M. Tech (Software Engineering)

Shweta Meena

2018

Design and Development of a Tool for analyzing effect of Refactoring on Maintainability

A PROJECT REPORT

SUBMITTED IN PARTIAL FUL LLMENT OF THE REQUIREMENTS FOR THE AWARD OF

DEGREE OF MASTER OF TECHNOLOGY
IN
SOFTWARE ENGINEERING

Submitted by
Shweta Meena
2K16/SWE/16

Under the supervision of

DR. RUCHIKA MALHOTRA
Associate Professor
Department of Computer Science and Engineering
Delhi Technological University

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
(FORMERLY DELHI COLLEGE OF ENGINEERING)
SHAHABAD DAULATPUR, BAWANA ROAD, DELHI – 110042

JUNE, 2018

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College Of Engineering)
Bawana Road, Delhi – 110042

CANDIDATE'S DECLARATION

I, (Shweta Meena), 2K16/SWE/16, student of M.Tech, Software Engineering, hereby declare that the project Dissertation titled, "**Design and Development of a Tool for analyzing effect of Refactoring on Maintainability**" which is submitted by us to the Department of Computer Science and Engineering, Delhi Technological University, Delhi in partial fulfillment of the requirement for the award of the degree of Master of Technology, is original and not copied from any source without any citation. The work has not previously formed the basis of any Degree, Diploma Associateship, Fellowship or other similar title or recognition.

Place: Delhi	SHWETA MEENA
Date:	2K16/SWE/16

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College Of Engineering)
Bawana Road, Delhi – 110042

CERTIFICATE

I hereby certify that the project Dissertation titled, "**Design and Development of a Tool for analyzing effect of Refactoring on Maintainability**" which is submitted by Shweta Meena, 2K16/SWE/16, Software Engineering, Delhi Technological University, Delhi in partial fulfillment of the requirement for the award of the degree of Master of Technology, was carried out by the students, under my supervision. To the best of my knowledge this work has not been submitted in part or full for any Degree or Diploma to this university or elsewhere.

Place: Delhi
Date:

Dr. Ruchika Malhotra
(Supervisor)
Associate Professor
Discipline of Software Engineering
CSE Department
Delhi Technological University
(Formerly Delhi College of Engineering)
Shahbad, Daulatpur, Bawana Road, Delhi – 110042

ABSTRACT

Delivering programming is an exceptionally perplexing and troublesome process that sets aside an impressive opportunity to advance. Inadequately planned programming frameworks are hard to comprehend and keep up. Programming upkeep can take around half of the general advancement expenses of delivering programming. Here we have basically address prototype of a tool which will perform the automatic refactoring of five different methods in our work. One of the fundamental characteristics of these high expenses is inadequately planned code, which makes it troublesome for engineers to comprehend the framework even before considering executing new code. With regards to programming building process, software refactoring affects diminishing the cost of programming upkeep through changing the inside structure of the code, without changing its outside conduct. Refactoring is a method for rebuilding a current group of code, its heart is a progression of little conduct safeguarding changes. Every change called a ‘refactoring’ does pretty much nothing, however, a succession of changes can create a huge rebuilding.

ACKNOWLEDGEMENT

The successful completion of any task would be incomplete without accomplishing the people who made it all possible and whose constant guidance and encouragement secured us the success.

First of all, I would like to thank the Almighty, who has always guided me to work on the right path of the life. My greatest thanks are to my parents who best owed ability and strength in me to complete this work.

I owe a profound gratitude to my project guide **Dr. Ruchika Malhotra,** who has been a constant source of inspiration to me throughout the period of this project. It was her competent guidance, constant encouragement and critical evaluation that helped me to develop a new insight into my project. Her calm, collected and professionally impeccable style of handling situations not only steered me through every problem, but also helped me to grow as a matured person. I am also thankful to her for trusting my capabilities to develop this project under her guidance.

Shweta Meena

M.Tech(SWE) – 2nd Semester

2K16/SWE/16

Table of Contents

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Refactoring is basically the process of improving software internally without affecting external behavior. It aims to make code easier to understand, easier to maintain and easier to extend. Refactoring also helps in identifying the error prone parts of your program which can be corrected by refactor your code.

It was basically Martin Fowler who made refactoring prominent by publishing the mechanisms of 72 refactorings for object-oriented programs in his influential book Refactoring: Improving the Design of Existing Code [1] . Though refactorings within Fowler's book are well described, time and skill and quite good test coverage is needed, if one wants to apply them with confidence [1]. It was also indicated that all such manual refactoring is tedious and error-prone. Proper tool support is therefore considered crucial. But unfortunately automating the various refactorings is not easy. It is a challenge and requires some serious work.

We basically aims to review various existing refactoring tools. Instead, it shall be a starting point for researchers, as well as for IDE_and tool developers willing to implement refactorings. It shall help researches in finding open topics and provide tool developers with an overview over current research and useful implementation techniques. Such a starting point is needed because these two groups rarely work together.

1.1 RESEARCH OBJECTIVE

The main objective of developing this prototype of a tool is to automate the refactoring in comparison to other refactoring tools which are available on the web or in the market.

The way in which this tool is different from other refactoring tool is that it provides the cleaned file of the code or the file which user provides it as a input. The most important part of this tool is that it is developed in Python language. There are many advantages of Python over C++ and Java. This prototype is open source so that anybody can use it . After cleaning the actual code we have implemented the five types of refactoring which are commonly used in the published research papers. It will also reduce the execution time of the projects so that they would take less time in development and to test this we will also refactor the individual modules of the refactoring projects. Nowadays, it seems to be difficult to deliver the product in a reasonable amount of time, so that task can be achieved by automatically finding out the errors in the project and then correcting them using the tool performs the automatic refactoring. In the past, developers achieved such goals by poring over hardware and software manuals, trying to locate the proper combination of assembly language instructions that would result in the level of performance that they desired. Since the computers they had available were all well documented and functioned in a wholly deterministic manner, it was relatively easy for a developer to determine the types of source-code adjustments that would work best on a given architecture.

In the past ten years, however, the software development landscape has evolved dramatically as the general public has embraced computing devices of all types and become increasingly reliant on them to accomplish everyday tasks. As the demand for more sophisticated applications has increased, designers have turned to use the use of higher-level languages and frameworks in an attempt to reduce development costs and remain competitive in the marketplace. As a result, applications have grown increasingly complex in terms of both code size and the interactions that occur within them. Therefore, while this approach to development may save time and money in the short run, it complicates the task of determining whether an observed performance issue is internal to an application or caused by the frameworks that it is built upon.

Beyond this, computer hardware itself has been forced to change dramatically to keep up with the unrelenting demand for more computing power. The simple single-issue processors of the past have given way to super-scalar designs capable of executing multiple instructions in a single cycle while simultaneously reordering operations to maximize overall performance. As a result, the instructions passed into the processor have become merely a guideline for execution, as opposed to the written rules they were viewed as in the past. Since a developer now has no way to determine precisely how the processor will operate, the act of hand-tuning an application at the assembly level is no longer a straightforward task.

- **Open Source :-** It is open source so that anyone can make changes in the source code of the project according to their needs and requirements. Being open source its code can be used in other projects as an API.
- **Platform Independent :**- This prototype is developed under Anaconda, which is an open source distribution of the Python. It is available for all major platforms and this project has been developed using standard library only. Therefore this code can be deployed to any platform and executable can be made for that platform.
- **Easy to Integrate :-** This prototype is easy to integrate with other web or stand-alone applications because this tool doesn't require any virtual environment or specific technologies for its execution.
- **Fast Performance :-** These prototype may require to perform refactoring fast so that it will not take more time during its working thus it is very important to make it fast. In order to make it fast, we have developed it using Python, which is compiled language and very close to machine level.
- **Light Weight :-** This prototype uses command line interface and do not require any special environment for its execution which makes it light and portable.

1.2 SCOPE

As programming are developing in intricacy, designers are winding up progressively dependent on the utilization of structures and reflection layers to actualize new functionalities in an opportune way. In doing as such, the capacity to comprehend the exact way in which a given section of code works has been decreased, prompting programming ventures in which nobody individual very sees how the whole application capacities. Accordingly, it has turned out to be about difficult to upgrade applications utilizing the manual (or "by-hand") tuning systems of the past. In light of this issue, computerized investigation devices can have a vital part in programming improvement. We have kept this prototype open-source, platform independent and easy to integrate. This increases its scope, anyone can access and use it online. It can be used as an API in other programs which wants to use its functionality. Being light and easy to integrate, it can be integrate with other applications. E.g. It can be used to develop plugin for IDE like Netbeans or Eclipse , it would be really helpful for these tools.

1.3 ORGANIZATION OF THE DISSERTATION

In this thesis, we have introduced the procedure followed by us to develop a prototype for a tool. The rest of the thesis has been divided into various chapters. This Chapter contains the general summary of the project. In Chapter 2 the work done in the research regarding refactoring and refactoring tool has been discussed. In Chapter 3 we have explained the refactoring techniques which have been implemented in the tool. In Chapter 4 we have mentioned the technologies used for the development of this tool. In Chapter 5 we have discussed the results drawn and our analysis for effect of refactoring on maintainability. The conclusion and future work have been explained in Chapter 6. Finally, all the references used in the research have been mentioned. The appendix contains relevant code snippets and screenshots.

CHAPTER 2

LITERATURE REVIEW

There are many IDEs and commercial software exists for refactoring. All of them don’t do the refactoring automatically. The user and developers will check at each and every line about the type of refactoring needs to be performed. It is very time consuming.

2.1 BACKGROUND WORK

Most of the analysis software are available under commercial license therefore their source code can’t be alter to make new application. The existing open source software are heavy and can’t be integrated with other applications easily.

The research which has done till now in this field has basically compared the existing tools on the basis if the languages and the type of refactoring they perform and the software requirements for such IDE installation. According to Rani et al. [1], IntelliJ IDEA was the primary Java™ IDE to broadly execute a significant number of refactoring which will work out and prescribed in the pivotal book Refactoring: Improving the Design of Existing Code by Martin Fowler et al. From that point forward, refactoring a code is considered as a noteworthy focal point of improvement, along each discharge to date including different as well as extended refactoring.

Rani et al. [1] also introduces that Eclipse is the multi-dialect programming improvement environment it is composed generally in java. It can be utilized to create applications in Java and, by methods for different modules other programming dialect like including R, ruby Ada ,C,C++, COBOL, FORTAN, php and so on

In spite of the fact that it is conceivable to refactor physically, instrument bolster is viewed as essential. Today, an extensive variety of devices is accessible that robotize different parts of refactoring.

In this research we have found that Bois et al. [27] introduced that some tools for example, the Refactoring Browser, XRefactory, jFactor bolster a self-loader approach. A few analysts showed the practicality of completely mechanized refactoring. For instance, Guru is a completely robotized apparatus for rebuilding legacy progressive systems and refactoring strategies in SELF projects. Other programmed refactoring approaches are proposed in. There is additionally an inclination to coordinate refactoring apparatuses straightforwardly into modern quality programming advancement situations. This is for instance the case for Smalltalk VisualWorks v7, Eclipse v2, Borland Together Control-Center v6, IntelliJ IDEA v3, Borland JBuilder v6, and so forth... The focal point of every one of these apparatuses is on applying a refactoring upon demand of the client. There is significantly less help accessible for identifying where and when a refactoring can be connected. Simon et al. [28] proposes to do this by methods for measurements, while Kataoka et al. [29] demonstrates where refactoring may be appropriate via naturally distinguishing program invariants utilizing the Daikon instrument. This approach depends on unique examination of the runtime conduct, and is by all accounts integral to different methodologies.

CHAPTER 3

REFACTORING TECHNIQUES

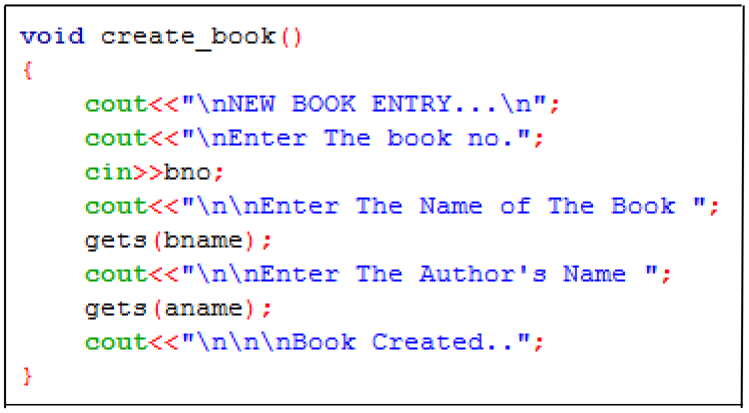
There are 74 refactoring methods which were proposed by Fowler [11]. Five refactoring methods have been implement in this tool. In this chapter we have basically discussed the refactoring methods, their reason to occur, solution to avoid them, advantages and example etc.

3.1 REFACTORING METHODS:- There are different types of refactoring. Some of them which are implemented in this prototype are described as follows.

3.1.1 Hide Method :-

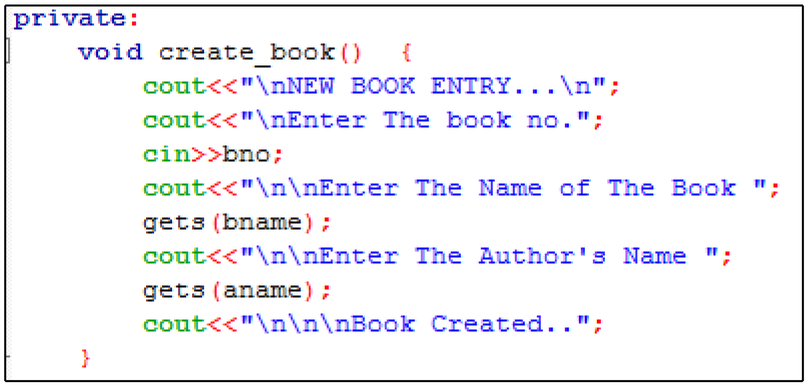
Issue:- A method isn't utilized by totally different categories or it is utilized just inside its own particular class chain of importance.

Solution :- Change the access modifier of a method or private or protected.



```
void create_book()
{
    cout<<"\nNEW BOOK ENTRY...\n";
    cout<<"\nEnter The book no.";
    cin>>bno;
    cout<<"\n\nEnter The Name of The Book ";
    gets(bname);
    cout<<"\n\nEnter The Author's Name ";
    gets(aname);
    cout<<"\n\n\nBook Created..";
}
```

Fig 3.1: Before applying Hide Method Refactoring



```
private:
    void create_book()   {
        cout<<"\nNEW BOOK ENTRY...\n";
        cout<<"\nEnter The book no.";
        cin>>bno;
        cout<<"\n\nEnter The Name of The Book ";
        gets(bname);
        cout<<"\n\nEnter The Author's Name ";
        gets(aname);
        cout<<"\n\n\nBook Created..";
    }
```

Fig 3.2: After applying Hide Method Refactoring

In Fig 3.1 we have taken the code from comment cleaned file on which we have applied Hide method refactoring in Fig 3.2. In the above figure before applying refactoring the access modifier or visibility of the method is public. The public method can be accessible from objects of other classes also. We have written the code for making public methods to private. In Fig 3.2 the create_book() access modifier changes to private. After this the create_book() can be accessed by the object of same class only.

Reason of Refactoring

Quite usually, the requirement to cover ways for obtaining and setting values is attributable to improvement of a richer interface that gives extra behavior, particularly if you started with a category that additional very little on the far side mere knowledge encapsulation. As new behavior is constructed into the category, you will realize that public getter and setter ways aren't any longer necessary and may be hidden. If you create getter or setter ways personal and apply direct access to variables, you'll delete the strategy.

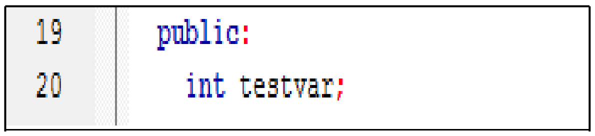
Advantages

1. Hidden methods makes it more easy for our code to evolve. If we change the access modifier of a function to private, then we will only worry about how can we not break the present class because we know that the function is not used at any other place in our code.

2. By making the access modifier of a function as private, we underscore the significance of general interface of the class and of the functions whose access modifier remains public.

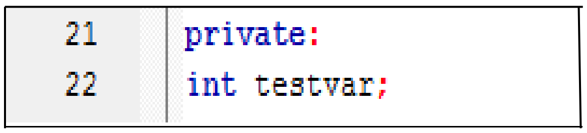
3.1.2 Encapsulate Field :-

Issue :- We have a public member or field.
Solution :- Make that public member or field to private member or field and create access methods for it.



```
public:
  int testvar;
```

Fig 3.3: Before applying Encapsulate Field Refactoring



```
private:
int testvar;
```

Fig 3.4: After applying Encapsulate Field Refactoring

In Fig 3.3 the access modifier of data member testvar is public in cleaned comment file of the code. We will apply Encapsulate Field Refactoring to change its accessibility to private. We have changed the accessibility of a data member because it will maintain the encapsulation property of object – oriented programming language. Encapsulation preserves the property of data members in the way that they would be accessible by the methods of the class. In Fig 3.4 we see that the access modifier changed to private.

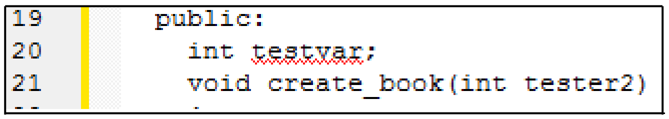
<u>Reason of Refactoring</u>
Encapsulation is one of the advantage or we can say it is a benefit of object-oriented programming, the capability to hide objects data. Otherwise, all objects would be public and these objects would access and modify the data of your object without any checks and alances. The Data which is associated with this data separate it from its behaviors, modularity of program sections is compromised, and it makes maintenance complicated.

Advantages :- If the data and behavior of a component are closely interrelated and are in the same place in the code, it is much easier for you to maintain and develop this component.

3.1.3 Self Encapsulate Field :-

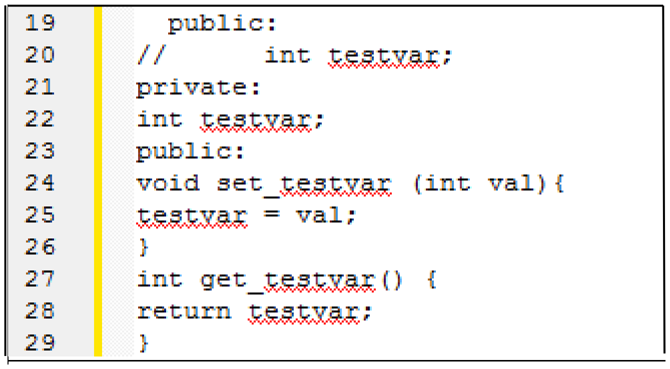
Issue :- You have a direct access to private members or fields inside a class.

Solution :- You can create a setter and getter for the that member or field, and use only them for accessing the field.



```
  public:
    int testvar;
    void create_book(int tester2)
```

Fig 3.5: Before applying Self Encapsulate Field Refactoring



```
  public:
//      int testvar;
private:
int testvar;
public:
void set_testvar (int val){
testvar = val;
}
int get_testvar() {
return testvar;
}
```

Fig 3.6: After applying Self Encapsulate Field Refactoring

In Fig 3.5 we see that it is before applying refactoring. The data member testvar is defined as public, which indicated that it can be accessible outside the class. It violates the Encapsulation property of object – oriented language. We have to make it private. In Fig 3.6 we have made the data member visibility to private and we have also created setter and getter for this data member. Now, we access this data member inside the class in which they are declared.

Reason of Refactoring

It is not flexible to directly accessing a private field inside a class. We want to be able to assign or initiate a member or filed value when the initial query is performed or we can also perform different operations on the new values of the member or field when they are assigned to the field, or we can also do all this in various ways in subclasses.

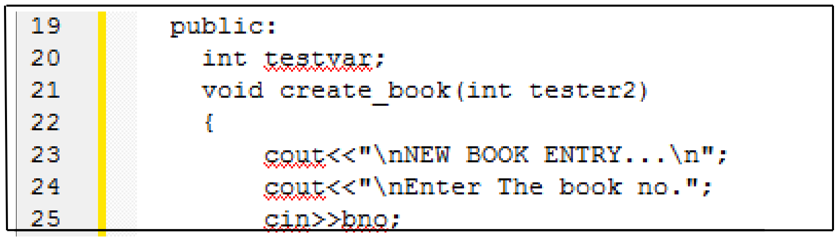
<u>Advantages</u>

Indirect access to fields is when a field is acted on via access methods (getters and setters). This approach is much more flexible than direct access to fields. First, you can perform complex operations when data in the field is set or received.

3.1.4 Remove Assignments to Parameters

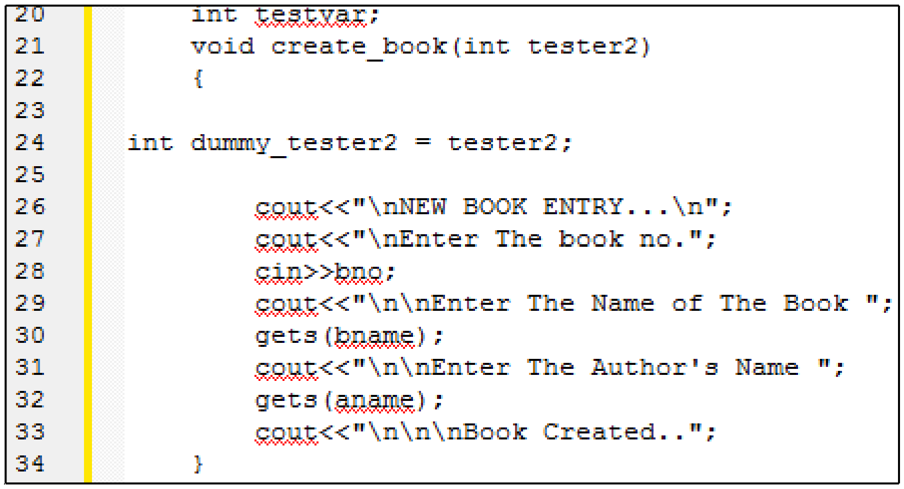
Issue :- Some value is assigned to a parameter within method's body.

Solution :- You can use a local variable instead of a parameter.



```
    public:
      int testvar;
      void create_book(int tester2)
      {
          cout<<"\nNEW BOOK ENTRY...\n";
          cout<<"\nEnter The book no.";
          cin>>bno;
```

Fig 3.7: Before applying Remove Assignments to Parameters Refactoring



```
      int testvar;
      void create_book(int tester2)
      {

int dummy_tester2 = tester2;

          cout<<"\nNEW BOOK ENTRY...\n";
          cout<<"\nEnter The book no.";
          cin>>bno;
          cout<<"\n\nEnter The Name of The Book ";
          gets(bname);
          cout<<"\n\nEnter The Author's Name ";
          gets(aname);
          cout<<"\n\n\nBook Created..";
      }
```

Fig 3.8: After applying Remove Assignments to Parameters Refactoring

In Fig 3.7 we see that we are passing tester2 data member as a actual parameter. It does not have any initial value. If user will use this same parameter and its value would after calling the create_book() method and before calling the create_book(). It will result in incorrect value of tester2. To avoid this, we have applied Remove Assignments to Parameters refactoring in Fig. 3.8. In Fig 3.8 we have declared a dummy variable which will store the value of tester2 inside the function. So that it won't affect the actual parameter value.

Reason of Refactoring

The reasons for this refactoring are a similar as for Split Temporary Variable, however during this case we tend to are managing a parameter, not an area variable. First, if a parameter is passed via reference, then once the parameter worth is modified within the strategy, this worth is passed to the argument that requested line this technique. Very often, this happens accidentally and ends up in unfortunate effects. albeit parameters are typically gone along worth (and not by reference) in your programing language, this writing quirk might alienate people who are unaccustomed to that.

Second, multiple assignments of various values to one parameter create it tough for you to understand what knowledge ought to be contained within the parameter at any explicit purpose in time. the matter worsens if your parameter and its contents are documented however the particular worth is capable of differing from what's expected within the strategy.

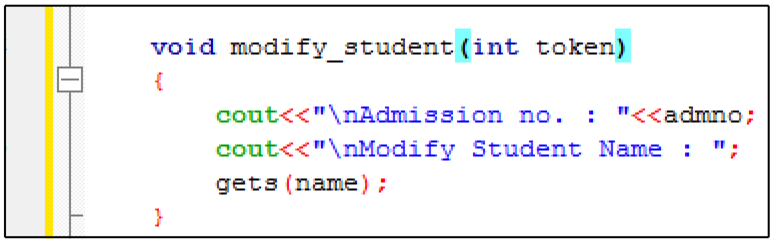
Advantages

Each component of the program ought to be liable for just one issue. This makes code maintenance a lot of easier going forward, since you'll be able to safely replace code with none facet effects.This refactoring helps to extract repetitive code to separate ways.

3.1.5 Remove Parameters

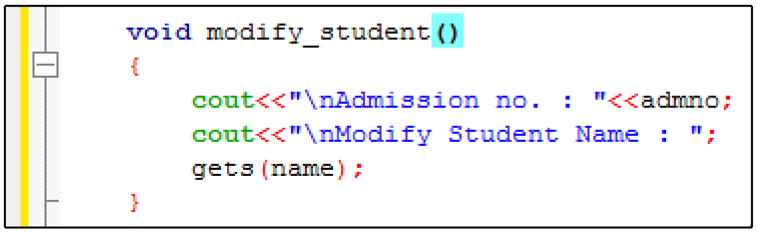
Issue :- The parameter is declared in the body of a method , but it is not used in the body of a method.

Solution :- We will remove the unused parameter.



```
void modify_student(int token)
{
    cout<<"\nAdmission no. : "<<admno;
    cout<<"\nModify Student Name : ";
    gets(name);
}
```

Fig 3.9 Before applying Remove Parameters Refactoring



```
void modify_student()
{
    cout<<"\nAdmission no. : "<<admno;
    cout<<"\nModify Student Name : ";
    gets(name);
}
```

Fig 3.10 After applying Remove Parameters Refactoring

In Fig 3.9 we have passed data member token as a actual parameter but it doesn't require to pass it to a method. We don't need token in the function definition. We will remove this by applying refactoring. In Fig 3.10 we have applied Remove parameter refactoring.

<u>Reason of Refactoring</u>

Every parameter during a methodology decision forces the coder reading it to work out what data is found during this parameter. Sometimes we tend to add parameters with a watch to the longer term, anticipating changes to the tactic that the parameter may well be required. All identical, expertise shows that it's higher to feature a parameter only if it's genuinely required. After all, anticipated changes usually stay simply that – anticipated.

<u>Advantages</u>

A method contains only those methods which are truly used by it.

CHAPTER 4

IMPLEMENTATION METHODOLOGY

The prototype of a tool which we have developed is in python. We will further discuss about the technologies which we used. We have taken different c++ files on which we have tested this tool. At each and every stage we have performed testing for the development of this tool. At the end of this chapter we will discuss about the implementation of this tool, the files that it will create automatically, cleaned file of a input file.

4.1 DATASETS USED

4.1.1 Data collection

We can execute any unlimited files on this prototype. There is no limit on the number of files or projects that developer and user execute. The user needs to provide this prototype a c++ file as a input, then it will provide a comments cleaned file and cpp file of five different refactorings which are implemented in this prototype.

4.2 TECHNOLOGIES USED

4.2.1 Python

Python is a broadly utilized high – level, general – reason, translated dialect. The
plan theory of Python underlines code readability and the linguistic structure has been intended to allow software engineers to express ideas and calculations in less lines of code than conceivable in dialects, for example, C++ or Java. There are develops given in Python expected to take into consideration the written work of clear and effective

projects on both a little and in addition on an extensive. Multiple assorted programming ideal models are upheld by Python, including object – situated, basic and utilitarian programming. Further, the highlights of a dynamic sort framework and programmed memory administration, joined with an expansive and exhaustive standard library, settle on Python a well known decision among developers.

The availability of Python interpreters for many operating systems allows Python code to run on a wide variety of systems. By the use of third – party tools such as Py2exe or Py installer. Python code can be packaged into stand – alone executable programs for some of the most popular operating systems, which allows the distribution and use of Python – based software on a variety of environments with no need to install a Python interpreter.

4.2.2 Anaconda

Anaconda might be a free and open source dispersion of the Python and R programming dialects for data science and machine learning associated applications (expansive scale handling, prophetical examination, logical processing), that means to change bundle administration and preparation. Package forms square measure oversaw by the bundle administration framework conda. The Anaconda distribution is employed by over vi million users, and it includes quite 250 in style information science packages appropriate for Windows, Linux, and MacOS.

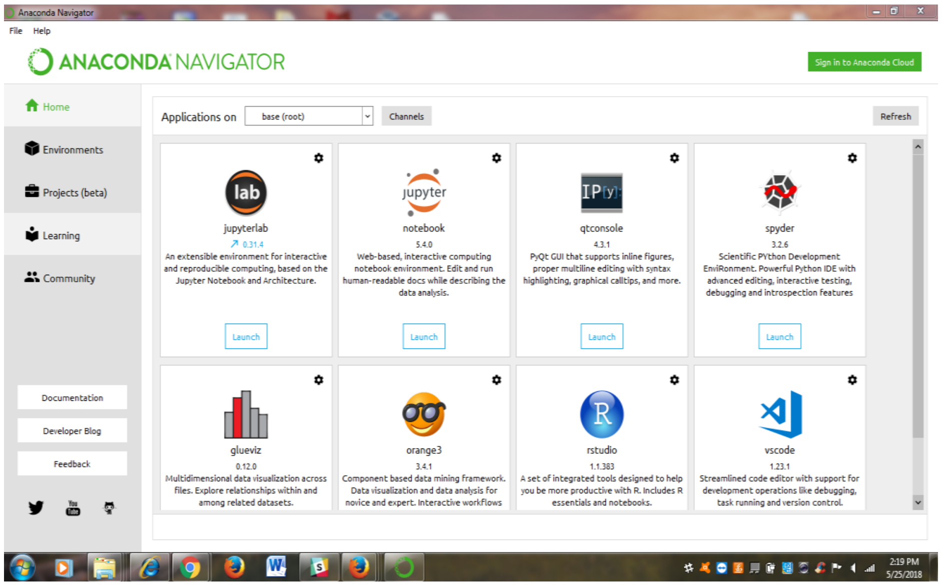
Steps to start with Anaconda

1. Go to start window and search for anaconda navigator. Click on Anaconda Navigator.

2. You will see Anaconda command prompt. It will start anaconda navigator after initializing it.

3. We will get this window of Anaconda Navigator. In Fig 4.1 we will open launch Anaconda Navigator. It will take somethime in initialization. In Fig 4.2 we have shown the window of launching jupyter notebook. In Fig 4.3 the screen

shows the sign in window. User has to enter its credentials for strtaing jupyter notebook.



An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

Multidimensional data visualization across files. Explore relationships within and among related datasets.

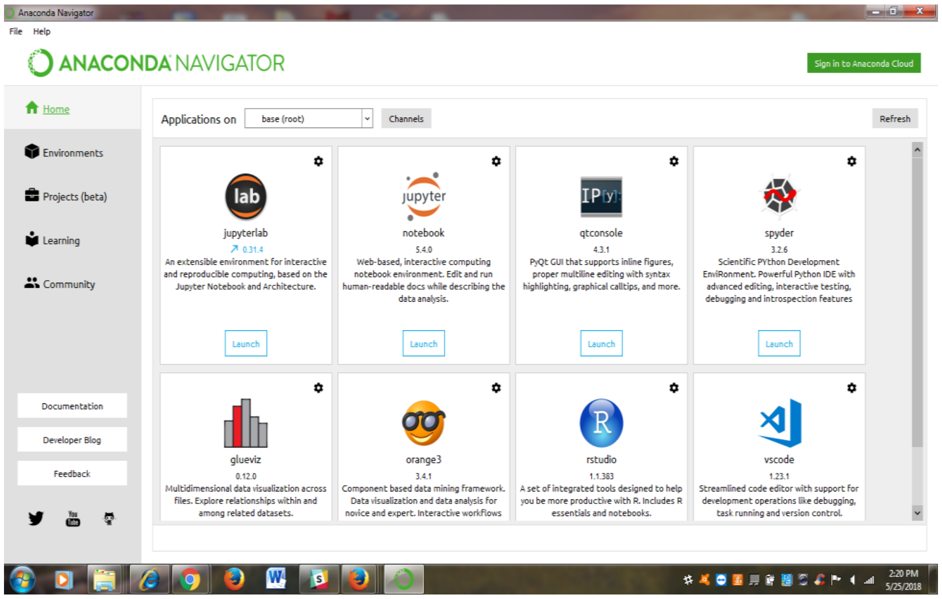
Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows

A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks.

Streamlined code editor with support for development operations like debugging, task running and version control.

Fig 4.11: Starting window of a Anaconda Navigator

4. Click on Sign in to Anaconda Cloud.



An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

Multidimensional data visualization across files. Explore relationships within and among related datasets.

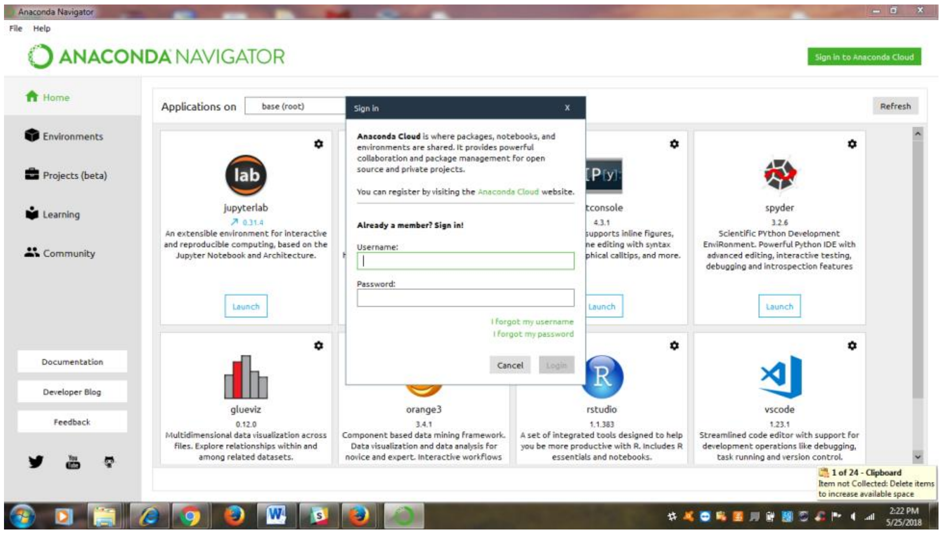
Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows

A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks.

Streamlined code editor with support for development operations like debugging, task running and version control.

Fig 4.12: Sign in to Anaconda cloud

5. Enter user credentials.



Sign in

Anaconda Cloud is where packages, notebooks, and environments are shared. It provides powerful collaboration and package management for open source and private projects.

You can register by visiting the Anaconda Cloud website.

Already a member? Sign in!

Username:

Password:

I forgot my username
I forgot my password

jupyterlab 0.31.4 An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

spyder 3.2.6 Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

glueviz 0.12.0 Multidimensional data visualization across files. Explore relationships within and among related datasets.

orange3 3.4.1 Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows

rstudio 1.1.383 A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks.

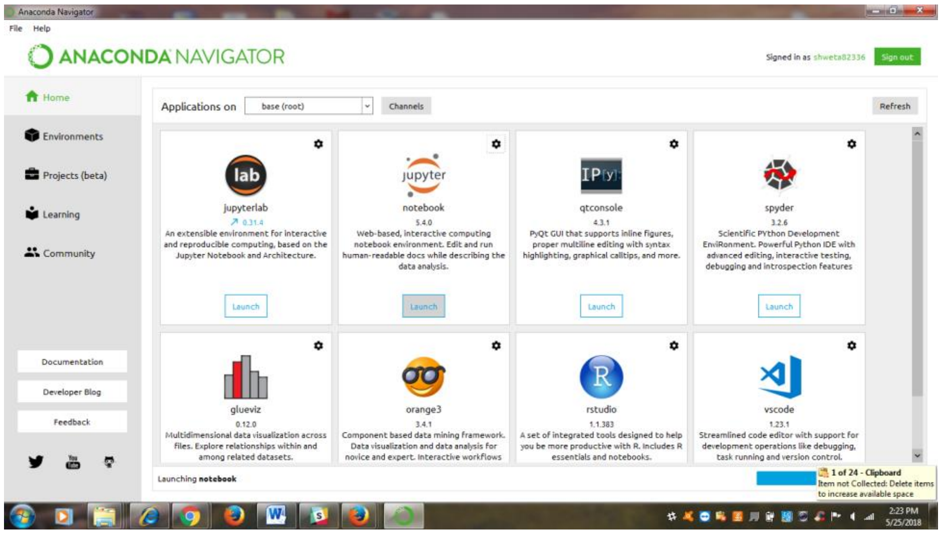
vscode 1.23.1 Streamlined code editor with support for development operations like debugging, task running and version control.

1 of 24 - Clipboard Item not Collected: Delete items to increase available space

2:22 PM 5/25/2018

Fig 4.13: Login window of Anaconda Cloud

6. Click on Launch Jupyter.



jupyterlab 0.31.4 An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

notebook 5.4.0 Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

qtconsole 4.3.1 PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

spyder 3.2.6 Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

glueviz 0.12.0 Multidimensional data visualization across files. Explore relationships within and among related datasets.

orange3 3.4.1 Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows

rstudio 1.1.383 A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks.

vscode 1.23.1 Streamlined code editor with support for development operations like debugging, task running and version control.

Launching **notebook**

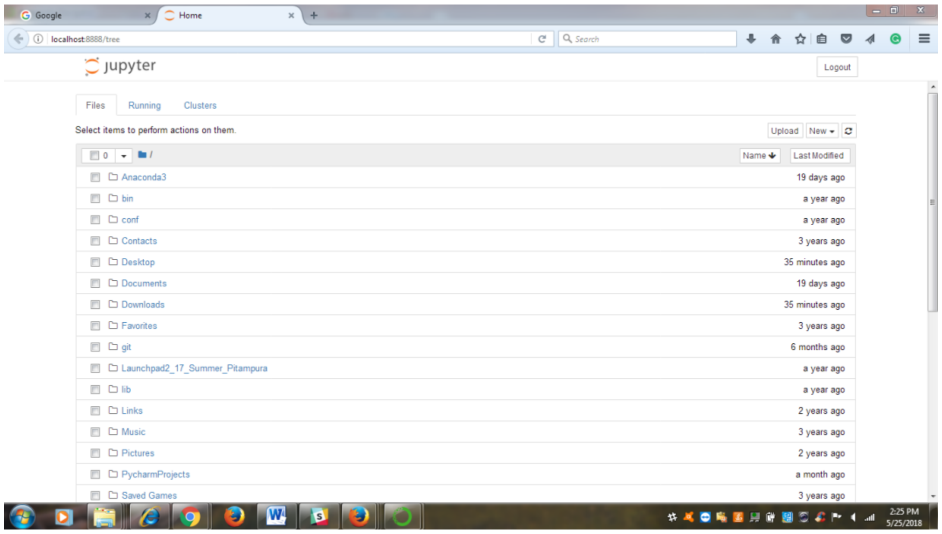
1 of 24 - Clipboard Item not Collected: Delete items to increase available space

2:23 PM 5/25/2018

Fig 4.14: Launching Jupyter Notebook

In Fig 4.4 User has to click on Launch icon below Jupyter notebook. In Fig 4.5 it will navigate to the jupyter notebook in browser. In the browser user can open any jupyter notebook on which they want to work.

7. After launching Jupyter Notebook



Select items to perform actions on them.

Name	Last Modified
Anaconda3	19 days ago
bin	a year ago
conf	a year ago
Contacts	3 years ago
Desktop	35 minutes ago
Documents	19 days ago
Downloads	35 minutes ago
Favorites	3 years ago
git	6 months ago
Launchpad2_17_Summer_Pitampura	a year ago
lib	a year ago
Links	2 years ago
Music	3 years ago
Pictures	2 years ago
PycharmProjects	a month ago
Saved Games	3 years ago

Fig 4.15: Launched Jupyter Notebook

4.2.3 Jupyter

The Jupyter Notebook is AN ASCII text file internet application that permits you to form and share documents that contain live code, equations, visualizations and narrative text. Uses include: knowledge cleansing and transformation, numerical simulation, applied mathematics modeling, knowledge image, machine learning, and far a lot of.

While Jupyter runs code in several programming languages, Python may be a demand (Python three.3 or larger, or Python two.7) for putting in the Jupyter Notebook.

4.2.4 Python Standard Libraries Used

A. Generic Operating System Services

1. import os:- This os stands for Miscllaneous Operating System Interface.This module gives a versatile method for utilizing working framework subordinate usefulness. In the event that we simply need to peruse or compose a document see open(), on the off chance that we need to control ways, see the os.path module, and on the off chance that

we need to peruse every one of the lines in every one of the records on the summon line see the file input module.

2. import time :- In import time library we have a Time access and conversions. This module gives different time-related capacities. For related usefulness, see additionally the date time and date-book modules. Despite the fact that this module is constantly accessible, not all capacities are accessible on all stages. The majority of the capacities characterized in this module call stage C library capacities with a similar name. It might some of the time be useful to counsel the stage documentation, on the grounds that the semantics of these capacities shifts among stages.

A. Python runtime services : - These are basically system – specific parameters and functions. This module provides access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter. It is always available.

B. Random :- This module implements pseudo-random number generators for various distributions. For integers, there is uniform selection from a range. For sequences, there is uniform selection of a random element, a function to generate a random permutation of a list in-place, and a function for random sampling without replacement. On the real line, there are functions to compute uniform, normal (Gaussian), lognormal, negative exponential, gamma, and beta distributions.

C. Numpy :- Numpy is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays. If you are already familiar with MATLAB, you might find this tutorial useful to get started with Numpy.

D. Matplotlib:- Matplotlib is a plotting library. In this section give a brief introduction to the matplotlib.pyplot module, which provides a plotting system similar to that of MATLAB.

4.3 TOOL IMPLEMENTATION

4.3.1 Comment Cleaning File

The first module of this prototype is basically comment cleaning. In Fig 4.6 we will provide a particular cpp file of a project as an input to this module then it will provide the cleaned file of the input file to the user. The cleaned file which it will provide would be different from the original file in the way that it has the no text after the back slash in single line comment and in multiline comment it will remove all the text between double back slash. This module work independently from all the files. It has no limit on the number of files that it can clean. The user can clean unlimited files. In Fig 4.6 we will provide cpp file as a input to the tool. It will provide cleaned .cpp file foe that input file.

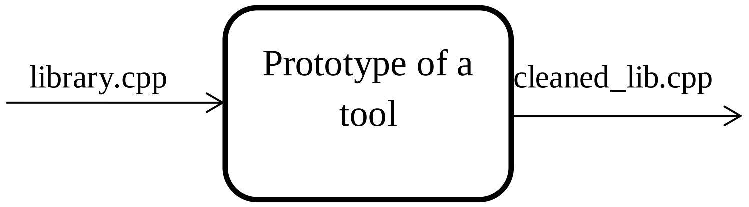


Fig 4.16: Process of a cleaning file

4.3.2 Retrieve all the parameters

After cleaning the file we need to retrieve all the parameters on the basis of which this prototype will choose the parameters on the basis which it performs refactoring. These parameters will in the form of a dictionary. We have basically created a dictionary of all the parameters. The dictionary contains the List of all Classes, List of class attributes, Dictionary of Classes, Data Types and Access Types. This output is used by the user while performing refactoring on a file that on what parameters which parameters are required and how to change them so that refactoring is performed.

In Fig 4.7 we have shown that the user will provide cleaned .cpp file as a input to a tool. The tool will provide dictionary of all the attributes, classes, methods, access modifier etc. to the user.

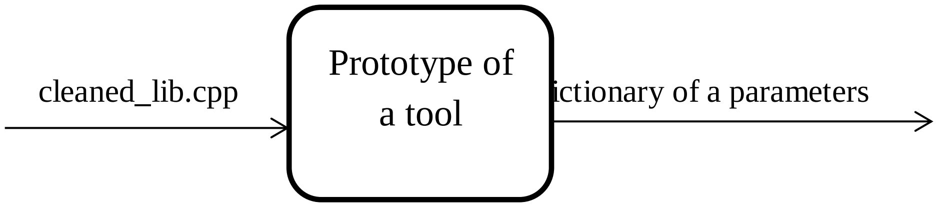


Fig 4.17: Retrieving Parameters

4.3.3 Refactoring Methods

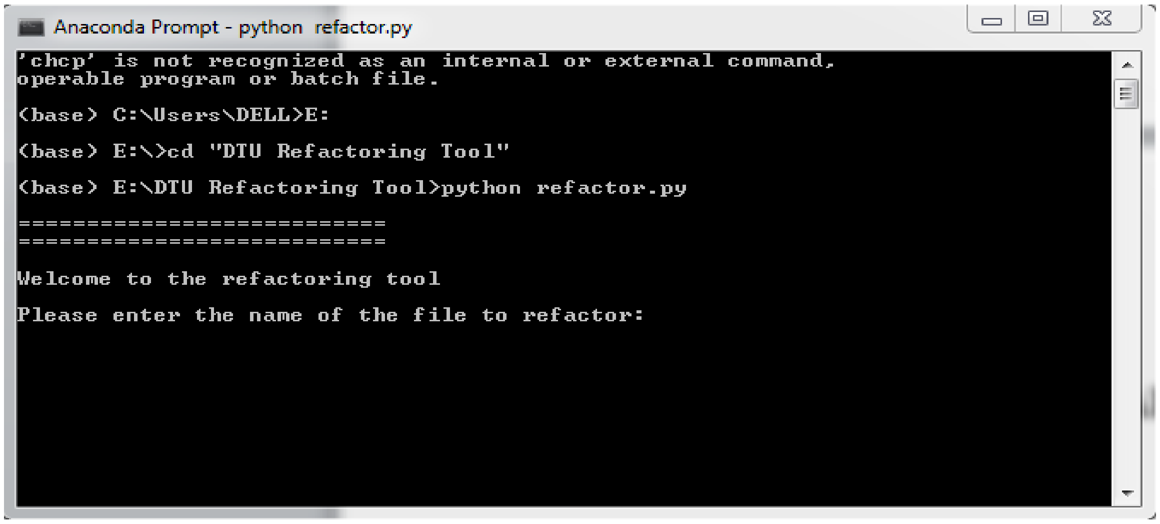
In this prototype we have implemented five refactoring methods which are most commonly used in the published research papers. We have not added support for File handling and Inheritance in this prototype right now. The File Handling and Inheritance increases complexity because of virtual functions. The user will provide cleaned_lib.cpp to the module then user want to apply. Then it will apply that particular refactoring. At the end it will provide the .cpp file of the particular refactoring method to the user. The five different refactoring methods which are implemented in this prototype are: Hide Method, Encapsulate Field, Self Encapsulate Field, Remove Assignment to Parameters and Remove unused Parameters.

CHAPTER 5

RESULTS AND CONCLUSION

In this chapter we have basically discussed about the results of a tool, what it will do. We will also discuss about the way it performs refactoring. The comparsion of existing tool with other existing tools of refactoring. The refactoring methods which are implemented in this tool are not available in other existing refactoring tools. We have discussed about the effect of implemented refactoring methods on maintainability. How does maintainability modify when user apply these refactoring methods on the code discussed in this chapter. Maintainability is basically defined as the feasibility of modification the attribute set. We have also discussed about the limitations of this refactoring tool.

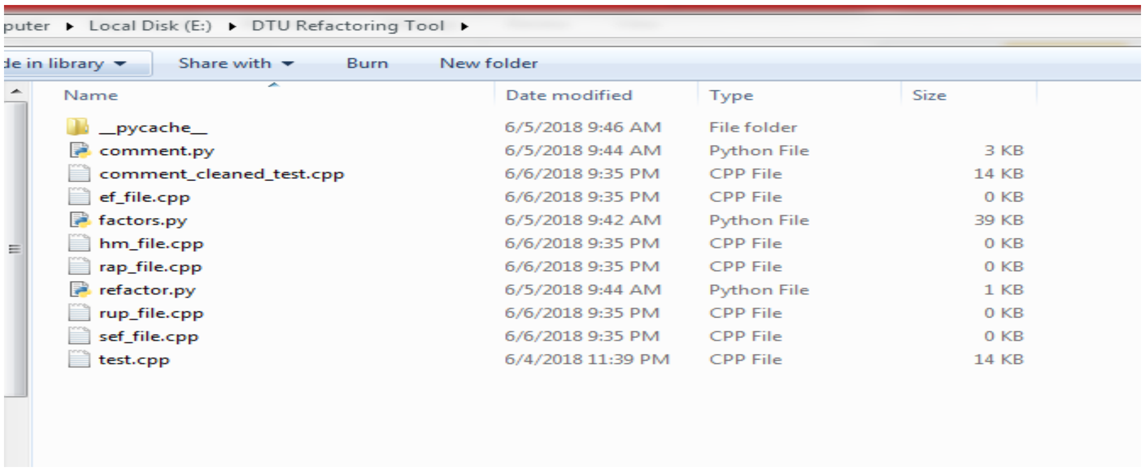
5.1 RESULTS



```
'chcp' is not recognized as an internal or external command,
operable program or batch file.

(base) C:\Users\DELL>E:

(base) E:\>cd "DTU Refactoring Tool"

(base) E:\DTU Refactoring Tool>python refactor.py

===========================
===========================

Welcome to the refactoring tool

Please enter the name of the file to refactor:
```

Fig 5.1: Executing refactor.py from Anaconda Prompt

In Fig 5.1 we are basically starting prototype of a tool from Anaconda Promopt. We have coded in such a way that it will automatically provide you all the refactoring performed .cpp file to the user. It will ask for the file name. User has to type the input file name. The other files in Fig 5.2 would be automatically created in tool folder.



Name	Date modified	Type	Size
__pycache__	6/5/2018 9:46 AM	File folder	
comment.py	6/5/2018 9:44 AM	Python File	3 KB
comment_cleaned_test.cpp	6/6/2018 9:35 PM	CPP File	14 KB
ef_file.cpp	6/6/2018 9:35 PM	CPP File	0 KB
factors.py	6/5/2018 9:42 AM	Python File	39 KB
hm_file.cpp	6/6/2018 9:35 PM	CPP File	0 KB
rap_file.cpp	6/6/2018 9:35 PM	CPP File	0 KB
refactor.py	6/5/2018 9:44 AM	Python File	1 KB
rup_file.cpp	6/6/2018 9:35 PM	CPP File	0 KB
sef_file.cpp	6/6/2018 9:35 PM	CPP File	0 KB
test.cpp	6/4/2018 11:39 PM	CPP File	14 KB

Fig 5.2: Automatically generated cleaned comment file and refactoring files

5.2 ANALYSIS AND COMPARISON

5.2.1 Tool Description

The refactoring tool is coded in Python (3.x) and is designed for handling C++(.cpp) files. The tool aims to be a 1st version of an advanced tool for refactoring the C++ code. We have considered five refactoring methods in our tool and it is also a tool for comment cleaning.

The comment cleaning file takes care of both types of comments, Single Line Comment and Multi Line Comment. The tool is not designed yet to handle, XML type codes or integration of other languages into the C++ files. It is designed to handle core C++ code currently in its first version.

Semantic rules have been used to identify the areas required as per each refactoring.

For comment cleaning:

We have used letter by letter parsing to identify for appearance of special character sequences such as “//” and “/*   */”. Hence one of the limitations, of our tool is appearance of these characters out of the required places. As the code syntax and semantics, plays an important role in the structure of the code, we identify the different attributes and classes present in the code. Hence the positive is to have the list of classes present in the code as well their attributes.

5.2.2 Effect of Refactoring Implemented in this tool on Maintainability

The five refactoring methods which have been implemented would be discussed here. We will discuss these refactoring methods on the basis of their comparison with other tools or in way it is different from other existing tools. We will discuss about the effect of applying these refactoring methods on maintainability on the basis of readability and understandability.

5.2.2.1. Hide Method (HM)

This refactoring changes the public methods of the class to private members, as public methods are loosely accessed, private access makes the calls secure, so if the member is only called from within the object of the class, making the method private makes it easier to keep the code secure.

This doesn't increase the lines of code much, and has the same readability and understandability. This refactoring changes only the access modifier of the function that's why it has no effect on lines of code. In Table 5.1 we have shown that the readability ad understandability will remain same even after Hide Method Refactoring.

5.2.2.2. Encapsulate Field (EF)

We have used the access types, to identify public attributes and create custom 'getter' and 'setter' functions and insert it into private access with their respective getter and setter methods as 'public'.

This creates a good level of abstraction, after this refactoring, as we can integrate type checking strongly using the getter and setter functions and also the members of the class are not directly exposed to user, making them more secure.

Although in Table 5.1 we have shown that it does increase the maintainability of the code as the lines of the code are increased and also access type of the member is changed, so updates have to be made accordingly.

Readability of the code and understandability is increased but proper checks have to be taken to replace the direct calls to the members to be replaced by the calls from the getter and setter and programmer has to adapt accordingly.

5.2.2.3. Self Encapsulate Field (SEF)

This refactoring is similar to Encapsulate Field (EF), but even the newly created getter and setter methods are set to private.

This decreases the maintainability of the code, as wherever the code is assigning the parameter, we have to update it to call the getter and setter method and too within the object, this might break the consistency of the code. In Table 5.1 we have shown that Self Encapsulate Field has no effect on readability and understandability starts decreasing.

5.2.2.4. Remove Assignment to Parameters (RAP)

We check for the parameters, creating a list of the parameters, we do this by iterating the whole file once, after removing the comments. On second iteration, on encountering the method, we keep check of the list of parameters and see whether they are being assigned; we replace that code by a dummy variable declaration and assignment to that dummy variable.

This is similar to creating a copy of the variable and updating the copy rather than the original variable itself.

In Table 5.1 we have shown that Remove Assignments to parameters refactoring increases the maintainability of the code, as any risk of modification is not there to the variable. Also increases the lines of code and understandability, making it very easy to maintain. It also increases the readability.

5.2.2.5. Remove Unused Parameters (RUP)

In this refactoring we identify the parameters of the methods and check whether it is ever used in the method, this is a type of obsolete code identification.

This does decrease lines of code wherever applicable, but makes it difficult to maintain, as the programmer needs to take care of the parameter removal and make sure to update the parameter list and also make sure that further calls don't refer to the removed parameter. In Table 5.1 we have shown that it increases the readability of the code as user does not need to read unused parameters. Sometimes developers and users get confused why they have written this parameter if it is not used then it creates confusion for them.

Table 5.1 Effect of Refactoring Methods on Maintainability

Refactoring Method	Readability	Understandability
Hide Method	—	↑
Encapsulate Field	↑	↑
Self Encapsulate Field	—	↓
Remove assignment to Parameters	↑	↑
Remove Parameters	↑	↑

In Table 5.1 the upper arrow ↑ indicates that the readability and understandability increases after applying refactoring and the downward arrow ↓ indicates that the readability and understandability decreases after applying refactoring. In Table 5.2 we have basically shown the comparison of three different tools. The three different tools that we have compared are as follows: Python refactoring Tool, Eclipse, Intellij Idea etc. The number of refactoring methods performed by Eclipse are more than Intellij Idea and Python Tool. Neither of them creates automatic refactoring files individually. It makes user more understandable.

Table 5.2 Comparison with other existing tool for refactoring

Tool	Python refactoring Tool	Intellij Idea	Eclipse
Automated Refactoring	It performs automatic refactoring on the input file provided to it. User does not need to worry where and which refactoring need to apply.	In Intellij Idea user will apply refactoring and then it will whether the applied refactoring is feasible to apply or not. It doesn't perform automatic refactoring.	In Eclipse the user will check at each line is it feasible to apply particular type of refactoring at a particular line of a code. It doesn't perform automatic refactoring.
System Requirements	Jupyter Notebook	Microsoft Windows 8/7/Vista/XP, 1 GB RAM minimum, 300 MB hard disk space +at least 1 G for caches.	Eclipse requires Java to run system needs the same Java version 315 MB available hard disk space Microsoft Windows,8/7/Vista/XP
Methods for Refactoring	1. Hide Method 2. Encapsulate Field 3. Self Encapsulate Field 4. Remove assignment to Parameters 5. Remove unused Parameters	1. Extract Method 2. Extract Method Object 3. Inline Superclass 4. Replace Method Code 5. Replace Temp with Query 6. Introduce constants	1. Extract Method 2. Rename 3. Simple Name Change 4. Extract Method Object
Languages Support	C++	Java JSP, XML, CSS, HTML and JavaScript	R, Ada, Java, PHP, C, C++
Generate automatic refactoring file for the input	It will generate automatically all refactorings cpp file which are performed on the input file code.	It will not generate any such file. It performs refactoring in the source file code only.	It will not generate any such file. It performs refactoring in the source file code only.
User Interface	It is compatible with user interface as user does not need to complete code.	Here, User needs to go through each line of code. So, it not compatible to user interfaces.	Here, User needs to go through each line of code. So, it is not compatible to user interface.

5.3 LIMITATIONS

The prototype of a refactoring tool is unable to separate comments if "/" is faced within a text string currently. It will work for c++ files only.

CHAPTER 6

CONCLUSION AND FUTURE WORK

We would conclude that the prototype of a refactoring tool is developed, which will work for c++ language only. This prototype will perform the comment cleaning in c++ projects and perform five different refactoring automatically without any human intervention. The individual file of each refactoring method would be generated. We have analyzed the effect of refactoring on maintainability on the basis of readability and understandability. We have also compared this prototype of a python tool with other existing refactoring tools on the basis of the platform, user interface, refactoring methods etc.

In the future, we will add other refactoring methods apart from these five refactoring methods. We will test this tool for large c++ projects and will produce complete automatically refactoring tool.

References

[1] A. Rani and H. Kaur, “Refactoring Methods and Tools”, *International Journal of Advanced Research in Computer Science and Software Engineering,* Volume 2, Issue 12, Dec. 2012.

[2] E. M. Hill, “Programmer-Friendly Refactoring Tools”.

[3] A. P. Black,“ Why Don’t People Use Refactoring Tools?”, *Computer Science Faculty Publications*, Portland State University, 2007.

[4] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman, “Compilers: Principles, Techniques, and Tools”, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

[5] D. Campbell and M. Miller, “ Designing refactoring tools for developers. *In Proceedings of the Workshop on Refactoring Tools”, ACM*, 2008.

[6] N. Chen and R. Johnson, “Toward Refactoring in a Polyglot World”, *In Proceedings of the 2nd Workshop on Refactoring Tools, ACM,* 2008.

[7] B. Daniel, D. Dig, K. Garcia, and D. Marinov, “Automated testing of refactoring engines”, *In Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering,* pp 194. ACM, 2007.

[8] Eclipse. The ASTRewriter class documentation, 2010. since Eclipse JDT Release 3.5.

[9] U. Eisenecker, K. Czarnecki, and S. Helsen, “Generative Programming”, *Addision Wesley*, 2000.

[10] T. Ekman, M. Schäfer, and M. Verbaere, "Refactoring is not (yet) about transformation", *In Proceedings of the 2nd Workshop on Refactoring Tools,* pp 1–4. ACM, 2008.

[11] M. Fowler, "Refactoring: improving the design of existing code", *Addison-Wesley Professional,* 1999.

[12] L. Frenzel, "The Language Toolkit: An API for Automated Refactorings in Eclipse-based IDEs", 2006. URL http://www.eclipse.org/articles/Article-LTK/ltk.html.

[13] R. M. Fuhrer, A. Kiezun, and K. M, "Refactoring in the Eclipse JDT: Past, present, and future", *In Proceedings of the 1st Workshop on Refactoring Tools,* pp 31–32. ACM, 2007.

[14] D. Jemerov, "Implementing refactorings in IntelliJ IDEA", *In Proceedings of the 2nd Workshop on Refactoring Tools,* pp 1–2. ACM, 2008.

[15] M. Klenk, R. Kleeb, and M. Kempf, "Refactoring Support for the Groovy-Eclipse Plugin", 2008.

[16] H. Li, C. Reinke, and S. Thompson, "Tool support for refactoring functional programs", *In Proceedings of the* 2003 *ACM SIGPLAN workshop on Haskell,* pp 38, ACM, 2003.

[17] E. Mealy and P. Strooper, "Evaluating software refactoring tool support", *In Software Engineering Conference, Australian,* pp 10, 2006.

[18] J. C. Miller and B. M. Strauss, III, "Implications of automated restructuring of cobol", *SIGPLAN Not..* 22 (6): pp76–82, 1987.

[19] M. Bisi and N. K. Goyal, \Software development e orts prediction using arti cial neural network," IET Software, vol. 10, no. 3, pp. 63-71, 2016, issn: 1751-8806.

[20] I. Moore, "Automatic inheritance hierarchy restructuring and method refactoring", *In Proceedings of the 11th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications,* pp 250. ACM, 1996.

[21] E. Murphy-Hill, "A Model of Refactoring Tool Use", *In Proceedings of the 3rd Workshop on Refactoring Tools,* ACM, 2009.

[22] E. Murphy-Hill and A. Black, "Making Refactoring Tools Part of the Programming Workflow", 2008.

[23] E. Murphy-Hill, C. Parnin, and A. Black, "How we refactor, and how we know it", *In International Conference on Software Engineering* 2009, *volume* 2, pp 0–24, 2009.

[24] J. Overbey and R. Johnson, "Generating Rewritable Abstract Syntax Trees", *Software Language Engineering,* pp 114–133, 2009.

[25] J. Kerievsky, " Refactoring to Patterns", 1st ed. Reading, MA: Addison Wesley, 2004. [Online] Available: Safari e-book.

[26] K. O. Elish and M. Alshayeb, "A Classification of Refactoring Methods Based on Software Quality Attributes", *Arabian Journal of Science and Engineering,* Volume 36, Issue 7, pp 1253–1267, Nov. 2011.L. Breiman, \Bagging predictors," Machine Learning, vol. 24, no. 2, pp. 123-140, 1996, issn: 1573-0565.

[27] B. D. Bois, P. V. Gorp and T. Mens,"A Discussion of Refactoring in Research and Practice".

[28] F. Simon, F. Steinbruckner, and C. Lewerentz. Metrics based refactoring. In Proc. European Conf. Software Maintenance and Reengineering, pp 30–38. IEEE Computer Society Press, 2001.

[29] Y. Kataoka, M. D. Ernst, W. G. Griswold, and D. Notkin. Automated support for program refactoring using invariants. In Proceedings of the International Conference on Software Maintenance, pp 736–743. IEEE Computer Society Press, 2001.