()
putTree()
contains()

Figure 4.17: The result obtained by Framework

b The second sample, the **Tree Utils** file was chosen from the two versions(gumtree-2.0.0, gumtree-2.1.1) , **Figure 4.18** shows the differences that were found with the Code Compare tool, namely the deletion of functions (**remove Mapped(), remove Completely Mapped(), remove Matched())** from version gumtree-2.0.0 . **Figure 4.19** shows the differences that were found in the framework, namely the function (**remove Mapped(), remove Completely Mapped(), remove Matched()**) and the variables (**t, trit**).


Code Compare - TreeUtils.java <-> TreeUtils.java*
File Edit View Comparison Changes Layout Tools Window Help
New Code Comparison
Refresh
Language Java
TreeUtils.java <-> TreeUtils.java*
Untitled1 <-> Untitled2
TreeUtils
removeCompletelyMapped(ITree tree)
TreeUtils
leafIterator(final Iterator<ITree> it)

C:\Users\user\eclipse-workspace\gumtree\gumtree-2.0.0\TreeUtils.java

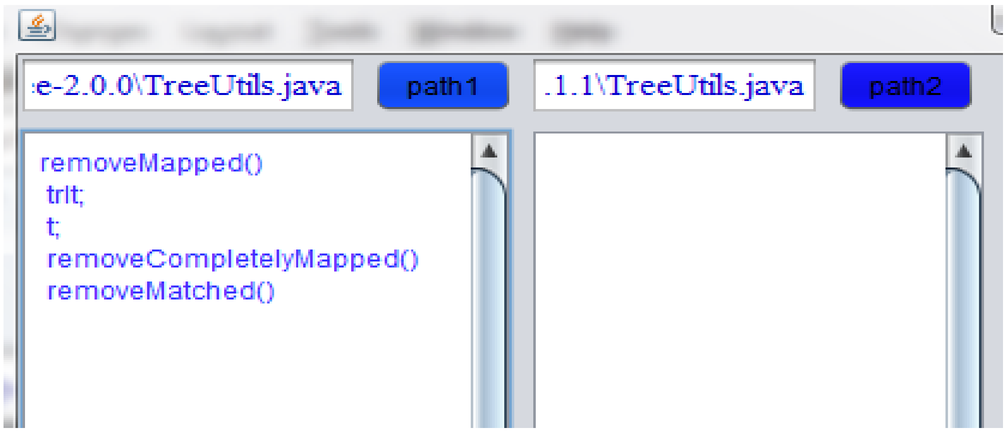
```
        public static List<ITree> removeMapped(List<
ITree> trees) {
            Iterator<ITree> trIt = trees.iterator();
            while (trIt.hasNext()) {
                ITree t = trIt.next();
                if (t.isMatched()) trIt.remove();
            }
            return trees;
        }
        public static ITree removeMatched(ITree tree)
{
            for (ITree t: tree.getTrees()) {
                if (t.isMatched()) {
                    if (t.getParent() != null) t.
getParent().getChildren().remove(t);
                    t.setParent(null);
                }
            }
            tree.refresh();
            return tree;
        }
        public static ITree removeCompletelyMapped(
ITree tree) {
            for (ITree t: tree.getTrees()) {
```

C:\Users\user\eclipse-workspace\gumtree\gumtree-2.1.1\TreeUtils.java

```
                public void remove() {
                    throw new RuntimeException("Not
yet implemented implemented.");
                }
            };
        }

        public static void postOrderNumbering(ITree
tree) {
            numbering(tree.postOrder());
        }
}
```

Files have changes. Encodings: | UTF-8 w/o BOM, Unix | UTF-8 w/o BOM, Unix | Ln 351 Col 61 Ch 61

Figure 4.18: The result obtained by CodeCompare tool



removeMapped()
trIt;
t;
removeCompletelyMapped()
removeMatched()

Figure 4.19: The result obtained by the framework

4.4 Case Study(akka actor):

We found in the previous section that the tool CodeCompre can only compare two files; to compare two versions we used the JAPICC tool.

As a case study, we used the akkaactor library because it is an open source library written in Java and for the availability of documents.

Table 4.9: Description of the datasets:

Name of Version	**Number of Version**	**Number of File**	**Size of Version**
akka-2.5.18	2.5.18	3040	36,2 MB
akka-2.5.19	2.5.19	3142	36, 9 MB
akka-2.5.20	2.5.20	3190	37, 4 MB
akka-2.5.22	2.5.22	3273	38,1 MB
Total		**18,825**	**221,9**

The following table shows the difference between the versions by using the framework and the tool; where the framework compares in terms of classes, functions and variables.

We find that there is a difference in the number of changes extracted from the versions by framework or tool because the tool compares only the classes and functions

Table 4.10: The result obtained by JAPICC and framework on dataset (versions)

Number of Version	Tool		Framework	
	deletions	additions	Deletions	Additions
Akka-2.5.18	7	12	6	89
Akka-2.5.19	117	39	145	16
Akka-2.5.20	53	4	82	40
Akka-2.5.22	136	62	57	133

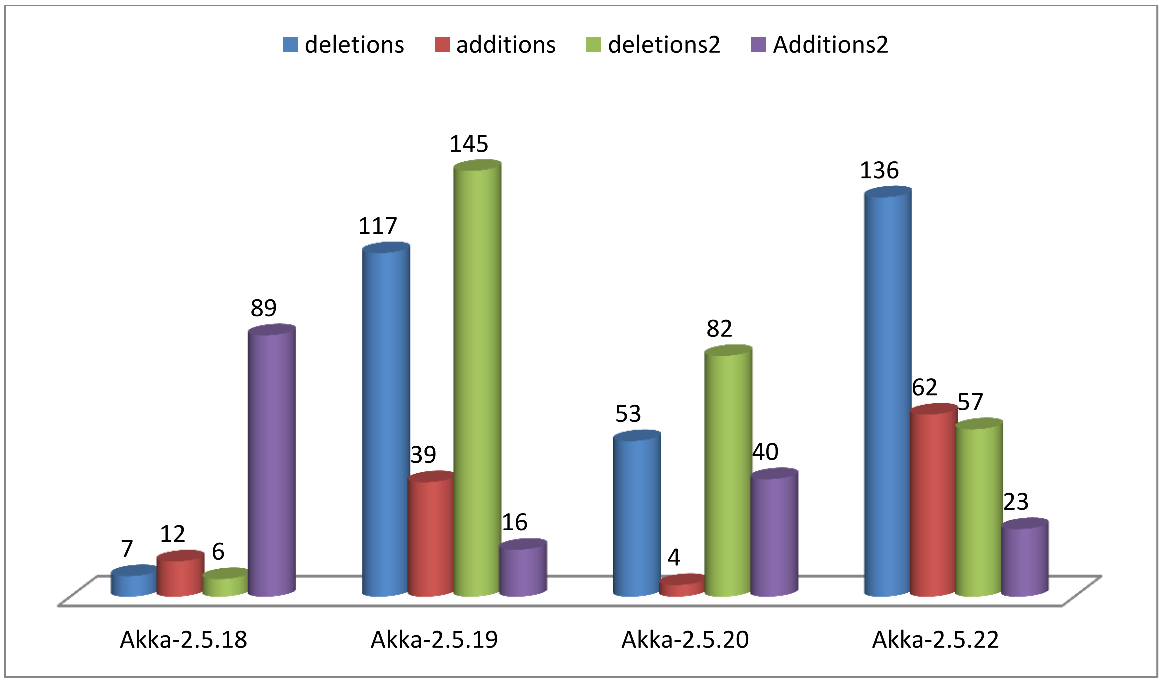


Figure 4.20: The result obtained by JAPICC and framework on dataset (versions).

Summary:

The framework performance validation section explains that it was able to find the difference between two versions in terms of classes, functions and variables, taking less than 114 Milli seconds in comparing files and less than 5870 Milli seconds in comparing versions.

Chapter Five

Discussion of the results

When comparing the performance of the framework with CodeCompare tool, it was found that the CodeCompare tool is better than framework it compares the whole files, even white spaces, but extracts a lot of changes inside the file. The framework compares only the functions and variables in the file

When comparing two versions, the difference between them was found in terms of classes, functions and variables, but it took more time when converting files to Abstract syntax trees, but sometimes it is difficult to compare it since the changes extracted from the version are 1,626.

The results show that a framework is better in finding the functions. The framework gave 48% in finding the added functions and 5% in finding the deleted functions. The tool showed 38% in finding the added functions and 9% in finding the deleted functions. The framework is distinct from the tool; it can find added and deleted variables.

The performance test when making comparison between two versions shows that the framework is better than the tool in finding functions, while for the classes showed similar percentages.

Conclusion:

This study provides proposed framework to find the difference between the versions by using the abstract syntax tree analysis algorithm in terms of classes, functions and variables.

We evaluated the performance of the framework by comparing it with the tools (Code Compre, JAPICC). The Code Compare tool is better because it compares the whole file, even white spaces, and the framework only compares the names of functions and variables.

When comparing two versions, the framework showed good results compared to the JAPICC tool. In addition, the framework can compare deleted and added variables, but tool cannot do so.

Future work:

- Adding the possibility to compare versions written in other languages such as C, php.
- Using of another algorithm to compare versions such as changedisiller algorithm.
- Develop the framework so that it can compare as many versions such as three or more versions.

.

REFERENCES

[1] A. Silva, G. Carneiro, A. Paula, M. Monteiro, F. Abreu, “Agility and Quality Attributes in Open Source Software Projects Release Practices,” in 2016 10th International Conference on the Quality of Information and Communications Technology (QUATIC), 2016, pp. 107–112.

[2] M. Mäntylä, B. Adams, F. Khomh, E. Engström and K. Petersen, “On rapid releases and software testing: a case study and a semi-systematic literature review,” Empir Software Eng, vol. 20, no. 5, pp. 1384–1425, Oct. 2015.

[3] A. Elhady, H. Abushama, “RACI Scrum Model For Controlling of Change User Requirement In Software Projects,” vol. 4, no. 1, p. 8, 2015.

[4] Y. Tseng, C. Lin, “Enhancing enterprise agility by deploying agile drivers, capabilities and providers,” Information Sciences, vol. 181, no. 17, pp. 3693–3708, Sep. 2011.

[5] P. Wang, “Toward developing agility evaluation of mass customization systems using 2-tuple linguistic computing” Expert Systems with Applications, 36(2), pp.3439-3447. 2009.

[6] P. Abrahamsson, O. Salo, J. Ronkainen and J. Warsta, “Agile Software Development Methods: Review and Analysis,” Sep. 2017.

[7] M. Tomanek, T. Klima, “Penetration Testing in Agile Software Development Projects,” IJCIS, vol. 5, no. 1, pp. 01–07, Mar. 2015.

[8] R. Naz, M. Khan, “Rapid Applications Development Techniques : A Critical Review,” International Journal of Software Engineering and Its Applications, vol. 9, no. 11, pp. 163–176.

[9] C. Gil, J. DIAZ, A. HOZ and R. MORALES, "Agile testing practices in software quality: State of the art review.” Journal of Theoretical and Applied Information Technology 92.1 (2016): 28.

[10] M. Viljan, N. Zabkar, “Measurement repository for Scrum-based software development process.” Conference on computer Engineering and Application (CEA, 08) Acapulco, Mexico. 2008.

[11] V. Vithana, “Scrum Requirements Engineering Practices and Challenges in Offshore Software Development,” International Journal of Computer Applications, vol. 116, pp. 0975 –8887, Apr. 2015.

[12] G. Canfora, L. Cerulo and M. Penta, “Identifying Changed Source Code Lines from Version Repositories,” in Fourth International Workshop on Mining Software Repositories (MSR’07:ICSE Workshops 2007), 2007, pp. 14–14.

[13] A. Sharma, M. Enhancing Q. Bali, “Comparative Study on Software Development Methods: Agile vs. Scrum.”, 2017.

[14] P. Gregory, L. Sharp, H. Deshpande, “The challenges that challenge: Engaging with agile practitioners’ concerns.” Information and Software Technology 77 (2016): 92-104.

[15] K. Schwaber, “SCRUM Development Process,” in Business Object Design and Implementation, 1997, pp. 117–134.

[16] K. Rao, A. Sastri, “Overcoming Testing Challenges in Project Life Cycle using Risk Based Validation Approach,” vol. 3, no. 3, p. 8, 2011.

[17] H. Dar, M. Ali and J. Shaikh, “INFLUENCE OF STATIC TESTING IN AGILE DEVELOPMENT: A CASE FROM PAKISTAN SOFTWARE MARKET,” p. 4, 2014.

[18] T. Monika, “Review on Structural Software Testing Coverage Approaches,” 2017.

[19] F. Musfira, M. Shakir and F. Munsifa, “Analysis of software testing challenges in Sri Lankan context,” Dec. 2016.

[20] I. Neamtiu, J. Foster and M. Hicks, “Understanding Source Code Evolution Using Abstract Syntax Tree Matching,” in Proceedings of the 2005 International Workshop on Mining Software Repositories, New York, NY, USA, 2005, pp. 1–5.

[21] M. Feist, E. Santos, I. Watts, A. Hindle, “Visualizing Project Evolution through Abstract Syntax Tree Analysis.“ Software Visualization (VISSOFT), 2016 IEEE Working Conference on. IEEE, 2016.

[22] J. Falleri, F. Morandat, X. Blanc, M. Martinez and M. Monperrus, “Fine-grained and accurate source code differencing,” presented at the Proceedings of the 29th ACM/IEEE international conference on Automated software engineering, 2014, pp. 313–324.

[23] N. Truong, P. Roe and P. Bancroft, “Static Analysis of Students’ Java Programs,” in Proceedings of the Sixth Australasian Conference on Computing Education - Volume 30, Darlinghurst, Australia, Australia, 2004, pp. 317–325.

[24] L. Lu, “Software testing techniques.” Institute for software research international Carnegie mellon university Pittsburgh, PA 15232.1-19 (2001): 19.

[25] M. Kumar, S. Manish, S. Singh, R. Dwivedi. “A Comparative Study of Black Box Testing and White Box Testing Techniques.” International Journal of Advance Research in Computer Science and Management Studies , 2015

[26] E. Collins, A. DiasNeto, F. Lucena, “Strategies for agile software testing automation: An industrial experience.” Computer Software and Applications Conference Workshops (COMPSACW), IEEE, 2012.

[27] P. Nidagundi, L. Novickis. “Introducing lean canvas model adaptation in the scrum software testing.”Procedia Computer Science 104, 2017, pp. 97-103.

[28] A. Bertolino, “Software testing research: Achievements, challenges, dreams.” Future of Software Engineering. IEEE Computer Society, 2007.

[29] S. Jan, S. Shah, Z. Johar, Y. Shah, F. Khan, “An Innovative Approach to Investigate Various Software Testing Techniques and Strategies,” p. 9

[30] M. Murugaiyan, “ WATEERFALLVs V-MODEL Vs AGILE: A COMPARATIVE STUDY ON SDLC” International Journal of Information Technology and Business Management , vol. 2, No. 1, June. 2012.

[31] R. Chauhan, I. Singh. “Latest research and development on Software Testing Techniques and Tools” International Journal of Current Engineering and Technology, 2014..

[32] A. Bansal, “Comparative Study of Software Testing Techniques,” Computer Science& Engg., Maharshi Dayanand University, India, vol. 3, no. 6, pp. 579–584, Jun. 2014.

[33] B. Hailpern, P. Santhanam, “Software debugging, testing, and verification,” IBM Systems Journal, vol. 41, no. 1, pp. 4–12, 2002.

[34] A. Zeller, “Program analysis: A hierarchy.” Proceedings of the ICSE Workshop on Dynamic Analysis (WODA 2003). 2003.

[35] M. Ernst, “Static and dynamic analysis: Synergy and duality.” WODA 2003: ICSE Workshop on Dynamic Analysis. 2003.

[36] N. Truong, P. Roe, P. Bancroft, “Static Analysis of Students’ Java Programs,” in Proceedings of the Sixth Australasian Conference on Computing Education - Volume 30, Darlinghurst, Australia, Australia, 2004, pp. 317–325.
. 2015.

[37] R. Löffler, B. Güldali, and S. Geisen, “Towards Model-based Acceptance Testing for Scrum,” Softwaretechnik-Trends, vol. 30, 2010.

[38] Y. Higo, A. Ohtani, and S. Kusumoto, “Generating Simpler AST Edit Scripts by Considering Copy-and-paste,” in Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering, Piscataway, NJ, USA, 2017, pp. 532–542.

[39] K. Pathak, A. Saha, “Review of agile software development methodologies.” International Journal of Advanced Research in Computer Science and Software Engineering 3.2 (2013).

[40] G. Matharu, H. Singh and P. Upadhyay, “Empirical study of agile software development methodologies: A comparative analysis.” ACM SIGSOFT Software Engineering Notes 40.1 (2015): 1-6.

[41] S. Balaji, M. Murugaiyan. “Waterfall vs. V-Model vs. Agile: A comparative study on SDLC.” International Journal of Information Technology and Business Management2.1 (2012): 26-30.

[42] A. Ovais, J. Markkula, M. Oivo, “Kanban in software development: A systematic literature review.” 2013 39th Euromicro conference on software engineering and advanced applications. IEEE, 2013..

[43] M. STOICA, M. MIRCEA, “Software Development: Agile vs. Traditional” Informatica Economicăvol. 17, no. 4, 2013.

[44] H. Flora, S. Chande, “A Systematic Study on Agile Software Development Methodologies and Practices,” (IJCSIT) International Journal of Computer Science and Information Technologies, vol. 5 (3), pp. 3626–3637, 2014.

[45] R. Pawlak, M. Monperrus, L. Seinturier, “Spoon: A library for implementing analyses and transformations of java source code.” Software: Practice and Experience 46.9 pp. 1155-1179, 2016.

[46] J. Jones, “Abstract syntax tree implementation idioms.” Proceedings of the 10th conference on pattern languages of programs (plop2003). 2003.

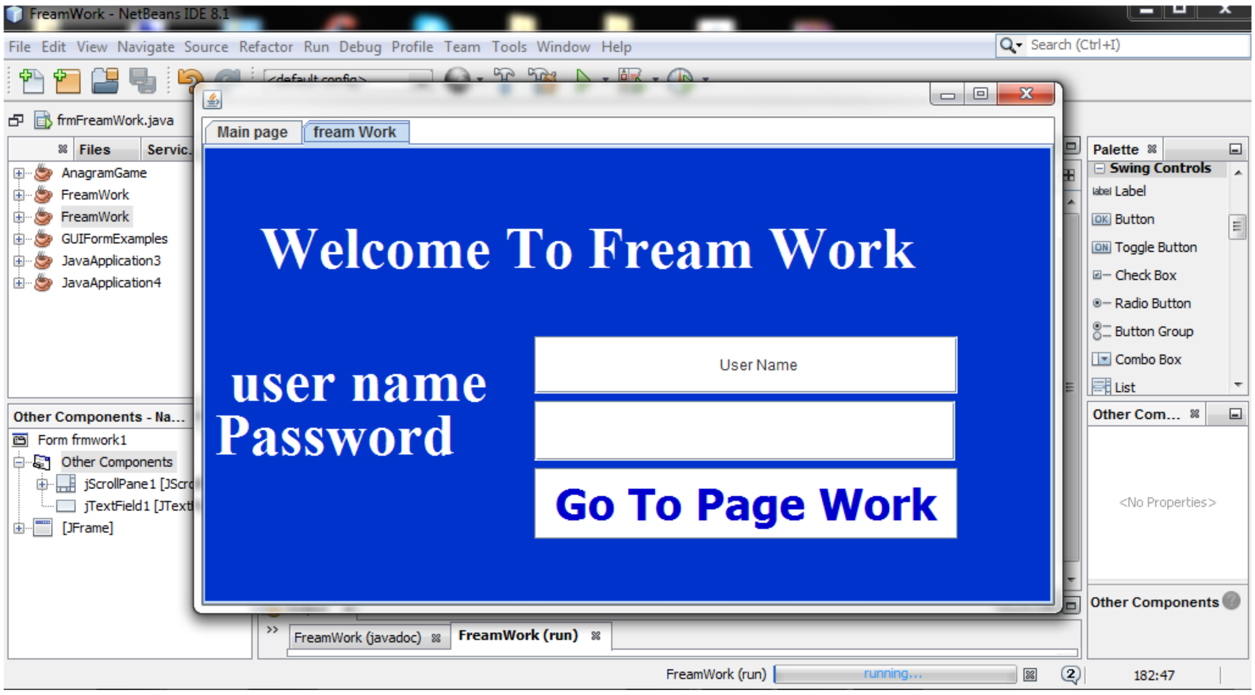
[47] A. Mukker, A. Mishra, and L. Singh, “Enhancing Quality in Scrum Software Projects,” vol. 3, no. 4, p. 7, 2014.

[48] https://lvc.github.io/japi-compliance-checker/

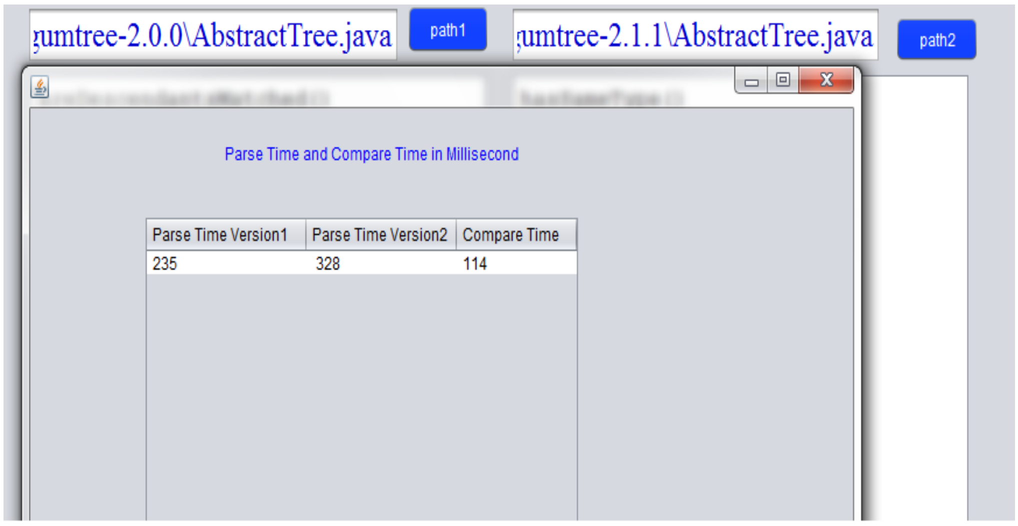
[49] https://www.devart.com/codecompare

[50 https://winmerge.org/

APPENDIX

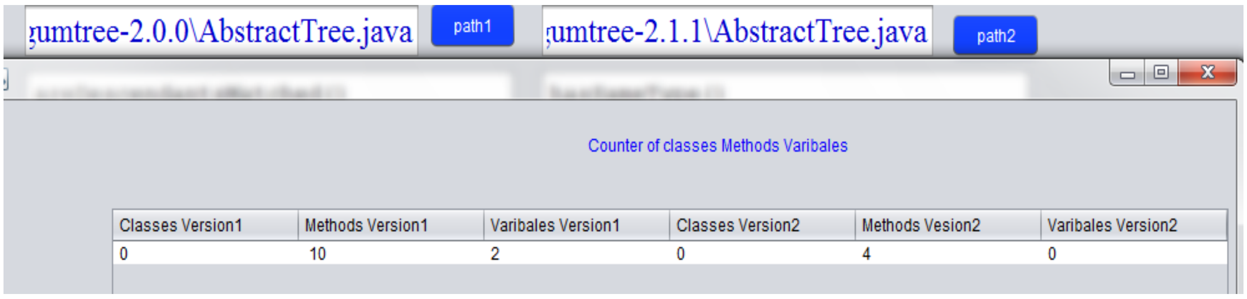


The main Screen of Framework


gumtree-2.0.0\AbstractTree.java
path1
gumtree-2.1.1\AbstractTree.java
path2
Parse Time and Compare Time in Millisecond

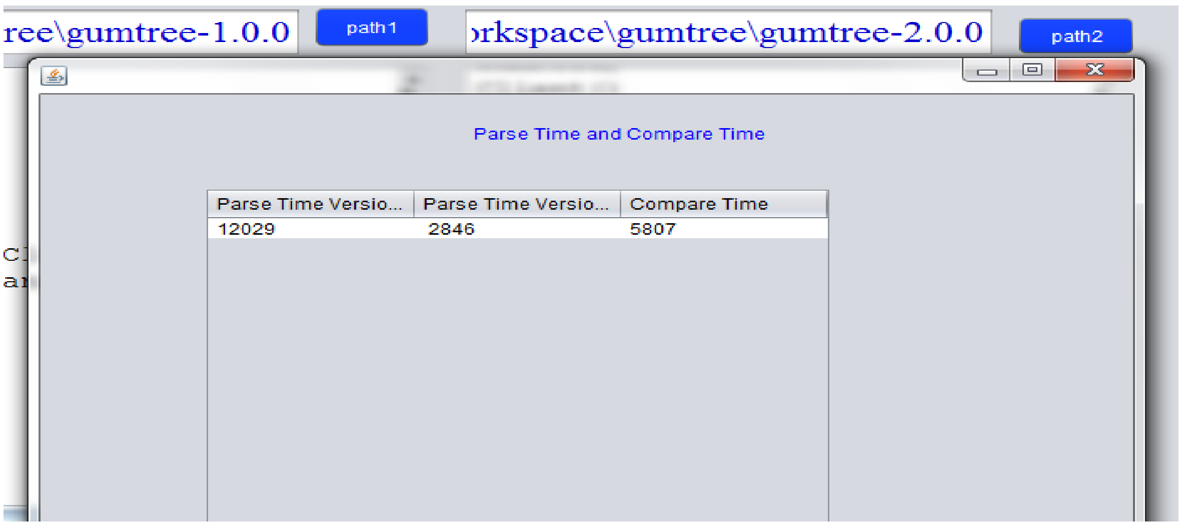
Parse Time Version1	Parse Time Version2	Compare Time
235	328	114

The result obtained by framework (parses time and compare time)


gumtree-2.0.0\AbstractTree.java
path1
gumtree-2.1.1\AbstractTree.java
path2
Counter of classes Methods Varibales

Classes Version1	Methods Version1	Varibales Version1	Classes Version2	Methods Vesion2	Varibales Version2
0	10	2	0	4	0

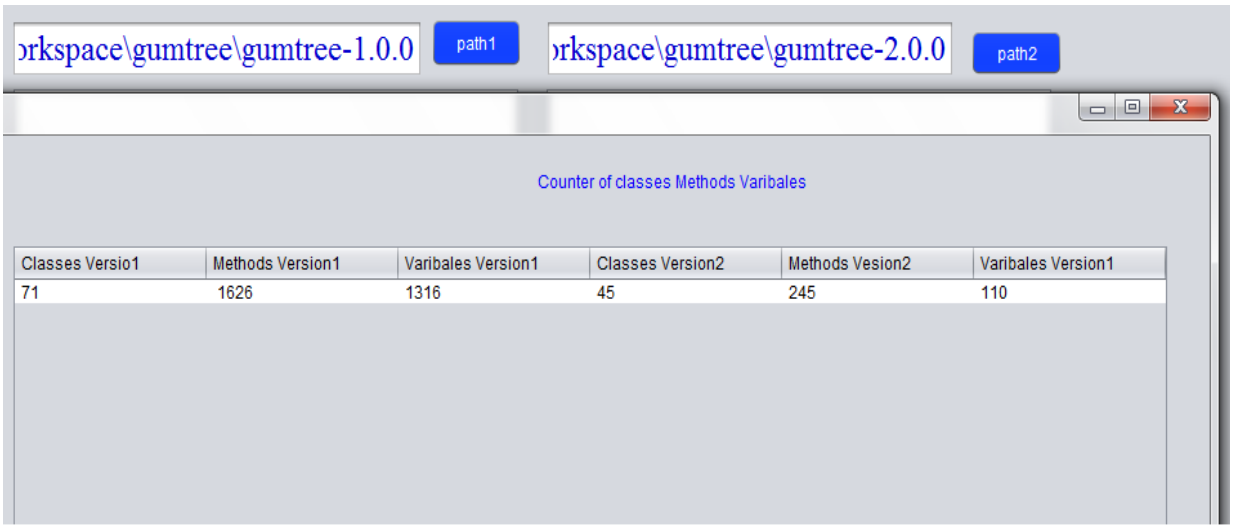
The result obtained by framework (counter of classes, methods and variables)



Parse Time and Compare Time

Parse Time Versio...	Parse Time Versio...	Compare Time
12029	2846	5807

The result obtained by framework (parses time and compare time)


›rkspace\gumtree\gumtree-1.0.0
path1
›rkspace\gumtree\gumtree-2.0.0
path2

Counter of classes Methods Varibales

Classes Versio1	Methods Version1	Varibales Version1	Classes Version2	Methods Vesion2	Varibales Version1
71	1626	1316	45	245	110

The result obtained by framework (parses time and compare time)