National University of Computer and Emerging Sciences

Comparative Study Of Rust and C++

FYP Team

Syeda Rabia Fatima…………19L-2364
Zainab Nauman……………..19L-2342
Hamna Shafqaat…………….18L-1033

Supervised by

Mr. Amir Raheem

FAST School of Computing

National University of Computer and Emerging Sciences

Lahore, Pakistan

Abstract

The purpose of this research is to study object oriented features introduced by Rust programming language and compare them to C++ object oriented features and find out the performance of Rust in object oriented paradigm. Rust is one of the latest programming languages. The designer of Rust programming language claims that it is better than C or C++ in terms of safety, memory usage and speed. The main focus is to compare Rust with C++ in object oriented programming paradigm. The comparison will be done by developing a system that follows all object oriented principles. These systems will be compared based on the performance, complexity and maintainability. Complexity and maintainability of a program will be computed from object oriented metrics. The tool used to compute object oriented metrics is Rust-Code-Analysis. The results of metrics will be used to understand the complexity, performance and maintainability of Rust code in object oriented paradigm. Small-scale programs, incorporating concepts such as polymorphism and inheritance, are implemented in both C++ and Rust. These programs are then passed through the Rust-Code-Analysis tool to obtain metric results. The findings indicate that Rust programs are more maintainable compared to C++ counterparts. It is mainly because there are only one functions implemented in small experiments. Furthermore, Rust demonstrates favorable performance when compared to C++. In terms of complexity, both Rust and C++ programs exhibit an average level. To further explore Rust's performance, complexity, and maintainability in the context of large-scale programs, additional large-scale programs will be developed in Rust and C++. After developing library management system in both rust and C++ the results of this large experiment shed light on the characteristics of Rust in the object-oriented programming paradigm. The findings indicate that C++ program is more maintainable than Rust program. Rust programs are not maintainable. Moreover, performance metrics indicates that rust does not perform well in object oriented paradigm. Furthermore, Rust programs are more complex than C++. It is mainly because of repetition of trait functions in rust program. Hence, Rust programming language is not suitable for this kind of object oriented projects.

Executive Summary

As Rust is one of the latest programming language that is emerging rapidly in the industry. Many researches have been done to study Rust behavior and its performance. There is a feature Rust Trait that is used to achieve object orientation in Rust. Traits show similar behavior to interfaces in C++. As object oriented programming provides many benefits to the programmers in terms of modularity and reusability. Rust also shows some object oriented features. The study aims to find out Rust performance in object oriented paradigm. To study the behavior of Rust, a Library management system will be developed using object oriented principles for comparing C++ and Rust.

We want to compare Rust language with C++ to compute Rust complexity and maintainability Therefore, we will create the Library Management system in C++ and Rust. In Chapter 4, we provide detailed information about the Library Management System, implemented in C++ using interfaces, and in Rust using traits as an alternative to interfaces. Subsequently, both systems are evaluated in Chapter 5, which describes the approach employed in this study. To assess Rust's complexity and maintainability, we utilize the Rust-Code-Analysis tool. This tool allows us to analyze Rust and C++ code and obtain metrics that provide insights into complexity and maintainability. By comparing the metric results provided by the tool, we can evaluate the complexity of Rust code relative to C++. The results obtained from the analysis and experiments conducted provide valuable insights into Rust's performance, complexity, and maintainability in the object-oriented paradigm.

The findings contribute to a deeper understanding of Rust's capabilities and suitability for object-oriented programming. Chapter 7 gives the detailed information of our experiments results. The evaluation of metrics between C++ and Rust provides valuable insights into their maintainability and performance. The maintainability index metric reveals that both languages present challenges in terms of maintainability, with C++ being slightly more maintainable than Rust. However, both languages exhibit negative values, indicating the need for improvement in this aspect. In terms of performance metrics, the volume metric indicates that Rust programs consume more memory than C++. This disparity can be attributed to the repetition of trait functions in Rust, which increases program’s cyclomatic complexity. Consequently, Rust demonstrates higher complexity compared to C++. Additionally, the difficulty and effort metrics indicate that Rust programs are more challenging to comprehend and require greater effort. The complexity arises from Rust's memory management and ownership models, which can be intricate to grasp. Rust's restriction on using the same variable in multiple places results in multiple variable declarations, further contributing to the program's complexity. Based on these results, it can be inferred that Rust may not be as suitable for object-oriented programming, as it introduces additional complexity. Overall, these findings shed light on the differences between C++ and Rust, highlighting areas of concern and potential avenues for improvement in Rust's object-oriented programming features.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Programming languages are used to maintain behaviour of computers. A programming language is a language that is used to create computer programs that direct a computer to carry out calculations and other tasks. Constructs for defining and manipulating data structures as well as for directing the course of execution may be found in programming languages. There are many programming languages used to write programs in object oriented paradigm. C++ is also an object oriented language. There are many new languages that compete with C++ in terms of object orientation and system programming[2]. Rust is a new and emerging programming language that is multi paradigm[7]. This language is created for memory safety purpose[6]. It also possesses certain object-oriented characteristics. In a very short time this language is getting popular. There is a keyword Trait used in Rust programming language that is similar to interfaces in C++ and helps in performing inheritance. Rust traits used to achieve object orientation[8].

This paper intends to study rust traits behaviour and how rust source code perform in object oriented paradigm. This can be done by developing a system using object oriented principles. After developing a system, these systems will pass to an object oriented metric evaluation tool to get the object oriented metrics results. These object oriented metrics will be helpful to compare Rust language with C++ in terms of object orientation and compare Rust complexity, maintainability and performance with C++.

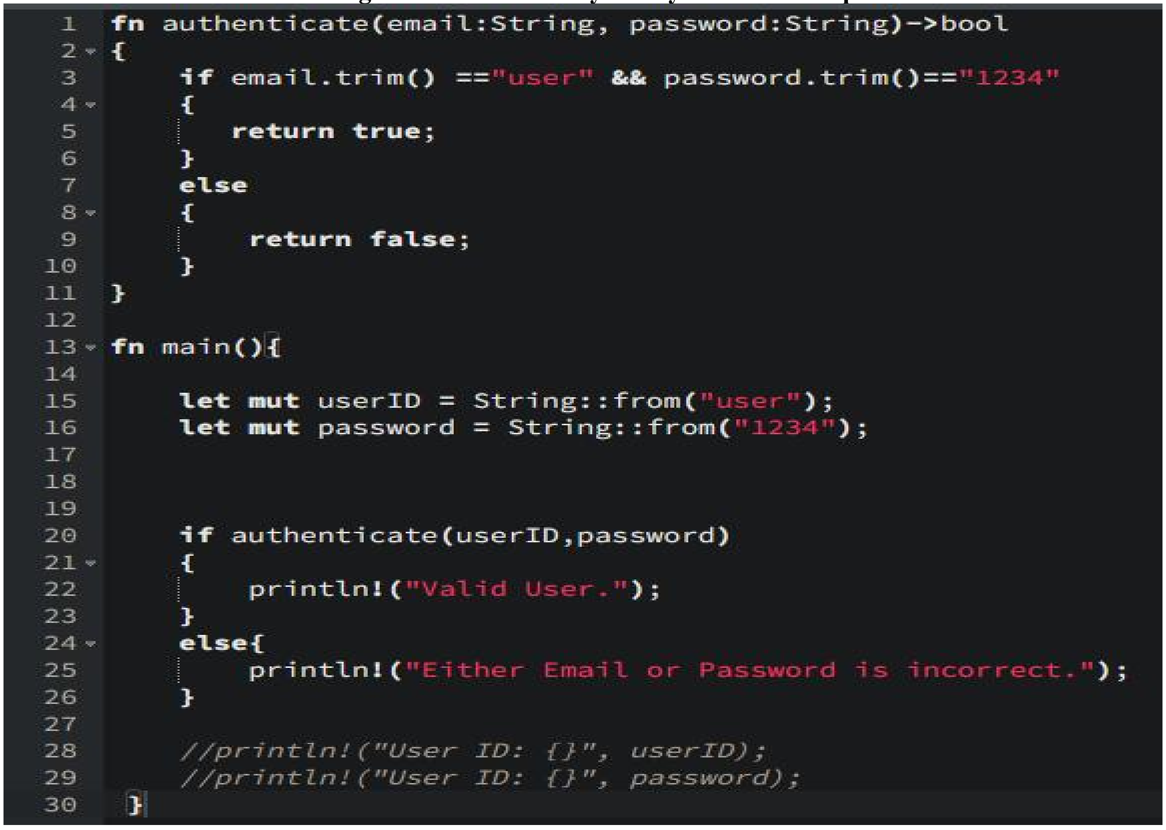
1.1 Rust Features

There are many features of Rust programming language. Out of these features memory management and traits are our main focus and will have a big role in our project so here it is discussed in a bit detail.

Memory management is one of the main features of rust. Memory management is manual, i.e. the programmer has explicit control over where and when memory is allocated and deallocated. In Rust, we can add abstractions without affecting the runtime performance of the code. It improves the code quality and readability of the code without any runtime performance cost. In C++ programming, there is an excellent improvement in error messages as compared to GCC. Rust goes one step further in case of clarity. Error messages are displayed with (formatting, colors) and also suggest misspellings in our program.

Furthermore, Rust offers memory safety. For languages that don't have a garbage collector, you need to explicitly allocate and free memory space. Thankfully, memory management is handled by the Rust compiler using the ownership model. Ownership works differently from a garbage collector in other languages because it simply consists of a set of rules that the compiler needs to check at compile time. The compiler will not compile if ownership rules are not followed. A Rust compiler will automatically insert a drop statement to free the memory. It uses the ownership model to decide where to free memory; when the owner goes out of scope, the memory is freed. Memory Management in Rust mainly consists of two things: Scope of an object and concept of Ownership. For example, see the code given in 'figure 1'. Here the last two print lines would not print the UserID and Password as the scope of these variables ended with the brackets of if and else. This is what Rust does i.e., it lets go of a memory space efficiently whenever it goes out of scope. The same code if written in C++ would have printed these two lines. Because Rust Ownership Model consists of many rules one of them states that: "There could be only one owner of an object". Here we tried to make two owners and it did not work. However, in order to make this work we use clones i.e., we simply make a clone of an object that we want assign to another object. In that way we don't violate the ownership rule.

Figure 1: Rust Memory Safety feature Example

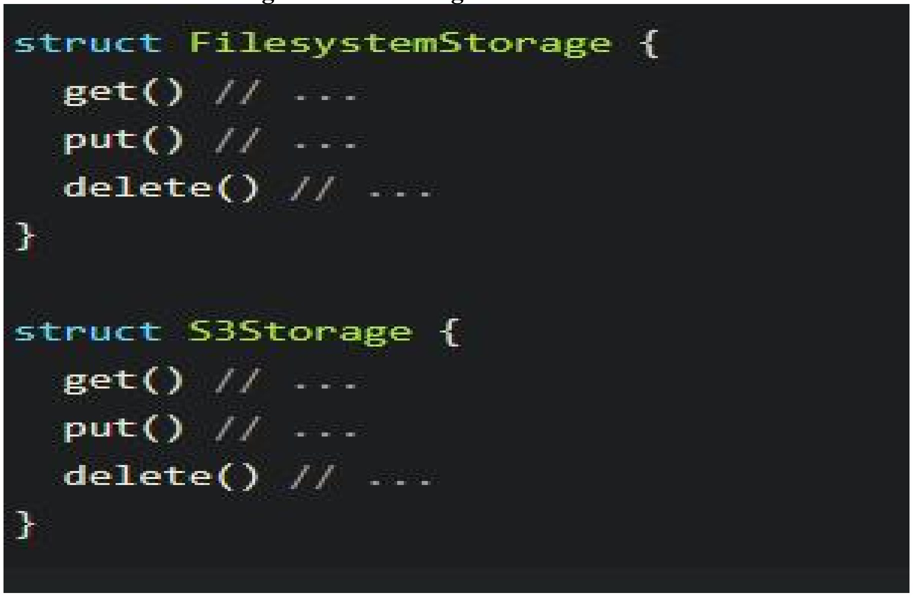


```rust
fn authenticate(email:String, password:String)->bool
{
    if email.trim() =="user" && password.trim()=="1234"
    {
       return true;
    }
    else
    {
        return false;
    }
}

fn main(){

    let mut userID = String::from("user");
    let mut password = String::from("1234");


    if authenticate(userID,password)
    {
        println!("Valid User.");
    }
    else{
        println!("Either Email or Password is incorrect.");
    }

    //println!("User ID: {}", userID);
    //println!("User ID: {}", password);
 }
```

One of the most talked about features of rust are Traits. Traits are used when we define a function and we want to use that function in other types with the same behavior i.e., implement that function, we use traits. So, in a way, traits are to Rust what interfaces are to Java and C++.

For example, Here, we have to create and implement the get put and delete function for every kind of file Storage system. As shown in 'figure 2'.

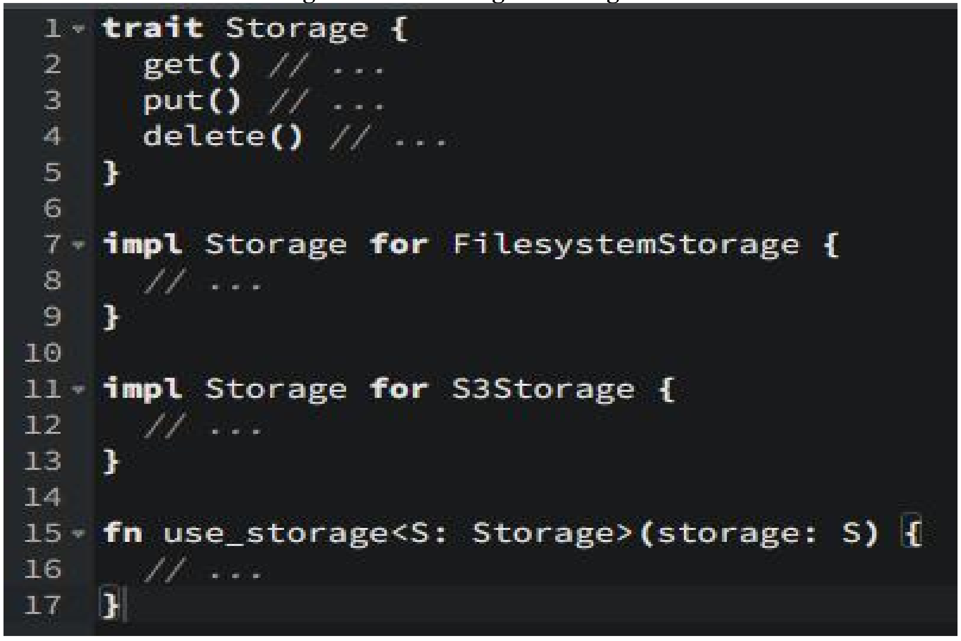
Figure 2: Rust Program Without Trait



```rust
struct FilesystemStorage {
  get() // ...
  put() // ...
  delete() // ...
}

struct S3Storage {
  get() // ...
  put() // ...
  delete() // ...
}
```

figure 3 shows that how to convert the code using traits.

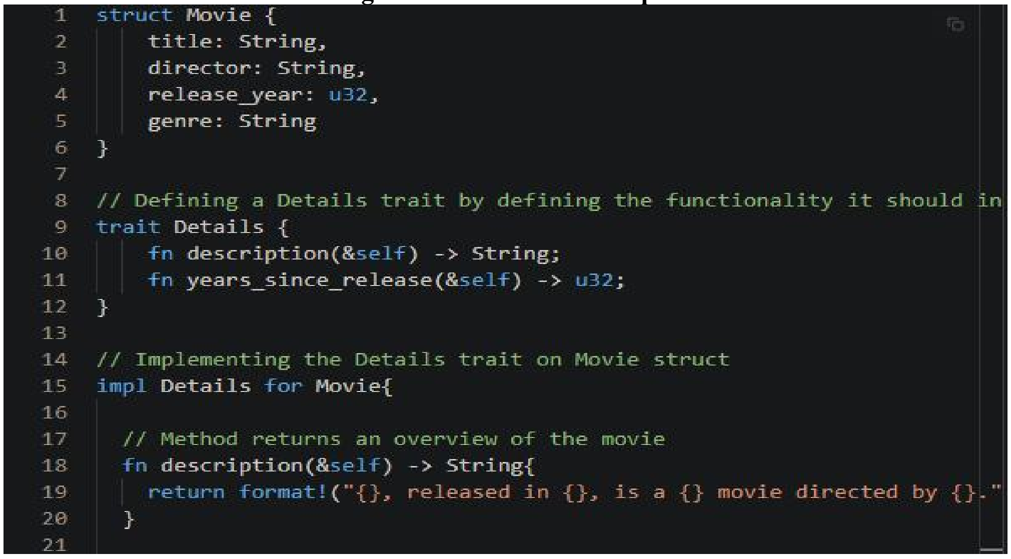
Figure 3: Rust Program using Trait



```
trait Storage {
  get() // ...
  put() // ...
  delete() // ...
}

impl Storage for FilesystemStorage {
  // ...
}

impl Storage for S3Storage {
  // ...
}

fn use_storage<S: Storage>(storage: S) {
  // ...
}
```

So, we simply made a Trait named storage and implemented it for every file system instance. This is just a simple and basic example of why traits are used and how they work. For better understanding, take a look at another example in 'figure 4':

Figure 4: Rust Trait Example



```
struct Movie {
    title: String,
    director: String,
    release_year: u32,
    genre: String
}

// Defining a Details trait by defining the functionality it should in
trait Details {
    fn description(&self) -> String;
    fn years_since_release(&self) -> u32;
}

// Implementing the Details trait on Movie struct
impl Details for Movie{

  // Method returns an overview of the movie
  fn description(&self) -> String{
    return format!("{}, released in {}, is a {} movie directed by {}."
  }

```

Here we have made a struct named Movie. Now every movie has some common details like their description and the date of releasing etc. So, instead of adding these components in every movie we simply made a trait called detail and implemented that for Movie.

1.2 Purpose of this Document

The main purpose of this project is to compare C++ programming language with a new and emerging language Rust. Rust programming language is getting popular due to its memory safety features and traits that shows similar behaviour to interfaces and used to perform inheritance. C++ is object oriented programming language and Rust is also showing object oriented characteristics. Furthermore, the study aims to compare object oriented metrics ‘see Table 1’ of C++ and Rust and how Rust behave in object oriented paradigm. The report helps those who wants to become a Rust developer in future.

1.3 Intended Audience

The intended audience of this project are the ones who are interested in learning new programming languages. The audience may be a CS or IT student or a professional programmer who wants to know about the new programming languages. The study help the audience to learn about the difference between new and old programming languages. The audience will get the idea about new programming language and how this language is better to use. This study will be helpful for those who want to learn Rust as this programming language is getting popular.

1.4 Definitions, Acronyms, and Abbreviations

Interface: Abstract class used to group together similar methods or functions.

Object Oriented Paradigm: Aims to integrate the benefits of modularity and reusability.

rust-code-analysis: Rust library used to evaluate object oriented metrics.

Modularity: Decomposition of software into smaller pieces.

Reusability: Use of existing components repeatedly.

SDG: Sustainable Development Goal.

IT: Information Technology.

CS: Computer Science

1.5 Conclusion

This chapter will give a brief overview of purpose of this study. Chapter two will give a detailed information about the the research statement, objectives and vision. Chapter three will give a detailed literature review about the researches that have done on Rust programming language in past. In later chapters experimental results will be share to give a clear understanding of the research.

Chapter 2: Project Vision

This chapter will give you a detailed overview about the problem statement, goals and objectives, and scope.

2.1 Problem Domain Overview

As Rust is a new emerging programming language. The Rust programming language is seldom ever used for study and limited researches and libraries are developed in Rust. There are no such research or experiments done on Rust in object oriented paradigm. Our research aims to study Rust programming language in object oriented paradigm and study how Rust traits used in object oriented programming. The Research aims to compare Rust with C++ in object oriented paradigm and comparison can be done based upon complexity, performance and maintainability indexes. For this purpose, rust-code-analysis tool will be used to analyze the difference between these languages based on different object oriented metrics 'see Table 1'. These metrics will be helpful to compute complexity, maintainability and performance of Rust and C++ source codes.

2.2 Problem Statement

The research main purpose is to study Rust programming language in object oriented paradigm. The study aims to compare Rust and C++ to evaluate Rust code complexity, performance and maintainability index. For evaluation of Rust code complexity, performance and maintainability indexes, a system or multiple systems will be developed in C++ and Rust using object oriented principles. After developing the systems, rust-code-analysis tool will be used to compare the performance of this system for both languages. More specifically, the research aims to find the answer of following question:

What will be the complexity and maintainability of Rust programming language in object oriented paradigm compared to C++?

2.3 Problem Elaboration

The research aims to study Rust programming language and compare it with C++ to know the performance, complexity and maintainability of Rust code. The study aims to compare the languages in object oriented paradigm. Object oriented programming provides many advantages in terms of modularity and reusability. Our study focuses on how Rust will behave in object oriented programming paradigm. In Rust, object oriented can be achieved by using Rust Traits. Traits are unique and used to achieve object orientation and also it shows similar characteristics to interfaces.

2.4 Goals and Objectives

The objectives of the study are:

- To study the programming languages C++ and Rust.
- Comparison of C++ to Rust programming language and see on what metrics is it better. For metrics list 'see Table 1'
- Comparison of Rust traits to Interfaces in C++ to find out Rust behaviour, complexity and performance in object oriented paradigm.
- Developing library management system in both languages for comparing Rust and C++ to see how Rust will perform in object oriented paradigm.

rust-code-analysis tool will be used for evaluating these systems that will be written in C++ and Rust for analyzing complexity, performance and maintainability in object oriented paradigm.The main focus of this study is to understand the performance of Rust in object oriented programming paradigm

2.5 Project Scope

The research main focus is to study Rust programming language and comparing Rust and C++ programming languages in order to know the performance of Rust programming language and give brief idea whether Rust suitable for future use or not. The comparison for our study can be done by developing a system or multiple systems in Rust and C++ programming languages and than compare the system or systems to find out complexity, performance and maintainability indexes of Rust code. Our study aims to study the behaviour of Rust in object oriented paradigm by computing object oriented metrics.

The tool used for the comparison is rust-code-analysis tool. This is the only tool available for Rust code evaluation. This tool also has a C++ support. This tool support eleven metrics for comparing different languages. These are the static metrics used to evaluate complexity and maintainability of source code. The tool will be helpful for our study to find out the performance of C++ and Rust source code. For list of metrics ‘see Table 1’.

2.6 Sustainable Development Goal (SDG)

The SDG are the guide for achieving a better, more sustainable future for everyone. They deal with issues like poverty, inequality, climate change, environmental degradation, peace, and justice, as well as other worldwide problems. In any situation, the SDG must be achieved with the help of all of society's innovation, expertise, technological advancements, and financial resources. There are seventeen sustainable development goals.

This research is targeting Quality Education of SDG area. The Quality Education SDG ‘as shown Figure5’ main aim is to provide equitable quality education and give opportunities to continue learning. The research aims to compare relatively new and old programming languages in order to identify the behaviour of Rust programming language in object oriented paradigm. The research aims to give education to the target audience by comparing the results of object oriented metrics of programming languages C++ and Rust. Moreover, the Research will give the clear idea whether Rust perform well in object oriented paradigm or not and motivate the target audience to learn new language to compete in IT market.

2.7 Conclusion

The main purpose of this study id to study Rust programming language and compare it to C++ in order to understand Rust behaviour in object oriented paradigm. In later chapters, details of different and relevant research papers will be shared that provides many related information about Rust programming language.

Figure 5: Sustainable Development Goals

This figure represents all the SDG's that can be target of a FYP

Table 1: List of Metrics

The list of Metrics used for comparison in our study.

Metrics	Description
CC	Cyclomatic complexity
SLOC	Source lines of code
PLOC	Physical lines of code
LLOC	Logical lines of code
CLOC	Comment lines of code
BLANK	Blank lines of code
HALSTEAD	A suite to measure tangible properties of software for example, volume, difficulty and effort "see Appendix A"
MI	Maintainability index "see Appendix A"
NOM	Number of methods
NEXITS	Number of exit points
NARGS	Number of arguments

Chapter 3: Literature Review / Related Work

The following section provides a detailed literature survey of previous researchers.

3.1 Definitions, Acronyms, and Abbreviations

Verbosity of Code: Use of longer words that is necessary to adequately express the intent of the code.

Cognitive Complexity: Measure of how difficult your code will be to read and understand.

Abstract Syntax Tree: tree representation of source code that tells the structure of the code.

Object Oriented Metrics: Units of measurement that are used to measure object oriented properties.

Garbage Collection: Process of automatic management of memory.

3.2 Detailed Literature Review

This section provides a detailed information about the work done by different researchers on Rust programming language that will be helpful for our research. Our research aims to study Rust programming language in depth and perform experiments to understand the behaviour of Rust in object oriented paradigm.

3.2.1 Evaluation of Rust Code verbosity, understandability and complexity

3.2.1.1 Summary of the research item

The purpose of this literature is to evaluate Rust code verbosity, understandability and complexity[1]. For that, researchers of this literature took nine different algorithms written in five languages. The languages they choose are C++, Python, JavaScript, TypeScript and Rust.they are comparing these algorithms by computing a set of metrics designed for Rust.They used a tool called rust-code-analysis[5] for evaluating those metrics because this is the only tool available to evaluate rust metrics. According to the data they gathered in their research, the complexity and maintainability of the Rust language are comparable to a group of widely used languages. There results show that Rust language is easily maintainable and less complex compared to C and C++. Source code written in Rust programming language shows lowest Cognitive complexity. This indicates that Rust code has a best understandability. These findings may help Rust programming language become more widely used in place of C in both open-source and commercial situations.

3.2.1.2 Critical analysis of the research item

The study[1] is a great initiative towards Rust programming language. Their results give us better understanding about Rust programming language and invoked many researchers to learn more about this language. The authors of this paper only study about the functional behviour of Rust programming language and they did not perform experiment on object oriented features of Rust like Traits. They perform experiment and statistical analysis to evaluate code verbosity, understandability and complexity of Rust programming language. Their experimental results shows that rust has lowest cognitive complexity which means rust code is easily understandable compared to other languages. Also, they show that Rust language is more organized and easily readable. Moreover, The maintainability of Rust and C++ did not significantly differ from one another. Maintainability of Rust programming language is similar to other languages but the authors only give conclusion about the projects that are written without object oriented features. Also, they compute these results on small scale projects and unable to say anything about if these results will be same in case of large scale projects. The authors did not claim that these results will also be same for large scale projects that follow all the object oriented features, so more research required in object oriented paradigm.

3.2.1.3 Relationship to the proposed research work

The study[1] is helpful for our research as they are also comparing old programming languages with the newly emerged Rust language. This literature only focuses on the Rust code verbosity, understandability and maintainability and they have done this work by comparing simple algorithms written in different languages. For this purpose they used a tool for evaluating the rust code and other languages. This information invoked us to study Rust programming language and find out how Rust language performs in object oriented programming paradigm. Our research main aim is to compare the behaviour of C++ and Rust in object oriented programming paradigm by computing different object oriented metrics. Also, our research aims to compare the behaviour of Rust traits and interfaces.

3.2.2 System Programming in Rust; Beyond Safety

3.2.2.1 Summary of the research item

In the world of new emerging programming languages Rust is really progressing as a safer alternative for C language[2]. The quality that makes it stand out is that it offers safety without overhead. In computer science, Overhead means time and space complexity and computation etc. take to run a program. Nowadays, overhead is a very big common problem in software and programs. Rust avoids overhead even without any collection of garbage memory. The perks of this language go further than this as Rust offers some of the functionalities that the traditional languages do not provide irrespective of them being safe or unsafe. This quality ensures the security and reliability of a system software to a very high level. These capabilities include zero-copy software fault isolation, efficient static information flow analysis, and automatic check-pointing. These qualities have been in the eyes of researchers for so long but they require high cost and complexities but they can be commoditized with Rust.

3.2.2.2 Critical analysis of the research item

The study[2] is a very simple, brief but important piece of article related to Rust. The reason is that this paper is not only giving us the list of capabilities that makes Rust better than C language but it also shed lights on how are those qualities achievable through this programming language. Overhead (the excess time and complexity) is a huge problem in all of the programming language. The paper tells us that Rust solves this problem but the question of how Rust is exactly able to do that remains the same. Therefore, a little more explanation would have been better in order to understand the process. Some more qualities are mentioned i.e., zero-copy software fault isolation, efficient static information flow analysis, and automatic check-pointing. Although its remarkable how Rust has these benefits but the paper lacks some sort of explanation to these.

3.2.2.3 Relationship to the proposed research work

The study[2] is helpful for our research as it hits our main aim of this project i.e., to compare Rust with C++ programming language. The paper mentions that Rust is able to avoid runtime as well as garbage collection overhead which is not a case in other programming languages. This paper helps us narrowing down the features we want to look for while comparing Rust with other programming languages.

3.2.3 Translating C to safer Rust

3.2.3.1 Summary of the research item

A relatively new programming language called Rust is designed to create secure and effective system-level applications. It is specifically made to replace unsafe languages like C and C++ in the coding ecosystem and has a robust type system that enables verifiable memory and thread safety[3]. There is a sizable current C and C++ code base that would benefit from being rewritten in Rust in order to eliminate a whole class of potential defects. However, manually converting these programs to Rust is a difficult task.

The automatic conversion of C programs into safer Rust programs[3]. That is, into Rust programs that enhances the safety guarantees of the original C programs that is the topic of this literature. We investigate the root reasons of error handling in translated programs in great detail, as well as the relative effects of addressing each source. We also present, assess, and discuss a novel method for automatically eliminating a specific source of risk. In contrast to programs that were developed in Rust initially, this work presents the first empirical investigation of problems in translated Rust programs as well as the first method for automatically deleting error related causes.

3.2.3.2 Critical analysis of the research item

The article translates the C languages to Rust. As the article claims that it is a more efficient language. The technical aspects of two programming languages, such as performance benchmarks, are among the things they frequently consider when evaluating them. It is considerably simpler to overlook bugs and errors in your code. Rust's code-validating process is much stricter than C++'s, making it a statically-typed language on steroids. One of the greatest reasons in favor of Rust is that it offers a stronger approach to security and code quality than C/C++. For instance, Rust's ownership system makes sure that no two threads can reference the same data without borrowing it or taking ownership of it, making it unavailable to other threads and preventing data races. The language you select ultimately comes down to your comfort level. You won't have to suffer from your choice to use Rust or C++. Both are reliable, widely used, and respected languages that will accomplish the job. But more research required on the categories of unsafety that were mentioned in the paper.

3.2.3.3 Relationship to the proposed research work

We are developing a program in C++. We will translate the same program in Rust. As we want to compare and study these languages. We will compare the behavior of interfaces of C++ and traits of Rust. From a high perspective, C++ has a larger community, a broader range of applications, more frameworks, and is well known throughout the coding industry. The statically-typed properties of Rust, on the other hand, make it better for security, speed, and preventing incorrect or dangerous code. So we will develop same codes in both languages and we will make their comparison on some tools. We have selected some metrics. On the basis of these metrics we will compare the behavior of these languages.

3.3 Literature Review Summary Table

The summary of the literature review is given in the form of table to give a brief overview of the studies that have done in past on Rust programming language. Our research aims to study Rust programming language in depth and perform experiments to understand the behaviour of Rust in object oriented paradigm. Different literature helped us narrow down the qualities of Rust.

Table 2: Literature Review Summary

This is brief description of Literature Review

No.	Name, reference	Inventor	Year	Description
1.	Evaluation of Rust code verbosity, understandability, and maintainability [1]	L. Ardito, L.Barbato, R. Coppola and M.Valsesia,	2021	The literature aims to evaluate the Rust source code complexity and maintainability.
2.	System programming in Rust: Beyond Safety[2]	A. Balasubramanian, M. Baranowski, A. Burtsev, A. Panda, Z. Rakamaric, L. Ryzhyk	2017	The literature aims to practically prove that System programmers can develop strong security and reliability methods using Rust.
3.	Translating C to safer Rust[3]	M. Emre, R. Schroeder, K. Dewey, B. Hardekopf	2021	The literature aims to study the unsafety in translated Rust codes and describe how unsafe codes can be removed automatically.

3.4 Conclusion

Study of Rust is quite interesting as it is one of the newest programming languages. Our project aims to compare Rust with C++ to see which programming language is better on what parameters. Moreover, we intent to study Traits in Rust and compare it with interfaces. We dug into some research papers and got major knowledge about Rust language. Our research paper helped us understand where to start digging more information. Moreover, different literature helped us narrow down the qualities of Rust. Therefore, it assists in the comparison of languages. With the results of these articles and the use of rust-code-analysis tool it is evident that our project is going in the right direction. The papers tell us that Rust is more maintainable and avoid overhead and provide safety. Our project goal is to check these claims by comparison and provide proof of these abilities.

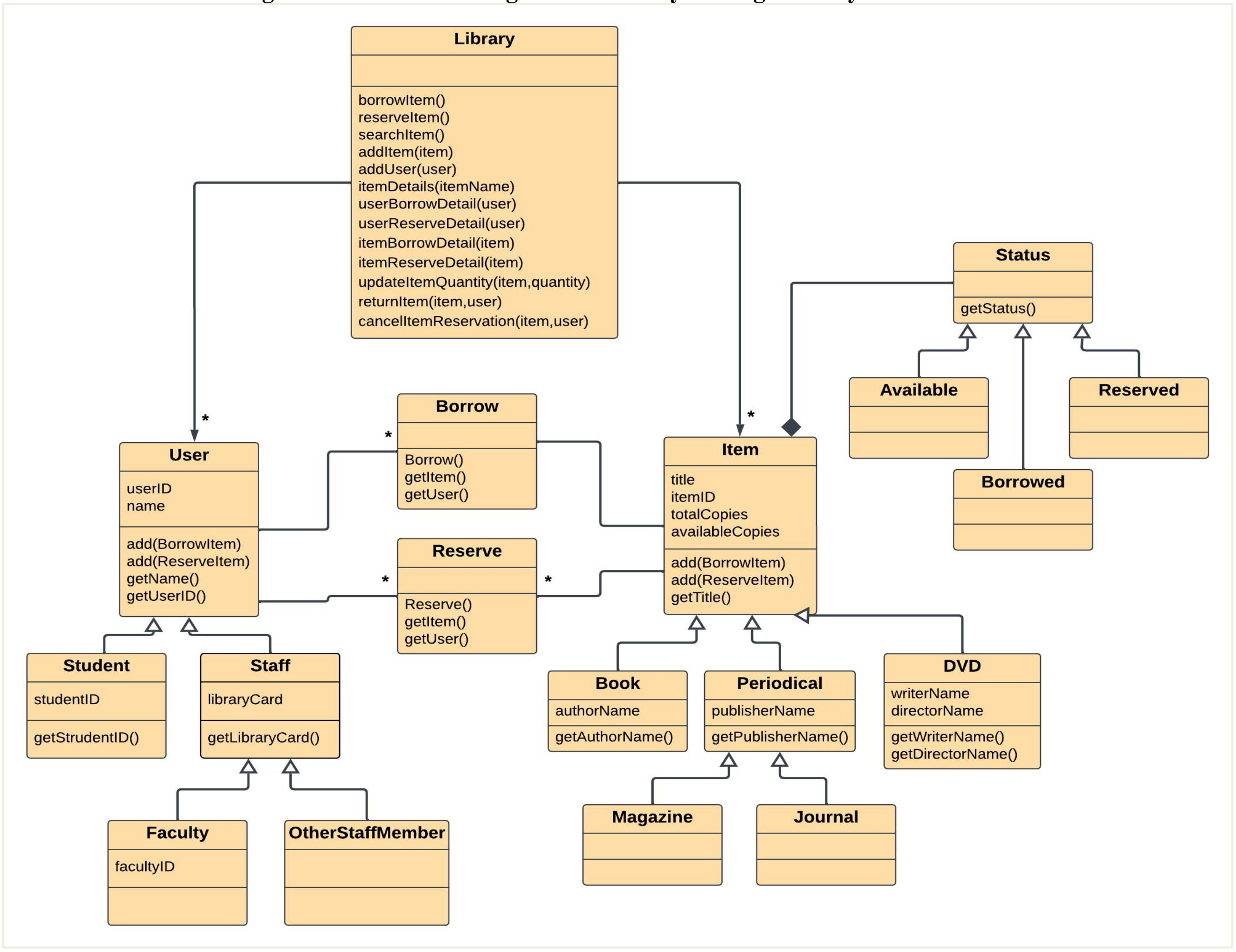
Chapter 4: Software Requirement Specifications

As this is the research project and for the research a system is required for the comparison. The research aims to compare Rust and C++ in object oriented paradigm and wants to find the performance of Rust programming language in object oriented paradigm.The research aims to find out does Rust suitable for object oriented programming. For this purpose, a system will be developed in both languages for comparison. For this research we will be developing Library management system for the comparison. The reason to choose this system is because this system is following all the object oriented features. As our aim to compare Rust and C++ in object oriented paradigm so this system will help us to study behaviour of Rust in object oriented paradigm. This chapter will give the features of our system so that readers can find out the scope of our system.

4.1 UML Class Diagram

Before developing library management system, we design a UML class diagram. UML design helps in developing the system easily. In this diagram, library is the main class and it has all the list of users and items. A user can be a Student, Faculty and Other staff members. An item can be a Book, Periodical and DVD. Furthermore, a Periodical can be a journal and magazine. There are two association classes i.e Borrow and Reserve. These association classes helps in implementing many to many relationship. Also, there is a Status class that is used to identify the status of an item. Figure 6 shows the detail of library management system that is used in this research.

Figure 6: UML class Diagram of Library Management system



4.2 Features

This section will tell the features of the system so that readers will know the scope of the system. A piece of software known as a library management system is responsible for handling the majority of a library's administrative duties. Systems are necessary for overseeing patron interactions and asset collections in libraries. Thanks to library management systems, libraries can keep track of the books and their checkouts as well as the subscriptions and member profiles.Another feature of library management systems is the upkeep of the database that is used to record new books and track borrowed books with their due dates. The main features of this system are:

- Members of the library can search for any book
- Members can borrow books from library
- System will be able to retrieve information about the books available
- System will be able to retrieve information about the borrower of the book.
- Librarian can enter new books
- Librarian can update quantity of the books.
- Librarian can update status of the book.
- Librarian can add new members.
- Members should be able to reserve a book.
- A member can borrow maximum 5 books at a time.

Chapter 5: Proposed Approach and Methodology

This chapter reports the proposed approach and methodology for the research. The research aims to find behaviour of Rust in object oriented paradigm. This can be done comparing Rust and C++. The aim of the research will be achieved by collecting quantitative data. This data will be collected by performing experiments on Rust and C++ programs. After getting the experimental results, statistical analysis will be performed to know the performance, complexity and maintainability of Rust and C++ program in object oriented paradigm.

5.1 Objects

For the research it is necessary to develop a system by following all the object oriented principles to analyze Rust code in object oriented paradigm and compare it with C++ programming language. Detailed information about the system that we are developing for our research is given in chapter 4. The reason behind choosing this system is because it will be developed using all the object oriented programming principles.

5.2 Tool

Rust-Code-Analysis tool will be used for computing object oriented metrics for Rust and C++ programs. It is a Rust library[5] that evaluates the Rust, C++ and many other languages and extracts a set of eleven object oriented metrics. For list of supported metrics ‘see Table 1’. The tool calculate object oriented metrics of source code. The reason to choose this tool is because of its features[5]. The tool either except one file or a complete directory containing multiple files. This is the advantage of this tool as whole directory will be computed. The tool give the results in JSON format so metrics results can be easily readable. The results of object oriented metrics extracted from the tool are divided into two sections.

- First section of the results are the overall results of the whole source program.
- Second section of the results are for each function and classes of the program.

Both sections of the metrics results will be helpful for the research. Whole program results will be helpful to compare the whole system of Rust and C++. Also, each function’s metrics results of Rust and C++ can be compared to understand Rust functions performance compared to C++ functions. The object oriented metrics results are considered as experimental data. This experimental data will be helpful for comparing Rust and C++ programming languages in object oriented paradigm. The comparison can be done by comparing the results of metrics of Rust and C++ programs.

5.3 Analysis Procedure

The experimental data will be used for comparative analysis. To collect the experimental data, the system developed in Rust and C++ will pass to the rust-code-analysis tool which will give the results of object oriented metrics. As experimental data is in JSON format, it will be converted to a tabular form. Tabular form will help to clearly identify metrics result. The criteria to compare two languages using experimental data are depends on metrics values. For any metric, one language might show maximum value result for that metric while other language show minimum value result. Different metrics are of different nature. Some metrics required minimum value while some required maximum value for a program to be considered as a good software.

5.3.1 Metrics Description

Each metrics are of different nature and some metrics required maximum value while some required minimum value for a program to be considered as a good software. Detailed information about the metrics are given in' Table 3'. Also, the table describes the values of metrics as minimum or maximum. Maximum or Minimum value mean that the metric value should have maximum or minimum value for one language than other language.

Table 3: Metric Description Table

This is brief description metrics and their expected acceptable values.

Metrics	**Description**	**Value**
SLOC	It counts the number of lines in a source file. This shows the size of the program. Its value must be low	Minimum
PLOC	It counts the instructions contained in a source file. Its value must be low.	Minimum
LLOC	It counts the logical lines of code that contains statements.	Minimum
CLOC	It counts the comment lines of code.	Maximum
BLANK	It counts the blank lines of source code.	Maximum
CC	It calculates complexity of a program by determining control flow of the program. Its value should be low.	Minimum
HALSTEAD	It is a suite that provides information about the effort, difficulty, volume of the source code.	Minimum
NOM	It counts the number of methods in a source code.	Maximum
NEXITS	It counts the number of exits from a function	Minimum
NARGS	It counts the number of arguments in a function	Minimum
MI	It evaluates the maintainability of source code	Maximum

SLOC, PLOC,LLOC metrics describes different kinds of lines of code. Lines of code metric value should be minimum because these metrics represent the size of the program.

CLOC and BLANK are the comment and blank lines of code respectively. These metrics will be used to show that how much code is organized and how much code is readable that's why there values should be maximum. Maximum number of blank lines shows that code is more readable and maximum number of comment lines represent code is more organized and easily understandable. But there values can be used only to show how much code is organized. There values are directly proportional to SLOC, PLOC and LLOC lines of code metrics. Less SLOC values ultimately reduce the BLANK and CLOC metric values.

CC and HALSTEAD metrics are used to determine the complexity of the program. Minimum value represents that program is less complex that's why there values should be minimum.

NOM metric value should be maximum because more number of methods or function shows that code is more modular.

NEXITS and NARGS are used to describe the exit points and arguments of a functions. There values should be minimum. Minimum values represent that the programs are less complex and easily maintainable.

MI used to describes how much maintainable the source code is. Its value should be maximum because high value shows that the program is more maintainable.

5.4 Conclusion

These methods will be helpful for the research because the research main aim is to study Rust programming language in object oriented paradigm. Quantitative data of Rust and C++ programs will be collected by performing the experiments and these experimental results will be useful to compare Rust and C++ programs in object oriented paradigm. These Results will give the clear idea what will be the performance of Rust in object oriented paradigm.

Chapter 6: Implementation and Test Cases

This section will give the details about the work that we have done so far. First we studied different research paper.Than, we design the approach to compare the Rust and C++ programming languages. By following the methodology, we implement a simple program following the approach mentioned in chapter 5 and gather the results that will be discussed in Chapter 7.

6.1 Implementation

C++ supports explicit allocation and deallocation with **new** and **delete**, as well as manual memory management. It is prone to memory-related issues like memory leaks, dangling pointers, and null pointer dereference. By imposing stringent ownership, borrowing, and lifespan constraints, Rust emphasizes memory safety. It becomes memory-safe without a garbage collector by preventing typical memory-related issues at compile time. Although C++ includes built-in concurrency and multi-threading capability, it does not have built-in thread safety methods. It is the duty of the programmer to provide good synchronization and prevent data races. On the other hand, a strong concurrency mechanism with integrated thread safety is offered by Rust. Through its ownership and borrowing scheme, it imposes stringent ownership and borrowing regulations, assuring thread safety at compile time. Keeping these things ion mind we implement some programs.

For this research, we perform some experiments to understand the complexity, maintainability and performance in object oriented paradigm. In rust, structs are similar to class in C++. Structs are used to encapsulate behaviour into single unit. Furthermore in rust, polymorphism can be achieved using traits. Traits act as an interface and defines a set of functions that a struct must implement. Traits can give the default implementation of function and structs can override that function. Moreover, there is no concept of inheritence in rust programming language. But if a developer wants to implement inheritence, one can achieve this using the concept of composition. In this experiment, we use these two approaches of rust programming language to implement polymorphism and inheritence. First we implement inheritence and polymorphism individually so that we can understand the individual performance of rust inheritence and polymorphism. After that we use these concepts to develop the library management system.

6.1.1 Implementation of Inheritence

In this experiment, Person class is used to to understand the inheritence in rust. In rust, inheritence is implemented using the concept of composition. We design a very simple program for this purpose. There is a base class name Person that holds the information about a person like name and age. There is only one method i.e print details. This method is used for printing details of a person. There are two derived classes. One is a Student class and other is a Teacher class. Both classes are inherited from Person class. Student holds a data member GPA and a Teacher has a telephone extension. In rust, this type of inheritence can be achieved by using the concept of composition. Student and Teacher classes both holds the object of Person class. This is how Student and Teacher classes can call the functions of Person class. UML class diagram of this experiment is given in “Appendix B”.

6.1.2 Implementation of Polymorphism

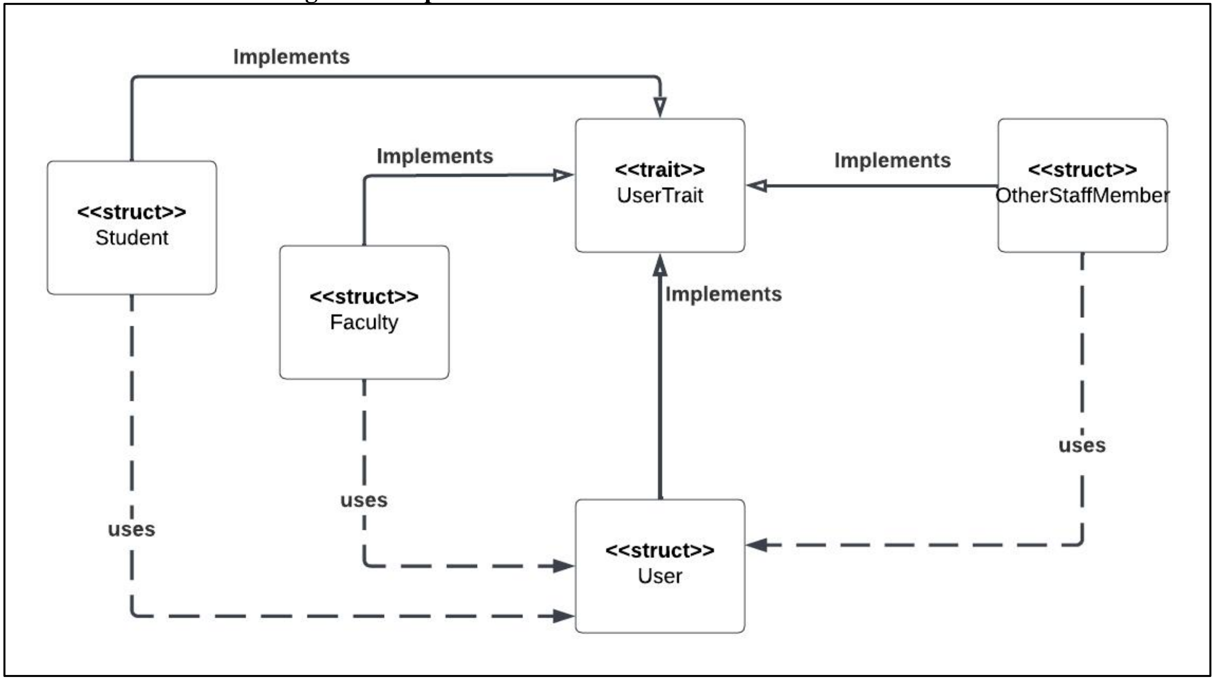
In this experiment, shape class is used to understand the polymorphism in rust. In rust, polymorphism is implemented using rust Traits. We design a very simple program of shape hierarchy that is calculating area of different shapes. In this design, Shape is a base class. In

C++ program, pure virtual functions are use to implement this this hierarchy. But in rust program, Shape is not implemented as Struct. Shape is defined as trait and it has only one function i.e calculate area. In rust program, shape class does not give the implementation of the function that is calculating area. There are three different shapes i.e Triangle, Rectangle and Circle. Each shape has different formula for calculating area. Triangle, Rectangle and Circle are defined as struct and all are implementing the Shape trait and gives different definition for calculating area. This is how runtime polymorphism will be achieved in rust programming language. UML class diagram of this experiment is given in "Appendix B"

6.1.3 Implementation of Library Management System

To implement this system in rust There are two traits. One is for User and other is for Item. User trait holds all the common functions that each user must have. Similarly Item traits holds all the functions that each item must implement. Since In Rust, default method implementations are not supported by traits. The implementation of every method in a trait must be specified explicitly by each type that implements the trait. There is a one base struct for User and Student, Faculty and OtherStaffMember structs uses the User struct object and each class is implementing the User Trait. 'figure 7' shows how User Hierarchy is implemented in rust.

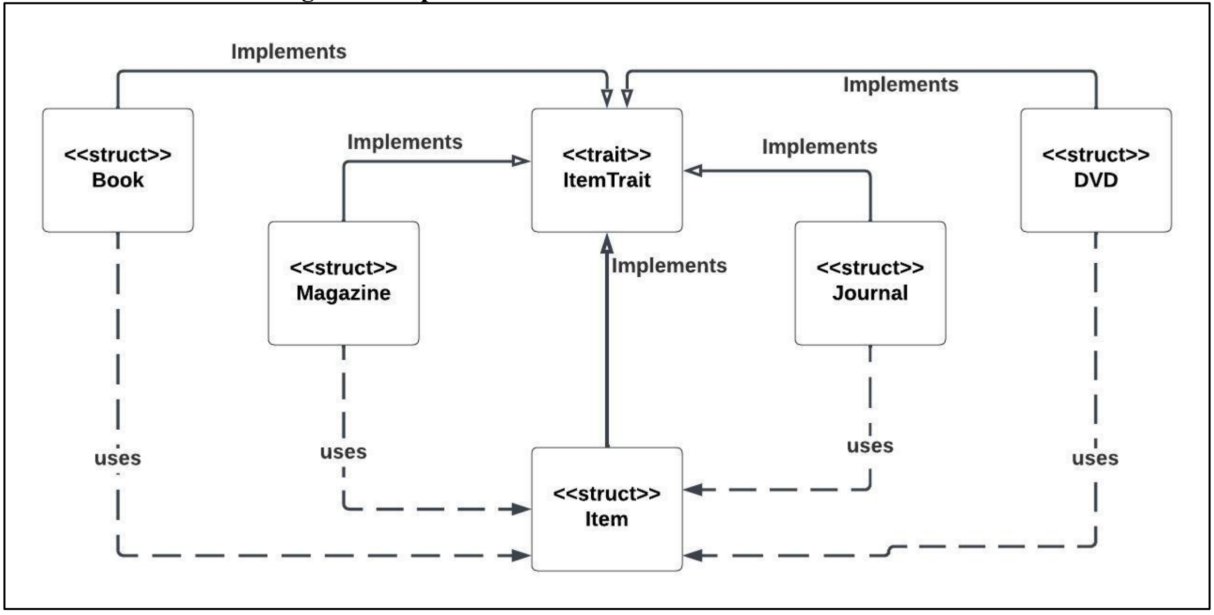
Figure 7: Implementation of Inheritence for User in Rust



UserTrait has the common functions and User, Student, Faculty and OtherStaffMember are implementing those functions. Moreover, Student, Faculty and OtherStaffMember uses the User struct for accessing the properties that are common to every users.

Similarly, there is a one base struct for Item and Book, Magazine, Journal and DVD structs uses the Item struct object and each class is implementing the Item Trait. 'figure 8' shows how Item Hierarchy is implemented in rust. ItemTrait has the common functions and Item, Book, Magazine, Journal and DVD are implementing those functions. Moreover, Book, Magazine, Journal and DVD uses the Item struct for accessing the properties that are common to every items. Moreover, Status hierarchy is implemented using Traits. Status is defined as Trait and Available, Borrowed, Reserved classes implementing the Status Trait.

Figure 8: Implementation of Inheritence For Item in Rust



6.2 Conclusion

In this chapter, we presented the implementation and test cases for comparing the Rust and C++ programming languages in terms of complexity, maintainability, and performance in the object-oriented paradigm. We described the two approaches that we used to implement polymorphism and inheritance in Rust, one using traits and the other using composition, and provided details on how we developed a library management system using both approaches. We also included UML class diagrams to illustrate the different implementations. Overall, this chapter provides a detail of the our work and sets the stage for the discussion of results in the next chapter.

Chapter 7: Experimental Results and Discussion

This chapter will provide the detailed information about the results that were collected by following the methodology described in previous chapter. These experiments are performed on a very small programs that are written in C++ and Rust.

7.1 Experiment Results Discussion

For all experiments, we gather object oriented metrics results to compare rust program with C++. The tool gives the results of many metrics but the metrics that are helpful for the experiment are Maintainability index, Performance metrics and Cyclomatic Complexity. For Performance Metrics we use HALSTEAD volume, Difficulty and effort metrics. Maintainability index metrics helps in identifying maintainability of a program. Performance metrics helps for comparing the performance of two programs and Cyclomatic complexity gives the program complexity results. For UML class diagram and detailed metrics result 'see Appendix B'.

7.1.1 Inheritence Results

SLOC,PLOC and LLOC metrics values for rust program are maximum than C++ program. Rust program required more lines of code to achieve the same functionality but C++ programs needs less lines of code. It shows that Rust inheritence programs are larger in size than C++ program.

NARGS and NEXITS metrics values for Rust program are maximum than C++ program. C++ programs need less number of arguments to achieve the functionality. It shows that inheritence in rust are more complex than C++.

MI metric values for Rust program is maximum than C++ program. Maximum value shows that Rust program is more maintainable than C++ program in object oriented paradigm. These results shows that if new functionality required than Rust do not require any major changes because Rust programs are more maintainable but in case of C++ it is complex to add new functionalities because it is less maintainable according to the metrics values obtained.

In case of CC metrics, sum for Rust and C++ program is same for inheritence. On the other hand, HALSTED metric suite has many features and all the features shows that Rust program values are minimum than C++ program. HALSTED Length and volume of Rust program is less than C++ program which shows that Rust programs has small size than C++ programs to achieve the same functionality. Also, HALSTED difficulty shows that Rust program are less difficult to write than C++ programs and HALSTED effort values shows that rust programs can be written with minimum effort. HALSTEAD time shows that how much estimated time required to write a program and Rust program shows that lesser time required to write a program than C++. HALSTEAD bugs values shows that C++ programs are expected to have more bugs than Rust program. HALSTEAD level values depend on the difficulty values. If difficulty value increases it means that level of the program decreases 'see Appendix B'. So Rust has high leveled than C++ program and Rust program is less difficult to write than C++. HALSTEAD vocabulary shows that Rust inheritence program use greater number of unique operators than C++. Rust use more keywords to achieve inheritence. All these HALSTEAD metric features shows that rust inheritence program is slightly complex than C++. 'Table 4' gives the summary of inheritence experiment results.

Table 4: Summary of Inheritence Experiment

Metrics / Programming Language		C++	Rust
Maintainability Index Metric	MI	66.7858448744038	67.60536547302534
Performance Metrics	Volume	810	691.8949
	Difficulty	17.809523	13.909090
	Effort	14425.7142857	9623.629709
Cyclomatic Complexity	CC	11	11

7.1.2 Polymorphism Results

SLOC,PLOC and LLOC metrics values for rust program are minimum than C++ program. C++ program required more lines of code to achieve the same functionality but Rust programs needs less lines of code. It shows that Rust programs are lesser in size than C++ program and Rust programs can achieve same functionality in less number of lines than C++.

NARGS and NEXITS metrics values for Rust program are minimum than C++ program. C++ programs need more number of arguments to achieve the functionality but in case of Rust minimum number is enough to achieve the functionality. It shows that Rust programs are less complex and more maintainable in object oriented paradigm because more number of arguments are very difficult to maintain and its very difficult to find any bugs in the program. Similarly, multiple exit points makes the program more complex and makes it difficult to find any errors in the program.

MI metric values for Rust program is maximum than C++ program. Maximum value shows that Rust program is more maintainable than C++ program in object oriented paradigm. These results shows that if new functionality required than Rust do not require any major changes because Rust programs are more maintainable but in case of C++ it is complex to add new functionalities because it is less maintainable according to the metrics values.

HALSTEAD and CC metrics for Rust program are minimum than C++ programs. In case of CC metrics, sum for Rust program is minimum than C++ program which shows that Rust is less complex than C++ program in object oriented paradigm. On the other hand, HALSTED metric suite has many features and all the features shows that Rust program values are minimum than C++ program. HALSTED Length, estimated program length and volume of Rust program is less than C++ program which shows that Rust programs has small size than C++ programs to achieve the same functionality. Also, HALSTED difficulty shows that Rust program are less difficult to write than C++ programs and HALSTED effort values shows that rust programs can be written with minimum effort. HALSTEAD time shows that how much estimated time required to write a program and Rust program shows that lesser time required to write a program than C++. HALSTEAD bugs values shows that C++ programs are expected to have more bugs than Rust program. HALSTEAD level values depend on the difficulty values. If difficulty value increases it means that level of the program decreases 'see Appendix B'. So Rust has high leveled than C++ program and Rust program is less difficult to write than C++. HALSTEAD vocabulary shows that Rust has less number of unique operators than C++. C++ has more keywords that are used in a program. All these

HALSTEAD metric features shows that rust is less complex than C++. ‘Table 5’ gives the summary of polymorphism experiment results.

Table 5: Summary of Polymorphism Experiment

Metrics / Programming Language		C++	Rust
Maintainability Index Metric	**MI**	44.413232739021595	60.15527591255034
Performance Metrics	**Volume**	1956.565652372609	1010.5009697246602
	Difficulty	30.825	17.5
	Effort	60311.13623438567	17683.766970181554
Cyclomatic Complexity	**CC**	22	16

7.1.3 Library System Results

The results for library management system are given in ‘Table 6’. Table gives the information that C++ and Rust both shows the negative values of Maintainability index metric. It shows that both programs are difficult to maintain and C++ is slightly more maintainable than C++. From ‘Figure 6’ , User and Item are the base classes and Rust implements them using Traits. Traits only give the prototype of functions and all the derived classes has to implement these functions. If a programmer wants to add some new functionality, than all types or structs must implement that new functionality. This approach requires more effort and making the program more complex. That is the reason maintainability index shows negative values. Furthermore, from ‘Figure 7’ and ‘Figure 8’, derived classes uses the object of base class and some functions of derived classes are dependent on the functionality of base class in rust implementation. If base class changed, than it is very difficult to maintain the derived classes. This is also the reason that maintainability index shows negative values. Since C++ maintainability index value is greater than Rust program. It shows that here C++ is slightly more maintainable than Rust. In case of C++, there are many control structures used in the program. These control structures are making the program more complex and it is more difficult to maintain. That is why c++ program shows negative values of maintainability index.

Table 6: Summary of Library System Results

Metrics / Programming Language		C++	Rust
Maintainability Index Metric	**MI**	-56.08644654835	-100.78426641805
Performance Metrics	**Volume**	40207.581778643	56721.2472335881
	Difficulty	93.434628975265	101.608695652173
	Effort	3756780.4854801	5763371.94716936
Cyclomatic Complexity	**CC**	238	396

Volume metric shows that Rust program takes more memory than C++. In Rust program, the functions that are part of trait needs to be repeated for every struct that implements the trait. From ‘Figure 7’ and ‘Figure 8’, it can be seen that UserTrait and ItemTrait functions are implemented by all structs. If there are ten functions than each type has to implement them all ten functions. Hence rust takes more space than C++ because of repetition of trait functions.

Difficulty and Effort metric shows that rust program is more difficult to understand than C++ and rust program required more effort. It is because memory management and ownership models of rust are very complex to understand and due to this rust do not allow to use the variable in multiple places. Due to this problem, there are multiple declaration of same variable that makes it difficult to maintain. Moreover, rust do not allow to use a variable that pass to some function and programmers are restricted to modularize the code because once the variable pass to the function it will not be useful for that particular function. It is because of its ownership model. In Rust it is very difficult to modularize the code. That is why rust program is more difficult than C++.

Repetition of function makes the program more complex and it also increases the cyclomatic complexity of program. Rust program is more complex than C++. It is mainly because rust ownership model. Due to this model, one function is doing all the jobs while it can be divided into small functions but rust do not allow this due to ownership model. Same variable cannot pass to multiple functions. It makes the rust program more complex than C++. As function is doing all the jobs, it makes the code block large. It also makes the program more complex. Furthermore, rust program uses many control structures and there are functions that are very large with large number of control structures. This makes the rust program more complex than C++.

7.2 Conclusion

To conclude, from the small experiments that is performed and from the results of library management system results it was found that rust program is more complex than C++. It is mainly because in Rust program there are repetition of functions making it more complex than C++. Repetition of functions takes more memory and hence it makes the program slow. There are many control structures used in Rust program because one variable is not valid when used multiple times due to its ownership model. Hence rust is not suitable for object oriented programming.

Chapter 8: Conclusion

To conclude, the report tries to investigate whether Rust actually offers the advantages it advertises or what benefits it has by contrasting it with another famous and common programming language C++ by creating a similar program in both of these languages to compare them all practically in terms of object oriented programming metrics. This experiment compares C++ and Rust to determine which programming language performs better across many metrics. Additionally, we intend to research whether Rust is suitable for object oriented programming or not. Rust uses Traits and many more features to achieve object oriented programming. Additionally, the literature assisted us in defining Rust's characteristics. As a result, it aids with language comparison.

For the purpose of comparing these languages a code is intended to be developed in both C++ and Rust. Some small examples related to polymorphism and inheritence was developed to check the rust program performance. It can be seen from the results of small experiments that rust is suitable for object oriented programming. These small experiments shows positive results because there are only one functions implemented in rust programs as they are the sample functions. These results are not enough to conclude anything that is why it is necessary to implement a large program. After that a Library management System was developed. The main findings of this experiment are as followed:

- Maintainability index values shows that, Rust program is less maintainable than C++. It is mainly because of repetition of functions. If new functionality added in the trait, than all types must implement that new functionality. This makes the program less maintainable.
- Cyclomatic Complexity metric shows that C++ program is less complex than Rust program. The complexity of rust program increases mainly because of using many control structures in rust. Rust do not allow to use same variable in multiple places like passing the variable to another function makes it invalid for the current function. For this reason there is no modularity in the function and one function has to do many jobs. This increases the program complexity.
- Performance metrics like Volume, Difficulty and Effort metrics shows that C++ program perform well than rust program. Volume of rust program increases because rust has repetition of functions. Due to this repetition rust takes more memory than C++ program. This makes the rust program slower than C++ program. This repetition also makes the rust program difficult and and extra effort required to add new functions in rust program.

From all these results, it is concluded that rust is not suitable for object oriented programming. The main reason is because there are repetition of functions, hence making rust program more complex. In Object oriented programming, Inheritance enables the development of specialized classes while preserving code coherence and minimizing duplication. This is easy in C++ but in case of Rust, traits are not suitable for inheritence because every function of traits needs to be duplicated for every type that is implementing the trait. As rust shows duplication of code that is why Rust programming language is not suitable to for object oriented programming.

References

[1] L. Ardito, L.Barbato, R. Coppola and M.Valsesia, “Evaluation of Rust code verbosity, understandability and complexity”,National Center for Biotechnology Information, Feb.26,2021.[Online].Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7959618/#sec-17title [Accessed: September 5,2022]

[2] A. Balasubramanian, M. S Baranowski, A. Burtsev, A. Panda, Z. Rakamaric, L. Ryzhyk. “System Programming in Rust: Beyond Safety”, ACM Digital Library, May 2017.[Online]. Available: https://dl.acm.org/doi/abs/10.1145/3102980.3103006 [Accessed: October 1,2022]

[3] M..Emre, R. Schroeder, K. Dewey, B. Hardekopf, “Translating C to Safer Rust”, ACM Digital Library, October 2021.[Online]. Available: https://dl.acm.org/doi/abs/10.1145/3485498 [Accessed: October 8,2022]

[4] “Design a Library Management System”, educative.io, [Online]. Available at: https://www.educative.io/courses/grokking-the-object-oriented-design-interview/RMlM3NgjAyR [Accessed: November 10, 2022]

[5] L. Ardito, L. Barbato, M. Castelluccio, R. Coppola, C. Denizet, S. Ledru, M. Valsesia, “rust-code-analysis: A Rust library to analyze and extract maintainability information from source codes”, ResearchGate, July, 2020. [Online]. Available: https://www.researchgate.net/publication/347700460_rust-code-analysis_A_Rust_library_to_analyze_and_extract_maintainability_information_from_source_codes. [Accessed: November 18, 2022]

[6] A. Amorim, C. Hritcu, B. C. Pierce, “The Meaning Of Memory Safety”, Springer Link, April.14,2018. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-89722-6_4 [Accessed: November 3, 2022]

[7] E. Reed, “Patina: A Formalization Of The Rust Programming Language”, University Of Washington, Feb,2015. [Online]. Available: https://dada.cs.washington.edu/research/tr/2015/03/UW-CSE-15-03-02.pdf [Accessed: November 15, 2022]

[8] J. Beckmann, “Verifying Safe Clients of Unsafe code and Trait Implementations in Rust”, Institute of Technology Zurich, Sept.28,2020. [Online]. Available: https://ethz.ch/content/dam/ethz/special-interest/infk/chair-program-method/pm/documents/Education/Theses/Jakob_Beckmann_MS_Report.pdf [Accessed: November 20, 2022]

[9] H. M. J. Almohri, D. Evans, “Fidelius Charm: Isolating Unsafe Rust Code”, ACM Digital Library, March, 2018. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/3176258.3176330. [Accessed: December 1, 2022]

[10] V. Astrauskas, C. Matheja, F. Poli, P. Muller, A. J. Summers, “How do Programmers Use Unsafe Rust”, ACM Digital Library, November, 2020. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/3428204. [Accessed: December 1, 2022]

[11] N. D. Matsakis, F. S. Klock, “The Rust Language”, ACM Digital Library, October,2014.[Online]. Available: https://dl.acm.org/doi/abs/10.1145/2692956.2663188 [Accessed: October 5, 2022]

[12] B. Qin, Y. Chen, Z. Yu, L. Song, Y, Zhang, “Understanding memory and thread safety practices and issues in real-world Rust programs”, ACM Digital Library, June,20.

[Online]. Available: https://dl.acm.org/doi/abs/10.1145/3385412.3386036 [Accessed: November 5, 2022]

[13] Z. Zhnag, B. Qin, Y. Chen, L. Song, Y. Zhang, “VRLifeTime-- An IDE Tool to Avoid Concurrency and Memory Bugs in Rust”, ACM Digital Library, November, 2020. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/3372297.3420024. [Accessed: October 5, 2022]

Appendix

Appendix A: Additional Information

A. HALSTED Metric Suite

It is a suite [1] that is used to measure the software properties like volume, program vocabulary, length, difficulty and effort. There are few base measures calculated before computing the actual measures ‘as shown in Table 7’. Base measures are:

n1 = Number of distinct operators

n2 = Number of distinct operands

N1 = Total number of operators

N2 = Total number of operands

‘Table 7’ shows halstead metric measures and their formulas:

Table 7: HALSTED metric table
This is brief description how HALSTED metric suite is used to evaluate software.

Measures	**Symbol**	**Formula**
Program Length	N	N = N1 + N2
Program vocabulary	n	N= n1+ n2
Volume	V	$V = N * \log_2(n)$
Difficulty	D	$D = n1/2 * N2/n2$
Program level	L	$L = 1/D$
Effort	E	E= D * V
Estimated program length	H	$H = n1 * \log_2(n1) + n2 * \log_2(n2)$
Time	T	$T = E/18$
Bugs	B	$B = E^{2/3}/3000$

B. Maintainability Index Suite

Maintainability index is used to evaluate maintainability of a program. Its formula is given as:

MI = 171.0 - 5.2*ln(V) - 0.23 * CC - 16.2 * ln(SLOC)

V= HALSTED Volume

CC = Cyclomatic Complexity

SLOC = source lines of code

Appendix B: Experiments Results

A. Polymorphism Experiment

Class diagram for this class is shown in ‘figure 9’. This algorithm is calculating areas of different shapes as shown in the figure 9. the results for this class is given in ‘Table 8’. HALSTEAD metric suite is used to evaluate the complexity of a program. The results for this metric is shown in ‘Table 9’. Cyclomatic Complexity has two parameters. One is sum and other is average. The results for this metrics of shape class is shown in ‘Table 10’.

Figure 9: Class Diagram of Shape Class

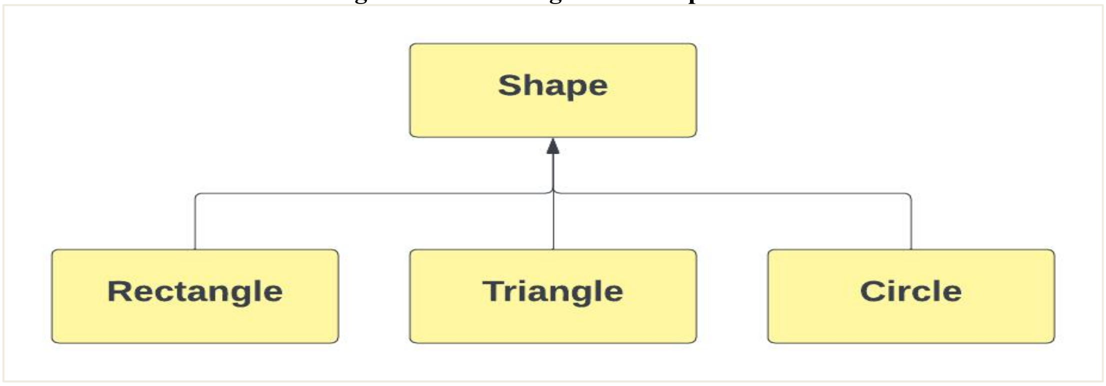


Table 8: Metrics Results for shape class.

C++		Rust	
Metrics	**Values**	**Metrics**	**Values**
NARGS	9	**NARGS**	6
NEXITS	1	**NEXITS**	0
SLOC	159	**SLOC**	81
PLOC	109	**PLOC**	67
LLOC	30	**LLOC**	16
CLOC	21	**CLOC**	8
BLANK	29	**BLANK**	14
NOM	16	**NOM**	7
MI	44.413232739021595	**MI**	60.15527591255034

Table 9: HALSTEAD Metrics Results of Shape class

C++		Rust	
HALSTEAD Metrics	**Values**	**HALSTEAD Metrics**	**Values**
length	334	**length**	184
Estimated program	287.9357738214561	**Estimated program**	205.81007680238335

length		**length**	
vocabulary	58	**vocabulary**	45
volume	1956.565652372609	**volume**	1010.5009697246602
difficulty	30.825	**difficulty**	17.5
level	0.032441200324412	**level**	0.0571428571428571
effort	60311.13623438567	**effort**	17683.766970181554
time	3350.618679688093	**time**	982.4314983434197
bugs	0.51263755522764	**bugs**	0.2262534798345405

Table 10: Cyclomatic Complexity Sum and Average results of shape class

C++		**Rust**	
CC	**Values**	**CC**	**Values**
Sum	22	**Sum**	16

B. Inheritence Experiment

Class diagram for this class is shown in ‘figure 10’. This algorithm is using inheritence as shown in the ‘figure 10’. the results for this class is given in ‘table 11’. HALSTEAD metric suite is used to evaluate the complexity of a program. The results for this metric is shown in ‘table 12’. Cyclomatic Complexity results shown in ‘table 13’.

Figure 10: Class Diagram of Person Class

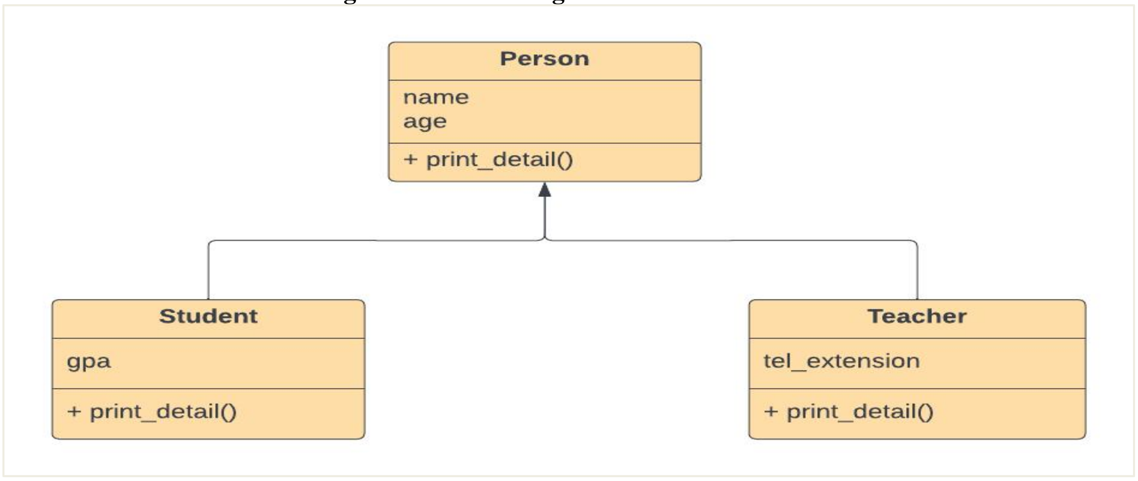


Table 11: Metrics Results for Person class.

C++		**Rust**	
Metrics	**Values**	**Metrics**	**Values**
NARGS	8	**NARGS**	9
NEXITS	0	**NEXITS**	3

SLOC	62	**SLOC**	621
PLOC	49	**PLOC**	51
LLOC	11	**LLOC**	16
CLOC	3	**CLOC**	8
BLANK	10	**BLANK**	0
NOM	7	**NOM**	11
MI	66.7858448744038	**MI**	67.60536547302534

Table 12: HALSTEAD Metrics Results of Person class

C++		**Rust**	
HALSTEAD Metrics	**Values**	**HALSTEAD Metrics**	**Values**
length	162	**length**	136
Estimated program length	130.29241368	**Estimated program length**	141.1270
vocabulary	32	**vocabulary**	34
volume	810	**volume**	691.8949
difficulty	17.809523	**difficulty**	13.909090
level	0.0561497	**level**	0.0718954
effort	14425.7142857	**effort**	9623.629709
time	801.428571428	**time**	534.64609
bugs	0.197531773	**bugs**	0.1508127

Table 13: Cyclomatic Complexity results of Person class

C++		**Rust**	
CC	**Values**	**CC**	**Values**
Sum	11	**Sum**	11

C. Library Management System

Detailed results of all metrics is given in 'Table 14'. HALSTEAD metric suite is used to evaluate the complexity of a program. The results for this metric is shown in 'Table 15'. Cyclomatic Complexity results shown in 'Table 16'.

Table 14: Metrics Result of Library system

C++		**Rust**	
Metrics	**Values**	**Metrics**	**Values**
NARGS	140	**NARGS**	411

NEXITS	40	**NEXITS**	131
SLOC	1388	**SLOC**	2082
PLOC	1188	**PLOC**	1609
LLOC	375	**LLOC**	564
CLOC	8	**CLOC**	10
BLANK	193	**BLANK**	468
NOM	135	**NOM**	257
MI	-56.0864465483	**MI**	-100.784266418

Table 15: HALSTEAD Metrics Result of Library system

C++		**Rust**	
HALSTEAD Metrics	**Values**	**HALSTEAD Metrics**	**Values**
length	4861	**length**	6893
Estimated program length	2427.149715393091	**Estimated program length**	2347.991850088082
vocabulary	309	**vocabulary**	300
volume	40207.581778643165	**volume**	56721.24723358811
difficulty	93.43462897526501	**difficulty**	101.6086956521739
level	0.0107026699947053	**level**	0.00984167736414206
effort	3756780.48548015	**effort**	5763371.947169365
time	208710.02697111946	**time**	320187.3303982981
bugs	8.05544045026371	**bugs**	10.715106964942601

Table 16: Cyclomatic Complexity results of Library System

C++		**Rust**	
CC	**Values**	**CC**	**Values**
Sum	238	**Sum**	396