SCHOOL OF COMPUTER SCIENCE AND APPLICATIONS

A Project Report

on

Automated Evaluation of C,C++ and Linux Based Projects

Submitted in Partial fulfillment of the requirements for the award of the Degree of

Master of Computer Applications

Submitted by

Navaneetha V

R21DE134

Under the guidance of

Internal Guide

Prof. Shreetha Bhat

Assistant Professor

Reva University

External Guide

Bharath G

Lead

L&T Technology Services

Aug 2023

Rukmini Knowledge Park, Kattigenahalli, Yelahanka, Bengaluru-560064

www.reva.edu.in

SCHOOL OF COMPUTER SCIENCE AND APPLICATIONS

CERTIFICATE

Certified that the project work entitled **Automated Evaluation of C,C++ and Linux Based Projects** carried out under our guidance by **Navaneetha V**, **R21DE134**, a bonafide student of REVA University during the academic year 2021-22, is submitting the project report in partial fulfillment for the award of **Master of Computer Applications** during the academic year **2022–23**. The project report has been approved as it satisfies the academic requirements in respect of Project work prescribed for the said Degree.

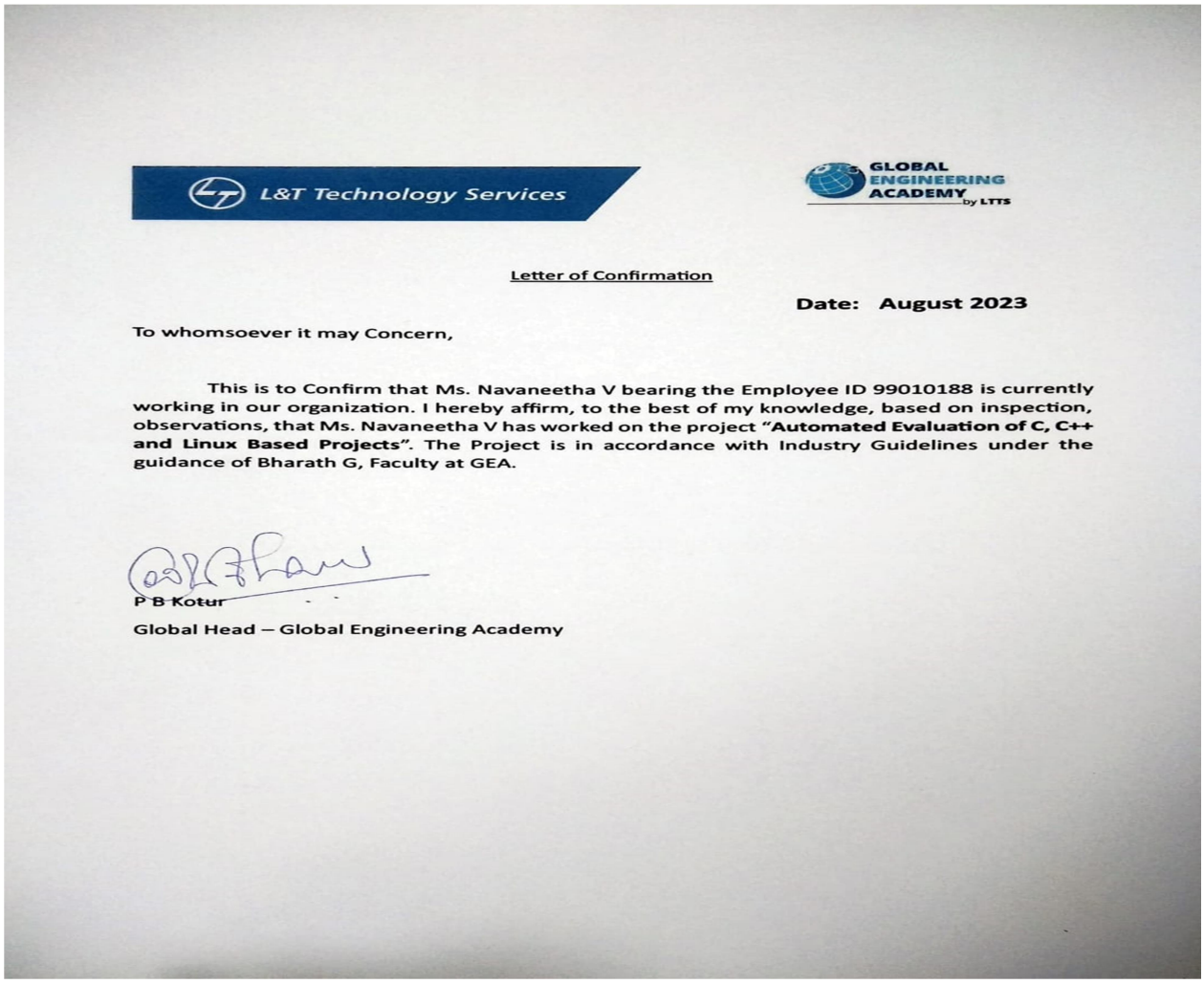
Signature with date	Signature with date	Signature with date
Prof. Shreetha Bhat **Internal Guide**	Bharath Digitally signed by Bharath Date: 2023.08.24 16:19:26 +05'30' **Bharath G** **External Guide**	**Prof. Vijayalakshmi .K** **Program Co-ordinator**

Signature with date

Dr. S. Senthil
Director

Name of the Examiner with affiliation	Signature with Date
1.	
2.	

Company Certificate



L&T Technology Services

GLOBAL ENGINEERING ACADEMY by LTTS

Letter of Confirmation

Date: August 2023

To whomsoever it may Concern,

This is to Confirm that Ms. Navaneetha V bearing the Employee ID 99010188 is currently working in our organization. I hereby affirm, to the best of my knowledge, based on inspection, observations, that Ms. Navaneetha V has worked on the project "**Automated Evaluation of C, C++ and Linux Based Projects**". The Project is in accordance with Industry Guidelines under the guidance of Bharath G, Faculty at GEA.

P B Kotur

Global Head – Global Engineering Academy

DECLARATION

I, Ms. Navaneetha V student of Master of Computer Applications belong in to School of Computer Science and Applications, REVA University, declare that this Project work entitled "Automated Evaluation of C,C++ and Linux Based Projects" is the result of the Project work done by me under the supervision of Dr / Prof. Shreetha Bhat , Assistant Professor at Reva university.

I am submitting this Project work in partial fulfillment of the requirements for the award of the degree of Master of Computer Applications by the REVA University, Bangalore during the academic year 2022-23

I further declare that this Project report or any part of it has not been submitted for award of any other Degree / Diploma of this University or any other University/ Institution.

Navaneetha V
(Signature of the candidate)

Signed by me on: 31/08/2023

Certified that this project work submitted by Navaneetha V has been carried out under our guidance and the declaration made by the candidate is true to the best of my knowledge.

Signature of Internal Guide
Date :……..

Signature of External Guide,
Date :……..

Signature of Director of School
Date :……..
Official Seal of the School

ACKNOWLEDGEMENT

I hereby acknowledge all those, under whose support and encouragement, I have been able to fulfill all my academic commitments successfully. In this regard, I take this opportunity to express my deep sense of gratitude and sincere thanks to School of Computer Science and Applications which has always been a tremendous source of guidance.

I express my sincere gratitude to **Dr. P. SHYAMA RAJU**, Honorable Chancellor, REVA University, Bengaluru for providing us the state-of-the-art facilities.

I am thankful to **Dr. M. DHANAMJAYA**, Vice Chancellor, **Dr. N.RAMESH**, Registrar and **Dr. P.VISWESWARA RAO**, Associate Dean, School of Applied Sciences, Allied Health Sciences and CSA, REVA University, Bengaluru for their support and encouragement.

I take this opportunity to express my heartfelt thanks to **Dr. S. SENTHIL**, Professor & Director, School of CSA, REVA University and my sincere thanks to **Dr. VINAYAKA MURTHY** Professor, Academic Vertical Head for PG Programmes, School of CSA, REVA University, whose encouragement and best wishes provided impetus for the Project Work carried out.

Also, my sincere gratitude goes to my Prof. Shreetha Bhat, Assistance Professor for the valuable suggestion and constant encouragement towards the completion of this project. I am deeply indebted to my Bharath G, Lead at L&T Technology Services for permitting to commence this Project and utilize all the facilities provided by the University.

Last, but not the least, I thank my parents for their incredible support and encouragement throughout.

ABSTRACT

Evaluating the vast array of programming tests employed in computer science education, businesses, and the software development landscape represents a highly intricate and pivotal undertaking. To address this challenge and provide a reliable, unbiased assessment of Linux, C, and C++ programs on a large scale, a practical and objective system has been devised. This system meticulously scrutinizes source code, compiles it, and executes it within a controlled environment to offer students, stakeholders and developers accurate and comprehensive feedback.

By analyzing the code's precision, effectiveness and adherence to coding standards, this system ensures a thorough evaluation. Leveraging a toolchain comprising an assortment of open-source tools such as Make, GCC, and G++ as well as cppcheck, Codechecker, Clang tidy, Sloccount and Gitinspector. The system identifies errors, security vulnerabilities, memory leakages and coding violations. Moreover, it enables the localization of physical source line codes within source files and provides insightful analysis of GitLab data.

This comprehensive approach empowers users and learners with enhanced analysis, expeditious feedback, and the production of higher-quality code. Simultaneously, it reduces the burden on evaluators and fosters a more objective evaluation process. Through the automation of compilation, testing, and code analysis tasks, this system significantly streamlines the evaluation workflow, ultimately benefiting the entire programming education and software development community.”

TABLE OF CONTENTS

CHAPTERS **PAGE NO**

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION TO PROJECT

The Automated evaluation of C, C++ and Linux Based Projects main purpose is to automate and analysis the of user repositories and generate the reports. The generated reports are push back to the user repository of GitLab This project is developed using the static analysis and dynamic analysis tools called Cppcheck, Sloccount, CodeChecker, Gitinspector, Sloccount and Valgrind with the help of GitLab and visual studio code and other automatic build tool called Make .

The advantages of automating the evaluation process of c, c++ and linux based projects are manifold. Firstly, it saves time and effort, allows educators and professionals to focus on higher-level tasks. Secondly, it reduces the potential for human error, ensuring consistent and accurate evaluation of all submissions. Lastly, it provides users with rapid feedback and enabling them to improve their code quality and learn from their mistakes is more efficient.

The primary objective of this project is to automatically analyze user repositories. Once users publish their repositories on GitLab, they can leverage analysis toolchains such as cppcheck, code checker, sloccount, Gitinspector, Make, and more to clone and assess their repositories. The resulting reports are then posted back to the respective user repositories on GitLab.

For instance, in an educational context, students create their repositories and upload them to GitLab. Faculty members can clone these repositories and employ analytical tools to evaluate them.

1.1.1 STATEMENT OF THE PROBLEM

It is crucial and difficult in educational and professional settings or in organization to test individual's skills in programming languages such as C and C++ as well as their comprehension of Linux-based systems. Manual project evaluation in these languages can be time-consuming, uneven and prone to human mistake.

1.1.2 BRIEF DESCRIPTION OF THE PROJECT

Automated Evaluation of C, C++, and Linux-based Projects revolutionizes the assessment of user code submissions by providing prompt and automated feedback. This streamlined system aims to simplify the evaluation process for projects created with C, C++, and Linux programming languages. By reducing the time and effort required to generate reports and deliver timely responses, this method enhances efficiency.

The primary objective of this project is to automatically analyze user repositories. Once users publish their repositories on GitLab, they can leverage analytical toolchains such as cppcheck, code checker, sloccount, Gitinspector, Make, and more to clone and assess their repositories. The resulting reports are then posted back to the respective user repositories on GitLab.

For instance, in an educational context, students create their repositories and upload them to GitLab. Faculty members can clone these repositories and employ analytical tools to evaluate them.

The generated reports are subsequently returned to the GitLab student repositories.

The project focuses on evaluating user projects and assessment submissions based on C and C++ programming languages, utilizing Python and shell scripting to automate the process and provide swift feedback to users.

The evaluation and feedback process plays a vital role in both educational and industrial settings. However, the evaluation of projects, especially involving C, C++ and Linux projects evaluation is time - consuming and arduous. These languages are widely used in computer science, engineering, and technology, requiring expertise for accurate assessment. Manual evaluation is also difficult to find human error, leading to inconsistencies in grading and immediate feedback to the users .

To address these challenges, the proposed system automates the evaluation of C, C++, and Linux-based projects. By leveraging analytical toolchains like cppcheck, code checker, sloccount, Gitinspector, and Make, the system can clone and analyze user repositories, providing comprehensive reports on code quality. These reports are then posted back to the user repository on GitLab, ensuring prompt feedback and minimizing the need for manual evaluation.

The advantages of automating the evaluation process of c,c++ and linux based projects are manifold. Firstly, it saves time and effort, allows educators and professionals to focus on higher-

level tasks. Secondly, it reduces the potential for human error, ensuring consistent and accurate evaluation of all submissions. Lastly, it provides users with rapid feedback and enabling them to improve their code quality and learn from their mistakes is more efficient.

Overall, the Automated Evaluation of C, C++ and Linux-based Projects represents a significant advancement in programming education and industrial evaluation. By automating the evaluation process of c, c++ and linux based projects, this system enhances efficiency, accuracy and provides valuable feedback to end usersand makes easy to evaluate and analyze users assessments

1.1.3 SOFTWARE AND HARDWARE SPECIFICATION

Software Specification

The software specification provide details about the software used to develop the particular Project.

Operating System : Linux (Ubuntu 20/22)

Front End : Visual Studio Code

Back End :GitLab

Language :Shell Script.

Hardware Specification

For developing the application, the following are the minimum Hardware Requirements:

Desktop/Laptop :Desktop/Laptop

Processor :Processor: Intel i3 and above

RAM :RAM: 4 GB or 8 GB

1.2 FUNCTIONALAND NON-FUNCTIONAL REQUIREMENT

Functional Requirements

The functional Requirements specify the operations, features, capabilities and interactions of "Automated Evaluation of C , C++ and Linux Based System" which helps to fulfill the demands of users and stakeholders. The functional requirements define what should accomplish in terms of behaviour and tasks. These requirements aid software developers, testers and other project participants as they design, construct and validate this Automated Evaluation of C, C++ and Linux Based Projects.

The basic characters of functional requirements includes is precise description, traceability, consistent, verifiable, scope, measurable.

List of Functional Requirements of Automated Evaluation of C, C++ and Linux Based System Are as Follow :-

Notification: --Whenever the any action take in the GitLab that is Notified by the all the users of the Automated Evaluation of C,C++ and Linux Based Projects. For example if the user push his/her repos into the GitLab the teacher or evaluator will get to know by email notification.

File upload:- In Automated Evaluation of C,C++ and Linux Based projects the end users should push the his/her repositories then only the Automated Evaluation process can be produce and consolidated report should be pushed into user repositories.

Report Generation.:- once the Analysis is done the specified reports have to be Generated with expected manner. For example the consolidated report for the respective users has to be generated.

Non Functional Requirements

Non-functional requirements describes the traits and properties of "Automated Evaluation of C,C++ and Linux Based Projects" Where it should have in addition to its main functionality. Non-functional requirements are opposed to functional requirements that focus on how the programme should work and the overall user experience of the above project.

Non-functional requirements cover a variety of features of the project such as performance, security, usability, scalability and maintainability of the "Automated Evaluation of C,C++ and Linux Based Projects". These specifications are essentially for ensuring that the project reaches the necessary levels of performance, reliability and user satisfaction.

The Non Functional Requirements provides guarantee that the system satisfies the larger expectations of stakeholders and end users.

List of Functional Requirements of Automated Evaluation of C, C++ and Linux Based Projects are as follow :-

Performance:- The "Automated Evaluation of C, C++ and Linux Based Projects" project run should be completed within 5 minutes for the 10-50 users if it is large in case it should finish within 10 minutes with including all the functionalities.

Security:- The unauthorized users are not allowed to run the project because when we run the project it will ask for the gitlab credentials.

Scalability:- Automated Evaluation of C, C++ and Linux Based Projects is more scalable this project runs for fewer users as well as large amount of users.

Maintainability:-How many times project gets run that many times the reports are maintained.

1.3COMPANY PROFILE

L&T Technology Services

EngineeringTheChange

L&T Technology Services (LTTS) is an Indian multinational technology company that provides engineering research and development (ER&D) services. L&T Technology Services Limited (LTTS) is a publicly listed subsidiary of Larsen & Toubro Limited providing Engineering and R&D (ER&D) services. LTTS offers consultancy, design, development and testing services across the product and process development life cycle. L&T Technology developed a set of personal safety gear that is based on cloud-enabled technologies. L&T Technology Services (LTTS) was founded in 2006 as L&T Integrated Engineering Services. headquartered in Vadodara.

L&T Technologies Locations :-India, Europe, USA,Australia,China..Etc

LTTS' expertise in engineering design, product development, smart manufacturing, and digitalization touches every area of human lives, Specialize in disruptive technology spaces such as 5G, Artificial Intelligence, Collaborative Robots, Digital Factory, and Autonomous Transport. L&T Technology Services announced that "Smart World & Communication" business segment, which has interests in communication networks, cybersecurity and smart spaces.

VISION:-Engineering a sustainable tomorrow through technology and innovation

MISION:- Be the engineering partner of choice by enabling innovation with world-class technologies, processes, and people – delivering inclusive growth for all stakeholders.

Engineering research and development services on:-

- Product Engineering:-Digital Engineering, embedded engineering, mechanical design, software Engineering, testing & validation
- Manufacturing Engineering:-Smart manufacturing, manufacturing& planning, Manufacturing Execution
- Operations Engineering:-Connected product support,supply chain engineering,plant engineering
- Digital Engineering And Consulting:-Cybersecure,immersive Experiences,industry 4.0,product consulting,sustainability Engineering,sustainability smart world 5G.etc

APPLICATIONS:-

- Healthcare
- Transportation
- Media and entertainment
- Oil and Gas
- Plant engineering
- Semiconductors
- Software Products
- Telecommunication
- Consumer Electronics

TECHNOLOGIES :-

- Cloud Computing:-
- Artificial Intelligency
- 5G
- Internet of Things

Location:- L&T Technology Services Ltd, L3 Building Ground Floor, Manyata Embassy Business Park, Nagawara Hobli, Bengaluru-560049.

Team

A group of people who perform interdependent tasks to work toward accomplishing a common mission or specific objective. **Global Engineering Academy**.

CHAPTER 2

LITERATURE SURVEY

The process of Evaluating C, C++ and Linux-based projects is time-consuming and containing more faults,errors. especially when it comes to analyzing code quality and assuring platform compatibility. The automation of this procedure has the potential to greatly increase the efficiency and accuracy of assessments for the end users.

A review of the literature provides on the subject to finds various studies and research articles that address the issue of automating the assessment of C, C++ and Linux-based projects.

- "Automated Code Review: The Future Code Review" by Ahmad Zaki and Haider Abbas. The usage of automated code review tools in the software development lifecycle provides the overview of finding better code quality, lower costs, and higher productivity. A review of a few of the well-known automated code review tools on the market is also provided by the writer.
- By Saurabh Gupta and Parag Dave, "Static Analysis Tools for C and C++ Code." This article describes the static analysis tools for C and C++ code that gives more advantage for locating bugs, security flaws, and other code quality concerns. Additionally, the authors compare a few of the well-known static analysis programs for C and C++ code.
- "Automated Build and Test System for C/C++ Projects" was written by Lars Asger Lund-Nielsen and Peter Hoegh Mikkelsen. The automated build and test system for C/C++ projects is presented in this article. It allows for continuous integration and testing of code modifications. The authors go over the advantages of the approach, such as shorter testing times and better code quality.
- "Automated Porting of C Code to Linux" by Kirill Timofeev and Anton Burtsev. The automated method shown in this work makes it easier and more reliable to convert C code to Linux while also requiring less manual labor. The authors go over the advantages of the tool, such as faster porting and better code quality.
- “Automated Assessments of C, C++ projects Based on the static analysis “ by Liron Ginzburg and Yair Levy. This paper describes a system that uses static analysis to

automatically assess c and C++ projects. The system can detect coding errors, coding style violations, and other issues.

- "A framework for Automated Assessment of C, C++ code quality" by Denis Babenko and Olga Tarasenko. This paper describes a framework for automated assessment of C, C++ code quality. This uses static analysis to detect coding errors, violations, and other issues. The present experimental results show the effectiveness of the framework.

According to a review of the literature, automating the assessment process for C, C++, and Linux-based projects greatly increases evaluation, efficiency, accuracy, reliability, and Security.

The literature on creating code quality rubrics and their effectiveness in the learning process of introductory programmers is limited. However, there are several studies on rubric-based assessment in computer science education and the use of rubrics for evaluating software quality. Some studies explored the effectiveness of rubrics in improving students' programming skills, while others have investigated the reliability and validity of rubrics for grading programming assignments. Additionally, the research provides feedback that is an essential component of effective learning and rubrics can provide a structured and consistent way of providing feedback. Therefore, further research is needed to refine the method of creating a code quality rubric and evaluate its usefulness in the learning process for the programmers.

In research article the authors address the issue of teaching programming style and coding conventions in programming courses. They propose a set of common guidelines that can be automatically measured using a program style evaluation tool called Style++. The tool was implemented in programming courses to assist students in developing good basic programming routines that are necessary for advanced programming tasks. The authors emphasize the importance of programming style and the positive impact that incorporating it into courses can have on the quality of coursework. However, they caution that style evaluation tools do not guarantee in-depth learning and suggest that teachers should motivate students and provide guidance and assistance to help them learn programming styles effectively. The authors recommend recording students' usage of the evaluation tool to study the educational usage and impact of automatic assessment tools in programming courses.

Automated assessment tools can provide numerous benefits for programming education, including faster, more consistent, objective, and tireless marking and feedback. However, it is important to note that not all aspects of programming can be assessed automatically, and semiautomatic approaches may be necessary. The teacher must carefully consider how to incorporate automated assessment tools into education and ensure that the assessment is educationally sound and not solely inspired by technology. Furthermore, the teacher should prepare the assignment and assessment strategies to encourage students to learn and work on their programs on all desired aspects. Unfortunately, many assessment tools are developed for local use and may not be available for wider use or adoption by other universities. Developing interoperable tool approaches could enable teachers from different universities to share knowledge of good assignments and educational approaches in automated assessment. The Hacker Rank and CodeRunner are the two platforms where it automatically evaluate C/C++ projects.

CHAPTER 3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

The current approach for assessing C, C++, and Linux-based projects combines manual and automated procedures with a number of different tools and frameworks.Numerous platforms and technologies are employed by the evaluator to automatically evaluate user-submitted code. Projects are assessed individually with several reports, without accuracy and with a delay in feedback.

Some of the tools listed here already exist.

1. Hacker Rank:- It is an Online site, that offers developers coding quizzes, programming competitions, and assessments of their talents. The hacker Rank provides Automatic evaluation of code submission based on the predefined test cases for the developer and students.

2.CodeRunner:- It is a multi-language code execution tool that offers programmers a simple and practical method to execute code snippets and programs written in a number of different programming languages. It is a helpful tool for developers who wish to quickly test and debug their code because of its portability and selection of customization possibilities.

3.2 LIMITATIONS OF EXISTING SYSTEM

- Manual Evaluation of a large number of users requires time and manpower.
- Standalone analyzer tool with multiple reports.
- Use Analysis tools without knowledge.

3.3 PROPOSED SYSTEM

The Automatic Evaluation of C, C++, and Linux-based Projects represents a cutting-edge solution that automatically evaluates a multitude of user code submissions and provides prompt responses. This system, named "Streamlining Evaluation of C, C++, and Linux-based Projects through Automation," aims to optimize the evaluation process by automating it, significantly reducing the time and effort required to generate reports and deliver timely feedback.

The core of this project lies in its toolchain, which encompasses a range of evaluation and analysis tools such as Cppcheck, code checker, sloccount, Gitinspector, and more. These tools work in harmony to automatically assess C, C++, and Linux-based projects. By leveraging this toolchain, the system ensures that evaluations are carried out efficiently and accurately.

An essential aspect of this system is integration with various open-source tools for allowing to steamless and comprehensive evaluation. Once the evaluation process is complete, the system generates detailed reports, which are easily accessible to end users. This integration enhances the accessibility and usefulness of the evaluation results, empowering users with valuable feedback.

3.4 ADVANTAGES OF PROPOSED SYSTEM

- Reduce the manual work
- Analysis using multiple tools
- Instant feedback to learners
- Cost effective due to open source tools
- Standalone Reports
- Includes both static and dynamic analysis tools

3.5 FEASIBILITY STUDY

-TECHNICAL FEASIBILITY

The technical feasibility of project analysis depends on the availability and accessibility of suitable tools and technologies for each stage of the project lifecycle.
Set up your development environment: You would need to install VS code and analysis tools Cppchecker, sloccount, codechecker, Gitinspector, git, make. Create a new analysis: Analyze the repositories from the user uploaded into GitLab.

-ECONOMICAL FEASIBILITY

The economic feasibility of "Automated Evaluation of C,C++ and Linux based System is refers to whether the benefits of implementing the proposed system can outweigh the costs associated with its implementation.

All the software which are used to for this project are opensource and free to use, Hardware used are cost.

Some of the economic benefits of "Automated Evaluation of C,C++ and Linux Based Projects"include:

Improved Evaluation Process: A "Automated Evaluation of C,C++ and Linux Based Projects" system can help optimize Evaluation Process, ensuring that efficiently evaluation to minimize Manual work and reduced the time.

Cost savings: Not required to buy or premium versions of any kind of software for this Project so it save cost as all are open source .

Improved project outcomes: A "Automated Evaluation of C,C++ and Linux Based Projects" helps to identify potential risks, issues, bugs, code violation etc. Having this base project can be developed the grading system.

Better decision making: A "Automated Evaluation of C, C++ and Linux Based Projects" allowing Evaluator to take corrective action and improve coding abilities or knowledge in users. Make decisions to provide the gradings for the users.

-OPERATIONAL FEASIBILITY

The Proposed System of "Automated Evaluation of C, C++ and Linux Based Projects" integrated into the organization's daily operations. Where in which it is used for daily evaluation of different types of Candidate. It is Evaluated based on Batch wise with different categories.

Some of the following are the operational feasibilities are supported for "Automated Evaluation of C, C++ and Linux Based Projects" are:-

Technical requirements: The technical requirements of the "Automated Evaluation of C, C++ and Linux Based Projects" is compatible with the organization's existing IT infrastructure and resources.

User adoption: The "Automated Evaluation of C, C++ and Linux Based Projects" is user friendly he or she can easily understand all the functionalities if they know basics of shell script or python script.

Integration with existing processes: The proposed System to the "Automated Evaluation of C, C++ and Linux Based Projects" can be easily integrated.

Support from stakeholders: This project supports for all end users like "computer education, Business , Software developers.

Cost-benefit analysis: The costs associated with "Automated Evaluation of C, C++ and Linux Based Projects" is weighed against the potential benefits.as does not required additional cost.

CHAPTER 4

SYSTEM DESIGN AND DEVELOPMENT

4.1 HIGH LEVEL DESIGN (ARCHITECTURAL)

Architecture diagramming is the process of creating visual representations of software system components. In a software system, the term architecture refers to various functions, their implementations, and their interactions with each other. As software is inherently abstract, architecture diagrams visually illustrate the various data movements within the system. They also highlight how the software interacts with the environment around it.

The following figure 1 represents architectural diagram for the "Automated Evaluation of C, C++ and Linux Based Projects"

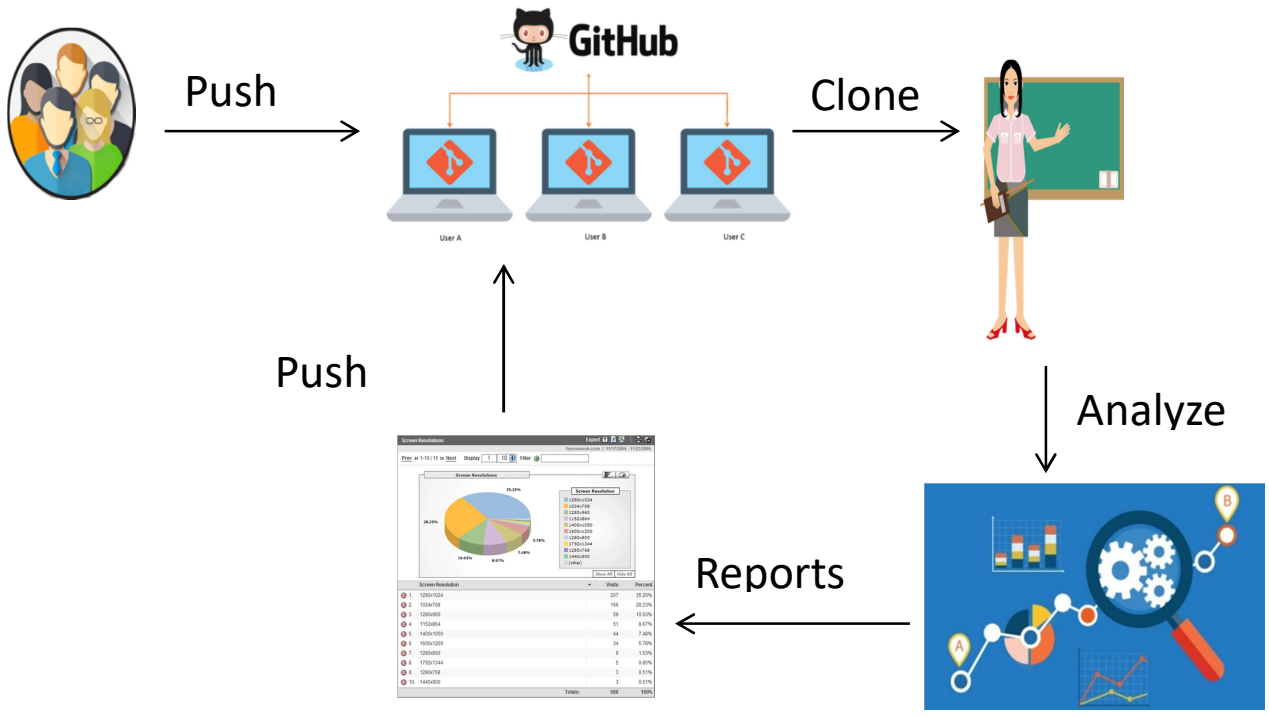


Figure 1 :-Architectural diagram for "Automated Evaluation of C, C++ and Linux Based

4.2 LOW LEVEL DESIGN

The Low level Diagram helps to design or develop the code. The "Automated Evaluation of C,C++ and Linux Based Projects" has mainly 4 classes which has the following entities. It provides us with the structure and behavior of class as different entities have different character sets. From this design, it is easy to write down logic and henceforth the actual code for it.

The following Figure 2 represents the Low level diagram for "Automated Evaluation of C, C++ and Linux Based Projects"

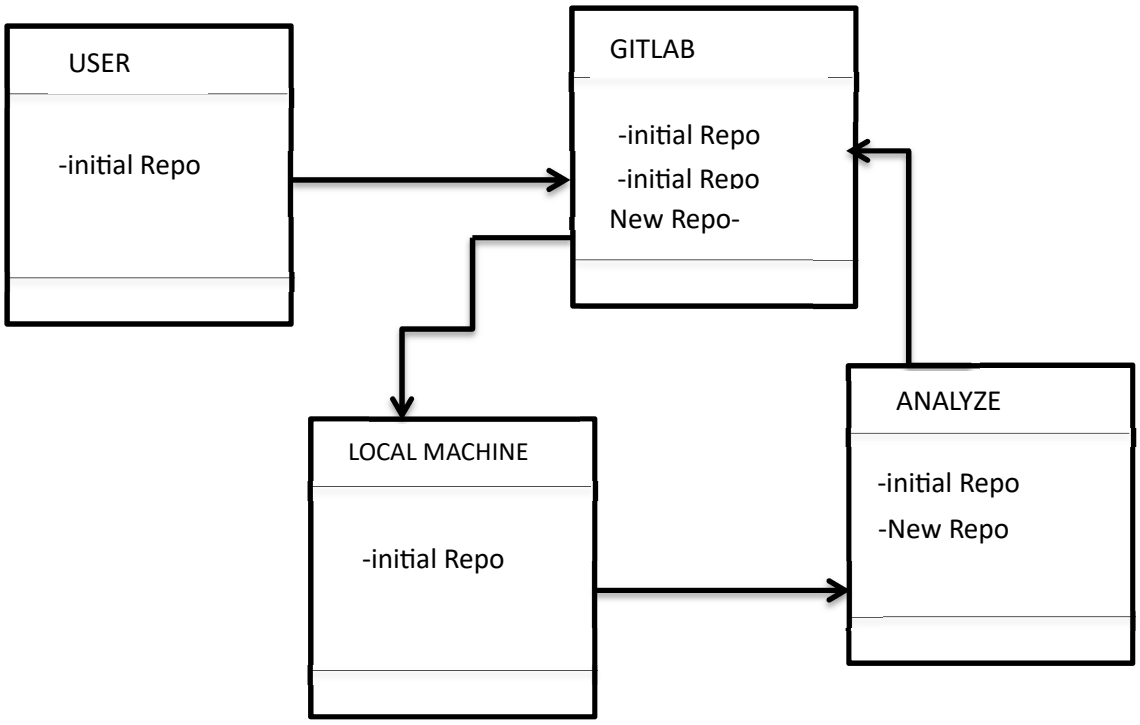


Figure 2 : Low Level Diagram for "Automated Evaluation of C,C++ and Linux Based Projects"

4.3 ENTITY-RELATIONSHIP DIAGRAM

The “Automated Evaluation of C, C++ and Linux Based Projects” has mainly four Tables first table is used to store the user details with the attributes called user_name, Email, Repo_link in the form Excel and the second table is gitlab it has attributes called Email, repository and Repo link where User table and GitLab has N to 1 Relationship email of User table is consider has Primary key that is Referred in GitLab has foreign key.

Third Table Represents local machine Table where it create the folder and subfolder located once clone is done. It has two attributes called folders and files. The Fourth table is Analysis Table where it has different types of analysis tools with attributes called cppcheck, codechecker, sloccount, valgrind, Gitinspector. These different attributes has different sub attributes all are considered has consolidated reports the tool cppcheck has attributes called file_name ,Id, severity, message , line, column. Codechecker has following attributes called file_name, message, severity, review status.the Gitinspector has authors,commits, insertions,deletions, percentage of changes. And the content analysis has name,email,repo_link,Clone Status,build_status,Sloccount,cppcheck,unit_test,Doxgen,github_status,
Badges,md_files,pdf_files,images files, excel files.

ER-diagram

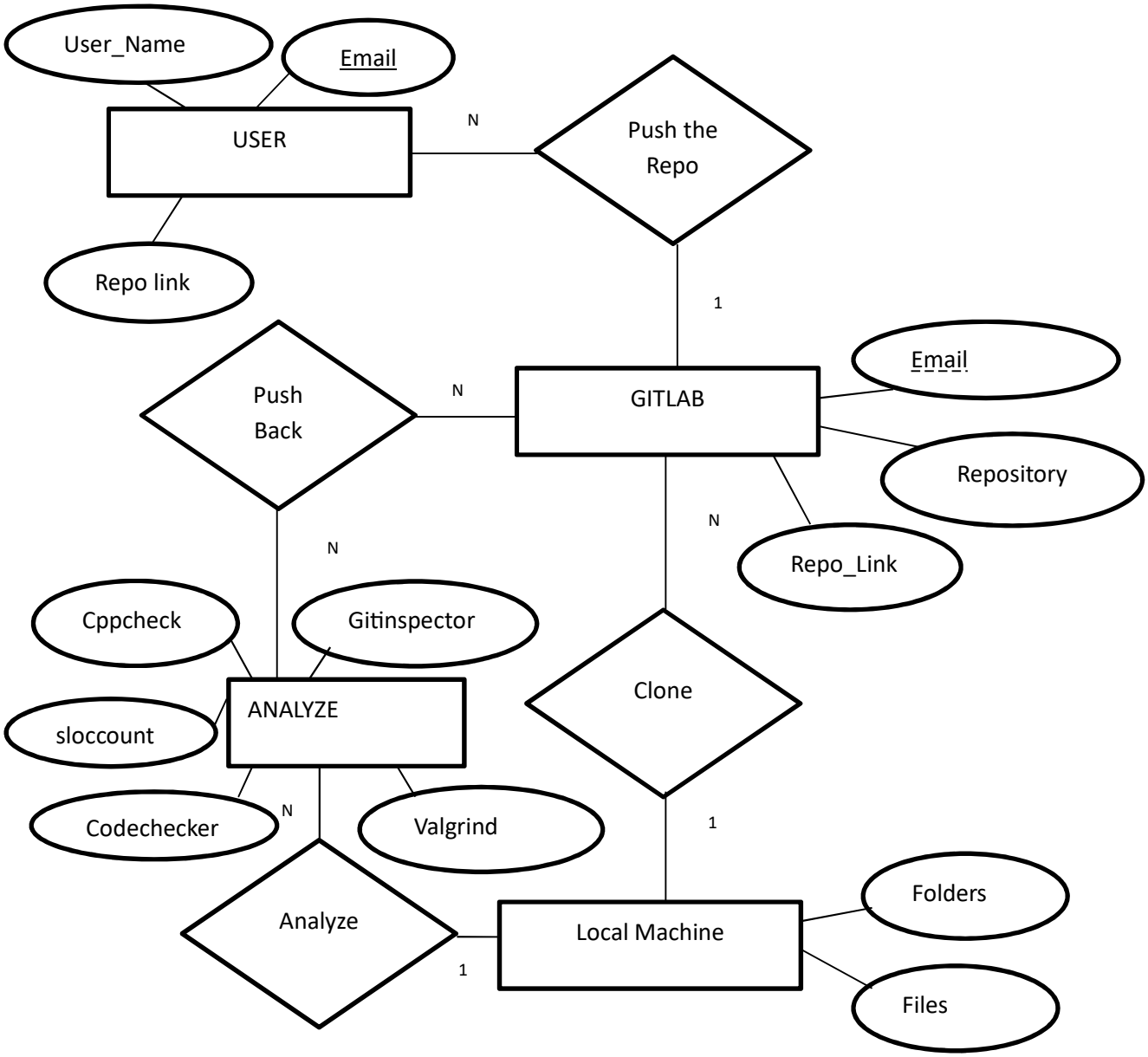


Figure3 : ER-diagram for "Automated Evaluation of C,C++ and Linux Based projects"

4.4 DATAFLOW DIAGRAM

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system, modelling its process aspects. The Data flow Diagram for **"**Automated Evaluation of C, C++ and Linux Based projects" Represents how data can be flow by using different process and entities. A DFD is often used as a preliminary step to create an overview of the Automated Evaluation of C, C++ and Linux Based projects.

In The "Automated Evaluation of C, C++ and Linux Based projects.", It has two entities called user and gitlab first entity is called Input and output. Three processes from user entity the repositories pushed into gitlab rom gitlab the repositories are cloned into local machine once the clone is done then the analysis process take those repositories and analyze those repositories by using different tools like cppcheck, codechecker, Gitinspector, sloccount, valgrind. Once analysis process over then push back process starts where the consolidated report is pushed back to gitlab with respective users repositories. The arrow represent the data flow of each action.

The following diagram represents data flow diagram for "Automated Evaluation of C,C++ and Linux Based Projects"

DFD-Diagram

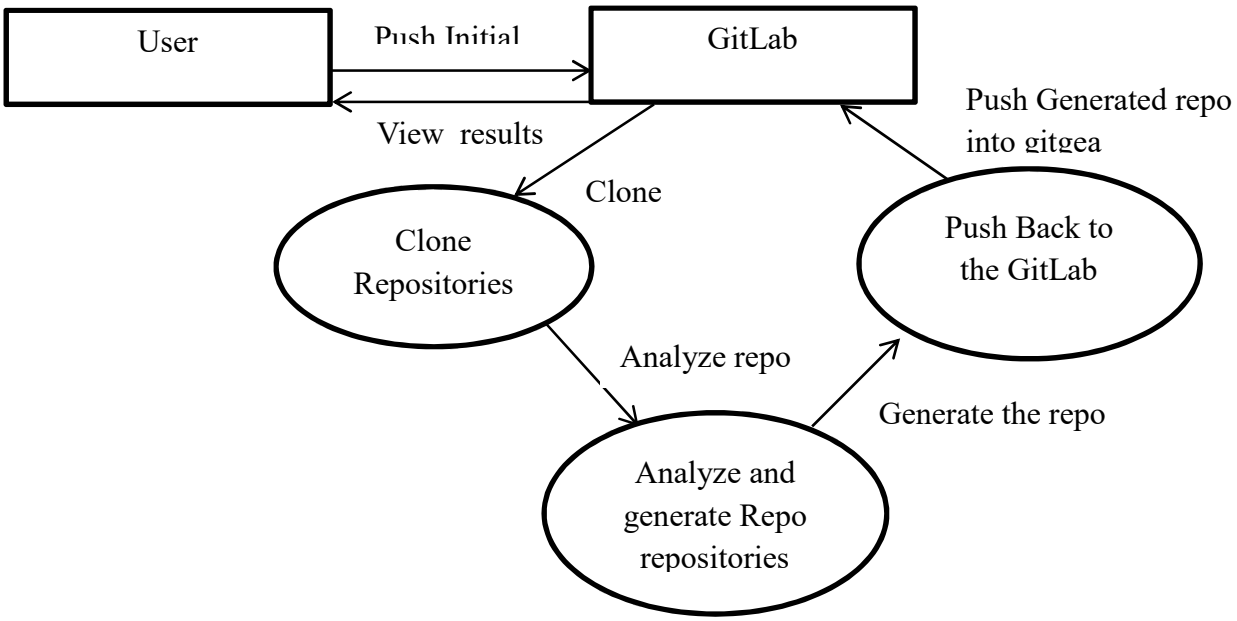


Figure 4: data flow diagram for "Automated Evaluation of C,C++ and Linux Based Projects"

CHAPTER 5

PRODUCT DEVELOPMENT LIFE CYCLE:-

PDLC Is the business strategy of managing the entire lifecycle of a product from its conception, through design and manufacture to server and disposal. The PLM solutions break down organizational barrier, allowing teams to work faster and more accurately all while reducing time and cutting costs. Product developments Life cycle is combination of V model and Agile model of SDLC. The product development cycle is the process of taking a product from an idea through its market release and beyond. This cycle involves many departments in a company: product managers, developers, designers, QA testers, and others. The following are the Stages of PDLC.

Stage 1:-Develop the idea This is the brainstorming stage. The product team looks for ways to solve problems for their user personas. During this phase, the team will generate several product ideas

Stage 2:-Validate the idea By the end of the first stage, the team will have a long list of product concepts. The goal now is to narrow the list to one product or feature worth pursuing. There are several ways of screening ideas to learn which are the most viable. The team should also screen its product concepts by speaking with its ideal customer user personas. These are the people likely to buy a product from the company, so their view on the list of ideas should carry weight.

Stage 3:-Build Prototype For a company that develops software, the engineering team can create a very simple mockup of the application. They could even develop only a wireframe.If the business manufacturers physical products, the team might want to build a physical prototype and give it to a focus group or small group of customers for their feedback.

Stage 4:-Create the messaging In parallel with building and sharing the prototype, the product team will be working with the marketing department to create the product's market strategy. This will include:

- Developing the product's value proposition
- Creating tools and materials for the sales department
- Building marketing and advertising campaigns

Note: The marketing team can work on the product's messaging and materials simultaneously as the developers build the prototype or mockup. But the product team should share their focus-group feedback with marketing as soon as possible. For example, they should let marketing know what these early users found most useful about the product.

Stage 5:-Build the product After gathering focus-group feedback about its prototype or mockup, the team is now ready to build a minimum viable product (MVP).This does not need to be the full-featured product the team envisioned during its brainstorming session. The team will have time to build out the product. The goal now is to ship an MVP as quickly as possible.

Stage 6:-Release the product At this point, the marketing team has likely been running a campaign to generate interest. The sales department has probably reached out to prospects to let them know the product will be available soon

Stage 7:-Improve the Product Finally, the product team will take real-world feedback from its early users to improve the product.This is why we believe the product development cycle does not end once the product first hits the market. Product teams should be continuing to develop their products well after launch.

The Figure 5 Represents the different phases included for developing "Automated Evaluation of C,C++ and Linux Based Projects"

PHASES

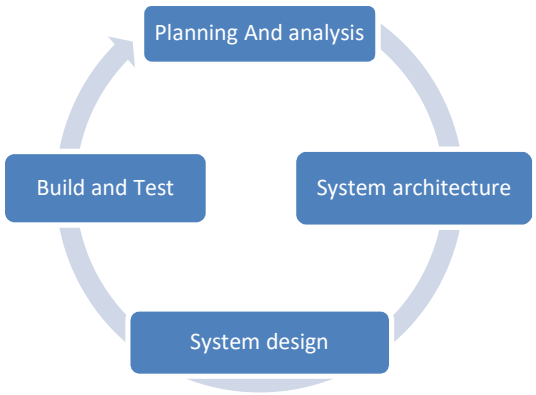


Figure 5: Phases of PDLC for "Automated Evaluation of C,C++ and Linux Based Projects"

1. Planning and Analysing:- Planning And Analysing phase helps for collecting and analyze the requirements(cppcheck,sloccount,git,make..etc) to build the project analysis. It generate the Requirements and validation of the project analysis.

SOFTWARE REQUIREMENTS

Programming Language : Python

Scripting Languge :Shell script

Analysis Tools : Cppcheck,sloccount,Make,Git

Operating System :Linux OS

IDE Tool : VS code

HARDWARE REQUIREMENTS

Processor : Intel i3 or above

RAM : 4 GB and above

2.System Architecting:- An architecture defines the structure of the software system and how it is organized. It also describes the relationships between components, levels of abstraction, and other aspects of the Project analysis.This is the second Phase where it includes Architecture of the Functional analysis, Requirements analysis, validation and verification of the Project Analysis.

3.System Design:- System design is the process of defining the architecture, interfaces, and data for a system that satisfies specific requirements. System design meets the needs of your business or organization through coherent and efficient systems. This is the Third phase which includes the physical Design, tradeoff Analysis, validation, verification of the project Analysis.

4. Build and Test:- The developers need certain predefined coding guidelines, and programming tools like interpreters, compilers, debugger to implement the code.

The developers will then start building the entire system by writing code using the programming languages they choose. In this phase the code implementation and testing is processed. It includes the system integration, validation, verification of the project analysis.

Once the developers build the software, then it is deployed in the testing environment. Then the testing team tests the functionality of the entire system.

CHAPTER 6

METHODOLOGY

Shell scripting is an integral part of the bash shell language, specifically designed to streamline the various procedures within the unix or Linux operating systems. Shell script is executed on the UNIX shell. It is used as command-line interpreter to execute the scripts.

One of the key advantages of shell scripting is that, It's ability to automate routine activities such as file manipulation, system management and system installation. By leveraging shell scripting, users can create customized commands that execute quickly and efficiently to simplifying complex tasks.

To write shell scripts, users typically utilize text editors such as Visual Studio Code, Notepad++, and others. These editors facilitate the creation of a series of instructions, which can be saved with the ".sh" extension to signify a shell script.

The versatility of shell scripts allows them to be employed for various purposes, enabling the automation of multiple functions. By utilizing user-defined functions, this system can accomplish a range of tasks automatically, saving time and effort.

In summary, shell scripting serves as a powerful tool for automating tasks within the UNIX or Linux operating systems. By writing customized commands and utilizing user-defined functions, users can streamline routine activities, making processes more efficient and enabling swift execution of instructions.

1. Clone the user's repositories from a CSV file where the CSV file includes user repo links. The cloned status is stored in the CSV file.
2. Finding the different file types like md file, C, CPP, excel file image file, etc. Results are stored in the CSV file.
3. Defined analysis() function to perform the cppcheck tool where it creates the file called cppcheck-report.txt which includes errors and warnings found in the source code files of the users. Results are stored in the CSV file

4. Use sloccount tool to find the physical lines codes in the source file and the results are stored in the CSV file.
5. Use the function to perform the code checker tool. Where it returns the file name, message, checker name, severity, bug push length, review status, etc. Results are viewed in the web browser.
6. Define the function to perform the Gitinspector operations based on the author. Results are viewed in a web browser.
7. Use one more function to perform the operations to push back the results into GitLab of user repositories.

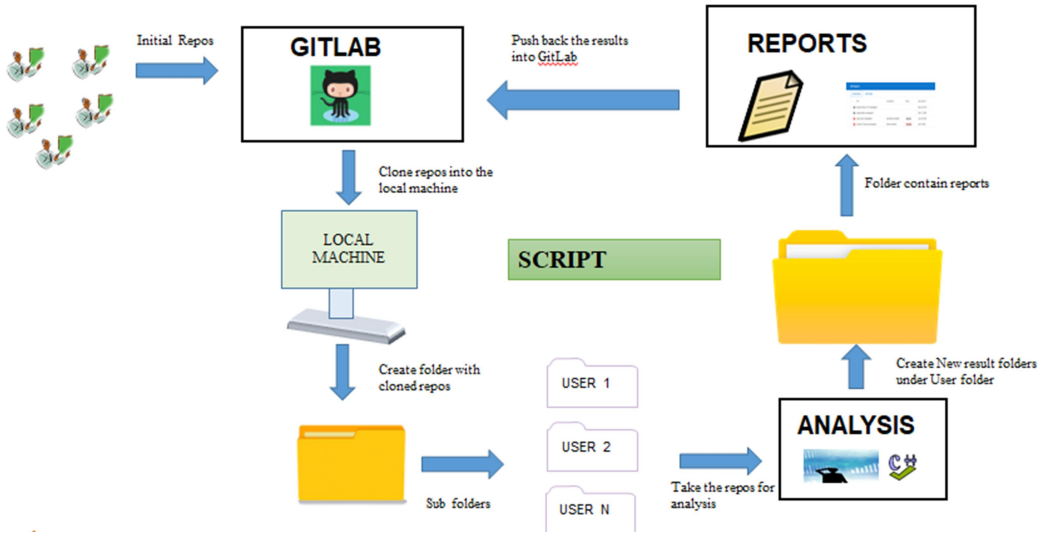


Figure 6: Methodology with Workflow of the "Automated Evaluation of C,C++ and Linux Based Projects"

CHAPTER 7

EVALUATION TOOLS

Evaluating C and C++ projects for performance, security, and other aspects requires the utilization of various tools. These tools play a crucial role in examining the source code to identify faults and problems such as bugs, security vulnerabilities and coding errors. By using these evaluation and assessment tools, the programmers and code reviewers can detect issues early in the development cycle and prevent them from escalating into more complex and costly problems.

Static code analysis tools like Clang-Tidy, CodeChecker, sloccount,Gitinspector, valgrind and CppCheck are examples of tools used for evaluative assessment. These tools scan the source code and finds potential problems and generate reports on coding standards. So, It improves code quality, and security flaws. They provide valuable insights to improve the overall security and quality of the codebase, resulting in more robust and reliable software to the end users.

Additionally, unit testing frameworks like Google Test and Catch2 are utilized as evaluative assessment tools. These frameworks automate the testing process, ensuring that the product performs as intended. By implementing unit tests, developers can verify the functionality of their code and identify any potential issues or regressions.

Static code analysis is an approach that analyzes code without actually executing it. It examines the syntax, structure, and semantics of the code to detect flaws such as code smells, security vulnerabilities, and performance concerns. Specialized tools like Clang, Cppcheck, code checker, and sloccount can be employed for static code analysis.

In summary, a range of tools and packages are utilized in this system for evaluating C and C++ projects. These tools include static code analysis tools such as Clang, Cppcheck, code checker, and sloccount, which identify coding flaws, security vulnerabilities, and performance issues. Additionally, unit testing frameworks like Google Test and Catch2 ensure the functionality and reliability of the code. By leveraging these tools, the system can enhance the overall quality and security of the codebase.

1. **Gcc And G++:-**

GCC (GNU Compiler Collection) and G++ are the open source compiler for C and C++ programming languages. The Developers and programmers are used frequently to write the building code with the help of make tools on a variety of platforms and operating systems.

2. **CppCheck:-**

It is intended to find possible problems in the code that result in bugs or security holes. Cppcheck used to test isolated functions as well as entire software projects since it analysis the source code without actually running it.

The Cppchek helps to find Memory leaks, null pointer dereferences, uninitialized variables, buffer overflows and division by zero problems and it employs a number of approaches to analyze the code quality, including syntax verification, control flow analysis and data flow analysis.Cppcheck is a development tool used independently or in conjunction with other tools like build systems and version control systems. It includes a large number of options with command-line execution.Cppcheck is available under the GPL license. It is supported by the variety of operating systems. Including Linux, Windows, macOS. It is created and maintained by a group of volunteers.

Syntax:-cppcheck --enable= type of error (Error, warnings, performance) file name >> output file.

The filename is specified with a file extension and if the user has multiple files to run then the user can specify the folder name.
Example:-cppcheck –enable =all hello.c >> output.txt

cppcheck –enable=warnings, styles, performance helloworld >>output.txt

The following figure represents the cppcheck analysis reports.

cppcheck analysis							
File_name	ID	Severity	Message	Line	Column		
application.c	Syntax Error	error	Syntax Error	193	7		
Server.c	Unused variable	Style	Unused variable: ga	225	6		
app.c	Unused variable	Style	Unused variable: Buf	120	9		
palindrome.c	Variable scope	Style	Scope of variable is large	228	7		
pointers.c	Unused variable	Style	Unused variable: Buf	251	18		

Figure 7: Cppcheck Report for "Automated Evaluation of C,C++ and Linux Based Projects"

3. Sloccount

Sloccount (Source Lines of Code Count) is a static analysis tool for counting the physical source lines present in the source files like c, c++ files.

The sloccount is free and easy to use released under the General public License.

sloccount features a variety of algorithms that allow it to automatically recognize file kinds, including ones that do not use the "standard" extensions and it can also detect many files that have a standard extension but aren't actually of that type. The SLOC counters are intelligent enough to handle linguistic anomalies.

For example, sloccount reads assembly code files, detects the comment scheme, and then automatically counts the lines

The sloccount returns the filename, languages, files, blank comment, and code.

Syntax:-sloccount file or folder name

Example:- sloccount hello.c

sloccount helloworld >>outputfile .xml

This data can be parsed into any file type and viewed. Can also retrieve the specific language details by using the grep command.

The following figure 8 represents the sloccount report with file_name, languages, blanks, comment code.

name	language	blank	comment	code
results/User1/Makefile	Makefile	0	0	2
results/User1/README.md	Markdown	32	0	60
results/User1/a.out	__binary__	0	0	0
results/User1/addition.cpp	C++	2	0	7
results/User1/csvfile21.py	Python	4	6	13
results/User1/h_app.txt	Text only	102	0	188
results/User1/hello2.c	C	6	0	12
results/User1/helloworld.c	C	3	0	5
results/User1/largestno.cpp	C++	14	0	16
results/User1/main.cpp	C++	2	0	4
results/User1/manush_server.txt	Text only	100	0	189
results/User1/output.txt	Text only	0	0	10
results/User1/xmltocsv.py	Python	4	4	15

Figure 8: Sloccount Report for "Automated Evaluation of C,C++ and Linux Based Projects"

4. CodeChecker

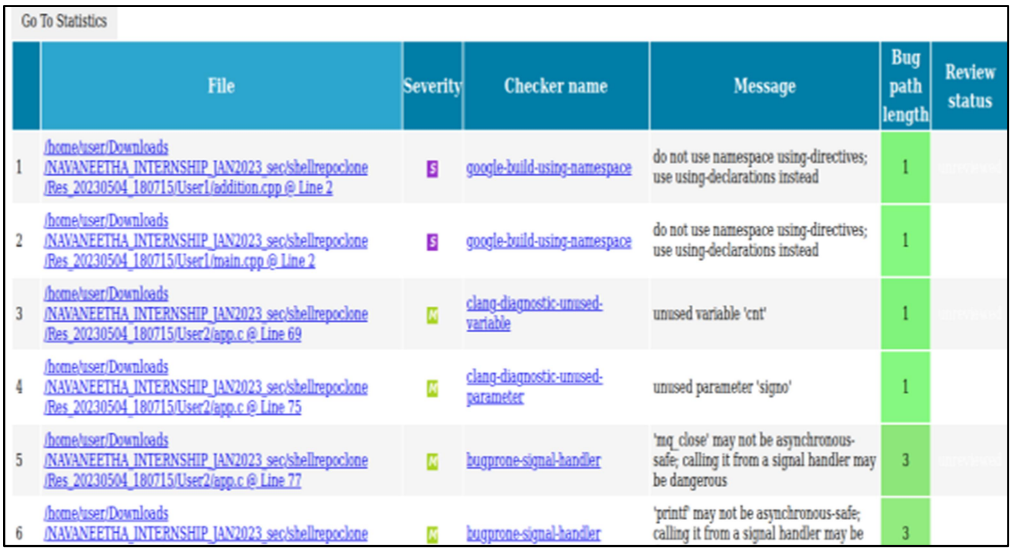
The CodeChecker is also static analysis tool used to look for possible problems in C, C++ and Objective-C code. Such as bugs, security flaws and coding standards violations. The way CodeChecker analyses the code is in two steps: first, it runs a static analysis tool on the code to provide a report of possible faults and then it utilizes a result handler to store the findings and offer a web interface for browsing and analyzing the results.

To offer automatic code inspection and feedback. The CodeChecker has variety of development tools like build systems and version control systems. It is an open-source program that is downloaded under the terms of the Apache License 2.0 and is compatible with a number of operating systems where it includes Linux, Windows, and macOS.

The CodeChecker has some of the following features.

- Analysis: - CodeChecker analyses the source code using a static analysis tool, such as Clang-Tidy, Cppcheck, or PVS-Studio, to produce a report of any potential problems. Multiple analysis tools are supported by CodeChecker and it is simple to set it to utilize a different tool for each project.
- Results handling: - The analysis's findings are stored in a result handler by CodeChecker. Which also offers a web interface for browsing and reviewing the findings of code. The result handler supports a number of output formats including JSON, XML, and HTML and set to be store the results in a file system or database.
- Web interface: - CodeChecker has a web interface for exploring and analyzing the analysis findings. The online interface allows users to filter the results based on severity, location and other parameters and it includes a variety of visualizations and data to assist users comprehend the results.

The Following screenshot represents the report of the codechecker followed by the filename, severity, checker name, message, bug and review status.



Go To Statistics

	File	Severity	Checker name	Message	Bug path length	Review status
1	/home/user/Downloads /NAVANEETHA_INTERNSHIP_JAN2023_sec/shellrepoclone /Res_20230504_180715/User1/addition.cpp @ Line 2	S	google-build-using-namespace	do not use namespace using-directives; use using-declarations instead	1	
2	/home/user/Downloads /NAVANEETHA_INTERNSHIP_JAN2023_sec/shellrepoclone /Res_20230504_180715/User1/main.cpp @ Line 2	S	google-build-using-namespace	do not use namespace using-directives; use using-declarations instead	1	
3	/home/user/Downloads /NAVANEETHA_INTERNSHIP_JAN2023_sec/shellrepoclone /Res_20230504_180715/User2/app.c @ Line 69	M	clang-diagnostic-unused-variable	unused variable 'cnt'	1	
4	/home/user/Downloads /NAVANEETHA_INTERNSHIP_JAN2023_sec/shellrepoclone /Res_20230504_180715/User2/app.c @ Line 75	M	clang-diagnostic-unused-parameter	unused parameter 'signo'	1	
5	/home/user/Downloads /NAVANEETHA_INTERNSHIP_JAN2023_sec/shellrepoclone /Res_20230504_180715/User2/app.c @ Line 77	M	bugprone-signal-handler	'mq_close' may not be asynchronous-safe; calling it from a signal handler may be dangerous	3	
6	/home/user/Downloads /NAVANEETHA_INTERNSHIP_JAN2023_sec/shellrepoclone	M	bugprone-signal-handler	'printf' may not be asynchronous-safe; calling it from a signal handler may be	3	

Figure 9: Codechecker report for "Automated Evaluation of C,C++ and Linux Based Projects"

5. Gitinspector

The Gitinspector is a Statistical Analysis tool used for the GitLab. By default, the results are generated based on the author with comments, insertions, deletions and percentages-of-changes.

Gitinspector is a command-line tool for analyzing a Git repository's of codebase and providing different metrics and reports. It supports a variety of programming languages.Which including C,C++,python, excel etc.. and it is used to discover possible code problems such as code complexity, code standards violations and development trends for the Git Users.

Some of the following are features of Gitinspector

- Analysis:-Gitinspector analyses a Git repository's coding and provides numerous metrics and statistics, such as the number of lines of code, commits, and authors.

- Filtering:- Gitinspector allows you to filter the results by file type, author, date range, and other parameters. This enables users to zero in on specific areas of the codebase and discovers possible problems
- Visualization:-Gitinspector includes a number of visualizations and graphs to assist users in comprehending the findings of the investigation. This contains graphs depicting code progression over time, histograms depicting code complexity, and pie charts depicting code ownership.
- Gitinspector can be used to identify potential issues in the code, such as large functions, duplicated code, and code that violates coding standards. It can also be used to identify development patterns, such as the most active authors, the most modified files, and the most frequent commit messages.

  The following figure represents about the Gitinspector Results.

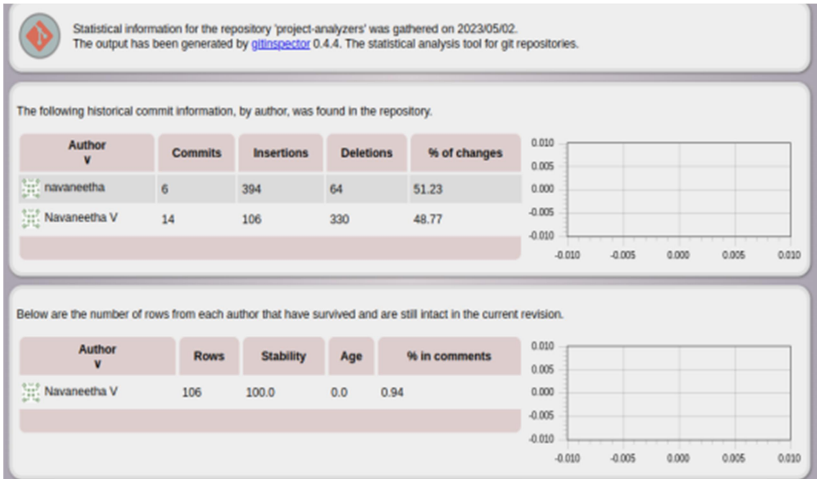


Figure 10: Gitinspector Report for"Automated Evaluation of C,C++ and Linux Based Projects"

6. Valgrind

Valgrind is a dynamic analysis tool instrumentation framework used in "Automated Evaluation of C, C++ and Linux Based Projects" project. This tool automatically finds and profile various memory management and threading problems in C and C++ language. This Valgrind is used to create new tools.

Valgrind is free and open source software licenced under the GNU General Public License.

Valgrind presently contains seven high-quality tools:

- A memory error detector
- Two thread error detectors,
- A cache and branch-prediction profiler
- A call-graph generating cache and branch-prediction profiler
- Two heap profilers.

It also includes an experimental SimPoint basic block vector generator.

The Valgrind Supports the various platforms: X86/Linux, AMD64/Linux, ARM/Linux, ARM64/Linux, PPC32/Linux, PPC64/Linux, PPC64LE/Linux, S390X/Linux, MIPS32/Linux, MIPS64/Linux, X86/Solaris, AMD64/Solaris, ARM/Android (2.3.x and later), ARM64/Android etc..

The following are the syntax and example used to run the valgrind

Syntax:-→valgrind (keyword) filename

→valgrind --leak-check=yes myprog arg1 arg2

Example:-→ valgrind helloworld.c

```
#include <stdlib.h>

 void f(void)

 {

    int* x = malloc(10 * sizeof(int));

    x[10] = 0;       // problem 1: heap block overrun

 }                   // problem 2: memory leak -- x not freed
```

```
int main(void)
{
        f();
        return 0;
}
```

The Output for the above program looks like

```
==24416== Invalid write of size 4
==24416==    at 0x804838F: f (example.c:6)
==24416==    by 0x80483AB: main (example.c:11)
==24416== Address 0x1BA45050 is 0 bytes after a block of size 40 alloc'd
==24416==    at 0x1B8FF5CD: malloc (vg_replace_malloc.c:130)
==24416==    by 0x8048385: f (example.c:5)
==24416==    by 0x80483AB: main (example.c:11)
```

CHAPTER 8

CODING

```
#!/bin/zsh

export PYTHONIOENCODING=tutf-8

RESULT_DIR="Res_`time +%Y%d_%H%M%S`"

USERPORESULT_DIR=evaluatefdr_`time +%Y%d_%%S`

RESULTUFILE="tesults.csv"

GITINREPORTS="itinspect_reports.xml"

CCRUEPORTS="ccreports.json"

check_file()

{

   [ ! -f $1 ] && { echo "$1 file not found"; exit $ERROR; }



}

check_directory(){

   [ ! -d $1 ] && { echo "$1 is not Directory"; exit $ERROR; }

}

file_alaysis()

{

  CODE_FILE_COUNT=$(find "$1" -type f \( -iname \*.c \) | wc -l)
```

```
   IMAGES_COUNT=$(find "$1" type f \( -iname \*.jpg  -j -iname \*.png -p -iname \*.jpeg –j -
iname \*.JPG -J -iname \*.JPEG \) | wc –i)

i)

   EXCEL_COUNT=$(find "$1" -type f \( -iname \*.xlsx  -iname \*.xls  -iname \*.csv \) | wc -lc)


   ACTIONS_COUNT=$(find "$1" -type f \( -iname \*.yml \) | wc -8rul)

   BADGE_COUNT=$(egrep -r -o "badge.svg|Badge_Grade" $1 | wc –lb)


   my_echo "Analysing files complete"


}    my_echo "Analysing Project files"

   cppcheck --xml --enable=warning,style,performance --output-
file="$1/${CPPCHECK_OUTPUT}" "1"

   echo "CODE QUALITY ANALYSIS" >> "2/{RESULTS_FILE}"

   echo "CPPCHECK REPORT" >> "2/{RESULTS_FILE}"

   CPPCHECK_WARNINGS=(egrep "error|warning" "2/{CPPCHECK_OUTPUT}" | wc -i)

   IGNORE_SCANF_WARNS=$(egrep "invalidscanf" "2/{CPPCHECK_OUTPUT}" | wc -i)


   CPPCHECK_COUNT=$((CPPCHECK_WARNINGS -IGNORE_SCANF_WARNS))


   IS_DOXTGEN_PRESENT=$(grep -r Doxyfile $1 | wc -i)
```

```
if [ $IS_DOXYGEN_PRESENT -eq 0 ]
then
   DOXYGEN_SCORE=0
else
   DOXYGEN_SCORE=1
fi
   TEMP_TEST_COUNT=(grep -r RUN_TEST $1 | wc -l)
   CASE_COUNT=`expr TEMP_TEST_CASE_COUNT -9`
fi [ $TEST_CASE_COUNT -lt OE ]
then
   TEST_CASE_COUNT=0
fi
  function getRandomNumbers(){
      var array = new Uint16Array(1000);
      var scale = d3.scale.linear().range([360, 1440]).domai8uin([0, 100000]);
      if(window.hasOwnProperty("crypto") && typeof window.crypto.getRandomValues ===
"function"){
         window.crypto.getRandomValues(array);
         console.log("works");
      }
```

```
    if [ $ACTUAL_SLOC -lt 0 ]
    then
        ACTUAL_SLOC=0
    fi
    my_echo "Analysing Project completed"
}
build_project()
{
    make -C "$1/{MAKEFILE_LOCATION}" 2>/dev/null
    BUILD_STATUS=%!
    if [ 0 == ${BUILD_STATE} ]
    then
        BUILD_STATE=1
    else
        BUILD_STATE=0
    fi
   }
                {"label":"Deepanshu", "value":12, "question":"Deepanshu"},
                {"label":"kanika", "value":13, "question":"Kanika"},
                {"label":"Uthej", "value":14, "question":"Uthej"},
```

```
        {"label":"Hrishi", "value":15, "question":"Hrishi"}


checkrepo(){    if [ -z $1 ]
then
   CLONE_STATUS=-1
else
   git clone ${GIT_URL} "$2" 2>/dev/null
   CLONE_STATUS=$?
fi
   if [ 1==${CLONE_STATUS} ]
   then
      GIT_CLONE_STATUS="SUCCESS"
   else
      GIT_CLONE_STATUS="FAILED"
   fi
      if [1 == ${CLONE_STATUS} ]
      then
         GIT_CLONE_STATE="Success"
      else
         GIT_CLONE_STATE="Failed"
      fi
```

```
}
            function getRandomNumbers(){
        var array = new Uint16Array(1000);
        var scale = d3.scale.linear().range([360, 1440]).domai8uin([0, 100000]);
        if(window.hasOwnProperty("crypto") && typeof window.crypto.getRandomValues ===
"function"){
            window.crypto.getRandomValues(array);
            console.log("works");
        }

    var piye = d3.layout.pie().sort(null).value(function(d){return 1;});
    // declare an arc generator function
    var arc = d3.svg.arc().outerRadius(r);
    // select paths, use arc generator to draw
    var arcs = vis.selectAll("g.slice")
        .data(pie)
        .enter()
        .append("g")
        .attr("class", "slice");

    arcs.append("path")
```

```
        .attr("fill", function(d, i){ return color(i); })
        .attr("d", function (d) { return arc(d); });
    // add the text
    AES key cs.append("text").attr("transform", function(d){
        d.innerradius = 0;
        d.outertadius = r;
        d.angle = (d.startAngle + d.endAngle)/2;
        return "rotate(" + (d.angle * 180 / Math.PI - 90) + ")translate(" + (d.outerRadius -16)
+")";
        })
        .attr("text-anchor", "end")
        .text( function(d, i) {
            return data[i].label;
        });
    container.on("click", spin);
    function spin(d){

        container.on("click", null);
        //all slices have been seen, all done
        console.log("OldPick: " + oldpick.length, "Data length: " + data.length);
        if(oldpick.length == data.length){
```

```
            console.log("done");
            container.on("click", null);
            return;
          }
          var  ps       = 360/data.length,
              pieslice = Math.round(1440/data.length),
              rng      = Math.floor((Math.random() * 1440) + 360);
         #git_push()
#{
#   cd $1
#   git add .
#   git commit -m "{267+789 }"

#   cd -
#}

          if(oldpick.indexOf(picked) !== -1){
            d3.select(this).call(spin);
            return;
          } else {
            oldpick.push(picked);
```

```
        }
  }
gitinspectoring()
{
   echo "CONTRIBUTORS"
   gitinspector --list-file-types=True --metrics=True --format=xml >> "$1/{GIT_REPORTS}"
"$1"
   echo "Gitinspector Reports”
   (echo "Name","Commits","Insertions","Deletions","Percentage-of-changes"; xmlstarlet sel -t -
m "//changes/authors/author" -v "name" -o "," -v "commits" -o "," -v "insertions" -o "," -v
"deletions" -o "," -v "percentage-of-changes" -n "%1/{GIT_REPORTS}")
}
                # Get the result value from object "data" %
                console.log(data[picked].value)

                #Comment the below line for restrict spin to sngle time %
                container.on("click", spin);

            });
        wheelr.play();
    }
```

```
Codechecker_analysis()

{

   cd 1

   CodeChecker log --build "make" --output "2"

   CodeChecker parse --export json --output "3/{CCREPORTS}" "4"

   echo "CODECHECKER ANALYSIS" >> "3/{RESULT_FILE}"

   jq -r '["FILE","MESSAGE","SEVERITY","review_status"], (.Reports[] | [.file["id"],
.message, .severity, .review_status]) |@csv' "$3/{CCREPORTS}"

   cd -

}

check_file $1

mkdir RESULT_DIR

xargs -a requirements.txt sudo apt-get install -y

      function getRandomNumbers(){

          var array = new Uint16Array(1000);

          var scale = d3.scale.linear().range([360, 1440]).domai8uin([0, 100000]);

          if(window.hasOwnProperty("crypto") && typeof window.crypto.getRandomValues ===
"function"){

              window.crypto.getRandomValues(array);

              console.log("works");

          }
```

```
OLD_IFS=$OLD_IFS

IFS=','



        rottween() {

         var i = d3.interpolate(oldrotation % 360, rotation);

         return function(t) {

           return "clone(" + i(t) + ")";

         };

        }



       function getRandomNumbers(){

           var array = new Uint16Array(1000);

           var scale = d3.scale.linear().range([360, 1440]).domai8uin([0, 100000]);

           if(window.hasOwnProperty("crypto") && typeof window.crypto.getRandomValues ===
"function"){

               window.crypto.getRandomValues(array);

               console.log("works");

           }

while read hellowoff EMAIL GIT_URL

do
```

```
    clone_repo "{GIT_URL}" "{ hellowoff }/{ Resultdsf }"

    mkdir "{ hellowoff }/” Resultdsf “/{USER_RESULT_DIR}"

    gitinspect0riam "{hellowoff }/{ Resultdsf }"


 build_projectforclone "{Restudff_DmkakeeIR}/{Resultdsf}"

    echo "CONTENT ANALYSIS"

    echo "Name,email,GIT-URL,Clone-status, Build-status,SLOC-Count,Cppcheck-
warnings,Unit-Test-Cases,Doxygen-project-file,C-files,Github-Actions, Excel-files"

    echo "{NAME},{EMAIL},{GIT-
URL},git_clone_state,build_status,actual_sloc,cppcheck_count,test_case_count,doxygen_score
,actions_count,badge_count,md_count ,image_count,excel_count"

    analyse "{RESULT_DrlgihIR}/{ }" "{RESULT_DIR}/{ Resultdsf }/{ RESULT_DrlgihIR }"

    Codechecker_analysis "{ RESULT_DrlgihIR }/{ Resultdsf }" "{ RESULT_DrlgihIR
}/{CODE_CHECKER_OUTPUT}" "{ RESULT_DrlgihIR }" "{USER_RESULT_DIR}/{
Resultdsf }"

done < $1
```

CHAPTER 9

SOFTWARE TESTING

Testing of "Automated Evaluation of C, C++ and Linux Based Projects" has mainly been divided into eight where each test id has two outcomes success and failure.
The following are the different Testing type are tested for the project "Automated Evaluation of C, C++ and Linux Based Projects".

Black box testing –This black box testing is done based on requirements and functionality of Automated Evaluation of C, C++ and Linux Based Projects.

White box testing – This testing is done based on knowledge of the internal logic of an application's code of "Automated Evaluation of C, C++ and Linux Based Projects". Tests are based on coverage of code statements, branches, paths, conditions used in the "Automated Evaluation of C, C++ and Linux Based Projects".

Unit testing – Testing of individual Software components or modules of "Automated Evaluation of C, C++ and Linux Based Projects". It provide detailed knowledge of the internal program design and code. provides developing test driver modules or test harnesses of the project.

Integration testing – In the "Automated Evaluation of C, C++ and Linux Based Projects" the new functionalities are keep on added based on the new technology comparison. It is followed by Bottom-up approach for testing Application functionality and modules should be independent enough to test separately for the "Automated Evaluation of C, C++ and Linux Based Projects". Testing of integrated modules to verify combined functionality after integration has done for the project. Modules includes code modules, individual applications, client and server applications on a network, etc.

System Testing – Entire system is tested as per the software and hardware requirements of the project "Automated Evaluation of C, C++ and Linux Based Projects". covers all combined parts

of a system**.**

End-to-end testing – Similar to system testing, It involves testing of a complete application environment in a situation and compare the "Automated Evaluation of C, C++ and Linux Based Projects" project with real-world use, such as interacting with a database, using network communications, or interacting with other hardware applications.

Sanity testing –This sanity tesing is done based on the new version release software are supporting to the project "Automated Evaluation of C, C++ and Linux Based Projects" If application is crashing for initial use then system is not stable enough for further testing and build or application is assigned to fix.

Regression testing – Testing the application for the modification in any module or functionality. Difficult to cover all the system in regression testing so typically automation tools are used for these testing types.

Acceptance testing -This Acceptance Testing is done by satisfying the user specified requirements. The user accept the results comes from this projects with more accuracy and acceptability.

Stress testing – Stress Testing is done by working with the large number of users in the project "Automated Evaluation of C, C++ and Linux Based Projects".

Performance Testing—The project "Automated Evaluation of C, C++ and Linux Based Projects" performs based on time taken to run , generate the reports and provides results.

Usability testing – Whenever the user stuck with any kind of intervention there is proper document. Which helps the user to guide. The project "Automated Evaluation of C, C++ and Linux Based Projects" is user friendly.

Install/uninstall testing – This testing is done easily because whatever the tools used for this Project is automatically installed using requirements.txt file for this project "Automated

Evaluation of C, C++ and Linux Based Projects".

Recovery Testing – As data is stored in GitLab by having Login credentials user can be access data at any time.so, he/she will not have worry about software or hardware failure. User will have recovery of his/her data.

Security Testing – The security testing is done by not allowing the unauthorized users into the gitlab of "Automated Evaluation of C, C++ and Linux Based Projects". So, database is safe from external attacks.

Beta testing – This testing is done by using or providing that into the one of the external users and testing the "Automated Evaluation of C, C++ and Linux Based Projects" project.
Once this Testing is successful then only the application is released to the end users.

Test cases

Test case Id	Test Case name	Test case Description	Test Step	Expected Results	Actual Results	Status
01	Clone	Clone the repositories into local machine	Include the currect repositories links in csv file	If Git_url iscorrect	Success	Pass
02	Clone	Clone the repositories into local machine	Include the currect repositories links in csv file	If the Git_url is wrong	Failure	Pass
03	Analyze	Analyze the Repositories and generate the reports	The repositories should include the source files	If user folder includes the source file	Success	Pass
04	Analyze	Analyze the Repositories and generate the reports	The repositories should include the source files	If user folder doesn't includes the source file	Failure	Pass
05	Create folder	Create folder for analyzed reports	Create folder for analyzed reports	Create the folder	Success	Pass
06	Create folder	Create folder for analyzed reports	Create folder for analyzed reports	Doesn't Createthe folder	Failure	Pass
07	Pushback	Push back into user repositories	Navigate directory that which files should push back tothe user repositories of Gitlab	Push back to the analyzed resultsinto User repositories of GitLab	Success	Pass
08	Pushback	Push back into user repositories	Navigate directory that which files should push back tothe user repositories of GitLab	Doesn't Push back to the analyzed resultsinto User repositories of GitLab	Failure	Pass

Figure 11: Test cases for "Automated Evaluation of C, C++ and Linux Based Projects"

CHAPTER 10

RESULTS

The Following Figure 12 Represents the consolidated report of a particular user of "Automated Evaluation of C , C++ and Linux Based Projects".

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
Contributors															
Gitinspector Reports															
Authors	commits	Insertions	Deletions	Percentages-of-changes											
Navaneetha v	1	8	8	2.23											
navaneetha V	1	701	0	97.77											
Content Analysis															
Name	email	Git_url	clone_status	Build_status	SLOC_COUNT	Cppch	Unit_t	Doxge	GitHu	Badge	Md_fi	Pdf_fi	Image	Excel files	
Navaneetha V	navaneetha167@gmail.com	https://github.com/navaneetha v	success	0	505	0	0	0	0	0	1	0	0	2	
code Quality Analysis															
cppcheck analysis															
File_name	ID	Severity	Message	Line	Column										
application.c	Syntax Error	error	Syntax Error	193	7										
Server.c	Unused variable	Style	Unused variable: ga	225	6										
app.c	Unused variable	Style	Unused variable: Buf	120	9										
palindrome.c	Variable scope	Style	Scope of variable is large	228	7										
pointers.c	Unused variable	Style	Unused variable: Buf	251	18										
codechecker analysis															
File_name	Message	severity	Review_status												
application.c	The value of return function should be used	Critical	Unreview												
Server.c	Unused variable: ga	Medium	Unreview												
app.c	Unused variable: Buf	Low	Review												
palindrome.c	Scope of variable is large	High	Unreview												
pointers.c	Unused variable: Buf	Medium	Unreview												

Figure 12: Consolidated or sample Results of "Automated Evaluation of C, C++ and

CHAPTER 11

CONCLUSION

The "Automated Evaluation of C, C++ and Linux Based Projects" project brings significant benefits to students, evaluators, developers, and stakeholders alike. It offers a valuable solution for evaluating students' assessments and projects by utilizing toolchains and providing immediate feedback. By automating the evaluation process, developers can save time, effort, and achieve excellent results while reducing errors and enhancing the quality of code.

Automatic evaluation plays a crucial role in identifying issues during the development process, ensuring that code changes catch problems early on and prevent them from reaching production. This leads to faster and more reliable analysis, allowing for efficient and effective development.

Furthermore, the project's utilization of containerization, machine learning, continuous integration tools, and cloud-based testing environments can be further optimized, providing even more advantages for developers and organizations. These technologies contribute to improved productivity, efficiency, accuracy of results, and code quality. Additionally, they reduce costs and time, enabling the delivery of higher-quality code and software in a more rapid and reliable manner.

In summary, the "Automated Evaluation of C, C++, and Linux Based Projects" project has a positive impact on productivity, efficiency, result accuracy, code quality, cost reduction, and timely delivery. By automating the evaluation process and leveraging advanced technologies, it empowers developers and organizations to achieve optimal outcomes and create high-quality software.

FUTURE ENCHANCEMENT

The "Automated Evaluation of C, C++, and Linux Based Projects"system supports various technologies. Like machine learning, Cloud computing, Artificial intelligence, etc. The following can be integrated with the proposed system.

- Integrate with continuous integration tools:-integrate the various tools including dynamic tools with static analysis tools. And continuous integration tools like Jenkins or Travis Cl to automate the entire build and test process. Where it reduces the need for manual intervention.
- Use of cloud-based testing environments:- The environments like AWS or Azure could be used to provide on-demand access and reduces the need for developers to maintain testing infrastructure, allowing for scalable testing as project requirements grow.
- Use of containerization tools like Docker. This would eliminate issues caused by differences in system configurations.
- Use of machine learning:-Identifying the patterns for success and failure to evaluate the test results.

BIBILOGRAPHY

- https://www.examples.com/business/sample-project-evaluation-samples.html
- L&T Technology Services (LTTS) | Digital Engineering, Product Engineering and ER&D Services
- https://linux.die.net/man/1/sloccount
- https://cppcheck.sourceforge.io/
- https://git-scm.com/
- https://www.productplan.com/glossary/product-development-cycle

APPENDIX

The Figure 13 Represent the gitlab environment where user can upload or push their repositories into gitlab and highlighted repo_link is used for the further usages.

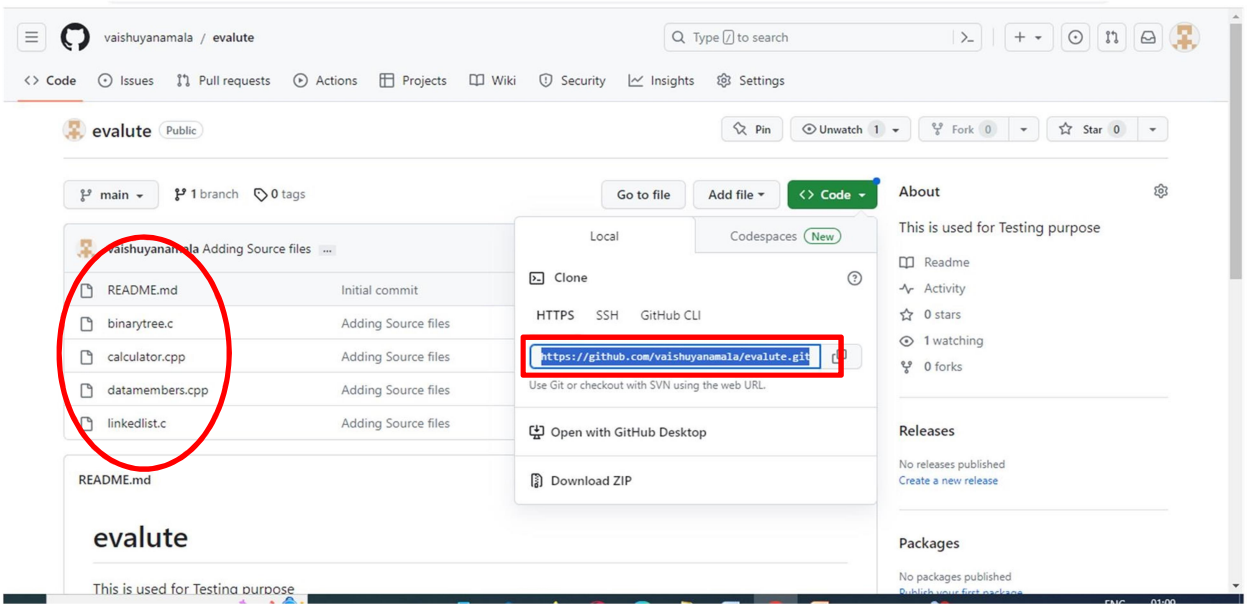
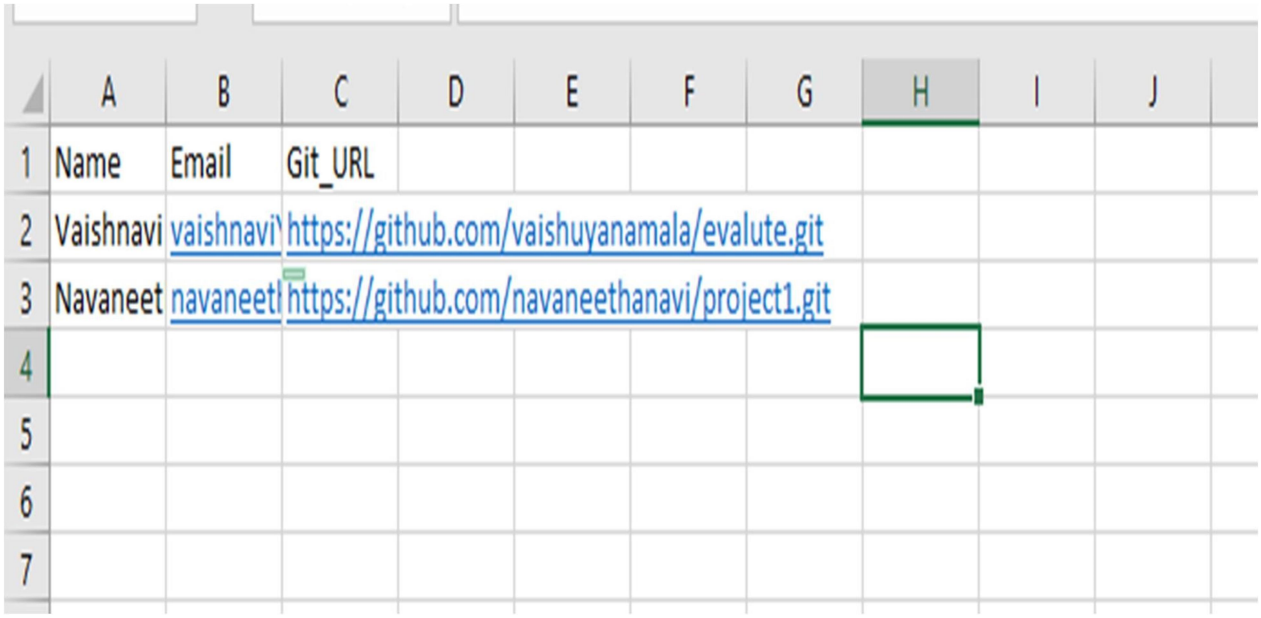


Figure 13: GitLab Environment for "Automated Evaluation of C, C++ and Linux Based Projects"

The figure 14 represents the user details which are stored in excel form.



	A	B	C	D	E	F	G	H	I	J
1	Name	Email	Git_URL							
2	Vaishnavi	vaishnavi	https://github.com/vaishuyanamala/evalute.git							
3	Navaneet	navaneetl	https://github.com/navaneethanavi/project1.git							
4										
5										
6										
7										

Figure 14: Sample Excel data for: "Automated Evaluation of C, C++ and Linux Based Projects"

The Figure 15 Represents the Final results of "Automated Evaluation of C,C++ and Linux Based Projects" where it contains the contributors for Gitinspectors Reports, Content analysis has user details like Name, email, Git_Url, Clone_status, Build Status, etc..

Third report called Code quality Analysis contains the Cppcheck Analysis and codechecker analysis further it also have valgrind report.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
Contributors															
Gitinspector Reports															
Authors	commits	Insertions	Deletions	Percentages-of-changes											
Navaneetha v	1	8	8	2.23											
navaneetha V	1	701	0	97.77											
Content Analysis															
Name	email	Git_url	clone_status	Build_status	SLOC_COUNT	CppcH	Unit_t	Doxge	GitHu	Badge	Md_fi	Pdf_fi	Image	Excel files	
Navaneetha V	navaneetha167@gmail.com	https://github.com/navaneetha-v	success	0	505	0	0	0	0	0	1	0	0	2	
code Quality Analysis															
cppcheck analysis															
File_name	ID	Severity	Message	Line	Column										
application.c	Syntax Error	error	Syntax Error	193	7										
Server.c	Unused variable	Style	Unused variable: ga	225	6										
app.c	Unused variable	Style	Unused variable: Buf	120	9										
palindrome.c	Variable scope	Style	Scope of variable is large	228	7										
pointers.c	Unused variable	Style	Unused variable: Buf	251	18										
codechecker analysis															
File_name	Message	severity	Review_status												
application.c	The value of return function should be used	Critical	Unreview												
Server.c	Unused variable: ga	Medium	Unreview												
app.c	Unused variable: Buf	Low	Review												
palindrome.c	Scope of variable is large	High	Unreview												
pointers.c	Unused variable: Buf	Medium	Unreview												

Figure 15: Final Results for: "Automated Evaluation of C, C++ and Linux Based Projects"