SWaP : Meta Analysis of Static Analyzer Reports for Accurate Warning Prioritization

By

Khushbu Yadav

IIIT-D-MTech-CS-IS-20-MT18087

Under the Guidance of
Dr. Rahul Purandare

Submitted in partial fulfillment of the requirements
for the Degree of M.Tech. in Computer Science & Engineering

To

Indraprastha Institute of Information Technology Delhi

Aug,2020

Certificate

This is to certify that the thesis titled **"SWaP : Meta Analysis of Static Analyzer Reports for Accurate Warning Prioritization"** submitted by **Khushbu Yadav** for the partial fulfillment of the requirements for the degree of *Master of Technology* in *Computer Science & Engineering* is a record of the bonafide work carried out by her under my guidance and supervision at Indraprastha Institute of Information Technology, Delhi. This work has not been submitted anywhere else for the reward of any other degree.

Dr. Rahul Purandare

Department of Computer Science

Indraprastha Institute of Information Technology, New Delhi

Abstract

Java being designed in a flexible and user-friendly demeanour, makes it the most accepted programming language for the development of web applications and platforms. Due to the immense popularity, there comes the responsibility of validation of the Java software when the software safety, reliability and quality control is of utmost importance. The detection of bugs in the software during the early stage helps to prevent the unbearable cost of human effort and time to fix them when captured at later stages. Hence many effective tools have been developed over the years to find potential bugs in the software by analysing the code statically.

The static analysis tools use different techniques to detect a variety of bugs in the software. As all of these tools follow distinct techniques, the bugs detected hold a minimal overlap, thereby making it difficult to merge the analysis reports generated by them. In this thesis, we propose a mechanism of merging the results of the static analysis tools namely SpotBugs, PMD, SonarScanner and CheckStyle and reporting analysis results in a generic manner. We have also incorporated the prioritizing policy to increase the overall efficiency of the final integrated tool. This way, the user can leverage the benefits from various static analyzers in order to improve the overall quality of the software.

Keywords: Static Analysis Tools, Bugs, Warnings

Acknowledgments

It is my privilege to express my sincerest gratitude to my advisor, Dr. Rahul Purandare, for giving me the opportunity to work on this thesis project and for his valuable inputs, guidance, and wholehearted support throughout the research. One simply could not wish for a better or friendlier supervisor. I would like to express my deepest appreciation to the program analysis group who have supported me throughout with their patience and knowledge and always showed me the right direction to proceed in order to achieve the goal. I would like to express my gratefulness to the institute, IIIT-Delhi, for providing me with all the necessary facilities to carry out my thesis.

Lastly but importantly I would like to express profound gratitude to my parents and to my friends for their constant encouragement throughout the process of research. This accomplishment would not have been possible without them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

In the development of safety and security-critical applications, it becomes very crucial to come up with a fail-safe design as even a minor problem or error could lead to catastrophic failure resulting in human loss. It's the responsibility of the software developers to avoid such fatal scenarios by preventing the programs from any unexpected failures or errors, otherwise this would lead to consequences which may not be even possible to take care of. Hence there arises a need of discovering the bugs or errors during the early stages in the development phase in order to prevent the additional cost to fix them. Therefore, the validation of the software applications ensuring the software quality plays a vital role and should be carried out throughout the development phase.

In recent years, various static analysis tools have been developed which automatically detect error prone anomalies in the program. All of these tools follow different strategies to identify the potential bugs that involve abstract interpretation, model checking, syntactic pattern matching, theorem proving, symbolic execution, data flow analysis, type systems,

etc. The static analysis tools analyse the source code using one or more of the above strategies and produces information regarding the bugs in the warning report. The bugs identified by multiple analysers cover a wide range of defects with a very little overlap. Hence due to high volume of warnings, it becomes cumbersome to figure out which warnings need an immediate fix.

There is a need for a meta tool which integrates the results of various static analysis tools and provides a way of prioritising the warnings generated by each of them as highlighted by [16]. In this paper, we propose an approach to combine the results from various static analysis tools and present the bug report in a generic form and thereby facilitating the user to leverage the benefits of multiple static analyzers. In our project, we have chosen four popular, publicly available static analysis tools namely SpotBugs, SonarScanner, PMD and CheckStyle.

1.2 Java Static Analysis Tool

We have discussed about each of the selected static analysis tools in detail below:

1.2.1 SpotBugs

SpotBugs [5] is the spiritual successor of FindBugs, which is an open source static code analyser that detects possible bugs in Java programs. It is basically a bug pattern detector that identifies more than 400 patterns in Java bytecode. Byte Code Engineering Library and ASM bytecode framework is used to analyse the Java classes and bugs found during analysis are then matched with source code. In order to balance the precision, accuracy and reliability, it uses a series of relevant techniques. The strategies adopted in SpotBugs static analyser can be categorized into the following:

(a)Linear Code Scan: The Java bytecode is scanned linearly by the detectors

(b)Control Sensitive: The control flow graph is taken into account for analysing the procedures.

(c)Analyse over Class Structure: Detectors concentrate over the structure without giving much importance to the actual code.

(d)Dataflow Analysis: Detectors perform simple interprocedural analysis using control and data flow graphs.

The potential bugs found after the analysis are classified into four categories based on the severity as (1) scariest (2) scary (3) troubling (4) of concern. SpotBugs can be further expanded by writing customised detectors in Java.

1.2.2 PMD-Project Mess Detector

PMD [2] is an open source code analyser that performs static analysis on the Java source code based on the ruleset selected during the execution. PMD creates Abstract Syntax Tree while checking the java source code and then applies the ruleset over this labelled tree to find potential bugs. Unlike SpotBugs, it lacks the knowledge of data flow analysis. The bugs found during the analysis majorly fall in the following categories: CPD (cut and paste detector), compliance with coding standards and coding anti patterns. PMD not only tries to capture the erroneous code but also looks for violation of stylistic conventions that could possibly lead to suspicious results. PMD incorporates various detectors which detects issues concerning stylistic rules and also permits us to choose a specific group of detectors to run. PMD has 22 default rule sets that cover multiple aspects of Java code and tries to detect bugs around these aspects. These rulesets check for security issues, performance issues, good practices issues and correctness issues. Programmers can easily extend the PMD ruleset by writing new bug pattern detectors using Java or XPath.

1.2.3 CheckStyle

Similar to PMD and SpotBugs, CheckStyle [1] is also an open source static code analysis tool for examining the Java source code. CheckStyle helps to ensure that source code adheres to the coding standards in order to improve the quality, reusability and readability of the source code. It does not check for the completeness of the code (i.e. the content is not being analysed). CheckStyle uses the configuration files namely "Google Java Style" and "Sun Java Code Conventions" by default. It captures the issues related to various aspects of source code such as class and method design issues. It also checks for code layout and formatting issues. CheckStyle is highly configurable and allows the programmer to perform customization by means of using multiple parameters. CheckStyle can be expanded to define custom rules of coding standards.

1.2.4 SonarScanner

SonarQube [3] is an open source platform, which serves as the central server, developed for the continuous inspection of the source code. SonarScanner [4] serves as the client application that helps to run the analysis over the target project and sends the outcome produced to the server (SonarQube) to process it. SonarQube provides an easier way to perform automatic reviews by statically analysing the code to detect bugs, code smells and security issues. SonarScanner, like SpotBugs, relies on syntactic checks and data flow analysis(interprocedural analysis) to detect bugs. It generates reports with warnings related to duplicated code, unit tests, coding standards, code coverage, cyclomatic complexity of the code, comments, potential errors etc. This static analysis tool is not easily expandable. Table 1.1 briefly summarizes each of the static analysis tools.

Static Analysis Tool Name	Version Used(Release Year)	Input Format	Interface	Technique
SpotBugs	4.0.4(2020)	Byte code	CL	Syntax,Data Flow
SonarScanner	4.3.0.2101(2020)	Source code	CL	Syntax,Data Flow
PMD	6.24.0(2020)	Source code	CL,GUI,Ant,IDE	Syntax
CheckStyle	8.33(2020)	Source code	CL,GUI	Syntax

Table 1.1: Summary of the static analysis tools

1.3 Research Motivation

Static Analysis Tools helps the programmers to check the source code and discover the susceptible error prone areas based on the ad hoc techniques used respectively. Moreover, each analyser follows a distinct pattern in order to detect the bugs lying in the code and thereby covering a wide range of defects with a very little overlap i.e. minimal correlation. Different analysers are good at capturing certain defects by checking against specific aspects of the code. Hence there is no single bug finding tool which gives the best results. With the large volume of warnings reported by the static analysers, the programmers find it difficult to look over each bug in order to fix them. Many times, the bugs are found to be false positives. This problem gives rise to the need of developing a meta tool that integrates the results of multiple static analyzers intelligently by combining the important warnings generated by each of the tools (SpotBugs, PMD, SonarScanner and CheckStyle). Moreover, prioritising the merged warnings would help the programmer to figure out which of the reported bugs needs immediate action to be performed.

1.4 Thesis Outline

Our dissertation embodies five chapters which are listed as (after the Introduction):

Chapter 2 provides a background on the evaluation of various static analysis tools. Furthermore, multiple techniques have been discussed to prioritize the warnings reported by static analysers.

Chapter 3 gives the detailed description of the methodology proposed to integrate the warning detected by multiple static analysis tool and provides a mechanism to assign priorities to the integrated warnings in the final merged report of the meta tool.

Chapter 4 presents the user study depicting the evaluation of the meta tool against the existing static analysers (SpotBugs, CheckStyle, PMD and SonarScanner).

Chapter 5 summarizes the entire thesis project in few lines and suggest the future work that can be done.

Chapter 2

Literature Review

The main goal of static analysis tools is to analyse the source code and discover certain flaw susceptible areas. With the passage of time, different analyzers have been developed to serve the purpose of detecting the bugs. The techniques to prioritise the warnings reported by multiple static analysis tools has also been the main concern of researchers. In addition to that, a lot of research has been done to evaluate the effectiveness of several analysers which has been discussed in this section.

2.1 Evaluation of Multiple Validation Tools

Almazan *et al.* [16] tries to compare the outputs generated by different static analyzers. In the experiment, they have focused on 5 bug finding tools namely Bandera, ESC/Java 2, FindBugs, JLint and PMD. They have run these tools over various Java programs(variable-sized) from multiple domains. As each analyser reports a bulk of warnings (see table 2.1), the evaluation process could become tedious and hence to ease up the task they have cross checked the common warnings and focused mainly on three checking tasks - concurrency errors, null dereferences and array bounds error. They have proposed two

metrics to compare the effectiveness of the static analyser. First metric is the normalized warning total which can be defined as the summation of the normalized warning count by each of the tools. Normalized warning count for tools is defined as the ratio of warnings reported for a class to the maximum number of warnings reported among all the classes. The second metric is the unique warning total, which gives the information about distinct warnings reported by each tool.

Warnings	**ESC/JAVA**	**FindBugs**	**Jlint**	**PMD**
Concurrency Warnings	126	122	8883	0
Null Dereferencing	9120	18	449	0
Null Assignment	0	0	0	594
Index out of Bounds	1810	0	264	0
Prefer Zero Length Array	0	36	0	0

Table 2.1: Warning count for the categories by Almazan *et al.* [16]

Koricherla *et al.* [9] also tried to compare the outputs generated by several validation tools. They have chosen four static analysis tools in lieu of the experiment i.e. FindBugs, PMD, CheckStyle and UCDetector. Their evaluation process involves a way of accumulating the warnings reported by the tools into two categories namely Important Bugs (Malicious code, clone, exception handling issue, etc.) and Unimportant Bugs (naming conventions, program styling, etc.). They calculated the percentage of warnings falling in the important bug category and based on that checked the efficiency of each static analyser. They showed that maximum warnings reported by these tools were non overlapping as they follow different techniques to capture them (see Fig 2.1). Moreover, they showed that FindBugs discovered 100% of the important warnings. They also concluded that PMD is more efficient than CheckStyle.

Barr *et al.* [14] have tried to compare two disparate approaches-statistical defect prediction and static bug finders with a similar footing of capturing the defect prone areas of the code. They have used the historical defect data to compare the two approaches and seek

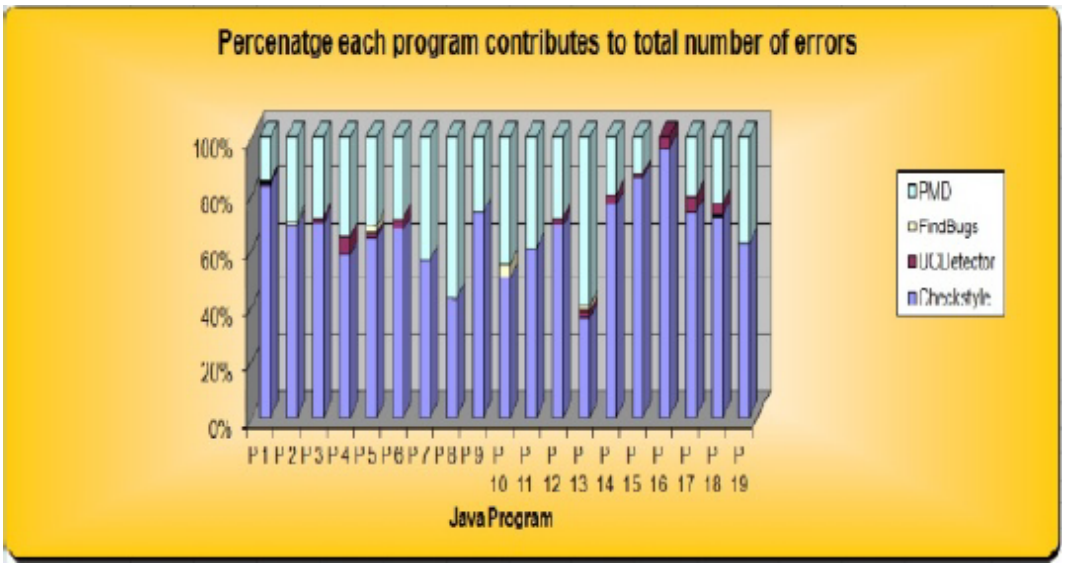

Figure 2.1: The percentage of errors detected by each tool for 19 Java programs by Koricherla *et al.* [9]

similarities. They have shown that statistical defect prediction performs better than PMD in most of the cases (partial as well as full credit accounting). Whereas statistical defect prediction tends to perform worse than FindBugs in full credit accounting. However, they have shown that when the ordering of warnings (generated by FindBugs) is done using priorities produced by defect predictors, it significantly improves the static bug finder priority levels in the majority of the cases.

Feldt *et al.* [13] have discussed the comparison regarding capabilities of various static analyzers to detect the Java concurrency bugs. The concurrency bugs in multithreaded programs are not easily detected compared to the bugs in sequential programs. The concurrency issues could be broadly classified into two intrinsic properties i.e. safety and liveness properties. The safety property can be stated as that nothing bad will happen during the program execution whereas the liveness property states that something good will eventually happen as execution progresses. The major issues under these properties are deadlocks, race conditions, livelocks, starvation, etc. They have selected FindBugs,

Joint, Coverity Prevent and Jtest for the experiment. They have run these tools on 20 multithreaded Java programs. Based on the defect detection(refer Table 2.2) and ration and by determining the false positive rates of the tools, they have shown that Jtest is best suited for capturing the bugs related to data race and atomicity violation, but it has a high false positive rate compared to other tools. FindBugs also turned out to be a better one as it checks for a large number of concurrency bug patterns compared to others with a reduced false positive.

Warning Type	**Coverity Prevent**	**FindBugs**	**Jtest**	**Jlint**
General	5	2	136	0
True	4	8	21	11
False Positive	4	5	16	20
Undetermined	3	1	8	3
Total	16	16	181	34

Table 2.2: Warning count per static analyser by Feldt *et al.* [13]

Valente *et al.* [7] have done a study to check on the relevance of warnings reported by FindBugs and PMD. They have taken into account the number of warnings reported and its lifetime to perform comparative analysis. They showed that the warning relevance rate of FindBugs was more than PMD (generating too many false positives).

2.2 Warning Classification and Prioritization Techniques

Ernst *et al.* [11] have proposed a way of eliminating the false positive warnings from a program by prioritizing them. They have inspected the warnings generated by three static analysers in the experiment- FindBugs, Jlint and PMD for three subject programs Columba, Lucene and Scarab and have found that the majority of the warnings do not indicate real bugs. They proposed a history-based warning prioritization algorithm (program specific

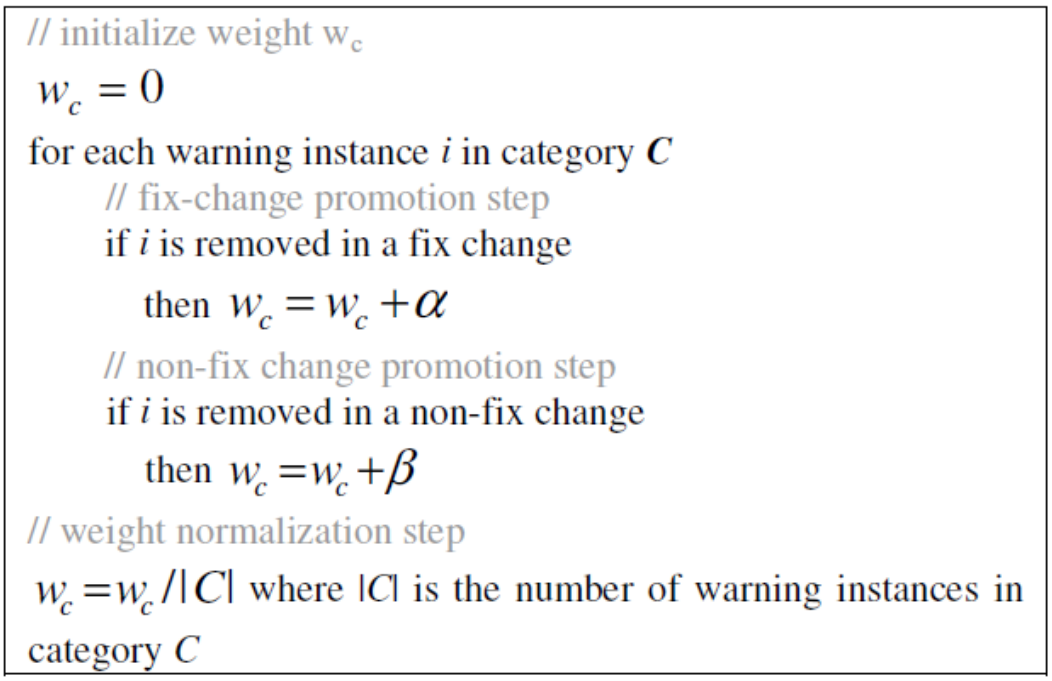

Figure 2.2: History Based Warning Prioritizing Algorithm by Kim *et al.* [11]

prioritization) by mining the software change history of removed warnings during the bug fixes (see Fig 2.2). Whenever the developer tries to fix the warning, it clearly shows that the warning was important. Moreover, when the warning remains unattended for a long period of time, it shows that the warning does not point to potential bugs. Based on this intuition, they have assigned weights to each of the warning categories which is directly proportional to the number of warning instances of the category being eliminated from the software history by a fixed change. This way they have assigned high priority to the warnings belonging to the categories with high weights and vice versa. Similarly, Kim *et al.* [10] also proposed a prioritizing algorithm based on the lifetime of warnings captured from the software change history.

Snavely *et al.* [8] have developed an automated statistical classifier which predicts whether the alerts generated by a static analyser are true or false positive by means of combining multiple static analysis tools, features from the alerts, alert fusion, code base metrics and

archived audit determination. The data used in the experiment consists of archives for 19 CERT audited codebases. The classification techniques being compared are Classification and Regression Trees, Lasso Logistic Regression, Extreme Gradient Boosting and Random Forest. They have developed the classifier using partition of the data and tested its performance based on merit measurements like specificity, sensitivity and accuracy. Similarly, Meirelles *et al.* [15] have also developed a prediction model that extracts the feature from the reports generated by different static analysers and are used to train a set of decision trees using AdaBoost to create a stronger classifier. At last this classifier is being used to rank the static alarms based on the probability of whether the alarm corresponds to the actual bug in the source code.

Allier *et al.* [6] have proposed a framework for comparing the alert ranking algorithms to check which one is best among them. The algorithms involved in this experiment are FeedBackRank, RPM, Z-Ranking, AWARE, AlertLifeTime and EFindBugs. Moreover, the framework uses a benchmark which covers two programming languages- Java and Smalltalk along with three bug finding tools namely FindBugs, PMD and SmallLint. They have shown that AWARE works best for ranking the alerts followed by FeedBackRank.

Liang *et al.* [12] have tried to improve the efficiency for warning prioritization by means of ranking scores of static analysis tools. In order to do so, they have taken into consideration three categories of impact factors as input features of the training set and have proposed a new heuristic for discovering the actionable alerts for labelling the training set. The training set is built by identifying the generic bug fix revisions, generic bug related lines then generating static analysis warnings and finally extracting the training set. They have used machine learning predictors to provide ranking scores for warnings. Bayesian Network, Logistic Regression, K-nearest Neighbours, Bootstrap Aggregating, Random Tree, and Decision Table are the algorithms used to train the predictor and 10 folds cross is used in the validation phase.

Ruthruff *et al.* [17] also provided automated support to meet the challenge of detecting the bogus false positive warnings and actionable warnings that are not acted on. They have used a logistic regression model which predicts the foregoing types of warnings from signals in the warnings and implicated code. In order to predict the actionable static analysis warning with high accuracy, they have used screening methodology to quickly discard factors with low predictive power.

As we cannot rely on a single static analysis tool to capture all the actionable alerts, our methodology tries to address the above problem and provides a way to combine the important warnings reported by multiple static analysis tools.

Chapter 3

Methodology

3.1 A Small Example

The sample code (processing sql statements with jdbc) in the Fig. 3.1 is compiled successfully by the Java 1.11 compiler without any warning or error. However, when it was brought to the four static analysers, several defects were detected in the source code. SpotBugs reported 6 warnings which includes: (1)Hardcoded constant database password at line 7; (2) main.java.CloseConn.main(String[]) may fail to close Connection at line 7; (3) main.java.CloseConn.main(String[]) may fail to close Statement at line 9;(4) main.java.CloseConn.main(String[]) may fail to clean up java.sql.Statement Obligation to clean up resource created at line 9. (5) main.java.CloseConn.main(String[]) may fail to clean up java.sql.ResultSet Obligation to clean up resource created at line 10 (6) Exception is caught when Exception is not thrown at line 17. PMD reported 5 warnings that mentions "All methods are static. Consider using a utility class instead. Alternatively, you could add a private constructor or make the class abstract to silence this warning at line 4", "Ensure that resources like this Connection object are closed after use at line 7", "Ensure that

```
1    package com.company;
2    import java.sql.*;
3    public class Main {
4    public static void main(String[] args) {
5               try{
6                  Class.forName("com.mysql.jdbc.Driver");
7                  Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/sonoo","root","root")
8                  //here sonoo is database name, root is username and password
9                  Statement stmt=con.createStatement();
10                 ResultSet rs=stmt.executeQuery("select * from emp");
11                 while(rs.next())
12                       System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
13                 }
14              catch(Exception e){
15                 System.out.println(e);
16               }
17          }
18      }
```

Figure 3.1: A sample Java code

resources like this Statement object are closed after use at line 9",“This statement should have braces at line 11”, “Ensure that resources like this ResultSet object are closed after use at line 10”. SonarScanner reported 9 issues which says: “The file should be located in main/java at line 1”, “Move the array designator from the variable to the type at line 4”, “Remove this Class.forName() as it is useless at line 6”, “Use try-with-resources or close this "Connection" in a "finally" clause at line 7”, “Use try-with-resources or close this "Statement" in a "finally" clause at line 9”, “ Use try-with-resources or close this "ResultSet" in a "finally" clause at line 10” and other warnings related to better programming practices. CheckStyle produced 36 warnings related to the formatting and class design issues. There is very little overlap between the errors detected by each of the static analysers.

3.2 Approach

In order to integrate the results generated by the four static analysis tools, we have proposed the following strategy which is illustrated in the figure 3.2.

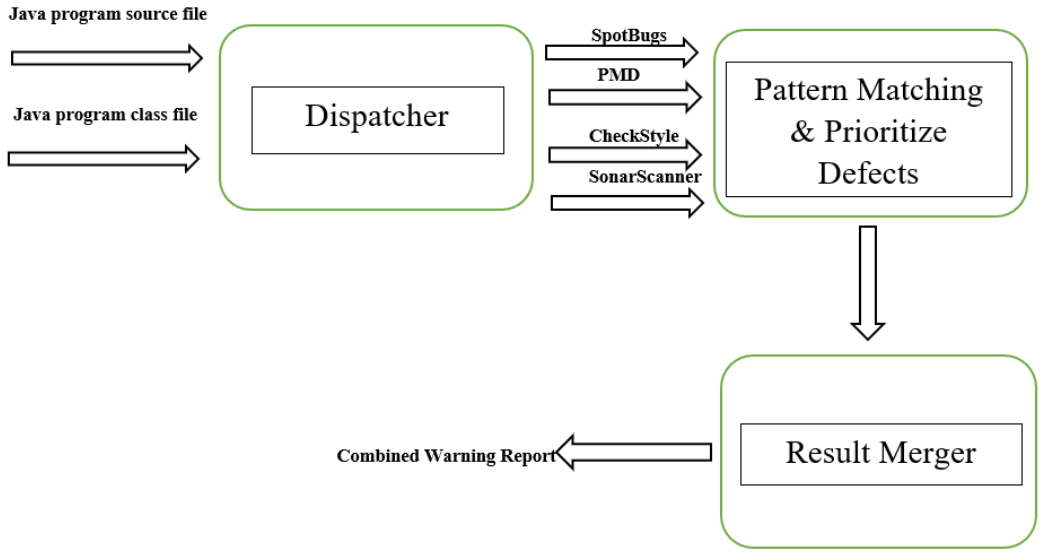

Figure 3.2: A general architecture to merge multiple static analysis tool's results

3.2.1 Algorithm

SWaP{

List<AnalysisReports> dispatcher(sourceCode, classFile);

List<AnalysedReports> patternMatcher(List<AnalysisReports>);

File finalResult=resultMerger(impSpot,impPmd,impSonar,impCheck);

Sort the finalResult based on lineNumber;

Display finalResult;

}

List<AnalysisReports> dispatcher(sourceCode, classFile){

Runs the script file to generate analysis reports w.r.t each static analyser

return SpotBugs_Analysis_Report,SonarScanner_Analysis_Report,

Pmd_Analysis_Report,CheckStyle_Analysis_Report

}

```
patternMatcher(List<AnalysisReports>){
impSpot=spotbugParser(SpotBugs_Analysis_Report)
impSonar=sonarParser(SonarScanner_Analysis_Report)
impPmd=pmdParser(Pmd_Analysis_Report)
impCheck=checkstyleParser(CheckStyle_Analysis_Report)
return impSpot, impSonar, impPmd, impCheck
}
spotbugParser(spotbugReport){
Scan each warning reported against the ruleset of SpotBugs.
if (warning.matches(ruleset))
        then mark warning as important;
else
        mark warning as unimportant.
add important warnings by overriding the spotbugReport.
return spotbugReport
}
pmdParser(pmdReport){
Scan each warning reported against the ruleset of Pmd.
if (warning.matches(ruleset))
        then mark warning as important;
else
        mark warning as unimportant.
add important warnings by overriding the pmdReport.
return pmdReport
}
```

```
sonarParser(sonarReport){
Scan each warning reported against the ruleset of SonarScanner.
if (warning.matches(ruleset))
        then mark warning as important;
else
        mark warning as unimportant.
add important warnings by overriding the sonarReport.
return sonarReport
}
checkParser(checkstyleReport){
Scan each warning reported against the ruleset of CheckStyle.
if (warning.matches(ruleset))
        then mark warning as important;
else
        mark warning as unimportant.
add important warnings by overriding the checkstyleReport.
return checkstyleReport
}
File resultMerger(impSpot,impPmd,impSonar,impCheck){
File finalReport;
Append the warnings of impSpot into the finalReport with default priority(low).
Foreach warning w in impSonar
    Foreach warning s in finalReport
        if (w.line==s.line)
            if(similarity(w.warning,s.warning)<0.5)
                add w to finalReport with default priority.
```

```
            else
                Set high priority for warning s in finalReport.
        else
            add w to the finalReport with default priority.
Foreach warning w in impPmd
    Foreach warning s in finalReport
        if (w.line==s.line)
            if(similarity(w.warning,s.warning)<0.5)
                add w to finalReport with default priority.
            else
                Set high priority for warning s in finalReport.
        else
            add w to the finalReport with default priority.
Foreach warning w in impCheck
    Foreach warning s in finalReport
        if (w.line==s.line)
            if(similarity(w.warning,s.warning)<0.5)
                add w to finalReport with default priority.
            else
                Set high priority for warning s in finalReport.
        else
            add w to the finalReport with default priority.
return finalReport
}
```

The target Java source code along with its class file is fed as an input to the dispatcher. The main task of the dispatcher is to run four static analysis tools namely SpotBugs,

CheckStyle, PMD and SonarScanner over this input program. Before the dispatcher tries to begin the analysis, a connection to SonarQube server is made to capture the results of SonarScanner static analyser for performing further analysis. Once the connection is established successfully, the dispatcher runs a script which helps the static analysers to execute parallelly. When all of the static analysis tools have complete their inspection on the java program, the dispatcher pipelines the analysis report generated by each of these tools to their corresponding files.

The pattern matcher uses four parsers corresponding to each static analysis tool which helps to scan the respective analysis report and classifies the warnings detected in the target source code into two categories i.e. Important warnings and Unimportant warnings. The parser compares the analysis report against the ruleset table of the respective static analyser. The ruleset table is prepared by scanning several bug tracking repositories like Apache Jira and Oracle Java Bug Repository to figure out which warnings lead to potential bugs. It is observed that warnings that are immediately fixed when being reported hold utmost importance. Hence, while scanning through the Oracle Java bug repositories and issue tracking system-JIRA, the major selection criterias to identify important warnings were status(closed), resolution(fixed), and the related priority of the warning.(P1-P3 in case of Oracle Java Bug repository and major-critical in case of Jira). The warnings falling in the above criteria are considered as important warnings. The important warnings includes issues related to Cyclomatic Complexity, Improper Boolean Checks and Complex Expressions, Multithreading and synchronization, Java Bean, Conditional loops and complex control statements, Abstract and Clone interface Rules, Null checks and Comparison,Unhandled Exceptions, and Out of Bound conditions. The ruleset table contains information regarding the important warnings of a particular static analyser. When the category of the detected warnings matches with the rules in the ruleset table, it is categorized as Important warning.The ruleset table 3.2 is the subset of

source ruleset [5].The ruleset table 3.3 is the subset of the source ruleset [2]. The ruleset table 3.4 is the subset of the source ruleset [1]. The ruleset table of SonarScanner is shown in the table 3.1.

Category	**Level**
Bug	Blocker
Bug	Critical
Bug	Major
Bug	Minor
Vulnerability	Blocker
Vulnerability	Critical
Vulnerability	Major
Vulnerability	Minor
Code Smell	Critical
Code Smell	Blocker

Table 3.1: Ruleset table for SonarScanner

The defects matched with the ruleset table are accumulated together in the respective analysis reports and sent for the next step to combine and prioritize the warnings. In the prioritization technique, warnings detected by more than one static analyser are assigned higher priority than the ones detected by a single tool. The result merger directs all the warnings of SpotBugs to meta tool report with priority set to "low". Then the important warnings reported by SonarScanner are compared against the warnings of meta tool to raise the priority level of the warning to "high" when similar warning is detected. Since every analyser has its own way of reporting warnings with somewhat different message, we have detected similarity between warnings based on the description of the warning i.e. if the warning reported by two different static analyser for particular line in the source code matches with each other with a probability$>$ 0.5, then it is highly likely that both are pointing to the same issue. If the warnings captured by SonarScanner do not match with meta tool report then they are simply appended to the SWaP report with priority set to low. Similar process is followed for adding the PMD and CheckStyle warnings to

Ruleset	Rule	Description
Bad Practices	BC	Argument type should not be pre assumed by Equals method.
Bad Practices	BIT	Bitwise operations check.
Bad Practices	CN	Class defines clone() but doesn't implement Cloneable, clone method does not call super.clone()
Bad Practices	DE	Exception may be drooped or ignored by the method.
Bad Practices	Eq	Covariant equals() method defined by abstract class
Bad Practices	HE	Class defines hashCode() but not equals(), Class inherits equals() and uses Object.hashCode(), Class defines equals() but not hashCode().
Bad Practices	NP	Possible null return by toString() and Clone method.
Bad Practices	OS	Method might not close the stream.
Bad Practices	RC	Suspicious reference comparison of Boolean values.
Bad Practices	Se	Non-serializable value stored into the instance field of a serializable class, Non-serializable class has a serializable inner class, declare Object as return type for readResolve method.
Bad Practices	FI	Finalizer nulls out the field.
Bad Practices	UI	Extension of class may lead to unsafe usage of GetResource.
Bad Practices	IMSE	Dubious catching of IllegalMonitorStateException.
Bad Practices	ES	Comparison of String parameter is on;y permitted using == or !=, Class defines compareTo(...) and uses Object.equals().
Bad Practices	It	NoSuchElementException cannot be thrown by iterator's next() method.
Bad Practices	ME	Enum field is public and mutable and unconditionally enum() sets its field.
Bad Practices	RR	Method ignores results of InputStream.read().
Bad Practices	RV	Negating the result of compareTo()/compare().
Bad Practices	ODR	Method may fail to close database resources.
Correctness	BIT	Check for signs of bitwise operation involving negative numbers.
Correctness	EC	Using pointer equality to compare different types, Call to equals(null).
Correctness	DMI	Reversed method arguments.
Correctness	Eq	Equals method always returns true, equals() method called doesn't override Object.equals(Object).
Correctness	NP	Method with Optional return type returns explicit null.

Ruleset	Rule	Description
Correctness	UR	The field method called from superclass's constructor performs uninitialized read
Correctness	NM	Class defines hashcode(); should it be hashCode()?, Apparent method/constructor confusion, Class defines tostring(); should it be toString()?
Correctness	HE	Hashed data structure used in class without invoking hashCode().
Correctness	RANGE	Array Index is out of bounds.
Correctness	SQL	Method tries to use a result set field with index 0, Unnecessary type check done using instanceof operator.
Multithreaded	DL	Synchronization on Boolean, Synchronization on boxed primitive.
Multithreaded	IS	Inconsistent synchronization.
Multithreaded	MWN	Mismatched wait() or notify()
Multithreaded	NP	Redundanct check of null value using instanceOf operator.
Multithreaded	TLW	Holding two locks waiting for another resource
Multithreaded	UG	Using synchronized set method and unsynchronized get method.
Multithreaded	UW	Unconditional wait.
Multithreaded	WS	Class's writeObject() method is synchronized but nothing else is.
Performance	DM	Explicit garbage collection; extremely dubious except in benchmarking code, Method invokes inefficient Boolean constructor; use Boolean.valueOf(...) instead.
Security	DM	Hardcoded constant / empty database password.
Security	HRS	HTTP Response splitting vulnerability.
Security	XSS	cross site scripting vulnerability is shown by the servlet in error page.
Dodgy Code	BC	instanceof will always return true, Unchecked/unconfirmed cast of return value from method.
Dodgy Code	CD	Test for circular dependencies among classes.
Dodgy Code	CI	Class is final but declares a protected field.
Dodgy Code	DM	Thread passed where Runnable expected.
Dodgy Code	DMI	Non serializable object written to ObjectOutput.
Dodgy Code	Eq	Unusual equals method, Class doesn't override equals in superclass.
Dodgy Code	NP	Dereferencing the readLine()'s result without performing any null check.

Ruleset	Rule	Description
Dodgy Code	QF	Improper and unconventional increment in conditional loop.
Dodgy Code	REC	Exception is caught when exception is not thrwon.
Dodgy Code	Se	Private readResolve method not inherited by subclasses.
Experimental	OBL	Over checked exception, method might not close the resource.

Table 3.2: Ruleset Table for SpotBugs

Ruleset	Rule	Description
Best Practices	AbstractClassWithoutAbstractMethod	Abstract class does not contain any abstract method.
Best Practices	ASwitchStmtsShouldHaveDefault	Add default option to switch to catch unspecified values.
Code Style	AbstractNaming	Abstract classes should be named 'AbstractXXX'.
Code Style	ControlStatementBraces	Braces are required around conditional statements.
Code Style	LocalVariableCouldBeFinal	Local variables declared only once can be declared as final.
Code Style	MethodArgumentCouldBeFinal	Method arguments never re-assigned within the method can be declared as final.
Code Style	SuspiciousConstantFieldName	Constant fields should be declared with uppercase to differentiate them with other variables.
Design	CyclomaticComplexity	Concentrating too much decisional logic within a single method makes it hard to read or modify.
Design	SimplifyBooleanExpressions	Avoid unnecessary comparisons in boolean expressions.

Ruleset	Rule	Description
Design	SimplifyBooleanReturns	Avoid unnecessary conditional tests while returning a boolean.
Design	SimplifyConditional	Do not check for null before an instanceof as it returns false when argument is null.
Error Prone	AvoidInstanceofChecksInCatchClause	Every exception that is caught should be handled in its own catch clause.
Error Prone	BeanMembersShouldSerialize	If a class itself is bean or being referenced indirectly by a bean should be serializable.
Error Prone	CloneMethodMustImplementCloneable	Method clone() should only be defined if the class implements a Cloneable interface.
Error Prone	CloneReturnTypeMustMatchClassName	If the class implements a Cloneable interface then its return type of clone() method should match the classname.
Error Prone	CloneThrowsNotSupportedException	Clone() method should throw a CloneNotSupportedException.
Error Prone	CloseResource	Ensure that the resources of any type are closed after its usage.
Error Prone	ConstructorCallsOverridableMethod	Calling an overridable method during the construction phase may result in imposing a risk of invoking methods on an incompletely constructed object and can be difficult to debug.
Error Prone	EqualsNull	Use == operator instead of equals method to check for null.
Error Prone	MissingBreakInSwitch	Switch case without a break or return for any case could result in problematic behaviour.

Ruleset	Rule	Description
Error Prone	ProperCloneImplementation	Object clone() should be implemented with super.clone().
Error Prone	SuspiciousEqualsMethodName	The method name closely resembles equals(Object).
Multithreading	AvoidsynchronizedAtMethodLevel	Method level synchronization should be avoided to prevent the issues when adding new code to it.

Table 3.3: Ruleset Table for PMD

Rule	Description
AbstractClassName	Ensures that the names of abstract classes conforming to some regular expression and check that abstract modifier exists.
AvoidInlineConditionals	Detects inline conditionals.
ClassFanOutComplexity	Checks the number of other classes a given class relies on.
CovariantEquals	Checks that classes which define a covariant equals() method also override method equals(Object).
CyclomaticComplexity	Checks cyclomatic complexity against a specified limit.
DesignForExtension	Checks that classes are designed for extension.
EqualAvoidsNull	Checks that any combination of String literals is on the left side of an equals() comparison.
EqualsHashCode	Checks that classes that either override equals() or hashCode() also overrides the other.
FallThrough	Checks for a switch case that lacks break, return, throw or continue statement.
FinalParameters	Checks that parameters for methods, constructors, catch and for-each blocks are final.
MethodLength	Long methods and constructors.
MissingSwitchDefault	Checks for a default clause in the switch statement.
NeedBraces	Checks for braces around code blocks.
NPathComplexity	Checks the number of possible execution paths through a function against a specified limit.
Linelength	Checks for long lines.
SimplifyBooleanExpression	Checks for over-complicated boolean expressions.
SimplifyBooleanReturn	Checks for over-complicated Boolean return statements.

Table 3.4: Ruleset Table for CheckStyle

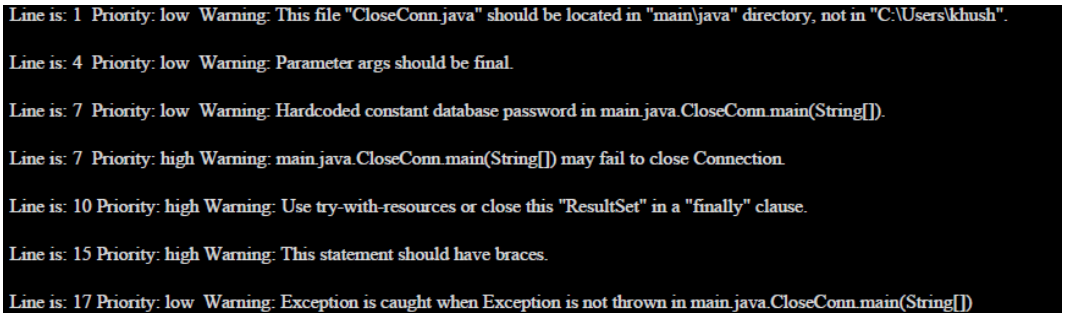

Figure 3.3: SWaP(Meta Tool) Output

the final report. Once the result merger has successfully integrated all of the reports, sort operation is performed on the warning set to arrange it in the order of line number. The final output is displayed in triplet form representing the line number, priority and detailed description of the warnings. The figure below shows the final output of the meta tool for the Sample Java code shown in Fig 3.3.

3.2.2 Worst-Case Analysis of Result Merger

Consider a scenario where each analyser reports n important warnings with no correlation. The important warnings reported by SpotBugs are directed to the SWaP report.Hence no comaprisons were made for merging SpotBugs warnings. Now, we have 'n' warnings in the merged report. The n number of important warnings reported by SonarScanner are compared against each warning in the merged report. The number of comparisons made for merging SonarScanner warnings are n^2 resulting in '2n' warnings in the SWaP report. Now, the n number of important warnings reported by PMD are compared against each warning in merged report.Therefore, the number of comparisons made for merging PMD warnings0 are $2n^2$. which results in '3n' warnings in the merged report. Finally, the n number of important warnings reported by CheckStyle are compared against each warning in the merged report. So, the number of comparisons made for merging CheckStyle

warnings are $3n^2$.

Worst Complexity for Result Merger for k analyzers $= 0 + n^2 + 2n^2 + 3n^2 ... + (k-1)n^2$

$$= n^2 \left(\frac{k(k-1)}{2}\right)$$

$$= \mathrm{O}(n^2k^2)$$

But in our thesis, we have combined the results of four static analysers i.e. k=4, therefore the worst-case complexity for Result Merger can be given as $\mathrm{O}(n^2)$.

Chapter 4

Experimental Results

We have conducted an experiment in order to compare and validate the performance of the meta tool against the existing static analysers. In this experiment, five Java developers have participated with a minimum of 2 years of industrial experience in java programming holding a bachelor degree in computer science and engineering. To carry out the preliminary study, each developer has been given five java programs(written by java developers other than the participants) along with five distinct analysis reports respectively (see Appendix). The analysis reports of five static analyzers namely SpotBugs, CheckStyle, PMD, SonarScanner and our meta tool have been renamed to Sam, Charlie, Paul, Sophie and Martha respectively in order to hide the actual identity of the tool from the users avoiding any kind of biasness.

The programmers were given a fixed time interval of 15 minutes to resolve the warnings detected in the analysis report of a program. The program lengths vary from 35-50 lines. The information about the warnings detected by multiple static analysers corresponding to each program is shown in the table 4.1.

The study is conducted in such a way that each programmer will receive one of the five renamed analysis report per program. This way, all the analysis reports were distributed

Analysis Tool	**Minimum number of warnings reported**	**Average number of warnings reported**	**Maximum number of warnings reported**
CheckStyle	21	47	104
PMD	1	5	8
SpotBugs	2	5	10
SonarScanner	9	13	18
SWaP	6	8	11

Table 4.1: Statistical information about warning counts of each static analyser

uniformly among the programmers. To compare the performance of our meta tool, the programmers were asked to tell about the number of warnings they were able to resolve per program in the given time frame. Based on the feedback collected from the developers, we have presented the data in the bar chart which gives a pictorial view of the evaluation of all the static analysis tools. The Fig. 4.1 displays the information about the number of warnings resolved in a program corresponding to each analysis report. We calculated the average and extremum counts of warnings resolved by the developers with respect to each report as represented in the table 4.2 .

Analysis Tool	**Minimum number of warnings resolved**	**Average number of warnings resolved**	**Maximum number of warnings resolved**
CheckStyle	1	3	5
PMD	0	3	6
SpotBugs	0	2	4
SonarScanner	1	3	6
SWaP	4	5	6

Table 4.2: Statistical information about resolved warning counts per static analyser

We can clearly state from the above statistical information that our meta tool (SWaP) helps the programmer to resolve maximum number of important warnings in a program with respect to other analysers.

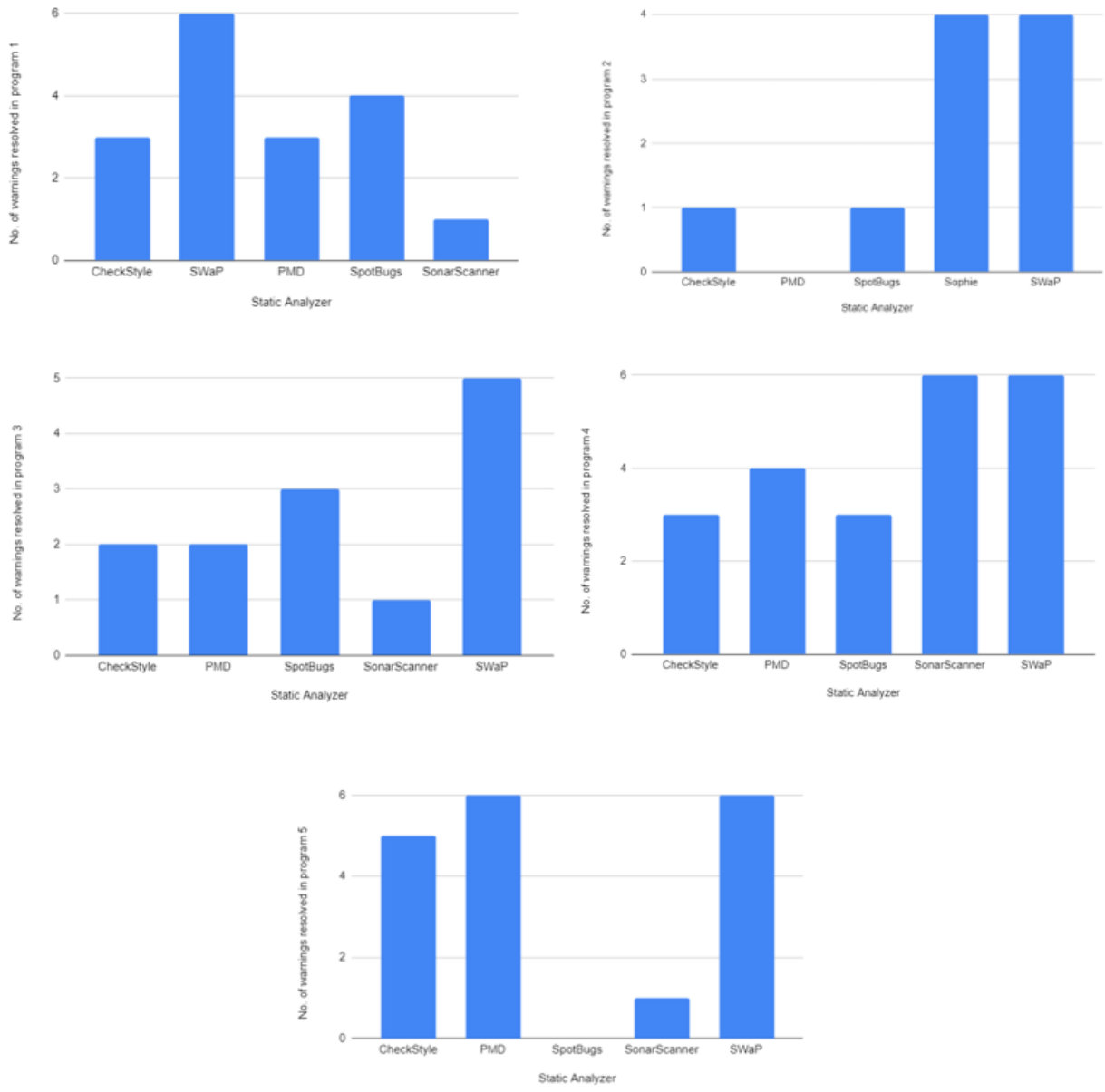

Figure 4.1: Warning counts Vs Static Analyser

Chapter 5

Conclusion and Future Work

In the thesis, we have presented an approach to integrate the warnings from the analysis reports of multiple static analysis tools namely SpotBugs, PMD, SonarScanner and CheckStyle in order to leverage the benefits of each of the static analysers as most of the times they report distinct warnings related to data flow analysis, syntactic pattern matching, symbolic execution etc. with a very little overlap. We have also proposed a mechanism to assign priorities to the warnings detected by the meta tool that facilitates the programmer to fix the important warnings first.

In the future, we would like to develop meta tools for other programming languages as well. Also, we would like to work on developing an easy to use GUI or plugin of the meta tool. As our research goes on, we would like to further improve the categories involving important and unimportant warnings to achieve a better accuracy for the actionable warnings. We would like to improve upon the prioritization technique by involving contextual information of the source code to decide which warning points out to immediate actionable alert along with introducing multiple levels of severity (range varying from 1 to 5).

Bibliography

[1] Checkstyle. https://checkstyle.sourceforge.io/cmdline.html.

[2] Pmd. https://pmd.github.io/latest/pmd_rules_java.html.

[3] Sonarqube. https://www.sonarsource.com/products/sonarqube/.

[4] Sonarscanner. https://docs.sonarqube.org/latest/analysis/scan/sonarscanner/.

[5] Spotbugs. https://spotbugs.readthedocs.io/en/stable/bugDescriptions.html.

[6] Simon Allier, Nicolas Anquetil, Andre Hora, and Stephane Ducasse. A framework to compare alert ranking algorithms. In *Proceedings of the 2012 19th Working Conference on Reverse Engineering*, WCRE '12, page 277–285, USA, 2012. IEEE Computer Society.

[7] J. E. M. Araújo, S. Souza, and M. T. Valente. Study on the relevance of the warnings reported by java bug-finding tools. *IET Software*, 5(4):366–374, 2011.

[8] Lori Flynn, William Snavely, David Svoboda, Nathan VanHoudnos, Richard Qin, Jennifer Burns, David Zubrow, Robert Stoddard, and Guillermo Marce-Santurio. Prioritizing alerts from multiple static analysis tools, using classification models. In

Proceedings of the 1st International Workshop on Software Qualities and Their Dependencies, SQUADE '18, page 13–20, New York, NY, USA, 2018. Association for Computing Machinery.

[9] Agata Gruza, Ramya Krishna Koricherla, and Clemente Izurieta. Evaluation of validation tools of java.

[10] S. Kim and M. D. Ernst. Prioritizing warning categories by analyzing software history. In *Fourth International Workshop on Mining Software Repositories (MSR'07:ICSE Workshops 2007)*, pages 27–27, 2007.

[11] Sunghun Kim and Michael D. Ernst. Which warnings should i fix first? 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*, ESEC-FSE '07, page 45–54, New York, NY, USA, 2007. Association for Computing Machinery.

[12] Guangtai Liang, Ling Wu, Qian Wu, Qianxiang Wang, Tao Xie, and Hong Mei. Automatic construction of an effective training set for prioritizing static analysis warnings. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, ASE '10, page 93–102, New York, NY, USA, 2010. Association for Computing Machinery.

[13] Md Abdullah Mamun, Aklima Khanam, Håkan Grahn, and Robert Feldt. Comparing four static analysis tools for java concurrency bugs. 09 2010.

[14] Foyzur Rahman, Sameer Khatri, Earl T. Barr, and Premkumar Devanbu. Comparing static bug finders and statistical prediction. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, page 424–434, New York, NY, USA, 2014. Association for Computing Machinery.

[15] Athos Ribeiro, Paulo Meirelles, Nelson Lago, and Fabio Kon. Ranking warnings from multiple source code static analyzers via ensemble learning. In *Proceedings of the 15th International Symposium on Open Collaboration*, OpenSym '19, New York, NY, USA, 2019. Association for Computing Machinery.

[16] N. Rutar, C. B. Almazan, and J. S. Foster. A comparison of bug finding tools for java. In *15th International Symposium on Software Reliability Engineering*, pages 245–256, 2004.

[17] Joseph R. Ruthruff, John Penix, J. David Morgenthaler, Sebastian Elbaum, and Gregg Rothermel. Predicting accurate and actionable static analysis warnings: An experimental approach. In *Proceedings of the 30th International Conference on Software Engineering*, ICSE '08, page 341–350, New York, NY, USA, 2008. Association for Computing Machinery.

Appendix A

Source Program

The five java programs along with their analysis reports used in the experiment are given as:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

//pmd -simplify conditional,local var final,method arg could be final
public class SimplifyBool {

    public static void main (String[] args){
        SimplifyBool bar=new SimplifyBool();
        if(bar!=null && bar instanceof SimplifyBool){
            System.out.println("bar belongs to class SimplifyBool");
        }
        int x=10;
        foo(x);

    }
   static void foo( int x){
    int y=x;
    y++;
    System.out.println(y);
       try{
           Class.forName("com.mysql.jdbc.Driver");
           Connection con=
DriverManager.getConnection("jdbc:mysql://localhost:3306/sonoo","root","root");//here sonoo is database
name, root is username and password
           Statement stmt=con.createStatement();
           ResultSet rs=stmt.executeQuery("select * from emp");
           while(rs.next())
               System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
           //        con.close();
       }catch(Exception e){ System.out.println(e);}
    }
}
```

Figure A.1: Source Program 1

```java
package main.java;
//PMD-cyclomatic complexity and clone,local var final
public class CycloComplexity implements Cloneable{

    public static void example(){
        int a=1;
        int  b=2;
        int c=3;
        int d=4;
        if(a!=b){
            if(a<b){
                System.out.println("ok");
            }
            else if(a==b){
                System.out.println("ok");
            }
            else{
                System.out.println("ok");
            }
        }
        else if(a==b-1){
            for(int i=0;i<c;i++){
                System.out.println("ok");
            }
        }
        else if(a==c+b){
            if(c==d){
                c++;
            }
            else{
                while(c+d!=2){
                    for(int j=3;j<c;j++){
                        System.out.println("ok");
                    }
                }
            }

        }
        else{
            if(b==d){
                c++;

            }
            else{
                while(c+d!=2){
                    for(int j=3;j<c;j++){
                        System.out.println("ok");
                    }
                }
            }
        }


    }
    public static void main(String[] args){
        CycloComplexity.example();
    }

}
```

Figure A.2: Source Program 2

```
package main.java;
import java.sql.*;
//PMD close connection
public class CloseConn {



       public static void main(String args[]){
           try{
               Class.forName("com.mysql.jdbc.Driver");
               Connection con=DriverManager.getConnection(
                       "jdbc:mysql://localhost:3306/sonoo","root","root");//here sonoo is database name,
root is username and password
               Statement stmt=con.createStatement();
               ResultSet rs=stmt.executeQuery("select * from emp");
               while(rs.next())
                   System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
        //       con.close();
           }
                   catch(Exception e)
                   {
                   System.out.println(e);
                   }
           }
   }
```

Figure A.3: Source Program 3

```
package main.java;

import java.util.ArrayList;
import java.util.List;

//PMD -missing break and equals method
public class NpEx {

    public static void main (String[] args){
        foo();
    int day = 4;
switch (day) {
        case 1:
            System.out.println("Monday");

        case 2:
            System.out.println("Tuesday");

        case 3:
            System.out.println("Wednesday");

        case 4:
            System.out.println("Thursday");
            break;
        case 5:
            System.out.println("Friday");
            break;
        case 6:
            System.out.println("Saturday");
            break;
        case 7:
            System.out.println("Sunday");
            break;
    }

    }
    List<String> elements = new ArrayList<String>();

    synchronized  public int count() {

        if(elements.size() == 0) {
            return 0;
        }
        else {
            elements.remove(2);
            return 1 + count();
        }
    }
   static void foo(){
        String x = "foo";
        if (x.equals(null)) // bad!
            System.out.println("Null String");

    }
}
```

Figure A.4: Source Program 4

```java
package main.java;

public abstract class Erroneous {
    double PI =3.78;
    public boolean equals(String S) {
        double d = Double.parseDouble(S);
        Object v=null;
        v=new Object();
        v=null;
        if (d == PI) {
            return true;
        } else {
            return false;
        }



    }
    void func(int n)
    {
        try {

            // this will throw ArithmeticException if n is 0
            int x = 10 / n;
            int y[] = new int[n];
            y[x] = 10;

            // this will throw ArrayIndexOutOfBoundsException
            // if the value of x surpasses
            // the highest index of this array
            System.out.println("No exception arose");
        }
        catch (Exception e) {
            if (e instanceof ArithmeticException)
                System.out.println("Can't divide by 0");
            if (e instanceof ArrayIndexOutOfBoundsException)
                System.out.println("This index doesn't exist in this array");
        }
    }
}
```

Figure A.5: Source Program 5

Appendix B

SpotBugs Report

The **SpotBugs** report generated for program 1 detects the following warnings:

- L D RCN: Redundant nullcheck of bar, which is known to be non-null in main.java.SimplifyBool.main(String[]) Redundant null check at SimplifyBool.java:[line 13].
- M S Dm: Hardcoded constant database password in main.java.SimplifyBool.foo(int) At SimplifyBool.java:[line 26]
- M B ODR: main.java.SimplifyBool.foo(int) may fail to close Connection At SimplifyBool.java:[line 26]
- M B ODR: main.java.SimplifyBool.foo(int) may fail to close Statement At SimplifyBool.java:[line 27]
- L C SIO: main.java.SimplifyBool.main(String[]) does an unnecessary type check using instanceof operator when it can be determined statically At SimplifyBool.java:[line 13]

- L B ISC: main.java.SimplifyBool.main(String[]) needlessly instantiates a class that only supplies static methods At SimplifyBool.java:[line 12]
- L D REC: Exception is caught when Exception is not thrown in main.java.SimplifyBool.foo(int) At SimplifyBool.java:[line 32]
- M X OBL: main.java.SimplifyBool.foo(int) may fail to clean up java.sql.Statement Obligation to clean up resource created at SimplifyBool.java:[line 27] is not discharged
- M X OBL: main.java.SimplifyBool.foo(int) may fail to clean up java.sql.ResultSet Obligation to clean up resource created at SimplifyBool.java:[line 28] is not discharged
- M D BC: instanceof will always return true for all non-null values in main.java.SimplifyBool.main(String[]), since all main.java.SimplifyBool are instances of main.java.SimplifyBool At SimplifyBool.java:[line 13]

The **SpotBugs** report generated for program 2 detects the following warnings:

- M B CN: Class main.java.CycloComplexity implements Cloneable but does not define or use clone method At CycloComplexity.java:[lines 3-57].
- L D DLS: Dead store to c in main.java.CycloComplexity.example() At CycloComplexity.java:[line 41].

The **SpotBugs** report generated for program 3 detects the following warnings:

- M S Dm: Hardcoded constant database password in main.java.CloseConn.main(String[]) At CloseConn.java:[line 11].

- M B ODR: main.java.CloseConn.main(String[]) may fail to close Connection At CloseConn.java:[line 11].
- M B ODR: main.java.CloseConn.main(String[]) may fail to close Statement At CloseConn.java:[line 14]
- L D REC: Exception is caught when Exception is not thrown in main.java.CloseConn.main(String[]) At CloseConn.java:[line 19]
- M X OBL: main.java.CloseConn.main(String[]) may fail to clean up java.sql.ResultSet Obligation to clean up resource created at CloseConn.java:[line 15] is not discharged.
- M X OBL: main.java.CloseConn.main(String[]) may fail to clean up java.sql.Statement Obligation to clean up resource created at CloseConn.java:[line 14] is not discharged.

The **SpotBugs** report generated for program 4 detects the following warnings:

- M D SF: Switch statement found in main.java.NpEx.main(String[]) where default case is missing At NpEx.java:[lines 12-32].
- M D SF: Switch statement found in main.java.NpEx.main(String[]) where one case falls through to the next case At NpEx.java:[lines 14-17].
- M C EC: Call to equals(null) in main.java.NpEx.foo() At NpEx.java:[line 51].

The **SpotBugs** report generated for program 5 detects the following warnings:

- M D UC: Useless object stored in variable y of method main.java.Erroneous.func(int) At Erroneous.java:[line 25].

- M D DLS: Dead store to v in main.java.Erroneous.equals(String) At Erroneous.java:[line 8].
- L D DLS: Dead store of null to v in main.java.Erroneous.equals(String) At Erroneous.java:[line 9].
- M D FE: Test for floating point equality in main.java.Erroneous.equals(String) At Erroneous.java:[line 10].

Appendix C

PMD Report

```
<?xml version="1.0" encoding="UTF-8"?>
<pmd xmlns="http://pmd.sourceforge.net/report/2.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://pmd.sourceforge.net/report/2.0.0 http://pmd.sourceforge.net/report_2_0_0.xsd"
    version="6.24.0" timestamp="2020-07-25T17:10:57.228">
<file name="C:\Users\khush\SimplifyBool.java">
<violation beginline="9" endline="34" begincolumn="27" endcolumn="1" rule="UseUtilityClass" ruleset="Design" package="main.java" class="SimplifyBool"
externalInfoUrl="https://pmd.github.io/pmd-6.24.0/pmd_rules_java_design.html#useutilityclass" priority="3">
All methods are static.  Consider using a utility class instead. Alternatively, you could add a private constructor or make the class abstract to silence this warning.
</violation>
<violation beginline="9" endline="34" begincolumn="8" endcolumn="1" rule="ClassNamingConventions" ruleset="Code Style" package="main.java" class="SimplifyBool"
externalInfoUrl="https://pmd.github.io/pmd-6.24.0/pmd_rules_java_codestyle.html#classnamingconventions" priority="1">
The utility class name 'SimplifyBool' doesn't match '[A-Z][a-zA-Z0-9]+(Utils?|Helper|Constants)'
</violation>
<violation beginline="13" endline="13" begincolumn="12" endcolumn="51" rule="SimplifyConditional" ruleset="Design" package="main.java" class="SimplifyBool" method="main"
externalInfoUrl="https://pmd.github.io/pmd-6.24.0/pmd_rules_java_design.html#simplifyconditional" priority="3">
No need to check for null before an instanceof
</violation>
<violation beginline="26" endline="26" begincolumn="23" endcolumn="25" rule="CloseResource" ruleset="Error Prone" package="main.java" class="SimplifyBool" method="foo"
variable="con" externalInfoUrl="https://pmd.github.io/pmd-6.24.0/pmd_rules_java_errorprone.html#closeresource" priority="3">
Ensure that resources like this Connection object are closed after use
</violation>
<violation beginline="27" endline="27" begincolumn="22" endcolumn="25" rule="CloseResource" ruleset="Error Prone" package="main.java" class="SimplifyBool" method="foo"
variable="stmt" externalInfoUrl="https://pmd.github.io/pmd-6.24.0/pmd_rules_java_errorprone.html#closeresource" priority="3">
Ensure that resources like this Statement object are closed after use
</violation>
<violation beginline="28" endline="28" begincolumn="22" endcolumn="23" rule="CloseResource" ruleset="Error Prone" package="main.java" class="SimplifyBool" method="foo"
variable="rs" externalInfoUrl="https://pmd.github.io/pmd-6.24.0/pmd_rules_java_errorprone.html#closeresource" priority="3">
Ensure that resources like this ResultSet object are closed after use
</violation>
<violation beginline="29" endline="30" begincolumn="12" endcolumn="90" rule="ControlStatementBraces" ruleset="Code Style" package="main.java" class="SimplifyBool"
method="foo" externalInfoUrl="https://pmd.github.io/pmd-6.24.0/pmd_rules_java_codestyle.html#controlstatementbraces" priority="3">
This statement should have braces
</violation>
</file>
</pmd>
```

Figure C.1: **PMD** report generated for program 1

```
<?xml version="1.0" encoding="UTF-8"?>
<pmd xmlns="http://pmd.sourceforge.net/report/2.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://pmd.sourceforge.net/report/2.0.0 http://pmd.sourceforge.net/report_2_0_0.xsd"
    version="6.24.0" timestamp="2020-07-26T02:40:52.529">
<file name="C:\Users\khush\CycloComplexity.java">
<violation beginline="3" endline="59" begincolumn="50" endcolumn="1" rule="UseUtilityClass" ruleset="Design" package="main.java"
class="CycloComplexity" externalInfoUrl="https://pmd.github.io/pmd-6.24.0/pmd_rules_java_design.html#useutilityclass" priority="3">
All methods are static.  Consider using a utility class instead. Alternatively, you could add a private constructor or make the cla
abstract to silence this warning.
</violation>
</file>
</pmd>
```

Figure C.2: **PMD** report generated for program 2

```
<?xml version="1.0" encoding="UTF-8"?>
<pmd xmlns="http://pmd.sourceforge.net/report/2.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://pmd.sourceforge.net/report/2.0.0 http://pmd.sourceforge.net/report_2_0_0.xsd"
    version="6.24.0" timestamp="2020-07-26T02:43:35.309">
<file name="C:\Users\khush\CloseConn.java">
<violation beginline="4" endline="21" begincolumn="24" endcolumn="5" rule="UseUtilityClass" ruleset="Design" package="main.java"
class="CloseConn" externalInfoUrl="https://pmd.github.io/pmd-6.24.0/pmd_rules_java_design.html#useutilityclass" priority="3">
All methods are static.  Consider using a utility class instead. Alternatively, you could add a private constructor or make the class
abstract to silence this warning.
</violation>
<violation beginline="11" endline="11" begincolumn="28" endcolumn="30" rule="CloseResource" ruleset="Error Prone" package="main.java"
class="CloseConn" method="main" variable="con" externalInfoUrl="https://pmd.github.io/pmd-
6.24.0/pmd_rules_java_errorprone.html#closeresource" priority="3">
Ensure that resources like this Connection object are closed after use
</violation>
<violation beginline="14" endline="14" begincolumn="27" endcolumn="30" rule="CloseResource" ruleset="Error Prone" package="main.java"
class="CloseConn" method="main" variable="stmt" externalInfoUrl="https://pmd.github.io/pmd-
6.24.0/pmd_rules_java_errorprone.html#closeresource" priority="3">
Ensure that resources like this Statement object are closed after use
</violation>
<violation beginline="15" endline="15" begincolumn="27" endcolumn="28" rule="CloseResource" ruleset="Error Prone" package="main.java"
class="CloseConn" method="main" variable="rs" externalInfoUrl="https://pmd.github.io/pmd-
6.24.0/pmd_rules_java_errorprone.html#closeresource" priority="3">
Ensure that resources like this ResultSet object are closed after use
</violation>
<violation beginline="16" endline="17" begincolumn="17" endcolumn="95" rule="ControlStatementBraces" ruleset="Code Style"
package="main.java" class="CloseConn" method="main" externalInfoUrl="https://pmd.github.io/pmd-
6.24.0/pmd_rules_java_codestyle.html#controlstatementbraces" priority="3">
This statement should have braces
</violation>
</file>
</pmd>
```

Figure C.3: **PMD** report generated for program 3

```
<?xml version="1.0" encoding="UTF-8"?>
<pmd xmlns="http://pmd.sourceforge.net/report/2.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://pmd.sourceforge.net/report/2.0.0 http://pmd.sourceforge.net/report_2_0_0.xsd"
    version="6.24.0" timestamp="2020-07-26T02:46:38.308">
<file name="C:\Users\khush\NpEx.java">
<violation beginline="12" endline="34" begincolumn="1" endcolumn="5" rule="MissingBreakInSwitch" ruleset="Error Prone"
package="main.java" class="NpEx" method="main" externalInfoUrl="https://pmd.github.io/pmd-
6.24.0/pmd_rules_java_errorprone.html#missingbreakinswitch" priority="3">
A switch statement does not contain a break
</violation>
<violation beginline="12" endline="34" begincolumn="1" endcolumn="5" rule="SwitchStmtsShouldHaveDefault" ruleset="Best Practices"
package="main.java" class="NpEx" method="main" externalInfoUrl="https://pmd.github.io/pmd-
6.24.0/pmd_rules_java_bestpractices.html#switchstmtsshouldhavedefault" priority="3">
Switch statements should have a default label
</violation>
<violation beginline="41" endline="41" begincolumn="12" endcolumn="24" rule="UseCollectionIsEmpty" ruleset="Best Practices"
package="main.java" class="NpEx" method="count" externalInfoUrl="https://pmd.github.io/pmd-
6.24.0/pmd_rules_java_bestpractices.html#usecollectionisempty" priority="3">
Substitute calls to size() == 0 (or size() != 0, size() &gt; 0, size() &lt; 1) with calls to isEmpty()
</violation>
<violation beginline="51" endline="51" begincolumn="13" endcolumn="26" rule="EqualsNull" ruleset="Error Prone" package="main.java"
class="NpEx" method="foo" externalInfoUrl="https://pmd.github.io/pmd-6.24.0/pmd_rules_java_errorprone.html#equalsnull" priority="1">
Avoid using equals() to compare against null
</violation>
<violation beginline="52" endline="52" begincolumn="13" endcolumn="46" rule="ControlStatementBraces" ruleset="Code Style"
package="main.java" class="NpEx" method="foo" externalInfoUrl="https://pmd.github.io/pmd-
6.24.0/pmd_rules_java_codestyle.html#controlstatementbraces" priority="3">
This statement should have braces
</violation>
</file>
</pmd>
```

Figure C.4: **PMD** report generated for program 4

```
<?xml version="1.0" encoding="UTF-8"?>
<pmd xmlns="http://pmd.sourceforge.net/report/2.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://pmd.sourceforge.net/report/2.0.0 http://pmd.sourceforge.net/report_2_0_0.xsd"
    version="6.24.0" timestamp="2020-07-26T02:48:58.659">
<file name="C:\Users\khush\Erroneous.java">
<violation beginline="5" endline="18" begincolumn="12" endcolumn="5" rule="SuspiciousEqualsMethodName" ruleset="Error Prone" package="main.java" class="Erroneous"
method="equals" externalInfoUrl="https://pmd.github.io/pmd-6.24.0/pmd_rules_java_errorprone.html#suspiciousequalsmethodname" priority="2">
The method name and parameter number are suspiciously close to equals(Object)
</violation>
<violation beginline="5" endline="5" begincolumn="34" endcolumn="34" rule="FormalParameterNamingConventions" ruleset="Code Style" package="main.java" class="Erroneous"
method="equals" variable="S" externalInfoUrl="https://pmd.github.io/pmd-6.24.0/pmd_rules_java_codestyle.html#formalparameternamingconventions" priority="1">
The method parameter name 'S' doesn't match '[a-z][a-zA-Z0-9]*'
</violation>
<violation beginline="7" endline="7" begincolumn="16" endcolumn="16" rule="UnusedLocalVariable" ruleset="Best Practices" package="main.java" class="Erroneous"
method="equals" variable="v" externalInfoUrl="https://pmd.github.io/pmd-6.24.0/pmd_rules_java_bestpractices.html#unusedlocalvariable" priority="3">
Avoid unused local variables such as 'v'.
</violation>
<violation beginline="10" endline="14" begincolumn="9" endcolumn="9" rule="SimplifyBooleanReturns" ruleset="Design" package="main.java" class="Erroneous" method="equals"
externalInfoUrl="https://pmd.github.io/pmd-6.24.0/pmd_rules_java_design.html#simplifybooleanreturns" priority="3">
Avoid unnecessary if..then..else statements when returning booleans
</violation>
<violation beginline="34" endline="34" begincolumn="17" endcolumn="17" rule="AvoidInstanceofChecksInCatchClause" ruleset="Error Prone" package="main.java"
class="Erroneous" method="func" externalInfoUrl="https://pmd.github.io/pmd-6.24.0/pmd_rules_java_errorprone.html#avoidinstanceofchecksincatchclause" priority="3">
An instanceof check is being performed on the caught exception.  Create a separate catch clause for this exception type.
</violation>
<violation beginline="35" endline="35" begincolumn="17" endcolumn="56" rule="ControlStatementBraces" ruleset="Code Style" package="main.java" class="Erroneous"
method="func" externalInfoUrl="https://pmd.github.io/pmd-6.24.0/pmd_rules_java_codestyle.html#controlstatementbraces" priority="3">
This statement should have braces
</violation>
<violation beginline="36" endline="36" begincolumn="17" endcolumn="17" rule="AvoidInstanceofChecksInCatchClause" ruleset="Error Prone" package="main.java"
class="Erroneous" method="func" externalInfoUrl="https://pmd.github.io/pmd-6.24.0/pmd_rules_java_errorprone.html#avoidinstanceofchecksincatchclause" priority="3">
An instanceof check is being performed on the caught exception.  Create a separate catch clause for this exception type.
</violation>
<violation beginline="37" endline="37" begincolumn="17" endcolumn="77" rule="ControlStatementBraces" ruleset="Code Style" package="main.java" class="Erroneous"
method="func" externalInfoUrl="https://pmd.github.io/pmd-6.24.0/pmd_rules_java_codestyle.html#controlstatementbraces" priority="3">
This statement should have braces
</violation>
</file>
</pmd>
```

Figure C.5: **PMD** report generated for program 5

Appendix D

CheckStyle Report

```
Starting audit...
[ERROR] C:\Users\khush\SimplifyBool.java:1: Missing package-info.java file. [JavadocPackage]
[ERROR] C:\Users\khush\SimplifyBool.java:9:1: Utility classes should not have a public or default constructor. [HideUtilityClassConstructor]
[ERROR] C:\Users\khush\SimplifyBool.java:11:5: Missing a Javadoc comment. [MissingJavadocMethod]
[ERROR] C:\Users\khush\SimplifyBool.java:11:29: '(' is preceded with whitespace. [MethodParamPad]
[ERROR] C:\Users\khush\SimplifyBool.java:11:30: Parameter args should be final. [FinalParameters]
[ERROR] C:\Users\khush\SimplifyBool.java:11:44: '{' is not preceded with whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\SimplifyBool.java:12:25: '=' is not followed by whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\SimplifyBool.java:12:25: '=' is not preceded with whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\SimplifyBool.java:13:9: 'if' is not followed by whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\SimplifyBool.java:13:15: '!=' is not followed by whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\SimplifyBool.java:13:15: '!=' is not preceded with whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\SimplifyBool.java:13:53: '{' is not preceded with whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\SimplifyBool.java:16:14: '=' is not followed by whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\SimplifyBool.java:16:14: '=' is not preceded with whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\SimplifyBool.java:16:15: '10' is a magic number. [MagicNumber]
[ERROR] C:\Users\khush\SimplifyBool.java:20:19: '(' is followed by whitespace. [ParenPad]
[ERROR] C:\Users\khush\SimplifyBool.java:20:21: Parameter x should be final. [FinalParameters]
[ERROR] C:\Users\khush\SimplifyBool.java:20:27: '{' is not preceded with whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\SimplifyBool.java:21:10: '=' is not followed by whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\SimplifyBool.java:21:10: '=' is not preceded with whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\SimplifyBool.java:24:8: 'try' is not followed by whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\SimplifyBool.java:24:11: '{' is not preceded with whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\SimplifyBool.java:26: Line is longer than 80 characters (found 166). [LineLength]
[ERROR] C:\Users\khush\SimplifyBool.java:26:26: '=' is not preceded with whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\SimplifyBool.java:26:91: ',' is not followed by whitespace. [WhitespaceAfter]
[ERROR] C:\Users\khush\SimplifyBool.java:26:98: ',' is not followed by whitespace. [WhitespaceAfter]
[ERROR] C:\Users\khush\SimplifyBool.java:26:106: ';' is not followed by whitespace. [WhitespaceAfter]
[ERROR] C:\Users\khush\SimplifyBool.java:27:26: '=' is not followed by whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\SimplifyBool.java:27:26: '=' is not preceded with whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\SimplifyBool.java:28:24: '=' is not followed by whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\SimplifyBool.java:28:24: '=' is not preceded with whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\SimplifyBool.java:29:12: 'while' construct must use '{}'s. [NeedBraces]
[ERROR] C:\Users\khush\SimplifyBool.java:29:12: 'while' is not followed by whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\SimplifyBool.java:30: Line is longer than 80 characters (found 90). [LineLength]
[ERROR] C:\Users\khush\SimplifyBool.java:30:47: '+' is not followed by whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\SimplifyBool.java:30:47: '+' is not preceded with whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\SimplifyBool.java:30:52: '+' is not followed by whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\SimplifyBool.java:30:52: '+' is not preceded with whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\SimplifyBool.java:30:68: '+' is not followed by whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\SimplifyBool.java:30:68: '+' is not preceded with whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\SimplifyBool.java:30:73: '+' is not followed by whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\SimplifyBool.java:30:73: '+' is not preceded with whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\SimplifyBool.java:30:87: '3' is a magic number. [MagicNumber]
[ERROR] C:\Users\khush\SimplifyBool.java:32:8: '}' is not followed by whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\SimplifyBool.java:32:9: 'catch' is not followed by whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\SimplifyBool.java:32:9: 'catch' is not preceded with whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\SimplifyBool.java:32:27: '{' at column 27 should have line break after. [LeftCurly]
[ERROR] C:\Users\khush\SimplifyBool.java:32:27: '{' is not preceded with whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\SimplifyBool.java:32:50: ';' is not followed by whitespace. [WhitespaceAfter]
[ERROR] C:\Users\khush\SimplifyBool.java:32:51: '}' is not preceded with whitespace. [WhitespaceAround]
Audit done.
```

Figure D.1: **CheckStyle** report for program 1

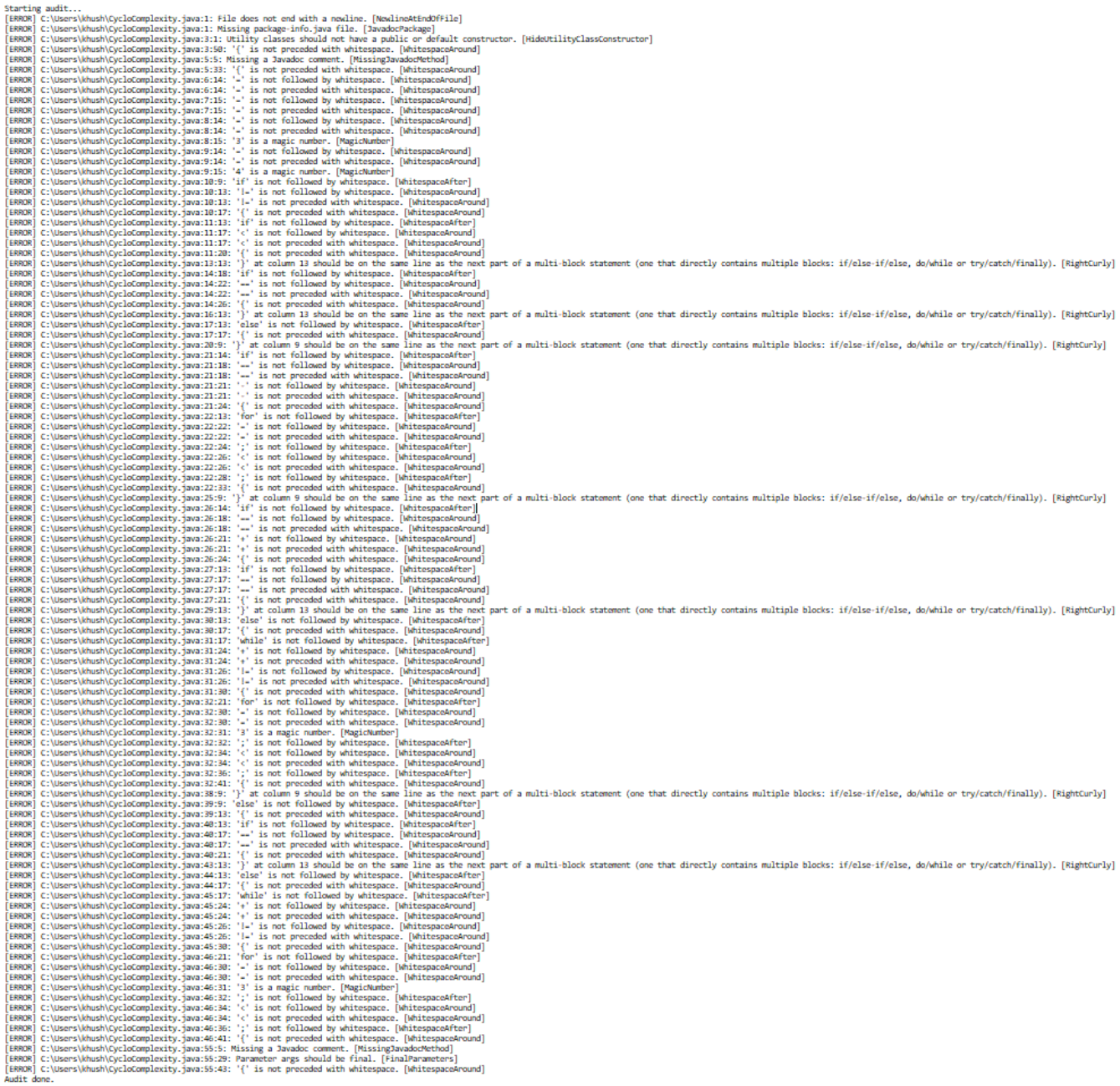

Figure D.2: **CheckStyle** report for program 2

```
Starting audit...
[ERROR] C:\Users\khush\CloseConn.java:1: Missing package-info.java file. [JavadocPackage]
[ERROR] C:\Users\khush\CloseConn.java:2:16: Using the '.*' form of import should be avoided - java.sql.*. [AvoidStarImport]
[ERROR] C:\Users\khush\CloseConn.java:4:1: Utility classes should not have a public or default constructor. [HideUtilityClassConstructor]
[ERROR] C:\Users\khush\CloseConn.java:8:9: Missing a Javadoc comment. [MissingJavadocMethod]
[ERROR] C:\Users\khush\CloseConn.java:8:33: Parameter args should be final. [FinalParameters]
[ERROR] C:\Users\khush\CloseConn.java:8:44: Array brackets at illegal position. [ArrayTypeStyle]
[ERROR] C:\Users\khush\CloseConn.java:8:47: '{' is not preceded with whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\CloseConn.java:9:13: 'try' is not followed by whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\CloseConn.java:9:16: '{' is not preceded with whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\CloseConn.java:11:31: '=' is not followed by whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\CloseConn.java:11:31: '=' is not preceded with whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\CloseConn.java:12:60: ',' is not followed by whitespace. [WhitespaceAfter]
[ERROR] C:\Users\khush\CloseConn.java:12:67: ',' is not followed by whitespace. [WhitespaceAfter]
[ERROR] C:\Users\khush\CloseConn.java:14:31: '=' is not followed by whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\CloseConn.java:14:31: '=' is not preceded with whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\CloseConn.java:15:29: '=' is not followed by whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\CloseConn.java:15:29: '=' is not preceded with whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\CloseConn.java:16:17: 'while' construct must use '{}'s. [NeedBraces]
[ERROR] C:\Users\khush\CloseConn.java:16:17: 'while' is not followed by whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\CloseConn.java:17: Line is longer than 80 characters (found 95). [LineLength]
[ERROR] C:\Users\khush\CloseConn.java:17:52: '+' is not followed by whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\CloseConn.java:17:52: '+' is not preceded with whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\CloseConn.java:17:57: '+' is not followed by whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\CloseConn.java:17:57: '+' is not preceded with whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\CloseConn.java:17:73: '+' is not followed by whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\CloseConn.java:17:73: '+' is not preceded with whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\CloseConn.java:17:78: '+' is not followed by whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\CloseConn.java:17:78: '+' is not preceded with whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\CloseConn.java:17:92: '3' is a magic number. [MagicNumber]
[ERROR] C:\Users\khush\CloseConn.java:19:13: '}' is not followed by whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\CloseConn.java:19:14: 'catch' is not followed by whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\CloseConn.java:19:14: 'catch' is not preceded with whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\CloseConn.java:19:32: '{' at column 32 should have line break after. [LeftCurly]
[ERROR] C:\Users\khush\CloseConn.java:19:32: '{' is not preceded with whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\CloseConn.java:19:55: ';' is not followed by whitespace. [WhitespaceAfter]
[ERROR] C:\Users\khush\CloseConn.java:19:56: '}' is not preceded with whitespace. [WhitespaceAround]
Audit done.
```

Figure D.3: **CheckStyle** report for program 3

```
Starting audit...
[ERROR] C:\Users\khush\NpEx.java:1: Missing package-info.java file. [JavadocPackage]
[ERROR] C:\Users\khush\NpEx.java:9:5: Missing a Javadoc comment. [MissingJavadocMethod]
[ERROR] C:\Users\khush\NpEx.java:9:29: '(' is preceded with whitespace. [MethodParamPad]
[ERROR] C:\Users\khush\NpEx.java:9:30: Parameter args should be final. [FinalParameters]
[ERROR] C:\Users\khush\NpEx.java:9:44: '{' is not preceded with whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\NpEx.java:11:15: '4' is a magic number. [MagicNumber]
[ERROR] C:\Users\khush\NpEx.java:12:1: switch without "default" clause. [MissingSwitchDefault]
[ERROR] C:\Users\khush\NpEx.java:19:14: '3' is a magic number. [MagicNumber]
[ERROR] C:\Users\khush\NpEx.java:22:14: '4' is a magic number. [MagicNumber]
[ERROR] C:\Users\khush\NpEx.java:25:14: '5' is a magic number. [MagicNumber]
[ERROR] C:\Users\khush\NpEx.java:28:14: '6' is a magic number. [MagicNumber]
[ERROR] C:\Users\khush\NpEx.java:31:14: '7' is a magic number. [MagicNumber]
[ERROR] C:\Users\khush\NpEx.java:37:5: Missing a Javadoc comment. [JavadocVariable]
[ERROR] C:\Users\khush\NpEx.java:37:18: Variable 'elements' must be private and have accessor methods. [VisibilityModifier]
[ERROR] C:\Users\khush\NpEx.java:39:5: Class 'NpEx' looks like designed for extension (can be subclassed), but the method 'count' does
not have javadoc that explains how to do that safely. If class is not designed for extension consider making the class 'NpEx' final or
making the method 'count' static/final/abstract/empty, or adding allowed annotation for the method. [DesignForExtension]
[ERROR] C:\Users\khush\NpEx.java:39:5: Missing a Javadoc comment. [MissingJavadocMethod]
[ERROR] C:\Users\khush\NpEx.java:39:19: 'public' modifier out of order with the JLS suggestions. [ModifierOrder]
[ERROR] C:\Users\khush\NpEx.java:41:9: 'if' is not followed by whitespace. [WhitespaceAfter]
[ERROR] C:\Users\khush\NpEx.java:43:9: '}' at column 9 should be on the same line as the next part of a multi-block statement (one
that directly contains multiple blocks: if/else-if/else, do/while or try/catch/finally). [RightCurly]
[ERROR] C:\Users\khush\NpEx.java:49:21: '{' is not preceded with whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\NpEx.java:51:9: 'if' construct must use '{}'s. [NeedBraces]
Audit done.
```

Figure D.4: **CheckStyle** report for program 4

```
Starting audit...
[ERROR] C:\Users\khush\Erroneous.java:1: Missing package-info.java file. [JavadocPackage]
[ERROR] C:\Users\khush\Erroneous.java:4:5: Missing a Javadoc comment. [JavadocVariable]
[ERROR] C:\Users\khush\Erroneous.java:4:12: Name 'PI' must match pattern '^[a-z][a-zA-Z0-9]*$'. [MemberName]
[ERROR] C:\Users\khush\Erroneous.java:4:12: Variable 'PI' must be private and have accessor methods. [VisibilityModifier]
[ERROR] C:\Users\khush\Erroneous.java:4:15: '=' is not followed by whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\Erroneous.java:4:16: '3.78' is a magic number. [MagicNumber]
[ERROR] C:\Users\khush\Erroneous.java:5:5: Class 'Erroneous' looks like designed for extension (can be subclassed), but the method 'equals' does
not have javadoc that explains how to do that safely. If class is not designed for extension consider making the class 'Erroneous' final or
making the method 'equals' static/final/abstract/empty, or adding allowed annotation for the method. [DesignForExtension]
[ERROR] C:\Users\khush\Erroneous.java:5:5: Missing a Javadoc comment. [MissingJavadocMethod]
[ERROR] C:\Users\khush\Erroneous.java:5:27: Parameter S should be final. [FinalParameters]
[ERROR] C:\Users\khush\Erroneous.java:5:34: Name 'S' must match pattern '^[a-z][a-zA-Z0-9]*$'. [ParameterName]
[ERROR] C:\Users\khush\Erroneous.java:7:17: '=' is not followed by whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\Erroneous.java:7:17: '=' is not preceded with whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\Erroneous.java:8:10: '=' is not followed by whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\Erroneous.java:8:10: '=' is not preceded with whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\Erroneous.java:9:10: '=' is not followed by whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\Erroneous.java:9:10: '=' is not preceded with whitespace. [WhitespaceAround]
[ERROR] C:\Users\khush\Erroneous.java:10:9: Conditional logic can be removed. [SimplifyBooleanReturn]
[ERROR] C:\Users\khush\Erroneous.java:19:5: Class 'Erroneous' looks like designed for extension (can be subclassed), but the method 'func' does
not have javadoc that explains how to do that safely. If class is not designed for extension consider making the class 'Erroneous' final or
making the method 'func' static/final/abstract/empty, or adding allowed annotation for the method. [DesignForExtension]
[ERROR] C:\Users\khush\Erroneous.java:19:15: Parameter n should be final. [FinalParameters]
[ERROR] C:\Users\khush\Erroneous.java:20:5: '{' at column 5 should be on the previous line. [LeftCurly]
[ERROR] C:\Users\khush\Erroneous.java:24:21: '10' is a magic number. [MagicNumber]
[ERROR] C:\Users\khush\Erroneous.java:25:18: Array brackets at illegal position. [ArrayTypeStyle]
[ERROR] C:\Users\khush\Erroneous.java:26:20: '10' is a magic number. [MagicNumber]
[ERROR] C:\Users\khush\Erroneous.java:32:9: '}' at column 9 should be on the same line as the next part of a multi-block statement (one that
directly contains multiple blocks: if/else-if/else, do/while or try/catch/finally). [RightCurly]
[ERROR] C:\Users\khush\Erroneous.java:34:13: 'if' construct must use '{}'s. [NeedBraces]
[ERROR] C:\Users\khush\Erroneous.java:36:13: 'if' construct must use '{}'s. [NeedBraces]
[ERROR] C:\Users\khush\Erroneous.java:41: Line is longer than 80 characters (found 116). [LineLength]
Audit done.
```

Figure D.5: **CheckStyle** report for program 5

Appendix E

SWaP Report

The **SWaP** report generated for program 1 detects the following warnings:

- Line is: 1 Priority: low Warning: This file "SimplifyBool.java" should be located in "main\java" directory, not in "C:\Users\khush".
- Line is: 11 Priority: low Warning: Parameter args should be final.
- Line is: 13 Priority: high Warning: instanceof will always return true for all non-null values in main.java.SimplifyBool.main(String[]), since all main.java.SimplifyBool are instances of main.java.SimplifyBool.
- Line is: 20 Priority: low Warning: Parameter x should be final.
- Line is: 26 Priority: low Warning: Hardcoded constant database password in main.java.SimplifyBool.foo(int).
- Line is: 26 Priority: high Warning: main.java.SimplifyBool.foo(int) may fail to close Connection.

- Line is: 27 Priority: high Warning: main.java.SimplifyBool.foo(int) may fail to close Statement.
- Line is: 28 Priority: high Warning: Use try-with-resources or close this "ResultSet" in a "finally" clause.
- Line is: 29 Priority: high Warning: This statement should have braces.
- Line is: 30 Priority: low Warning: Line is longer than 80 characters (found 90).
- Line is: 32 Priority: low Warning: Exception is caught when Exception is not thrown in main.java.SimplifyBool.foo(int).

The **SWaP** report generated for program 2 detects the following warnings:

- Line is: 1 Priority: low Warning: This file "CycloComplexity.java" should be located in "main\java" directory, not in "C:\Users\khush".
- Line is: 3 Priority: high Warning: Class main.java.CycloComplexity implements Cloneable but does not define or use clone method.
- Line is: 5 Priority: low Warning: Refactor this method to reduce its Cognitive Complexity from 30 to the 15 allowed.
- Line is: 11 Priority: low Warning: Remove this conditional structure or edit its code blocks so that they're not all the same.
- Line is: 14 Priority: low Warning: Change this condition so that it does not always evaluate to "false".
- Line is: 55 Priority: low Warning: Parameter args should be final.

The **SWaP** report generated for program 3 detects the following warnings:

- Line is: 1 Priority: low Warning: This file "CloseConn.java" should be located in "main\java" directory, not in "C:\Users\khush".
- Line is: 8 Priority: low Warning: Parameter args should be final.
- Line is: 11 Priority: low Warning: Hardcoded constant database password in main.java.CloseConn.main(String[]).
- Line is: 11 Priority: high Warning: main.java.CloseConn.main(String[]) may fail to close Connection.
- Line is: 15 Priority: high Warning: Use try-with-resources or close this "ResultSet" in a "finally" clause.
- Line is: 16 Priority: high Warning: This statement should have braces.
- Line is: 17 Priority: low Warning: Line is longer than 80 characters (found 95).
- Line is: 19 Priority: low Warning: Exception is caught when Exception is not thrown in main.java.CloseConn.main(String[]).

The **SWaP** report generated for program 4 detects the following warnings:

- Line is: 1 Priority: low Warning: This file "NpEx.java" should be located in "main\java" directory, not in "C:\Users\khush".
- Line is: 9 Priority: low Warning: Parameter args should be final.
- Line is: 12 Priority: high Warning: Add a default case to this switch.
- Line is: 13 Priority: high Warning: End this switch case with an unconditional break, return or throw statement.

- Line is: 16 Priority: high Warning: End this switch case with an unconditional break, return or throw statement.
- Line is: 19 Priority: high Warning: End this switch case with an unconditional break, return or throw statement.
- Line is: 39 Priority: low Warning: Class 'NpEx' looks like designed for extension (can be subclassed), but the method 'count' does not have javadoc that explains how to do that safely. If class is not designed for extension consider making the class 'NpEx' final or making the method 'count' static/final/abstract/empty, or adding allowed annotation for the method.
- Line is: 51 Priority: high Warning: Call to equals(null) in main.java.NpEx.foo()
- Line is: 51 Priority: highWarning: 'if' construct must use ''s.

The **SWaP** report generated for program 5 detects the following warnings:

- Line is: 1 Priority: low Warning: This file "Erroneous.java" should be located in "main\java" directory, not in "C:\Users\khush".
- Line is: 5 Priority: low Warning: Either override Object.equals(Object), or rename the method to prevent any confusion.
- Line is: 10 Priority: high Warning: Avoid unnecessary if..then..else statements when returning booleans
- Line is: 19 Priority: low Warning: Class 'Erroneous' looks like designed for extension (can be subclassed), but the method 'func' does not have javadoc that explains how to do that safely. If class is not designed for extension consider making the class 'Erroneous' final or making the method 'func' static/final/abstract/empty, or adding allowed annotation for the method.

- Line is: 34 Priority: high Warning: An instanceof check is being performed on the caught exception. Create a separate catch clause for this exception type.
- Line is: 34 Priority: high Warning: 'if' construct must use ''s.
- Line is: 36 Priority: high Warning: An instanceof check is being performed on the caught exception. Create a separate catch clause for this exception type.
- Line is: 36 Priority: high Warning: 'if' construct must use ''s.
- Line is: 41 Priority: low Warning: Line is longer than 80 characters (found 116).

Appendix F

SonarScanner Report

Figure F.1: **SonarScanner** report generated for program 1

Figure F.2: **SonarScanner** report generated for program 2

Figure F.3: **SonarScanner** report generated for program 3

Figure F.4: **SonarScanner** report generated for program 4

Figure F.5: **SonarScanner** report generated for program 5