An Anti-Plagiarism Add-on for Web-CAT

Lawan Thamsuhang Subba

Dissertation 2013
Erasmus Mundus MSc in Dependable Software Systems

NUI MAYNOOTH
Ollscoil na hÉireann Má Nuad

Department of Computer Science
National University of Ireland, Maynooth
Co.Kildare, Ireland

A dissertation submitted in partial fulfillment
of the requirements for the
Erasmus Mundus Msc Dependable Software Systems

Head of Department: Dr Adam Winstangley
Supervisors: Dr Aidan Mooney and Hugh Maher
July 2014

European Commission
ERASMUS
MUNDUS

Declaration

I hereby certify that this material, which I now submit for assessment of the program of study leading to the award of Master of Science in Dependable Software Systems, is entirely my own work and has not been taken from the work of other save and to the extent that such work has been cited and acknowledged within the text of my work.

Lawan Subba

____

Acknowledgment

I want to thank to Dr. Aidan Mooney for all his support, help, patience and contributions made to this project. I extend this thank to all the professors that have shared their knowledge with myself throughout the different lectures I have had at the National University of Ireland, and to my classmates whose support has been immensely helpful for the culmination of this research.

Abstract

Plagiarism is a major problem in every discipline and Computer Science courses are no different. It is very common for students to submit their peers programming assignment as their own. This practice is unfair and also halts the learning process of the students who choose to copy.

This research investigates the performance of various plagiarism detection tools like MOSS, JPlag and Plaggie2 . Controlled changes were made to a code file, and the sensitiveness of the various tools to those changes were determined. Plaggie2 with its algorithm of tokenization followed by string comparison was found to have acceptable performance for our tests. It is also open source whereas the other tools are proprietary and web based. Therefore Plaggie2 was incorporated with Web-CAT.

Web-CAT is a flexible, automated grading system designed to process computer programming assignments. It serves as a learning environment for software testing tasks and helps automatically assess student assignments. The developed system was tested by a variety of potential future users in a number of tests and the feedback received was very positive. In addition the tests presented us with a number of possible future enhancements.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter we will discuss plagiarism in academic institutions, what constitutes as plagiarism and its consequences. Next we discuss the problems and consequences of plagiarism in software context. Then we use a survey to determine how academics view and detect software plagiarism at NUIM. Next we then discuss and argue the benefits of test driven development. Finally we discuss the automated assignment grading system called Web-CAT and how adding a plagiarism detection functionality to it would be provide benefit to instructors.

1.1 Plagiarism

The Oxford English dictionary [Pla14] defines plagiarism as "The action or practice of taking someone else's work, idea, etc., and passing it off as one's own; literary theft". Therefore we can say that plagiarism has occurred when someone uses other peoples work without modification, or with subtle modifications without acknowledging the original author properly. In an academic environment, students are expected to put substantial effort into understanding the course work. Academics are able to judge students by evaluating the assignments given as part of the course. If students plagiarize they cannot be evaluated correctly as this evaluation will form a wrong impression of their understanding. Also the original author will not be given credit where it is due. Therefore it is considered a serious offense by all academics.

According to Joy *et al.* [JCYS11] students plagiarize for a number of reasons such as: inadequate time management, workload, laziness, not understanding what constitutes plagiarism, fear of failure, high achievement expectations, cryptomnesia, thrill of breaking the rules and risk of getting caught, work ethics, competitive achievement, low self-esteem, time pressure, and a desire to increase.

However, it is important to note that not all cases of plagiarism are intentional. For example, students may not be familiar with the methods of formal writing. Plagiarism may also occur as students are unfamiliar with the correct way to cite sources, or a policy on what constitutes as plagiarism has not been made clear by the department. Marshall *et al.* [MG05] concludes that "education programmes need to decode the more formal definitions of plagiarism into specific examples that illustrate the range of activities that are not permitted and how misconduct can be avoided".

In academic environment where the students submit essays, tools such as Turnitin [1] can help detect instances of plagiarism. The tool is widely used across various institutions including the National University of Ireland Maynooth (NUIM). All institutions have their own guidelines for dealing with instances of plagiarism, but the action will depend upon factors like students reluctance to admit the offense, the extent to which the student has gone to disguise his actions and the university policy. NUIM has its own definition and guidelines [NUI] to identify and tackle plagiarism.

1.2 Software Plagiarism

In Computer Science, like in other subjects as well, it is integral that students not only understand but are also able to apply what they have learned in a practical setting. Therefore, computer science courses expect students to submit code as part of their laboratory works and assignments. Besides the final examination, these submissions may account for some of the overall marks of the course. Students may be allowed to discuss with each other as discussions are an integral part of the learning process. However in the study by Cosma *et al* [CJ06] all tutors agreed on a "zero tolerance" policy, and many tutors even commented that all assignments should be investigated for plagiarism. In addition they also agreed that the guilty students should be penalised or warned even if the assignment was an unmarked one and had no contribution to the overall marks.

Culwin *et al.* [CML01] conducted a study on source code plagiarism in 55 UK higher education computing schools. They discovered that 20 out of 55 institutes believe that 20% to more than 50 % of students in an initial programming course are involved in plagiarism. Also, 13 of the 55 institutes regard the problem as bad and getting worse. Researchers at Dublin City University used a novel approach[DH05] to find who was the plagiarizer and who was the supplier. In their study they found more than 50% of students were involved in plagiarism.

Plagiarism itself is a vague term as, what constitutes as plagiarism varies in the views from person to person. What one academic might consider as plagiarism might not hold true for another. Cosma *et al* [CJ06] conducted a survey on 120 academics on what they considered as plagiarism. The majority of participants in that survey agreed with the following interpretation for Plagiarism.

1. Plagiarism of source code can occur in programming assignments if a student re-used source code written by somebody else. The student may have obtained the source code from the original author with or without permission. Also the student may have borrowed and submitted the source code as his/her own work either intentionally or unintentionally by not properly acknowledging the original author.

2. If a student reuses source code that has been submitted previously as part of another assessment for which the student has received academic credit. Then if s/he has failed to acknowledge this fact then, it can be considered self-plagiarism or another academic offense depending upon the regulations of the academic institution.

[1]http://turnitin.com/

3. If a student reuses source code that has been produced by someone else or one that has been produced by the same student in response to another assignment. Then it will not be considered plagiarism if the student provides proper acknowledgment but a breach of assignment regulations.

In a software context generally there are large number of students and files involved. Therefore it is very difficult for tutors to check for plagiarism. For example, at NUIM there are more than 400 students in first year Computer Science courses. For software submissions there are many automated approaches [HRvV11] that assist with detection of Plagiarism in the software context. Each one has varying levels of success, features and its associated problems. Some of the most peer reviewed ones are MOSS [A^+05] and JPLag [ASR06].

In a study by Charlie Daly *et al* [DH05] they found that the students who did not copy did better in end-semester examinations that students who choose to copy. It can be reasoned that the students who copy may not be understanding the course content properly. Therefore, it is important to identify and stop instances of plagiarism.

1.3 Software Plagiarism perspectives in NUIM

We wanted to know what kinds of programming languages were being used in NUIM computer courses, how the lecturer's were detecting plagiarism and the perspectives of the lecturers on software plagiarism. This information would provide crucial information for later design and implementation decisions. In order to find the answers we conducted an online survey on SurveyMonkey [2]. From the 16 lecturers available in the Computer Science department, 10 choose to take part in our survey. From the 10 that choose to take part, some have chosen to skip some of the questions. The details of the survey are discussed next.

Question 1: What programming languages are used by students to submit assignments in your course?

The answers to this question which is shown in Figure 1.1 show that the majority of the assignments submitted by students are in the Java programming language. This number supports the widespread usage and popularity of Java in student courses. Other programming languages like C# and Python are also being used in some of the courses. Besides the choices provided some of the faculty stated in the others section that, languages like MATLAB, Octave, Haskell, Scheme and Prolog are also being used to submit assignments. Also some of the faculty have also mentioned in the others section that there are some assignments that require no programming like HTML and UML.

Question 2: Do you check all submitted source code for plagiarism?
For this question, 4 people from the faculty answered that they do check every submitted file for instances of plagiarism, whereas 3 people from the faculty answered no. The number of students involved in some of the courses may make it difficult for the faculty

[2]https://www.surveymonkey.com/home/

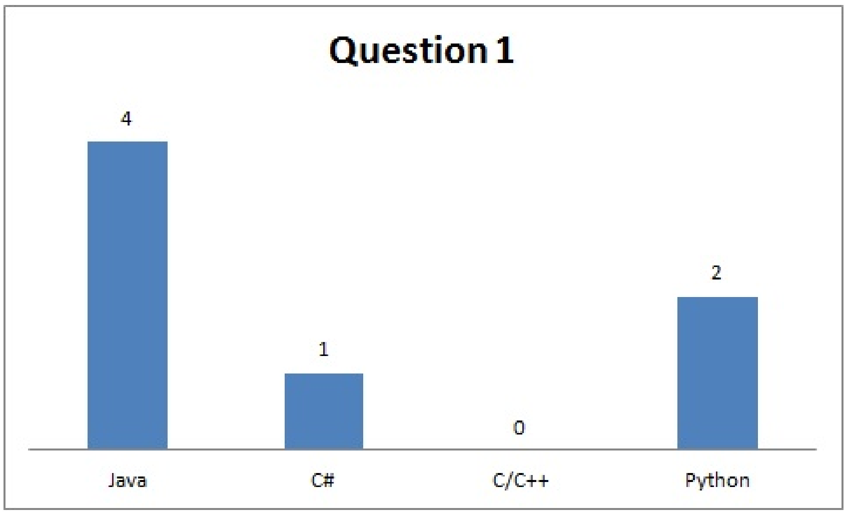


Figure 1.1: Response to Question 1

to check every file for plagiarism, when no automated system is in place.

Question 3: What system do you use for detecting plagiarism?

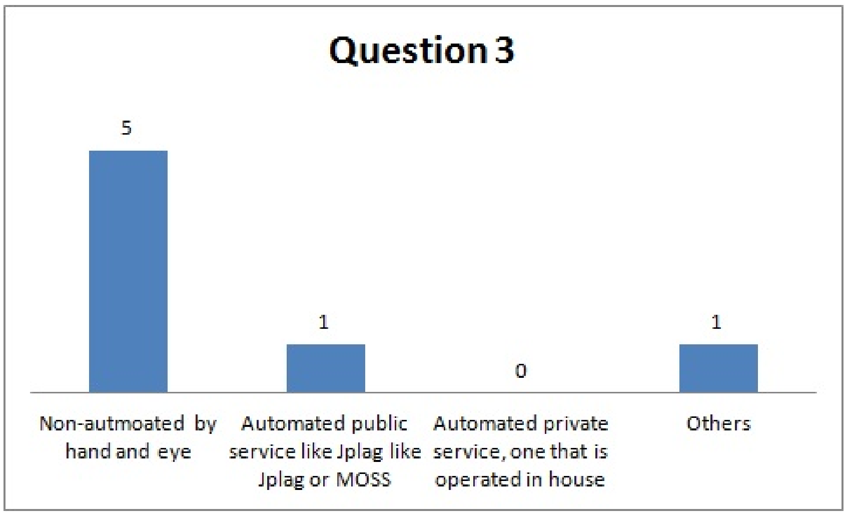


Figure 1.2: Response to Question 3

For this question, from the answers shown in Figure 1.2 we see that the majority of the faculty use a primitive non-automated hand and eye approach at detecting plagiarism. However a small minority of the faculty do use automated approaches like JPlag and MOSS.

Question 4: Do you think your approach at detecting plagiarism is effective?
For this question the answers are evenly divided. 8 members of the faculty choose to answer this question. Half of them are satisfied with the approach they are using at the moment and believe that their method is effective. Whereas the other half do not believe

that their method is effective at identifying plagiarism.
Question 5: I think that the proportion of students involved in source code plagiarism in my course is
From the answers to this question shown in Figure 1.3 we find that the majority of the members believe that less than 10% of the students are involved in plagiarism. Following that 3 members believe it to be in the range of 10% to 20%. Finally, 1 member believes the problem to be more severe, indicating a range of 20% to 30%. Overall the faculty seems to believe that not many students in NUIM are involved in plagiarism.

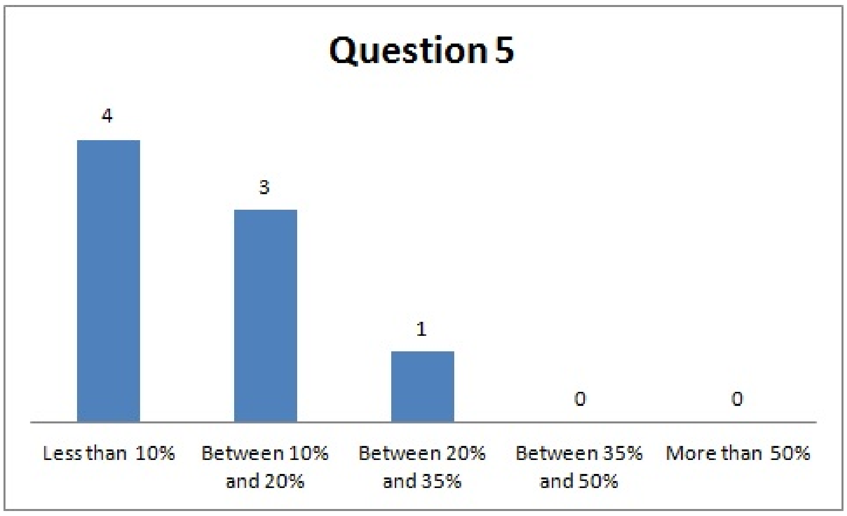


Figure 1.3: Response to Question 5

Question 6: I think that the problem of source code plagiarism is:
The answers to this question are shown in Figure 1.4 and seem to support the idea that faculty at NUIM do not consider plagiarism to be a major problem. Only one member of the faculty answering that it is bad and getting worse.

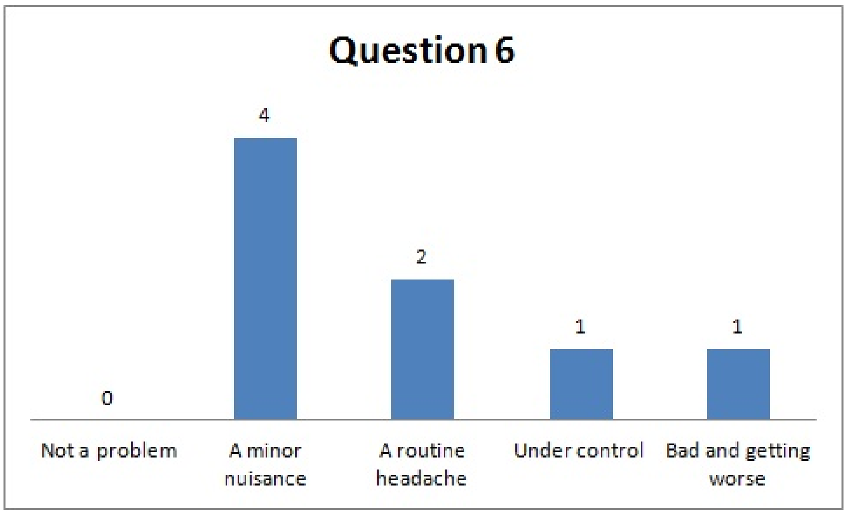


Figure 1.4: Response to Question 6

Question 7: Would you be interested in using an automated plagiarism detection tool?
From the 8 members who choose to answer this question, all unanimously answered that they would be interested in using a automated plagiarism detection tool.

From the survey we discovered that the majority of assignment submissions were in Java programming language. Next we found that the majority of the lecturers didn't use any automated detection tool instead relying on manually checking for instances of plagiarism. Overall the general consensus in the computer science department is that software plagiarism is not a big problem. Finally all the lectures showed interest in using automated plagiarism detection tools to help them in their work.

1.4 Test Driven Development

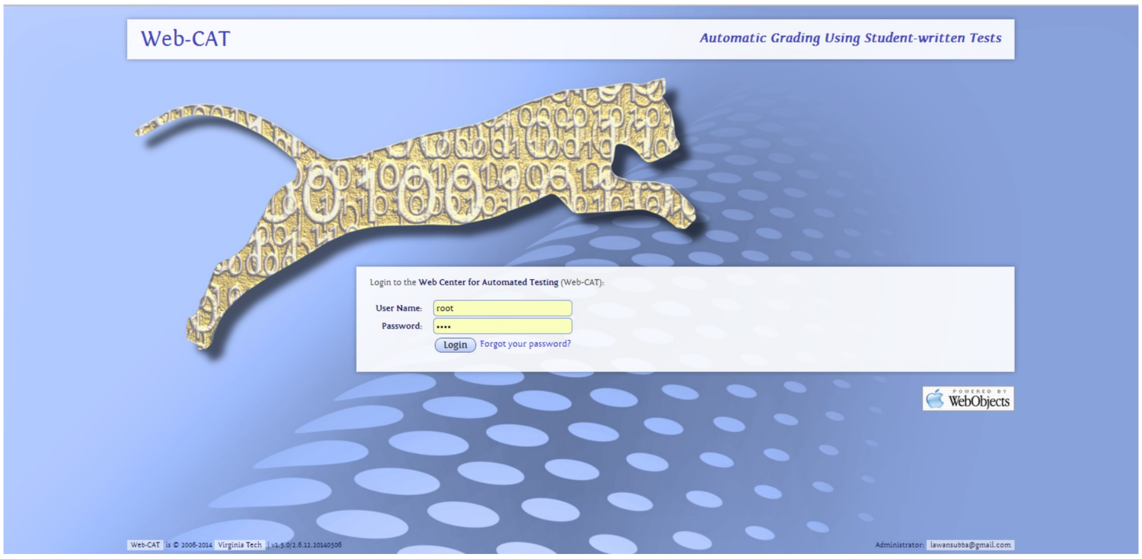
Test driven development (TDD) is a test first strategy, where there can be no code without test cases covering them. In this approach programmers are expected to write the test cases first and then code is written that will pass those tests. After every change of the code, all the existing tests are applied to it and any new functionality will have more test cases associated with it. Therefore TDD gives programmers the confidence that their program does exactly what they want and nothing more. TDD code development strategy has been popularised by extreme programming. The practice of TDD as summed by Beck [Bec03] is

1. Create new test case.
2. See if the new test case will fail by running all the test cases
3. Write just enough code to make the test pass.
4. Re-run the test cases and see them all pass.
5. Refactor code to remove duplication.

TDD is in contrast to "Big Bang" development, where programmers write all the code first and then try to run, debug and test it. This approach is counter productive as most bugs introduced early on are costly to fix later on, Therefore it introduces a significant overhead. TDD can help resolve those problems.

1.5 Web-CAT

Web-CAT is an acronym for web based center for automated testing. Web-CAT [Edw14] is a open-source automated grading system for programming assignments. It champions TDD by grading students on how well they have tested their programs. Students are encouraged to write test codes for their own programs, therefore the students are responsible for responsible for verifying the correctness of their programs. Web-CAT is licensed under the terms of the GNU General Public License and can be downloaded from its site [Edw14]. It won the 2006 premier award, recognising high-quality, non-commercial course-ware for engineering education. It is a comprehensive, extensible tool for automatically processing student program submissions, grading them, and generating feedback. Many institutions like University of Pisa (Italy), Virginia Tech (United States of America), University of the Basque Country (Spain) are actively running Web-CAT. Figure 1.5 shows how the web interface of Web-CAT looks on a browser.



Login to the **Web Center for Automated Testing** (Web-CAT):

User Name: root
Password: ••••

Web-CAT is © 2006-2014 Virginia Tech | v1.5.0/2.6.12.20140506

Administrator: lawansubba@gmail.com

Figure 1.5: Web-CAT Interface.

The creators of Web-CAT [Edw03] define the goals of web-CAT as:

1. For every assignment submission, Web-CAT require that the student provide tests as well.

2. Motivate students to write tests for their code from the very start, as opposed to testing only after the code is complete.

3. Support the principle of "write a little test, write a little code", which is one of the fundamental principles of TDD.

4. Provide timely and useful feedback to the student on the quality of their tests and the quality of their solution.

5. Utilise a grading/rewarding system that will promote practices and behaviors the students should possess.

Web-CAT is based on a three-tiered architecture. The system is implemented as a web application with a plug-in-style architecture, therefore it can be extended by adding new plugins to provide additional services. Its architecture is shown by Figure 1.6 [Sha03]. The main components of Web-CAT are:

1. Client Tier: At the client tier, we will have users who use the web browser to send request and will receive a response from the server.

2. Middle Tier: This tier will act as the web server tier and all the business logic will be located here. The web server will respond to user requests by serving them with HTML pages.

3. Database Tier: The third server is comprised of database and file servers. The file server contains files submitted by the user, whereas the database server stores information regarding the users, courses and wizard states.

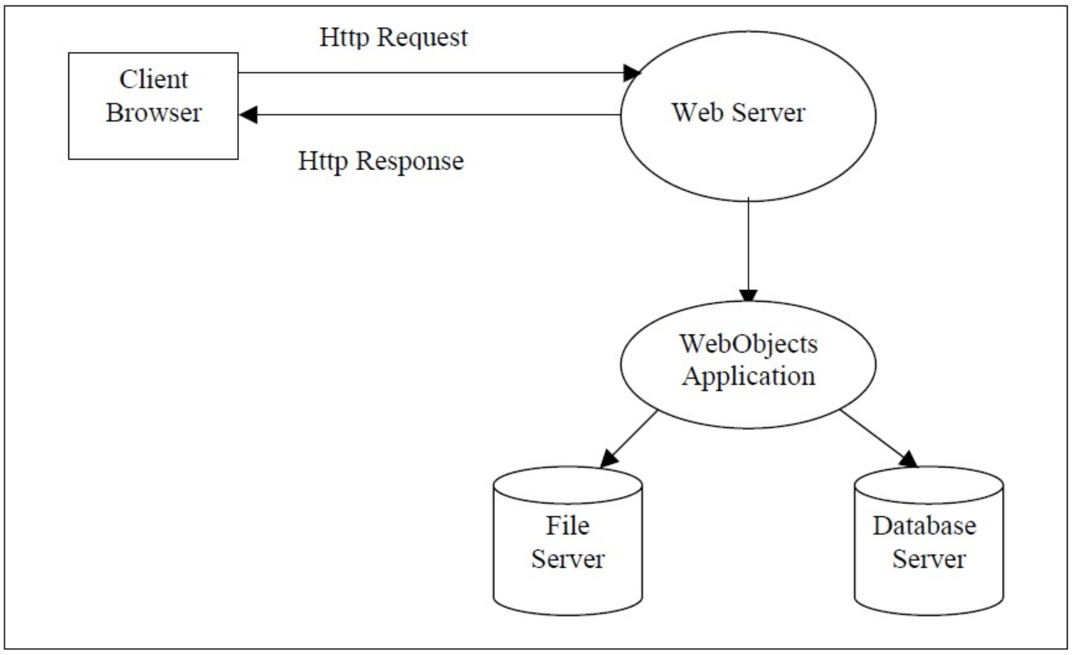


Figure 1.6: Web-CAT Architecture

Web-CAT supports the grading of assignments written in Java (1.4 and 1.5), Scheme (both MIT Scheme and DrScheme), Prolog (SWI), Pascal (FreePascal), Standard ML (SML/NJ), and c++ (g++ 3.4.x) by utilising language specific grading plug-ins. It allows further flexibility by allowing us to create grading plug-ins for newer languages by modifying existing plug-ins.

Web-CAT supports several different roles of users, like administrator, instructors and students, with each having its own access rights. In Web-CAT, the administrators are the superusers of the system. They can create courses (For e.g: Algorithms, Software Engineering) in Web-CAT. The administrator can create instructors and students and then assign them to the courses. The instructor can create assignments by setting submission rules in their course and publishing them for students to see. Finally students can submit their assignments. Web-CAT will then verify, grade and test the program based on a pre-set grading scheme set up by the instructor and return the score to the student. Therefore the instructor only has to create assignments, tests and grading schemes, while the system automatically performs grading and feedback generation. Students use Web-CAT with just a web browser, but it also supports electronic submissions and provides plug-ins for a number of IDEs to simplify student usage (Eclipse, NetBeans, etc.).

A study [Edw03] had been undertaken in 2003 in Virginia Tech University to investigate the usage of Web-CAT. The study compared students performance when using Web-CAT with student performance when a automated test script generator was provided by the instructor. The generator would create test scripts on the students submissions and provide feedback. Overtime there was a remarkable 28% reduction in defects per 1000 lines of code. The results showed that Web-CAT with its emphasis on TDD plays a role in improving students learning. As with every submission, students received a more clear perspective on how to test their code, so they were less likely to repeat the same mistakes in subsequent submissions.

However at the time of writing Web-CAT does not have functionality to support plagiarism detection. This means that the large volumes of submitted assignments are only graded on the quality of students test codes. If the instructor wants to check for plagiarism s/he has to manually compare the submitted programs. Manually checking is time consuming and requires a lot of effort if the number of students in a course is high.

1.6 Problem Statement

Web-CAT is a beneficial application as it is free to use. In addition it helps improve the quality of students programs by evaluating the quality of their test cases and providing immediate feedback. However it lacks support to detect plagiarism in the submitted programs. As mentioned in Section 1.1 plagiarism is a major concern in academics. It is wrong for students to receive academic credit for assignments that have been plagiarised. Furthermore we also discussed the detrimental effects of plagiarism in programming courses in Section 1.2. We realize that creating a automated plagiarism detection application and integrating it with Web-CAT would give added benefits to the academic institutions using it. They would have the benefits of TDD as described in Section 1.4 and also the ability to detect and stop software plagiarism.

Our research will try to answer the question: how to practically incorporate an automated plagiarism detection functionality in to Web-CAT? In order to answer this question, we will study the various plagiarism detection tools and their algorithms that will be discussed in Section 2.2. We will select one algorithm by using programming assignments at NUIM as reference. We realise that the Java programming language is immensely popular for programming courses. Therefore our research will focus only on detecting plagiarism on programs submitted using the Java programming language. Furthermore we will also select a metric that will allow us to best represent the measure of similarity between programs. Finally we realise that programs submitted as response to the same assignment is bound to have some degree of similarity. Therefore we will also explore the threshold values that can be used to detect plagiarism.

The institutions that are actively running Web-CAT would be interested in using our solution. As Web-CAT is an GNU general public licensed open source application. We can also release our solution under the very same license and allow current and future users of Web-CAT to download and install it. Also in the study conducted in 1.3 we found that all the members of the faculty were interested in using a plagiarism detection tool. Our solution would benefit the faculty members at NUIM also.

1.7 Summary

In this chapter we have described the problem statement which we are trying to address with our research. The rest of the paper is organised as follows. Chapter 2 provides an overview of the related work done in the area of automated assignment assessment and plagiarism detection. In Chapter 3 we will discuss the theoretical background on which we base our solution. In chapter 4 we will evaluate the effectiveness of our solution. Finally in chapter 5 we give our conclusions and discuss possible future work.

Chapter 2

Related Work

2.1 Automated Assessment

In any academic setting, tutor assessment is a fundamental part of the learning process. It provides a means to guide the learning process and receive feedback for both the tutor and the student involved. In computer courses, programming assignments help the tutor evaluate the students understanding of the course and provide feedback where necessary. Due to the large numbers of student involved it is usually the case that tutors don't have time to provide meaningful feedback. By using an Automated Assessment tool this problem can be reduced. Most of the tools will execute the students assignment and compare the results with a reference solution.

A study done by Ihantola *et al.* [IAKS10] discusses the various different Automated assessment tools available and their features. The study has also indicated fragmentation in the domain with too many solutions (For eg: Moodle, Automatic Grader, etc). Without an industry wide acceptance of single solution the field seems to be stagnant with solutions with limited functionality or that offer redundant features. The authors have also identified Web-CAT as a good candidate for extension and acceptance as a automated assessment tool for computer science courses. In 2013, NUIM installed Web-CAT and experimented with its usage. Therefore this research is directed on the usage of Web-CAT and not any other automated assessment tool.

2.2 Plagiarism Detection tools

In order to detect plagiarism, early systems used a attribute based approach(For example [Jon01]). Where several properties of the source code files are counted and each files are represented by a sequence of attribute based numbers. Finally, files were compared with each other using their sequence of numbers. However, Whale *et al.* [Wha90] demonstrated that such attribute based approaches were not capable of detecting similar programs.

Later structure based approaches were developed which compares programs using its structure instead of using attributes, therefore it provides an improved measure of similarity and is more effective at detecting plagiarism in programs [Wha90]. The first step in structure based approach usually involves tokenization where a program is converted to a token sequence by a lexical analyzer, next the token sequences are compared to find sim-

Feature	JPlag	MOSS	Plaggie
1 - Supported languages (#)	6	23	1
2 - Extendability	No	No	Yes
3 - Presentation of results	HTML	HTML	HTML and Text based
4 - Exclusion of template	Yes	Yes	Yes
5 - Submission or file-based rating	Submission	Submission	Submission
6 - Local or web-based	Web	Web	Local
7 - Open source	No	No	Yes

Table 2.1: Feature Matrix Plagiarism Detection Tools

ilar programs. For detection of plagiarism in software, there are various tools [HRvV11] available. Each of these tools has its own features, functionality and weaknesses. An overview of the features of 3 tools JPlag, Plaggie and MOSS are given in Table 2.1. We will discuss them in detail in the following subsections.

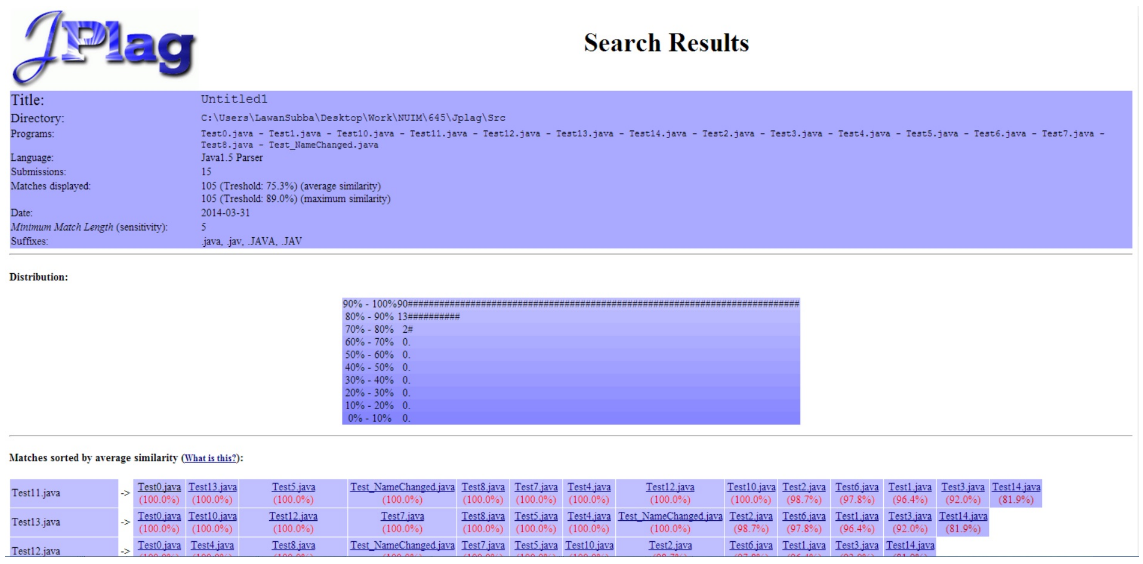
2.2.1 JPlag

JPlag [PMP00] was developed by Guido Malpohl at the University of Karlsruhe. It is closed source and therefore cannot be extended by the users. However it is publicly available for use as a WWW service. Potential users must register first with the service, and will be given credentials to use the JPlag service. JPlag is a structure-based approach. It converts programs into token strings that structurally represent the program. Subsequently programs are compared in pairs using the "'Greedy String Tiling"'. This algorithm was proposed by Michael Wise [Wis06]. The results are provided as a set of HTML files, Figure 2.1 shows how JPlag returns the overall results for all submitted files, the results are organized in a manner such that similar submissions are grouped together providing easy access. Figure 2.2 shows JPlag results for a pair of similar programs, where the sections of similar code are shown using color coding. The color coding again helps find sections of similar code quickly. JPlag supports the languages Java, C#, C, C++, Scheme and natural language text. It also provides the option to set the base code directory containing all the files that should be excluded from the comparison. Finally JPlag expects every program to be examined to be located in its own directory, the directories may also hold multiple files per submission.

2.2.2 Plaggie

Plaggie was developed in 2002 by Ahtiainen *et al.* [ASR06]. It is an open source plagiarism detection tool meant only for Java programs and can be extended. The results produced by Plaggie and configuration parameters required by it look very similar to JPlag, but Plaggie is a stand-alone command line Java application and must be installed locally.

The basic algorithm used for detecting similarity between two source code files is similar to JPlag, namely tokenisation followed by the Greedy String Tiling algorithm as described in [PMP00]. The results returned are also provided as a set of HTML files, Figure fig:Plaggie Results Overview Page shows the results overview page for all submissions. Unlike JPlag, Plaggie doesn't use grouping to show results instead it shows



Search Results

Title:	`Untitled1`
Directory:	`C:\Users\LawanSubba\Desktop\Work\NUIM\645\Jplag\Src`
Programs:	`Test0.java - Test1.java - Test10.java - Test11.java - Test12.java - Test13.java - Test14.java - Test2.java - Test3.java - Test4.java - Test5.java - Test6.java - Test7.java - Test8.java - Test_NameChanged.java`
Language:	Java1.5 Parser
Submissions:	15
Matches displayed:	105 (Treshold: 75.3%) (average similarity) 105 (Treshold: 89.0%) (maximum similarity)
Date:	2014-03-31
Minimum Match Length (sensitivity):	5
Suffixes:	.java, .jav, .JAVA, .JAV

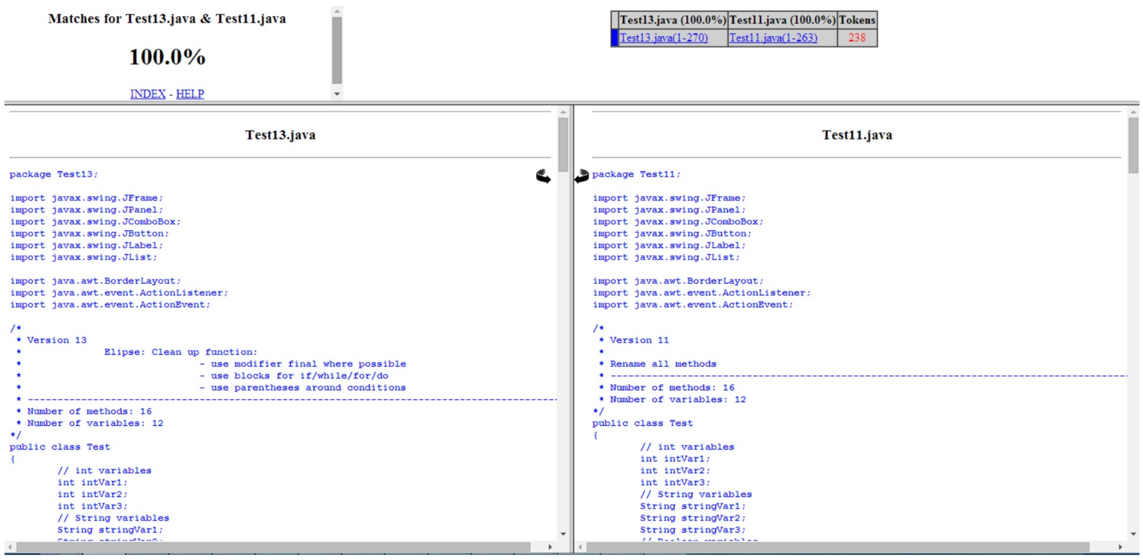
Distribution:

```
90% - 100%90##############################################################################
 80% - 90% 13##########
 70% - 80%   2#
 60% - 70%   0.
 50% - 60%   0.
 40% - 50%   0.
 30% - 40%   0.
 20% - 30%   0.
 10% - 20%   0.
  0% - 10%   0.
```

Matches sorted by average similarity (What is this?):

Test11.java	->	Test0.java (100.0%)	Test13.java (100.0%)	Test5.java (100.0%)	Test_NameChanged.java (100.0%)	Test8.java (100.0%)	Test7.java (100.0%)	Test4.java (100.0%)	Test12.java (100.0%)	Test10.java (100.0%)	Test2.java (98.7%)	Test6.java (97.8%)	Test1.java (96.4%)	Test3.java (92.0%)	Test14.java (81.9%)
Test13.java	->	Test0.java (100.0%)	Test10.java (100.0%)	Test12.java (100.0%)	Test7.java (100.0%)	Test8.java (100.0%)	Test5.java (100.0%)	Test4.java (100.0%)	Test_NameChanged.java (100.0%)	Test2.java (98.7%)	Test6.java (97.8%)	Test1.java (96.4%)	Test3.java (92.0%)	Test14.java (81.9%)	
Test12.java	->	Test0.java	Test4.java	Test8.java	Test_NameChanged.java	Test7.java	Test5.java	Test10.java	Test2.java	Test6.java	Test1.java	Test3.java	Test14.java		

Figure 2.1: JPlag Results Overview Page



Matches for Test13.java & Test11.java

100.0%

INDEX - HELP

Test13.java (100.0%)	Test11.java (100.0%)	Tokens
Test13.java(1-270)	Test11.java(1-263)	238

Test13.java

```
package Test13;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JComboBox;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JList;

import java.awt.BorderLayout;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

/*
 * Version 13
 *               Elipse: Clean up function:
 *                               - use modifier final where possible
 *                               - use blocks for if/while/for/do
 *                               - use parentheses around conditions
 * ---------------------------------------------------------------------------------------------
 * Number of methods: 16
 * Number of variables: 12
*/
public class Test
{
        // int variables
        int intVar1;
        int intVar2;
        int intVar3;
        // String variables
        String stringVar1;
```

Test11.java

```
package Test11;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JComboBox;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JList;

import java.awt.BorderLayout;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

/*
 * Version 11
 *
 * Rename all methods
 * ---------------------------------------------------------------------------------------------
 * Number of methods: 16
 * Number of variables: 12
*/
public class Test
{
        // int variables
        int intVar1;
        int intVar2;
        int intVar3;
        // String variables
        String stringVar1;
        String stringVar2;
        String stringVar3;
```

Figure 2.2: JPlag Result for a pair of programs

all similarities in a list. Therefore it will take more time and effort to interpret results of Plaggie if the number of submissions are large. Figure 2.4 shows the similarity results for a pair of programs, where we can view the similar sections in a program but it does not use colour coding like JPlag to show similar sections. Plaggie also allows us to exclude template code from the comparison by setting a file name in the configuration file. It also expects submissions to be separated into directories.

The authors also mention that fact that they have not implemented many optimizations that were implemented by the creators of JPlag. In addition in the read-me file of Plaggie, the authors have listed known successful attacks

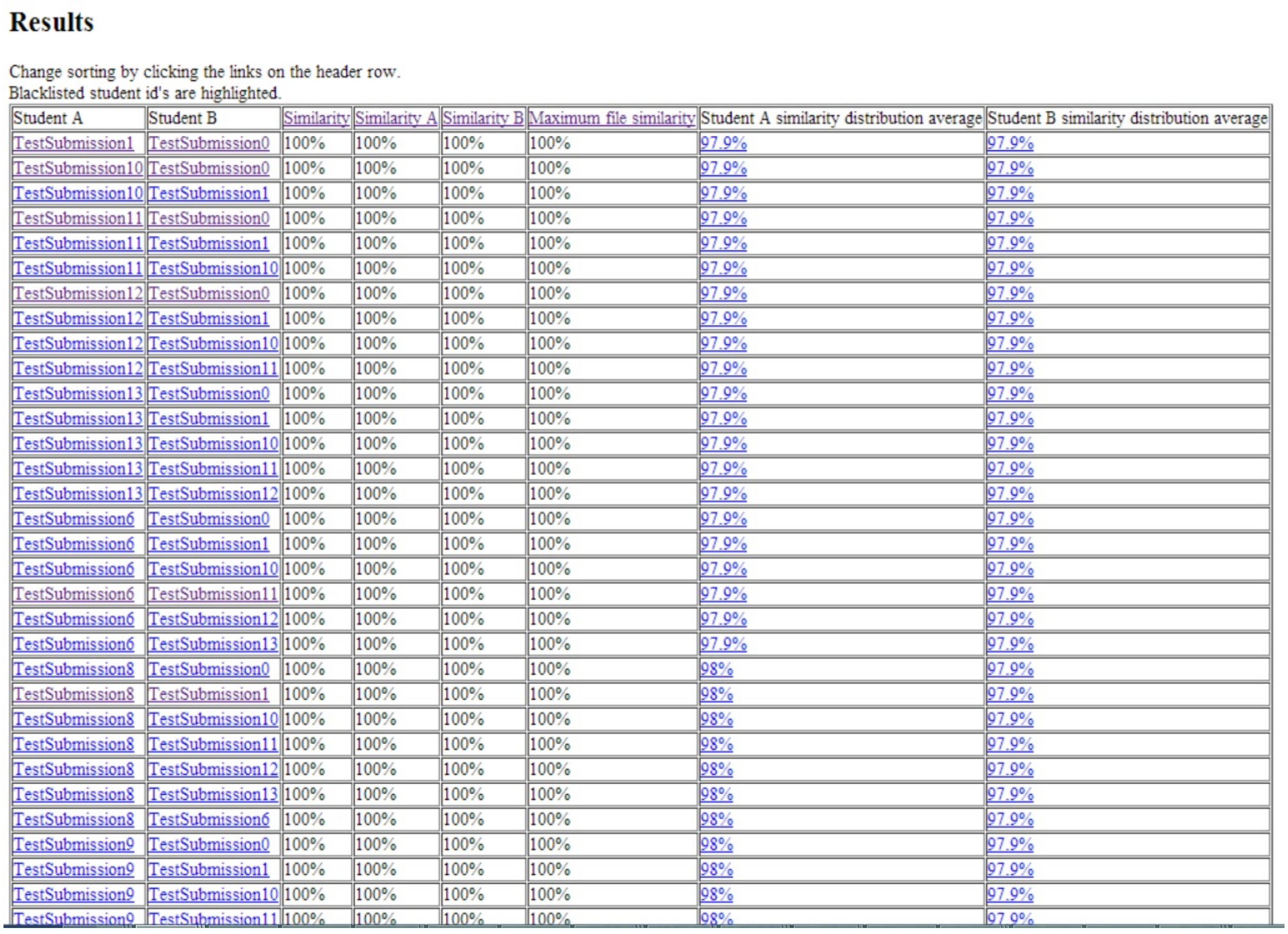
1. Inclusion of redundant program code
2. Changing the order of case-blocks and if-else blocks

3. Moving inline code to separate methods and vice versa

Also listed are unsuccessful attacks like

1. renaming the names of classes, methods or variables
2. changing comments or indentation

Furthermore the authors also state that the tool is not at its is best at comparing GUI codes. Currently Plaggie only works for Java 1.5, at present there are no plans to update support to a later version of java. In addition we found a modified version of Plaggie called Plaggie2. This version was created by Beata Katuscakova [Kat] in 2010 as part of her final undergraduate project in Univerzita Pavla Jozefa Safarika Prirodovedecka Fakulta. This version can be downloaded from its website[1]. An indepth discussion on the differences between the two tools will be provided in Section 4.1.



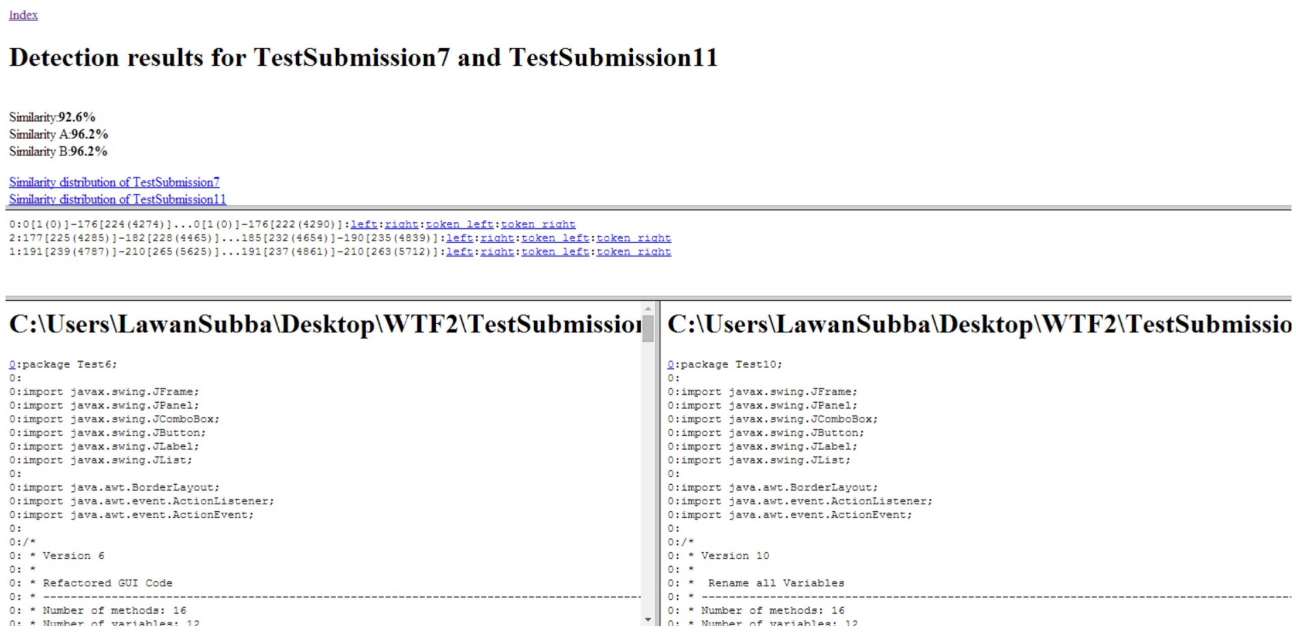
Results

Change sorting by clicking the links on the header row.
Blacklisted student id's are highlighted.

Student A	Student B	Similarity	Similarity A	Similarity B	Maximum file similarity	Student A similarity distribution average	Student B similarity distribution average
TestSubmission1	TestSubmission0	100%	100%	100%	100%	97.9%	97.9%
TestSubmission10	TestSubmission0	100%	100%	100%	100%	97.9%	97.9%
TestSubmission10	TestSubmission1	100%	100%	100%	100%	97.9%	97.9%
TestSubmission11	TestSubmission0	100%	100%	100%	100%	97.9%	97.9%
TestSubmission11	TestSubmission1	100%	100%	100%	100%	97.9%	97.9%
TestSubmission11	TestSubmission10	100%	100%	100%	100%	97.9%	97.9%
TestSubmission12	TestSubmission0	100%	100%	100%	100%	97.9%	97.9%
TestSubmission12	TestSubmission1	100%	100%	100%	100%	97.9%	97.9%
TestSubmission12	TestSubmission10	100%	100%	100%	100%	97.9%	97.9%
TestSubmission12	TestSubmission11	100%	100%	100%	100%	97.9%	97.9%
TestSubmission13	TestSubmission0	100%	100%	100%	100%	97.9%	97.9%
TestSubmission13	TestSubmission1	100%	100%	100%	100%	97.9%	97.9%
TestSubmission13	TestSubmission10	100%	100%	100%	100%	97.9%	97.9%
TestSubmission13	TestSubmission11	100%	100%	100%	100%	97.9%	97.9%
TestSubmission13	TestSubmission12	100%	100%	100%	100%	97.9%	97.9%
TestSubmission6	TestSubmission0	100%	100%	100%	100%	97.9%	97.9%
TestSubmission6	TestSubmission1	100%	100%	100%	100%	97.9%	97.9%
TestSubmission6	TestSubmission10	100%	100%	100%	100%	97.9%	97.9%
TestSubmission6	TestSubmission11	100%	100%	100%	100%	97.9%	97.9%
TestSubmission6	TestSubmission12	100%	100%	100%	100%	97.9%	97.9%
TestSubmission6	TestSubmission13	100%	100%	100%	100%	97.9%	97.9%
TestSubmission8	TestSubmission0	100%	100%	100%	100%	98%	97.9%
TestSubmission8	TestSubmission1	100%	100%	100%	100%	98%	97.9%
TestSubmission8	TestSubmission10	100%	100%	100%	100%	98%	97.9%
TestSubmission8	TestSubmission11	100%	100%	100%	100%	98%	97.9%
TestSubmission8	TestSubmission12	100%	100%	100%	100%	98%	97.9%
TestSubmission8	TestSubmission13	100%	100%	100%	100%	98%	97.9%
TestSubmission8	TestSubmission6	100%	100%	100%	100%	98%	97.9%
TestSubmission9	TestSubmission0	100%	100%	100%	100%	98%	97.9%
TestSubmission9	TestSubmission1	100%	100%	100%	100%	98%	97.9%
TestSubmission9	TestSubmission10	100%	100%	100%	100%	98%	97.9%
TestSubmission9	TestSubmission11	100%	100%	100%	100%	98%	97.9%

Figure 2.3: Plaggie Results Overview Page

[1]https://code.google.com/p/plaggie2/


Index

Detection results for TestSubmission7 and TestSubmission11

Similarity:**92.6%**
Similarity A:**96.2%**
Similarity B:**96.2%**

Similarity distribution of TestSubmission7
Similarity distribution of TestSubmission11

```
0:0[1(0)]-176[224(4274)]...0[1(0)]-176[222(4290)]:left:right:token left:token right
2:177[225(4285)]-182[228(4465)]...185[232(4654)]-190[235(4839)]:left:right:token left:token right
1:191[239(4787)]-210[265(5625)]...191[237(4861)]-210[263(5712)]:left:right:token left:token right
```

C:\Users\LawanSubba\Desktop\WTF2\TestSubmissio

```
0:package Test6;
0:
0:import javax.swing.JFrame;
0:import javax.swing.JPanel;
0:import javax.swing.JComboBox;
0:import javax.swing.JButton;
0:import javax.swing.JLabel;
0:import javax.swing.JList;
0:
0:import java.awt.BorderLayout;
0:import java.awt.event.ActionListener;
0:import java.awt.event.ActionEvent;
0:
0:/*
0: * Version 6
0: *
0: * Refactored GUI Code
0: * ---------------------------------------------------------------------------------------------
0: * Number of methods: 16
0: * Number of variables: 12
```

C:\Users\LawanSubba\Desktop\WTF2\TestSubmissio

```
0:package Test10;
0:
0:import javax.swing.JFrame;
0:import javax.swing.JPanel;
0:import javax.swing.JComboBox;
0:import javax.swing.JButton;
0:import javax.swing.JLabel;
0:import javax.swing.JList;
0:
0:import java.awt.BorderLayout;
0:import java.awt.event.ActionListener;
0:import java.awt.event.ActionEvent;
0:
0:/*
0: * Version 10
0: *
0: *  Rename all Variables
0: * ---------------------------------------------------------------------------------------------
0: * Number of methods: 16
0: * Number of variables: 12
```

Figure 2.4: Plaggie Result page for a pair of programs

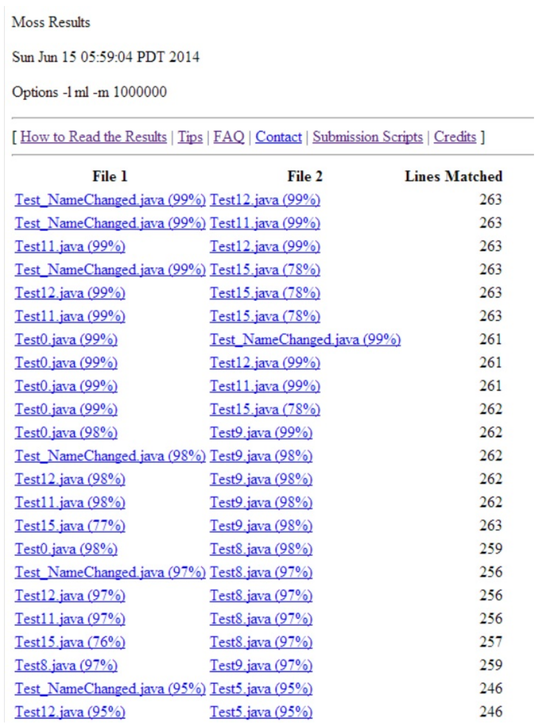
2.2.3 MOSS

MOSS stands for Measure Of Software Similarity. It was developed in 1994 at Stanford University by Aiken *et al.* [A$^+$05]. Moss is being provided as a web-service, the service can be consumed by using a Perl script which is provided by the moss website [A$^+$05]. You can download the submission script from the MOSS website[2], however in order to use it you need a moss id which can be obtained by sending an email to moss@moss.stanford.edu. In order to measure similarity between documents, moss uses a document finger printing algorithm called winnowing [Wis06]. Document fingerprinting divides a document into contiguous sub-strings, called k-grams. All the k-grams are hashed and a subset of them are selected as its fingerprint. These fingerprints are then used to compare pairs of programs. Finally MOSS creates a HTML output of the results on its own server and provides the URL to the user. The results may only remain on the server for a couple of weeks. Figure 2.5 shows the results overview page of MOSS for all submissions. Like Plaggie, MOSS shows all similarities in a list, which can get very large for huge number of submissions. Therefore it takes time and effort to interpret MOSS results for large number of submissions. Figure 2.6shows the similarity results for a pair of programs, like JPlag it shows similar sections of the program using color coding. MOSS also expects files to be examined to exist in its own directory.

2.2.4 Other Tools

In addition to these 3 tools, there are many others tools [HRvV11] like SIM, Marble which are available for plagiarism detection. However these will not be discussed in detail in our study as we will focus only on JPlag, Plaggie and MOSS. We choose JPlag and MOSS as they were most frequently cited and peer reviewed. Plaggie was chosen as it is based

[2]http://theory.stanford.edu/ aiken/moss/



Moss Results

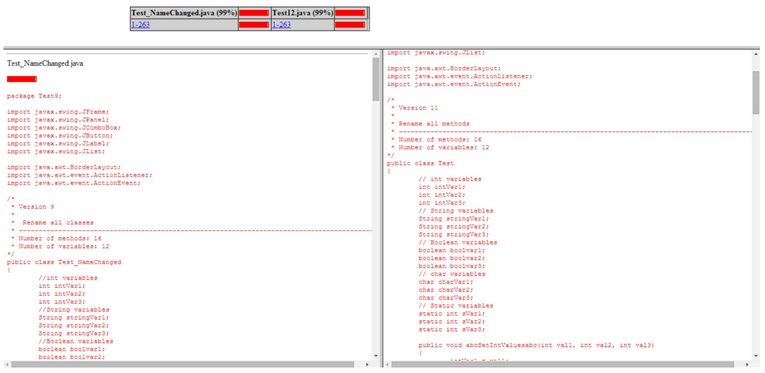
Sun Jun 15 05:59:04 PDT 2014

Options -l ml -m 1000000

[How to Read the Results | Tips | FAQ | Contact | Submission Scripts | Credits]

File 1	File 2	Lines Matched
Test_NameChanged.java (99%)	Test12.java (99%)	263
Test_NameChanged.java (99%)	Test11.java (99%)	263
Test11.java (99%)	Test12.java (99%)	263
Test_NameChanged.java (99%)	Test15.java (78%)	263
Test12.java (99%)	Test15.java (78%)	263
Test11.java (99%)	Test15.java (78%)	263
Test0.java (99%)	Test_NameChanged.java (99%)	261
Test0.java (99%)	Test12.java (99%)	261
Test0.java (99%)	Test11.java (99%)	261
Test0.java (99%)	Test15.java (78%)	262
Test0.java (98%)	Test9.java (99%)	262
Test_NameChanged.java (98%)	Test9.java (98%)	262
Test12.java (98%)	Test9.java (98%)	262
Test11.java (98%)	Test9.java (98%)	262
Test15.java (77%)	Test9.java (98%)	263
Test0.java (98%)	Test8.java (98%)	259
Test_NameChanged.java (97%)	Test8.java (97%)	256
Test12.java (97%)	Test8.java (97%)	256
Test11.java (97%)	Test8.java (97%)	256
Test15.java (76%)	Test8.java (97%)	257
Test8.java (97%)	Test9.java (97%)	259
Test_NameChanged.java (95%)	Test5.java (95%)	246
Test12.java (95%)	Test5.java (95%)	246

Figure 2.5: MOSS Results Overview Page



Test_NameChanged.java (99%)	🟥	Test12.java (99%)	🟥
1-263	🟥	1-263	🟥

Test_NameChanged.java

```
package Test9;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JComboBox;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JList;

import java.awt.BorderLayout;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

/*
 * Version 9
 *
 *  Rename all classes
 * -----------------------------------------------------------------------------------
 * Number of methods: 16
 * Number of variables: 12
*/
public class Test_NameChanged
{
        //int variables
        int intVar1;
        int intVar2;
        int intVar3;
        //String variables
        String stringVar1;
        String stringVar2;
        String stringVar3;
        //Boolean variables
        boolean boolvar1;
        boolean boolvar2;
```

```
import javax.swing.JList;

import java.awt.BorderLayout;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

/*
 * Version 11
 *
 * Rename all methods
 * -----------------------------------------------------------------------------------
 * Number of methods: 16
 * Number of variables: 12
*/
public class Test
{
        // int variables
        int intVar1;
        int intVar2;
        int intVar3;
        // String variables
        String stringVar1;
        String stringVar2;
        String stringVar3;
        // Boolean variables
        boolean boolvar1;
        boolean boolvar2;
        boolean boolvar3;
        // char variables
        char charVar1;
        char charVar2;
        char charVar3;
        // Static variables
        static int sVar1;
        static int sVar2;
        static int sVar3;

        public void abcSetIntValuesabc(int val1, int val2, int val3)
        {
```

Figure 2.6: MOSS Result page for a pair of programs

on JPlag and is open source.

- SIM [Hag06] is a software similarity tester created by Dick Grune at the VU University Amsterdam. The latest version is SIM 2.77. It supports the languages C, Java, Pascal, Modula-2, Lisp, Miranda. It works by creating tokens from the source code first, then it builds forward reference tables which are used to detect similarity between newly submitted files. SIM can be downloaded from its website[3].

- Marble [Gru06] is a software similarity tool developed in 2002 at Utrecht University by Jurriaan Hage. It supports Java language only. In order to find similarities between programs. It follows a process of splitting, normalization and comparison. Finally the results are outputted to a script file.

[3]http://dickgrune.com/Programs/similarity_tester/

2.3 Summary

Lancaster *et al.* [LC04] compared eleven different source code detection engines and recommended using JPlag and MOSS. In addition the study by Hage *et al.* [HRvV11] found that the performance of JPlag and MOSS were similar. However both these tools are closed source and require the users to send files to their servers. Due to the sensitive nature of plagiarism we agreed that such an option would not be appropriate. As every time we use those services, we would be giving out sensitive information. Therefore we had to decide whether to create a detection tool from scratch or extend an already existing one. We choose to use the codebase of Plaggie for the following reasons.

1. Plaggie is open source and is based on JPlag. Both use the tokenization followed by greedy string tilling algorithm approach.
2. The codebase of Plaggie is in Java programming language, which is the same language as the codebase of Web-CAT. Therefore the two would integrate very easily with each other.
3. Plaggie only supports detection in programs written in Java 1.5, however that won't be a problem as Web-CAT only supports Java 1.5 and 1.4 submissions.

We discussed in Section 2.2, that JPlag groups similar results together which saves time and effort when there are large number of submissions. Whereas Plaggie and MOSS don't provide this feature, due to the open source nature of Plaggie we can extend it to also group similar results. In addition, we will extend Plaggie to color code similar sections of code as we found it helps find similar sections of code quickly. Finally neither of the three tools show "groups of cheaters", For example: let's say Student A copies a program from Student B and another student Student C copies from Student A, in such a case all 3 of them should fall under a single group of cheaters. We believe that adding this feature will help identify groups of students who have plagiarized together. In addition, it will also help identify plagiarizing students who may be missed by a list wise comparison of JPlag, Plaggie and MOSS.

Chapter 3

Background

This chapter will aim to describe the various concepts, technologies and terminologies that are used in this paper. First we describe the tokenization process which is the first task of Plaggie. Tokenisation is used to retrieve meaningful elements from a program called tokens. Second we discuss the Greedy string tilling algorithm, this is the algorithm used by Plaggie to compare the tokens of various programs and check for similarity. Third we discuss WebObjects, which is the framework used to develop Web-CAT. Then we discuss the GNU general public license and how it gives us the right to modify both Web-CAT and Plaggie. Next we discuss the Graphical User Interface testing, which will allow us to test our modified version of Web-CAT with plagiarism detection add-on. Finally we discuss about the Usability testing and why it is important for us to perform a usability test on our plagiarism detection add-on.

3.1 Tokenisation

In order for compilers to understand what a program means, a compiler will have to break it apart and understand its structure and meaning. To accomplish this, a process called tokenization is used. The process produces tokens which are in fact words, phrases or any other element that holds meaningful value by breaking up a stream of text by following a set of rules. Tokenization is used in disciplines such as linguistics to find the components of a written language. Also it is used in Computer Science, where it acts as a part of the lexical analysis. The entire process consists of two parts Lexical analysis and Parsing as seen in Figure 3.1 and is discussed in sub-sections 3.1.1 and 3.1.2.

This separation is performed to simplify at least one of the processes involved [App98]. For example: a parser that would have to consider comments and white-spaces during operation would be substantially more complex than one that assumes that it has already been handled by the lexical analyser. This process is also used by Plagiarism detection tools as the initial phase, it generates a set of tokens of the programs which then can be compared to other programs. The results of this process can be seen in figure 3.2

3.1.1 Lexical Analysis

The fundamental job of the lexical analyser is first to process the source program by reading the input characters. Then it will classify the input characters as lexemes and

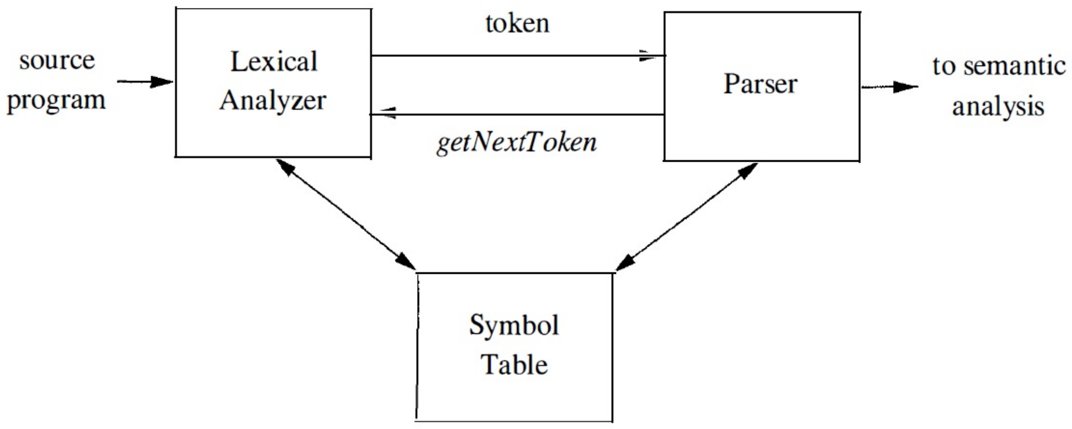


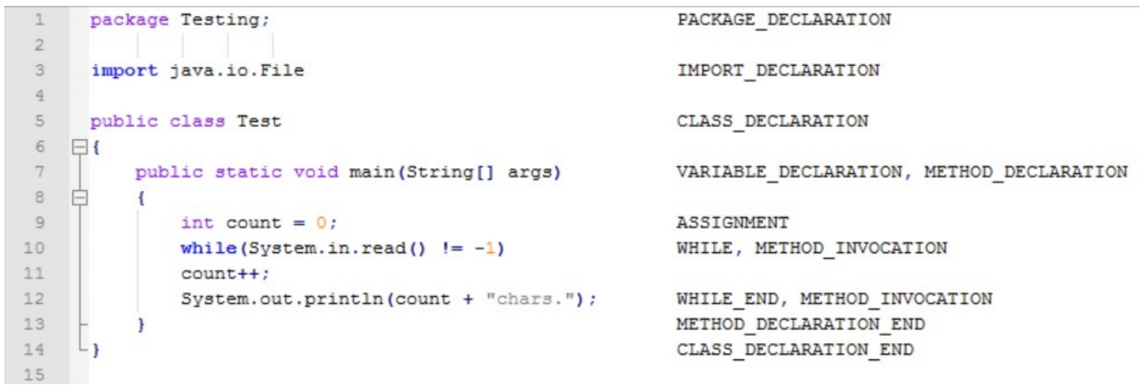
Figure 3.1: Interactions Between Lexical Analyzer and Parse

finally generate a series of tokens as output for every lexeme in the source program. Furthermore, it will discard things like comments, white space and tabs from the source program. During the analysis process the lexical analyser will communicate with the symbol table. If the lexical analyzer finds a lexeme that it regards as an identifier, it will write that lexeme into the symbol table. Finally the stream of tokens are passed to the parser to perform syntax analysis.

We can choose to write the Lexical Analysis code manually that will perform the task of identifying and returning the information of the identified tokens. However an automated approach would be to use a lexical analyser generator. We can specify a lexeme pattern to a generator and it will compile those patterns and generate code that will perform the role of a lexical analyzer. This approach makes modifying a lexical analyzer much easier, as we don't need to make changes to the program, we only need to modify the pattern. It is also makes the process of creating a lexical analyzer faster, as the programmer will specify the software at high level of abstraction and the tedious task of creating the actual program is passed to the generator. There are many lexer analyzer generators like Lex, Jlex, Flex. However, using lexer analyzer generators requires steep learning as we need to understand the complex grammar of the tool.

3.1.2 Parsing

A parser is a piece of software which takes in text data as input and outputs a data structure, either a abstract syntax tree, parse tree or any other hierarchical structure. Usually lexical analysis is followed by the parsing phase, where the parser uses the tokens created by the lexer to produce a data structure. Parsers can be written manually, but there are also options to use parser generators like JavaCC, Cup. Using parser generators too require a steep learning curve as we need to understand the complex grammar of the tool.



```
package Testing;                                PACKAGE_DECLARATION

import java.io.File                             IMPORT_DECLARATION

public class Test                               CLASS_DECLARATION
{
    public static void main(String[] args)      VARIABLE_DECLARATION, METHOD_DECLARATION
    {
        int count = 0;                          ASSIGNMENT
        while(System.in.read() != -1)           WHILE, METHOD_INVOCATION
        count++;
        System.out.println(count + "chars.");   WHILE_END, METHOD_INVOCATION
    }                                           METHOD_DECLARATION_END
}                                               CLASS_DECLARATION_END
```

Figure 3.2: Simple Tokenization Example

3.2 Greedy String Tilling Algorithm

After the tokenisation process, we will use the greedy string tilling algorithm introduced by Wise [Wis06] to compare token strings. When comparing strings generated from two code files A and B, the objective is to find a substring that is common in both and also satisfies the following rules [PMP00].

1. Any token taken from string A can only be matched with exactly one token taken from string B. This rule will prevent the source text from being matched to sections of text duplicated in the Plagiarised program.

2. The strings can be matched at any position such that reordering the parts of the source code cannot hide the similarity.

3. Longer matches of substring are preferred over shorter ones, as they are more reliable. The shorter matches are often cases of wrong detection.

By applying the third rule sequentially for every matching step, it leads to a greedy algorithm which will consist of two phases:

1. Phase 1:
   In this phase, we will search for the biggest contiguous match that can be found in both strings involved. This is done by 3 nested loops: the first one iterates over all tokens in the 1st string, the second will use that token and compare it with every other token in the 2nd string. The innermost loop will try to extend this match as far as possible until the strings are similar. The set of all the longest common substring will be found by these nested loops.

2. Phase 2:
   All the matches of maximal length that are found in phase 1 are marked. It means that their tokens may no longer be used in further searches in subsequent iterations of phase 1. This ensures that a string will only be used in one match and thus satisfies the first rule from above. Also, there is a probability that some of the matches may overlap. In such cases, only the first match is used.

These 2 phases are repeated until no more matches can be found. A pseudo-code for the algorithm can be seen in Figure 3.3

```
**Greedy-String-Tiling**(String $A$, String $B$) {
    $tiles = \{\}$;
    **do** {
        $maxmatch = MinimumMatchLength$;
        $matches = \{\}$;
        **Forall** unmarked tokens $A_a$ in $A$ {
            **Forall** unmarked tokens $B_b$ in $B$ {
                j = 0;
                **while** ($A_{a+j} == B_{b+j}$ &&
                        unmarked($A_{a+j}$) && unmarked($B_{b+j}$))
                    $j++$;
                **if** ($j == maxmatch$)
                    $matches = matches \oplus match(a, b, j)$;
                **else if** ($j > maxmatch$) {
                    $matches = \{match(a, b, j)\}$;
                    $maxmatch = j$;
                }
            }
        }
        **Forall** $match(a, b, maxmatch) \in matches$ {
            **For** $j = 0 \ldots (maxmatch - 1)$ {
                mark($A_{a+j}$);
                mark($B_{b+j}$);
            }
            $tiles = tiles \cup match(a, b, maxmatch)$;
        }
    } **while** ($maxmatch > MinimumMatchLength$);
    **return** $tiles$;
}
```

Figure 3.3: Greedy String Tilling Algorithm

3.3 WebObjects

Web-CAT was built on top of the WebObjects web applications framework. WebObjects [TD] allows you to develop and deploy World Wide Web applications by providing an object-oriented environment. The WebObjects application will be hosted on a server machine. Next the server will receive requests from client machine using web browsers and then the server will dynamically generate HTML pages as a response and returns the pages back to the clients browser. WebObjects supports rapid development of web applications by providing a web application server, prebuilt application components and a suite of different tools. WebObjects was purchased by Apple in 1997.

The various advantages [App] of WebObjects are

1. Streamlined Database Access
   Any dynamic web application is bound to use a database to store information and to provide rich content to it users. In WebObjects, database tables that are used by the application are represented as a collection of Java classes called Enterprise Objects.

This allows the developer to create a model that will map objects to data from the tables. This high level of abstraction frees the developer from having to write inflexible, database specific code. WebObjects automatically creates appropriate SQL code for all database operations by using drivers such as JDBC.

2. Separation of Model, View, and Controller
   WebObjects supports the MVC programming paradigm, and provides a clean separation of logic (Java code), presentation (Web pages) and data (data store).

3. State Management
   WebObjects provide state management by providing objects that will allow us to maintain information for a particular user session without the usage of cookies. WebObjects provides objects that allow you to maintain information for the life of a particular user session, or longer

4. Pure Java
   WebObjects applications are 100% pure Java, therefore they are platform independent and can be deployed on any system with a certified Java virtual machine.

5. Modular Development
   WebObjects give power to the user by providing an integrated set of tools and frameworks, that support the rapid assembly and deployment of complex web applications.

6. Scalability and Performance
   WebObjects also allow the administrator to host multiple instances of a web application. The application can be run either on one or several application servers. The developers then select from one of several load-balancing algorithms or may create their own algorithms too.

Client and server machines communicate with each other using request and response, during which various processes are involved as shown in Figure 3.4.

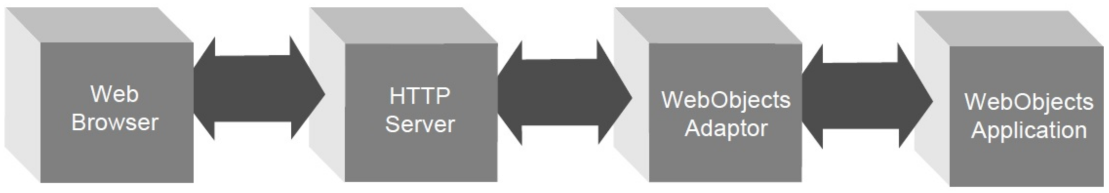


Figure 3.4: Chain of Communication between the browser and your applications

A brief description of these processes are as follows:

1. An HTTP server:
   The HTTP server receives requests from the client and uses either utilises the Common Gateway Interface (CGI), the Netscape Server API (NSAPI), or the Internet Server API (ISAPI).

2. A WebObjects adaptor:
   The WebObjects adaptor allows the web browser and WebObjects application to communicate by acting playing an intermediary role between HTTP server and WebObjects application.

3. A WebObjects application executable:
   The Webobjects application receives an incoming request and it will create a dynamically generated HTML page and return it as response.

In recent years we have seen that WebObjects is in a steady state of decline. During the writing of this thesis, we found it hard to find relevant up to date material on how to properly install WebObjects and use it. Apple itself has stopped supporting WebObjects and as of 2013 it is being maintained by the developer community WOCommunity Association [1].

3.4 GNU General Public License (GPL)

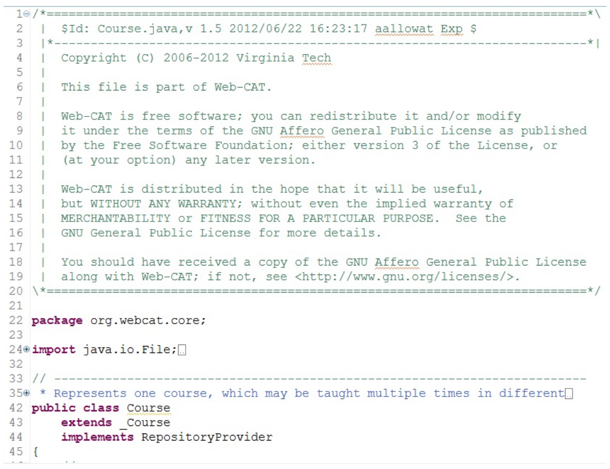
When you buy a proprietary software you are allowed to use it, but you are not provided rights to see the code, share and change the software. However you don't have to buy free software and it provides much more freedom than proprietary software. There are many different licenses for free software. Among them GNU general public license[2] is one of the most popular free software licenses, it provides the end users the freedom to use, study, share and modify the code of the software. Every software possessing these rights are called free software. Adding to that GPL also enforces the software to be copyleft. Copyleft means every modified or extended versions of the software will also retain the license, therefore the code and the license are legally inseparable. These rights provide an added incentive for programmers to release their code under the GNU general public license, as no one can misuse the software and everyone is obliged to release their modifications too.

Plaggie (ref Figure 3.6) is released under the GNU general public license. Whereas Web-CAT (ref Figure 3.5)is released under the GNU Affero General Public License [3] which is an extension of the GNU general public license. This extended license has an added requirement that if you run the free software on a server and allow users to access it on that server, the same server must provide the users the ability to download the modified version of the software. Therefore we have the right to make changes to both Plaggie and Web-CAT according to our needs, but our modified versions will also acquire the GNU general public license.

3.5 Graphical User Interface Testing with Selenium

Graphical user interfaces (GUI) are an integral part of any software. It is an interface that provides the ability for the user to interact with computer programs through visual

[1]http://wiki.wocommunity.org/display/WEB/Home
[2]http://www.gnu.org/copyleft/gpl.html
[3]http://www.gnu.org/licenses/agpl-3.0.html



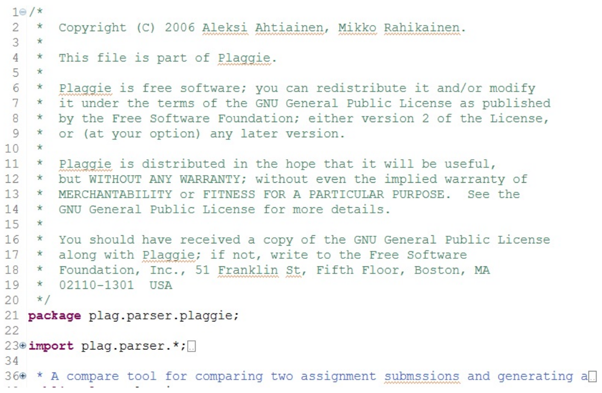
```
/*==========================================================================*\
 |  $Id: Course.java,v 1.5 2012/06/22 16:23:17 aallowat Exp $
 |*-------------------------------------------------------------------------*|
 |  Copyright (C) 2006-2012 Virginia Tech
 |
 |  This file is part of Web-CAT.
 |
 |  Web-CAT is free software; you can redistribute it and/or modify
 |  it under the terms of the GNU Affero General Public License as published
 |  by the Free Software Foundation; either version 3 of the License, or
 |  (at your option) any later version.
 |
 |  Web-CAT is distributed in the hope that it will be useful,
 |  but WITHOUT ANY WARRANTY; without even the implied warranty of
 |  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 |  GNU General Public License for more details.
 |
 |  You should have received a copy of the GNU Affero General Public License
 |  along with Web-CAT; if not, see <http://www.gnu.org/licenses/>.
\*==========================================================================*/

package org.webcat.core;

import java.io.File;

// -------------------------------------------------------------------------
 * Represents one course, which may be taught multiple times in different
public class Course
    extends _Course
    implements RepositoryProvider
{
```

Figure 3.5: GNU Affero General License Declaration for Web-CAT



```
/*
 *  Copyright (C) 2006 Aleksi Ahtiainen, Mikko Rahikainen.
 *
 *  This file is part of Plaggie.
 *
 *  Plaggie is free software; you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published
 *  by the Free Software Foundation; either version 2 of the License,
 *  or (at your option) any later version.
 *
 *  Plaggie is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with Plaggie; if not, write to the Free Software
 *  Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA
 *  02110-1301  USA
 */
package plag.parser.plaggie;

import plag.parser.*;

 * A compare tool for comparing two assignment submssions and generating a
```

Figure 3.6: GNU General License Declaration for Plaggie

icons and indicators. The user is able to access the functionality provided by the program by interacting with the various visual components and by following a number of predetermined sequence of instructions. The rise and popularity of GUI based applications is in stark contrast to previous Command Line Interface (CLI) applications which required steep learning curves and dedication. Right now we are surrounded by GUI applications; our operating systems, the programs on our computers, the websites we visit everyday and the mobile phones that keep us in touch with the rest of the world all have a GUI.

However,testing GUI based application is much harder than testing a command line interface based application. In the case of CLI applications, we can focus our tests on select classes and check for correctness, but GUI applications are event driven, we need to simulate the exact user event, wait for the system to process our request and check the results for correctness. In addition some functionality can only be accessed if we follow a certain predetermined sequence. For example, if a user wants to open a file, he/she has to click the file menu then click the open dialog and use the dialog to find and select the file.

Next, even though the underlying code does not change, the GUI components may change with every run of the program. For example, a button may move to another place, data in the tables may be sorted in a different order, or formats of entire pages may change as per user credentials. This causes significant problems in performing regression tests. On top of that GUI applications have a multitude of operations that can be performed. Testing all relevant test cases is a time consuming and laborious ordeal.

Regardless of the problems associated with GUI testing, it is critical that any GUI application is tested to prove that it meets its specification. In the past GUI testing was done manually, which was slow and error prone. This gave way to the popularity and use of automated GUI testing tools. There are many GUI testing tools available [4]. Some are open source whereas others are proprietary, each catering to a select operating system or type of GUI application. Some of the tools provide GUI testing functionality for web applications while others are for desktop applications like Windows applications, OSX [5] applications and Linux applications [6].

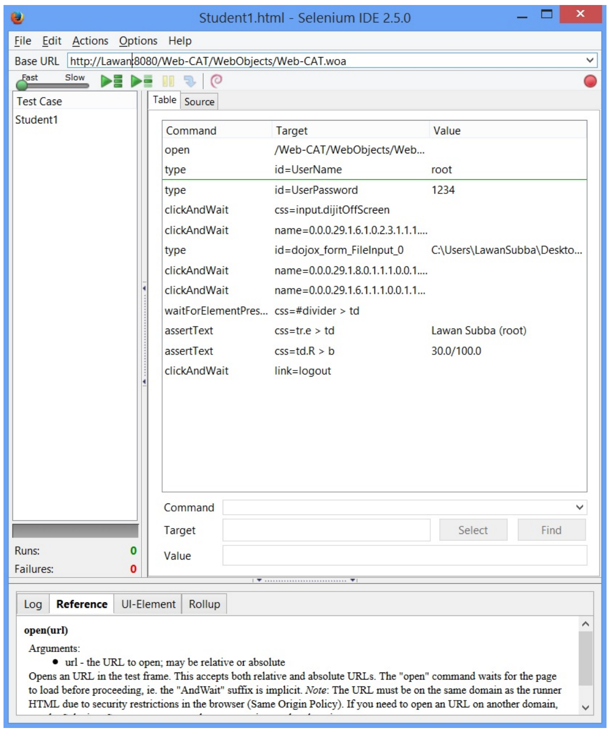
Figure 3.7: Selenium Logo

As Web-CAT is a web application, it is one of our requirements that we perform GUI testing to our modifications to Web-CAT. The purpose of the tests is to ensure not only that our modifications work but also that we have not introduced any bugs in the system with our changes. Also the modified version of Web-CAT must work on major browsers like Internet Explorer, Mozilla Firefox and on Google Chrome. For this purpose we will use the tool Selenium[7] (Figure 3.7). We made this decision as we are very familiar with Selenium as it is one of tools taught as part of the Software Testing (CS608) module. Selenium allows us to perform GUI testing on web applications and supports all the latest browsers and operating systems. It provides a wide array of functionality with the help of 4 selenium projects.

1. **Selenium IDE**

   Selenium IDE [8] (Figure 3.8) or Selenium 1.0 is an integrated development environment for Selenium scripts. It can be installed to our local installation of Mozilla Firefox as an extension. It allows us to record every event performed on the GUI, edit the recorded events and also debug the tests. Finally it is not only a recording tool, it allows us to record and play back the tests in the actual browser environment where the application runs. From Figure 3.8, it can be seen that that the IDE has created a test case that has captured all events on the web applications in the form of commands. We can edit the commands as per our requirements and

[4]http://en.wikipedia.org/wiki/List_of_GUI_testing_tools
[5]http://www.apple.com/ie/osx/
[6]http://en.wikipedia.org/wiki/Linux
[7]http://docs.seleniumhq.org
[8]http://docs.seleniumhq.org/projects/ide/


Student1.html - Selenium IDE 2.5.0
File Edit Actions Options Help
Fast Slow
Test Case
Student1
Table Source
Runs: 0
Failures: 0
Select
Find
Log Reference UI-Element Rollup

Base URL http://Lawan:8080/Web-CAT/WebObjects/Web-CAT.woa

Command	Target	Value
open	/Web-CAT/WebObjects/Web...	
type	id=UserName	root
type	id=UserPassword	1234
clickAndWait	css=input.dijitOffScreen	
clickAndWait	name=0.0.0.29.1.6.1.0.2.3.1.1.1....	
type	id=dojox_form_FileInput_0	C:\Users\LawanSubba\Deskto...
clickAndWait	name=0.0.0.29.1.8.0.1.1.1.0.0.1....	
clickAndWait	name=0.0.0.29.1.6.1.1.1.0.0.1.1...	
waitForElementPres...	css=#divider > td	
assertText	css=tr.e > td	Lawan Subba (root)
assertText	css=td.R > b	30.0/100.0
clickAndWait	link=logout	

Command
Target
Value

open(url)

Arguments:

- url - the URL to open; may be relative or absolute

Opens an URL in the test frame. This accepts both relative and absolute URLs. The "open" command waits for the page to load before proceeding, ie. the "AndWait" suffix is implicit. *Note*: The URL must be on the same domain as the runner HTML due to security restrictions in the browser (Same Origin Policy). If you need to open an URL on another domain,

Figure 3.8: Selenium IDE plugin for Mozilla

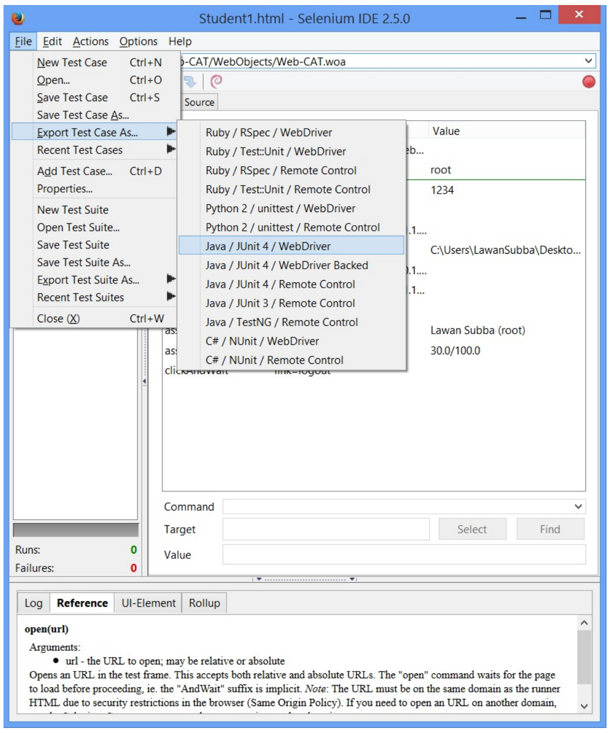
run through the commands to simulate actual user events on the web application. The IDE provides a user friendly way to run GUI tests. It also allows us to export the test case (Figure 3.9) in HTML, Ruby scripts, Java and many other formats.

2. **Selenium Remote Control**
   Selenium Remote Control (RC)[9] is a test tool that allows the user to write automated web application UI tests in any programming language against any HTTP website using any mainstream JavaScript-enabled browser. Selenium RC consists of 2 components, Figure 3.8 shows how the 2 components of Selenium RC work together.

   Firstly the Selenium server which is used to launch and then close browsers, it also analyses and runs the commands passed by the user's test program. It will then pass the results back to the users program after running the commands. The Selenium-core is a set of JavaScript functions which will interpret and then execute the commands passed by the user. In order to do the execution, it will use a built-in JavaScript interpreter in the browser. In addition to that,Selenium RC acts as an HTTP proxy, it intercepts and verifies HTTP messages passed between the application under test and the browser.

   The second component is the client library which provides the interface between the RC server and each of the programming languages that a user writes a test case in. There is a different client library for every supported language. The client library sends a command for execution to the Server. Then using the Selenium-Core

[9]http://docs.seleniumhq.org/docs/05_selenium_rc.jsp



open(url)

Arguments:

- url - the URL to open; may be relative or absolute

Opens an URL in the test frame. This accepts both relative and absolute URLs. The "open" command waits for the page to load before proceeding, ie. the "AndWait" suffix is implicit. *Note*: The URL must be on the same domain as the runner HTML due to security restrictions in the browser (Same Origin Policy). If you need to open an URL on another domain,

Figure 3.9: Export option for Selenium IDE plugin for Mozilla

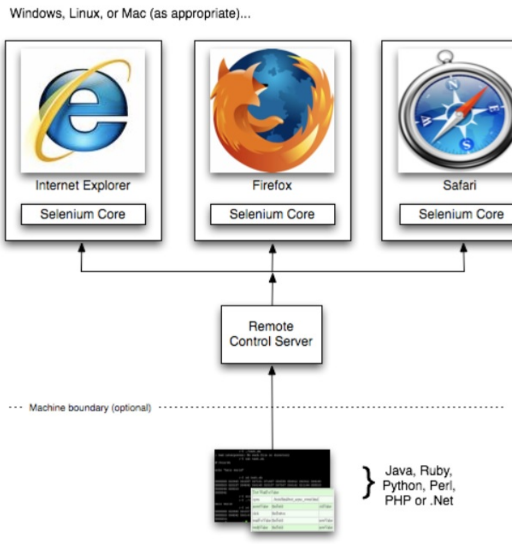

Windows, Linux, or Mac (as appropriate)...
Internet Explorer
Selenium Core
Firefox
Selenium Core
Safari
Selenium Core
Remote Control Server
Machine boundary (optional)
Java, Ruby, Python, Perl, PHP or .Net

Figure 3.10: Selenium RC in action

JavaScript commands, the server will pass the selenium command to the respective browser. Then the browser will use its JavaScript interpreter and execute the Selenium command. Finally the client library will receive the results of your set of commands and return it back to your program.

3. **Selenium WebDriver**
   The Selenium WebDriver or Selenium 2.0 adds the feature of integrating the Web-

Driver API. The WebDriver addresses the various limitations that existed in the Selenium RC API and also provides a more simpler interface for programming. The WebDriver is better at handling of dynamic web pages where the contents of the page change without the page being refreshed. It overcomes other limitations of Selenium 1.0 which hindered testing by providing better support for file uploading or downloading and handling pop-ups. Selenium-RC works in the same way for every browser, by injecting JavaScript functions to browser when the browser load. It then uses the browsers JavaScript to run test on the application under test. The Selenium WebDriver works differently, it directly accesses the browsers built in support for automation by making use of their drivers. There are drivers available for Chrome, Firefox and Internet Explorer. We can write application test code for the Selenium WebDriver by ourselves or use the Selenium IDE to create a test case and then export it to a language of our choice. We may then use the various browser drivers to test our application on various browsers.

4. **Selenium Grid**
   While Selenium RC and Selenium WebDriver focus on sequentially running tests on a single server, there may be cases when you might want to run tests on multiple servers or run parallel tests on the same server. Selenium Grid provides this functionality by allowing you to run tests on different browsers, different machines and in parallel.

In order to perform GUI testing on our modified version of Web-CAT. Selenium IDE was used to create test cases for different scenarios. This method was preferable rather than manually writing code as it requires less effort and is much quicker. In addition, we will run the test cases on the IDE itself for Mozilla. We will then export the test cases to Java format, and use Selenium WebDriver to run our tests on Chrome and Internet explorer. This will ensure that our modified version is stable for installation and use.

3.6 Usability Testing

Usability is defined as the ease of use of a human-made object and the effort required to learn to use that object. The object could be anything that a human interacts with like a software application, book, tool or a machine. The object of use can be a software application, website, book, tool, machine, process, or anything a human interacts with. Nielson [Nie03] proposed the 5 different attributes for usability which are described as

- Learnability: When a user encounters the design for the very first time, how easy it for them to perform basic tasks?
- Efficiency: Once the users grasped the design, how quickly can they perform basic tasks?
- Memorability: If the user returns to the design after a long interval of time, how easily can they re-establish competency?
- Errors: How often do users make errors, what is the severity of those errors, and how easily they can recover from such errors?

- Satisfaction: Do they users feel pleasant when they are using the design?

Usability testing [RC08] is the process of evaluating the degree to which a product satisfies usability criteria by using people as test participants who represent a certain target group. It is an effort to improve the productivity of the product and provide added benefits and satisfaction to the user. Design issues are exposed using the data gathered from the target group, and design decisions are formed which will attempt to remedy, minimize or eliminate the design flaws. Rubin *et al* describes the various types of testing that can be performed throughout a product life cycle as:

1. **Exploratory or Formative Study**
   The exploratory study is performed early on in the product development cycle , when the product is in the initial stages of being designed and defined. The main objective of this study is to determine if the initial design meets the users usability expectations. During this study an actual working product is not required, we can conduct this study using mockups, screen shots and illustrations. Designers benefit from this study, as it is performed early and minimizes the risk of putting effort on a product that is already flawed.

2. **Assessment or Summative Tests**
   Assessment tests are performed early or at some point in the middle of the product development cycle, but normally after some high-level implementation of the product has been finished. The objective of this test is to continue on the findings of the exploratory tests and examine the usability of developed product. In this study we are interested in actually seeing how a user performs realistic tasks on our system and identifying defects.

3. **Validation or Verification Test**
   The validation test, also referred to verification test is performed late in the development life cycle. The purpose of this test is ensure that problems discovered early on in the other tests have been fixed and new problems have not been introduced .With validation testing we also compare the usability of the product to previously established benchmarks.

If we fail to conduct a usability test on a software application, we risk releasing a software that has potential design problems. The design problems leads to an application that is hard to learn, inefficient in its operation, hard to remember and generates many errors. Such problems result in a decline in user satisfaction, and it is very likely users will opt out of our product and choose another one.

As Web-CAT is being used in many educational institutions, we cannot avoid performing usability testing on it. We will use the lecturers and teaching assistants of the Computer Science department as the target group for our Usability testing. The exploratory study was conducted with my supervisors at the beginning of this research, where we discussed the requirements of a plagiarism detection add-on for Web-CAT. After developing the add-on, we conducted a assessment test with the target group to evaluate the usability of our solution which will be discussed in detail in section ref-sec:Usability Testing on Web-CAT with Plagiarism Add-on. Finally if we have enough time, we will also conduct a validation test at the end of the development cycle.

Chapter 4

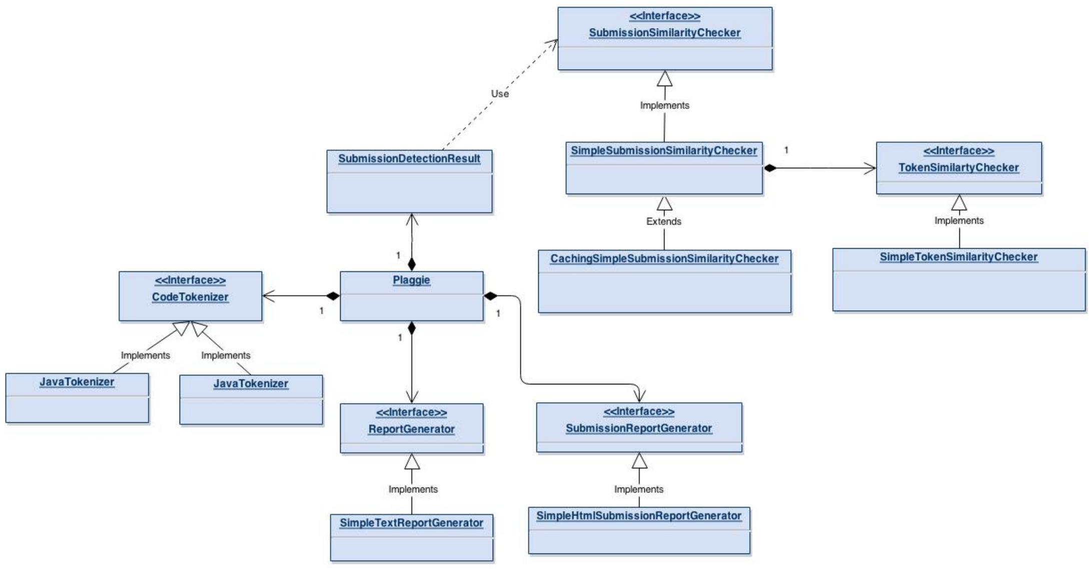
Solution

In this chapter we will discuss our approach at solving the problems we have discussed in chapters 1, 2, and 3. Firstly we will discuss how the 2 versions of Plaggie differ from each other and why we have chosen to integrate Plaggie2 with Web-CAT. Next we discuss the changes we have made in both Web-CAT and Plaggie2 to support this integration. Finally, as Web-CAT is an open source application, we discuss how our add-on can be installed on existing versions of Web-CAT.

4.1 Plaggie2 Versus Plaggie

In Section 2.3, we discussed our plan to use the open source code of Plaggie as a base and develop an add-on for Web-CAT. However, we mentioned in Section 2.2.2 that there are two versions of Plaggie available. The original version was created by Ahtiainen *et al.* [DH05] in 2002, while the newer version called Plaggie2 was created by Beata Katuscakova in 2010 [Kat] and both can be downloaded from their respective websites. During the literature review we found papers describing Plaggie, but we could find no papers on Plaggie2. However, after contacting the authors themselves directly, we were sent an final year undergraduate paper written in the Slovakian. As a result, at the beginning we found it hard to understand the significance of the modified version. However after studying the codebase of both versions, we were able to understand the significance of both and their differences. Firstly we will discuss how Plaggie works by discussing the major classes (Figure 4.1) that are contained in its implementation

1. Plaggie:
   This class is the entry point of the program, and it controls all the main operations. It compares all submissions for similarity by using algorithm 1, where a submission may comprise of a single file or a list of files. Next all of the submissions are compared for similarity, and if submissions consist of multiple files, the tool will compare every file from submission A with files from submission B. At the end of the process, we will have a SubmissionDetectionResult list holding results of our comparison. Finally, using this list we generate either a text or HTML report.



<<Interface>>
SubmissionSimilarityChecker
Use
Implements
SubmissionDetectionResult
SimpleSubmissionSimilarityChecker
1
<<Interface>>
TokenSimilartyChecker
Extends
Implements
CachingSimpleSubmissionSimilarityChecker
SimpleTokenSimilarityChecker
1
<<Interface>>
CodeTokenizer
Plaggie
1
1
1
Implements
Implements
JavaTokenizer
JavaTokenizer
<<Interface>>
ReportGenerator
<<Interface>>
SubmissionReportGenerator
Implements
Implements
SimpleTextReportGenerator
SimpleHtmlSubmissionReportGenerator

Figure 4.1: Overview of major classes in Plaggie

```
S is an array of submissions;
for i in 1 : length(S) - 1 do
    for j in 0 : (i-1) do
        submissionA ← S[i];
        submissionB ← S[j];
        Compare submissionA and submissionB for similarity;
    end
end
```

Algorithm 1: Plaggie Submission Comparison Algorithm

2. *CodeTokenizer*
   The CodeTokenizer interface defines the contractual obligations, that every class must satisfy if it wants to lex, parse and generate tokens from programs.

3. JavaTokenizer
   The JavaTokenizer class provides the implementation to the *CodeTokenizer* interface, and is responsbile for lexing, parsing and retrieving the tokens of a program. The Strategy design pattern is present here as the Plaggie class holds a reference to the CodeTokenizer interface. The Strategy design pattern allows us to use the JavaTokenizer class or any other class that implements the CodeTokenizer interface to perform lexing, parsing and retrieving of tokens.

4. *TokenSimilartyChecker*
   The TokenSimilartyChecker interface defines contractual obligations for classes that will actually compare the tokens of different programs. Here too we find the Strategy pattern, and we may supply a different class with a different algorithm.

5. SimpleTokenSimilarityChecker
   The SimpleTokenSimilarityChecker implements the *TokenSimilartyChecker* inter-

face by providing the implementation of the Greedy String Tilling algorithm discussed in Section 3.2. The SimpleTokenSimilarityChecker will take lists of tokens of two files as input and calculates similarity values. Two similarity values that range between 0 and 1 are computed as follows

$$SimilarityValueA = \frac{TotalNumberOfTokensMatchedBetweenTokenListAandB}{TotalNumberOfTokensInTokenListA} \tag{4.1}$$

$$SimilarityValueB = \frac{TotalNumberOfTokensMatchedBetweenTokenListAandB}{TotalNumberOfTokensInTokenListB} \tag{4.2}$$

6. *SubmissionSimilarityChecker*
   The SubmissionSimilarityChecker interface defines contractual obligations for classes that will use the TokenSimilarity interface to compare programs and hold the results of the comparison.

7. SimpleSubmissionSimilarityChecker
   The SimpleSubmissionSimilarityChecker class implements the *SubmissionSimilarityChecker* interface. This class will generate tokens of the files every time before a comparison. SimpleSubmissionSimilarityChecker class is used when preserving memory is not a concern, but as tokens are generated frequently the detection process is slowed down. This class also calculates the similarity on a submission level. As submission may consists of many files, this class will generate 2 similarity values for each pair of files using the formulas (4.1) and (4.2). Then using the SimilarityValueA and SimilarityValueB of each pair of files it computes SubmissionSimilarityValueA and SubmissionSimilarityValueB. Finally SubmissionSimilarityValueA, SubmissionSimilarityValueB values are used to compute SubmissionSimlarity, which ranges between 0 and 1. Where a value of 1 represents completely similar and 0 represents completely dissimilar. This process of calculating SubmissionSimlarity is also inherited by the CachingSimpleSubmissionSimilarityChecker class.

$$SubmissionSimilarityValueA = \frac{\sum SimilarityValueA}{NumberOfSimilarityValueA} \tag{4.3}$$

$$SubmissionSimilarityValueB = \frac{\sum SimilarityValueB}{NumberofSimilarityValueB} \tag{4.4}$$

$$SubmissionSimlarity = SubmissionSimilarityValueA * SubmissionSimilarityValueB \tag{4.5}$$

8. CachingSimpleSubmissionSimilarityChecker
   The CachingSimpleSubmissionSimilarityChecker inherits from the SimpleSubmissionSimilarityChecker class. This class compares the files in a manner such that tokens for files are generated, tokens are compared and saved to a token list. If the files appear again in another comparison, the tokens for those files are retrieved from the token list. For example: if the tool compares FileA.java and FileB.java, the token list will contain the tokens of both FileA.java and FileB.java. If at a later time the tool needs to compare FileA.java or FileB.java with other files, then the tokens are retrieved from the list. If memory is not a concern, CachingSimpleSubmissionSimilarityChecker class is used for comparison.

9. SubmissionDetectionResult
   The SubmissionDetectionResult class is the wrapper class that will use the implementations of the CodeTokenizer and SubmissionSimilarityChecker interface. For every comparison between programs, objects of SubmissionDetectionResult are created. These objects hold information about the submission pair and their similarity.

10. *ReportGenerator*
    The *ReportGenerator* interface defines contractual obligations for classes that provide functionality to create reports summarising the collected results in text format.

11. SimpleTextReportGenerator
    The SimpleTextReportGenerator class implements the *ReportGenerator* interface and provides the functionality to create results in text format.

12. *SubmissionReportGenerator*
    The *SubmissionReportGenerator* interface defines contractual obligations for classes that provide functionality to create reports summarising the collected results in HTML format.

13. SimpleHtmlSubmissionReportGenerator
    The SimpleTextReportGenerator class implements the *SubmissionReportGenerator* interface and provides the functionality to create results in HTML format.

Plaggie also uses a configuration file, where its settings are stored. There are a large number of settings that can be set, however some settings are more important that others, which we will discuss later on. In addition, the original authors of Plaggie designed it for future extensibility, their design allows us to write our own classes and replace core functionality. For example, if we want to use a different lexer and parser to return tokens we can do that by creating a new class that implements the *CodeTokenizer* interface. Next if want to use a string comparison algorithm other than the greedy string tilling algorithm then we do that by creating another class that implements the *TokenSimilarityChecker* interface. Likewise we can also change the way in which reports are generated. Katuscakova [Kat] has taken advantage of this extensibility property and extended Plaggie. The differences between the two versions is now presented:

1. Change in Tokenization:
   Plaggie uses the Jlex and Cup lexer and parser to generate tokens from programs.

In Plaggie2, Katuscakova [Kat] has replaced this with another implementation of *CodeTokenizer* interface. In this new implementation she has used an open source parser called Java Parser 1.5 [1]. Java parser returns the abstract syntax tree (AST) of a program, we can then transverse this AST and identify the tokens. We also found that using Java Parser is much more user friendly than using a CUP parser, as Java Parser gives us the AST of the program as a Java object. We can then use Java programming language to traverse and access the different parts of the AST. This is in stark contrast to the CUP parser, which involves complex lexing and parsing grammer and requires a steeper learning curve. Finally we believe that the major contribution of using Java Parser is that it is maintained by the open source community. The latest version even supports Java 1.7. Therefore it gives us the possibility to further extend Plaggie2 to support Java 1.7.

2. Added defense against certain types of attacks:
   Furthermore Katuscakova [Kat] has added preventative measures to certain kinds of attacks that can be found in assignments in small classes. The measures can be enabled by using an XML configuration file that we have to put together with the code we are checking. These measures are presented next:

   (a) When Plaggie creates tokens for a program, it removes all methods names. Therefore at times it also flags methods that are very different to be the same. Plaggie2 can avoid this by also considering the name of the method while looking for a match. This can help reduce false positives, where Sections of code are flagged incorrectly.

   (b) Sometimes in a programming assignment, some methods can be provided by the instructor for all students. Plaggie2 can be set to ignore some methods from the comparison.

   (c) Students may use a return call in a void function to fool the plagiarism detection tool, therefore Plaggie2 can be set to ignore the return statement in a void method.

   (d) Students may also add an empty if else block in order to trick the tool, Plaggie2 can be set to ignore an empty else.

   (e) Students may substitute for and while loops, Plaggie2 can be set to consider them both as the same.

From the different preventative measures Katuscakova[Kat] added, measure A looks the most promising as it has the possibility of reducing false positives, but what if 2 students share methods and rename their methods. Next, Plaggie already has the option to ignore certain files from the comparison process, so we believe measure B wont affect the outcome by much. Finally measure C, D and E do help improve the similarity when the cases are present but we cannot be certain that those attacks will exist in most programs. In addition her paper is in Slovakian, so we could not understand the significance of her tests and evaluations.

[1]https://code.google.com/p/javaparser/

Version	Description	Plaggie	Plaggie2
0	Original version	100	100
1	Translated comments and minor layout changes	100	100
2	Move 25% of the methods	82.1	82.1
3	Move 50% of the methods	94.3	94.3
4	Move 100% of the methods	81.6	81.6
5	Move 50% of the class attributes	94.3	94.3
6	Move 100% of all class attributes	100	100
7	Refactored GUI code	86.9	86.9
8	Changed Imports	100	100
9	Changed GUI text and colors	100	100
10	Rename all classes	100	100
11	Rename all variables	100	100
12	Rename all methods	100	100
13	Eclipse - Clean up function - use 'this' qualifier for field and method access	100	100
14	Elipse - Clean up function: - use modifier final where possible - use blocks for if/while/for/do - use parentheses around conditions	100	100
15	Eclipse - Generate getters and setters	69.7	69.7

Table 4.1: Semantics preserving modifications applied to version 0 and similarity results for Plaggie and Plaggie2

Next in order to compare the performance between Plaggie and Plaggie2, we performed a sensitivity analysis on their standalone versions. The sensitivity test is taken from the paper [HRvV11], where 17 different re-factoring are made to a single program. Most of the modifications are performed manually, but some are performed by using the re-factoring functionality provided by the eclipse IDE. We have taken 14 re-factoring from the original test, and skipped 3 Eclipse IDE based re-factoring as those were less likely to be used by students. The modifications are ones that students are most likely to use to hide plagiarism. We then run the plagiarism detection tool on the original and modified versions of the program. By performing this test we find the sensitivity of plagiarism detection tool towards a particular modification. Version 0 is the original file created by us, that tries to represent the various constructs, keywords and semantics a student is likely to use in an programming assignment. The details of the rest of the re-factoring and the similarity values returned by both tools are provided in Table 4.1.

After running both Plaggie and Plaggie2 on our re-factored set, we get the results as shown in Figure 4.2 Overall we saw that both tools returned the same similarity values. We will discuss the significance of these results in relation to the result of other detection tools in Section 5.1

Finally Plaggie2 uses JavaParser[2] which is more easier to use than the parser used in Plaggie, which requires us to utilize parser grammar in order to modify it. In addition the parser being used in Plaggie2 is maintained by the open source community, which

[2]https://code.google.com/p/javaparser/

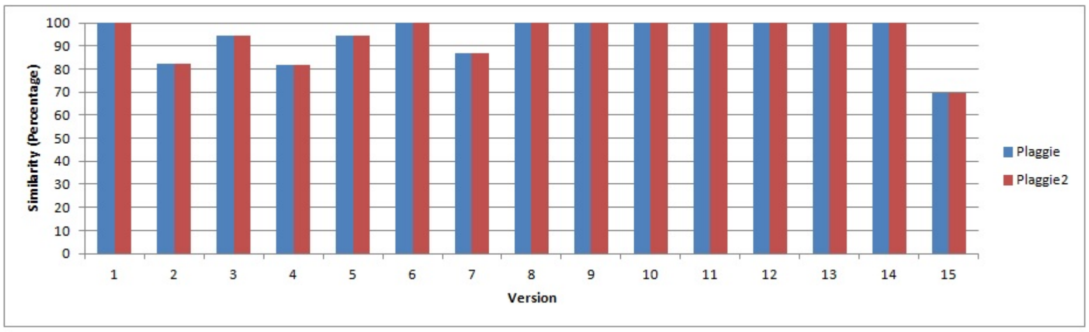


Figure 4.2: Plaggie vs Plaggie2 Sensitivity Test Results

gives us the possibility to extend Plaggie2 to support Java 1.7 in the future. Therefore we will work with the codebase of Plaggie2 in order to develop our add-on for Web-CAT. However we could not understand the significance of her tests and evaluation therefore we wont use her defense against certain types of attack feature. We will also evaluate this decision in the Section 5.2.

4.2 Changes to Web-CAT

In order to integrate Web-CAT and Plaggie2 we examined the architecture of Web-CAT. Web-CAT provides functionality to its users via various subsystems that work together, where each subsystem is responsible for providing a set of specific features. Now we will discuss the main subsystems and the features they provide.

- Core Subsystem:
  This subsystem provides common support for all the other subsystems. It defines the basic structure for the pages and also controls the look and feel of Web-CAT. In addition, it provides the code data model to all other subsystems. The code data model consists of entities like users (including instructors, students), courses and assignments. It also supports many to many relationships, which allows us to determine information like which users are instructors for which courses or which users are enrolled in each course. The core subsystem is also responsible for controlling authorisation in Web-CAT, for example the actions available to each user will be determined by the users role in relation to the course. A user who is a student in a course will have different access rights than a user who is a instructor in that course. These access rights can be further controlled by the administrator, by adding or removing the access rights of certain users.

- Grader Subsystem:
  This subsystem implements TDD in Web-CAT, it provides Web-CAT's assignments submission and grading functionality. This subsystem allows us to manage the electronically collected assignments and to grade programming assignments. Next it provides instructors with the functionality to create and edit programming assignments. Instructors may select various assignment parameters like due date, opening

date for submission, limit on number of submissions among many others. This subsystem also allows students to submit their assignments. Additionally it allows the instructors to create actions to compile, execute and evaluate the students code. Instructors can do this by using pre-built scripts provided by Web-CAT or they can choose to write their own scripts. Web-CAT comes pre-installed with scripts to evaluate Java and C++ assignments. Also instructors can add their own comments, deduct points to any line at the program and provide overall comments on the entire assignment.

- Admin Subsystem:
  This subsystem provides authorised users with the ability to search, view and update the database entities for all the subsystems. In addition this subsystem also allows authorized users to update and install new subsystems.

We decided to set up our add-on on the grader subsystem, as this subsystem controls the gathering and grading of assignments. After an instructor creates an assignment on Web-CAT and sets the rules for submission and execution, S/he will publish the assignment for all students to see. After students log in to Web-CAT they will find this call for submission and submit their assignments, Students work will then be graded by the Grader subsystem depending upon rules defined by the instructor. The default behavior of Web-CAT is to unzip the students submission, execute it, grade it and then delete it. However, Plaggie2 requires that all students submissions are in single folder. Therefore we modified the default behavior and now Web-CAT doesn't delete the submissions, instead for every assignment its submissions are kept a single location. The submissions in this location will be used by Plaggie2 to check for plagiarism.

The use case diagram in Figure 4.3 illustrates the process. At any time during the call for assignments, instructors are able to view all the submissions so far for an assignment on the "View Submissions" page. We have added the Plagiarism detection add-on at the top of the "View Submissions" (Figure: 4.4) page, where there are also other existing options to download scores and regrade all the latest submissions for the assignment. The link to open the add-on is "Run plagiarism check on this assignment".

When the instructor clicks this link the plagiarism check dialog box opens up(Figure: 4.5). This dialog box allows the instructor to set configuration parameters for Plaggie2 to run on this assignment. As mentioned in Section 4.1 there are many configuration parameters for Plaggie2 but we feel giving all those options to the user might overwhelm them.Therefore, we will only allow them to pass the more significant of the parameters. The parameters are discussed next:

1. Minimum number of tokens to match:
   As discussed in Section 3.1, before we can compare programs they must be converted to tokens. This parameter will set Plaggie2 to use its value as the minimum number of tokens to match for programs to be flagged as plagiarism. Program pairs will only be flagged if they each contain matching tokens of length equal or greater than this parameter. For example: if you set this parameter as 5, Plaggie will look for pairs of similar tokens of length 5, 6, 7 and so on. The allowed range of values for this parameters is from 1 to 100, while the default value is set as 9. Using the minimum match length of 9 is suggested by the authors of JPlag [PMP00] which uses a similar approach to Plaggie2.

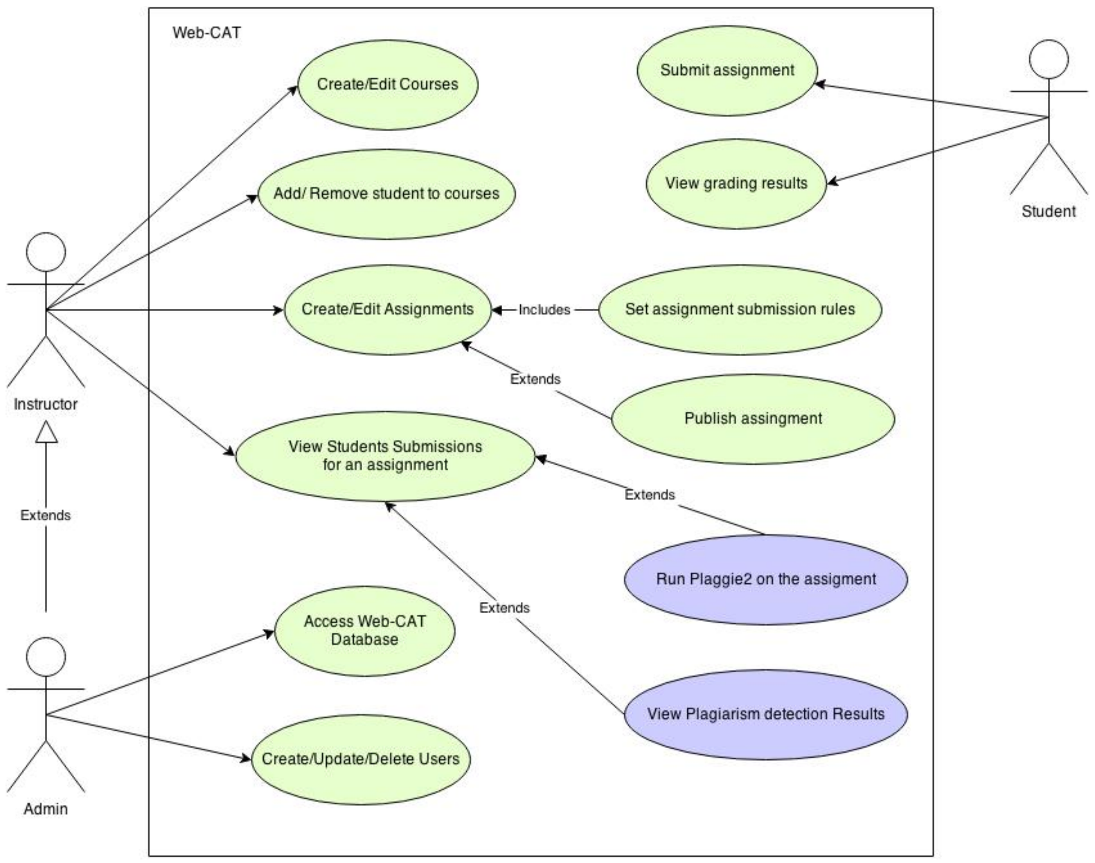
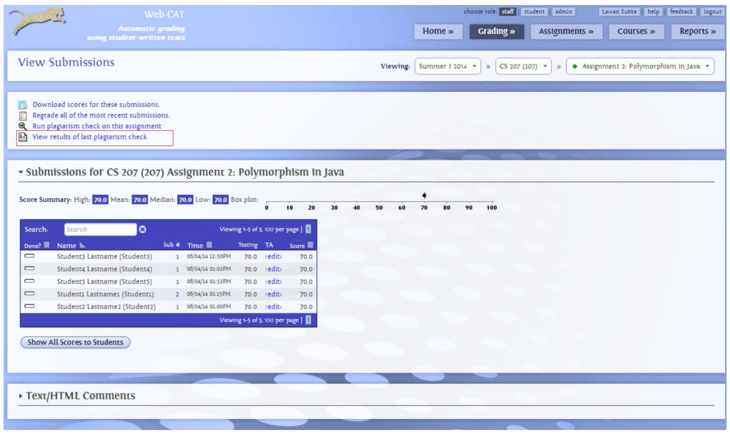

Web-CAT
Create/Edit Courses
Add/ Remove student to courses
Create/Edit Assignments
Includes
Set assignment submission rules
Extends
Publish assingment
View Students Submissions for an assignment
Extends
Run Plaggie2 on the assigment
Extends
View Plagiarism detection Results
Access Web-CAT Database
Create/Update/Delete Users
Submit assignment
View grading results
Student
Instructor
Extends
Admin

Figure 4.3: Web-CAT Users Use Case Diagram

2. Maximum number of detection results to report:
   When Plaggie2 compares the files, the number of detection results may get very large. For example if we set Plaggie2 to compare 10 files, the number of comparisons it has to perform is 90. This number can grow exponentially depending upon the number of submissions. Therefore we have set the default value for this parameter to 100 as it gave good results during our tests, so it will only return the top 100 results it has identified as plagiarism. The allowed ranges for this parameter is also from 1 to 1000.

3. Minimum Submission Similarity value between two files:
   Plaggie2 will compute a similarity percentage for every pair of files, some may have a 0% similarity while others may have some similarity and a value greater than 0%. This parameter will set Plaggie2 to return results for only those pairs whose similarity percentage exceeds this value. The allowed ranges for this parameter is from 50-100. We use this range as anything below 50% is very likely to be falsely identified cases. The default value is set to 50% as suggested by the authors of Jplag [PMP00] which uses a similar approach to Plaggie2.

4. Exclude these files from detection:
   In Web-CAT, students will normally write test code that will run on their assignments. This class will have no significance to the assignment, but will be used by


Web-CAT
Automatic grading using student-written tests
choose role: staff | student | admin
Lawan Subba | help | feedback | logout
Home »
Grading »
Assignments »
Courses »
Reports »

View Submissions

Viewing: Summer I 2014 » CS 207 (207) » Assignment 2: Polymorphism in Java

Download scores for these submissions.
Regrade all of the most recent submissions.
Run plagiarism check on this assignment
View results of last plagiarism check

Submissions for CS 207 (207) Assignment 2: Polymorphism in Java

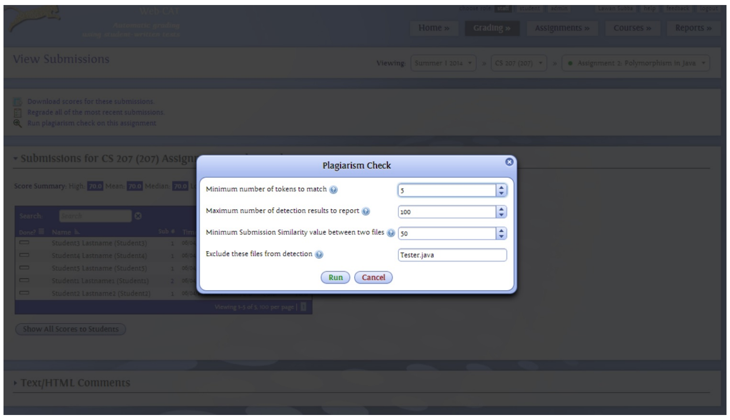
Score Summary: High: 70.0 Mean: 70.0 Median: 70.0 Low: 70.0 Box plot:

Search: Viewing 1-5 of 5, 100 per page | 1

Done?	Name	Sub #	Time	Testing	TA	Score
	Student3 Lastname (Student3)	1	06/04/14 12:50PM	70.0	‹edit›	70.0
	Student4 Lastname (Student4)	1	06/04/14 01:02PM	70.0	‹edit›	70.0
	Student5 Lastname (Student5)	1	06/04/14 01:52PM	70.0	‹edit›	70.0
	Student1 Lastname1 (Student1)	2	06/04/14 01:15PM	70.0	‹edit›	70.0
	Student2 Lastname2 (Student2)	1	06/04/14 01:00PM	70.0	‹edit›	70.0

Viewing 1-5 of 5, 100 per page | 1

Figure 4.4: Plagiarism detection add-on

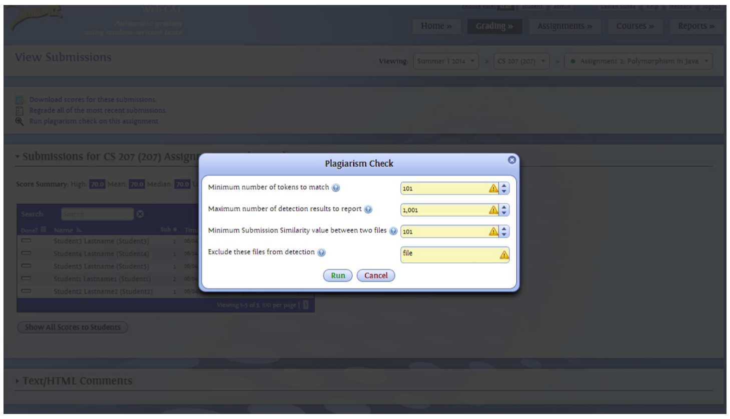


Minimum number of tokens to match	5
Maximum number of detection results to report	100
Minimum Submission Similarity value between two files	50
Exclude these files from detection	Tester.java

Figure 4.5: Plagiarism detection Dialog

Plaggie2 to check for similarity. This parameter will set Plaggie2 to ignore such files from being used in the comparison. We can define upto 3 java files to be ignored, by passing comma separated java file names to this parameter. The default value for this parameter is Tester.java as we found this file being commonly used as the tester file.

The instructor can use the default values or may choose to use a different set of parameters. However if the values go outside the allowed ranges, the dialog will show the error condition and will prevent the instructor from running the plagiarism tool(Figure:4.6). If the parameters fall within the valid ranges and the instructor chooses to click run on the dialog, the run button wont set Plaggie2 to execute on that assignment directly. Instead the request for plagiarism check will be added to the database record for that assignment, and a thread will be activated. The add-on will then disappear from the "View Submissions" page and a information label "This assignment is being processed" will appear. The newly created thread will then look for requests for plagiarism checks in the database and then execute all that it can find sequentially. This approach was chosen because running a plagiarism check is a time consuming process. If the number of submissions are large, it means that the instructor will have to wait a long time for the results to appear and cannot perform other tasks.


Plagiarism Check

Minimum number of tokens to match: 101

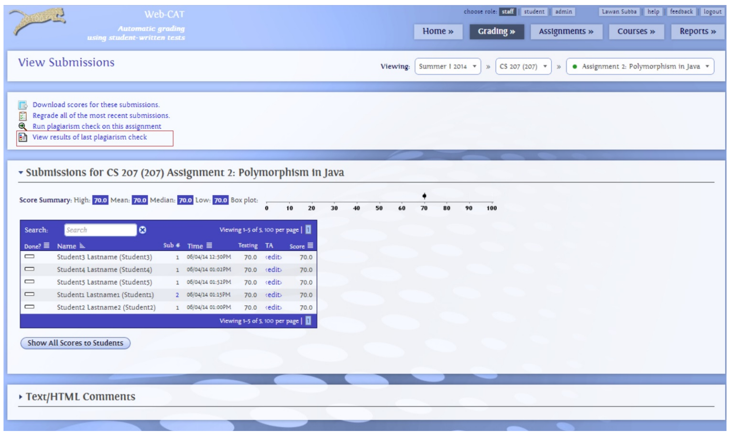
Maximum number of detection results to report: 1,001

Minimum Submission Similarity value between two files: 101

Exclude these files from detection: file

Figure 4.6: Plagiarism detection dialog failure case

This approach of initiating another thread to perform some of the work has been influenced by how Web-CAT evaluates the students submissions. When a student submits an assignment on Web-CAT, it initiates another thread that will be responsible for evaluating and grading the assignment so that the student does not have to wait. Then in our case, the thread will execute Plaggie2 on our assignment, generate results and then update the database records for that assignment. Finally the thread will go to sleep and wait for the next request. The instructor has to wait until the results are generated, which s/he can check this by refreshing the "View Submissions" page. Where the information label will disappear and option to view the results(Figure:4.7) will appear. When the user clicks the link "View results of last plagiarism check" the results created by Plaggie2 will open in another tab.


View Submissions

Viewing: Summer I 2014 » CS 207 (207) » Assignment 2: Polymorphism in Java

Download scores for these submissions.
Regrade all of the most recent submissions.
Run plagiarism check on this assignment
View results of last plagiarism check

Submissions for CS 207 (207) Assignment 2: Polymorphism in Java

Score Summary: High: 70.0 Mean: 70.0 Median: 70.0 Low: 70.0 Box plot:

Done?	Name	Sub #	Time	Testing	TA	Score
	Student3 Lastname (Student3)	1	06/04/14 12:50PM	70.0	‹edit›	70.0
	Student4 Lastname (Student4)	1	06/04/14 01:02PM	70.0	‹edit›	70.0
	Student5 Lastname (Student5)	1	06/04/14 01:52PM	70.0	‹edit›	70.0
	Student1 Lastname1 (Student1)	2	06/04/14 01:15PM	70.0	‹edit›	70.0
	Student2 Lastname2 (Student2)	1	06/04/14 01:00PM	70.0	‹edit›	70.0

Viewing 1-5 of 5, 100 per page | 1

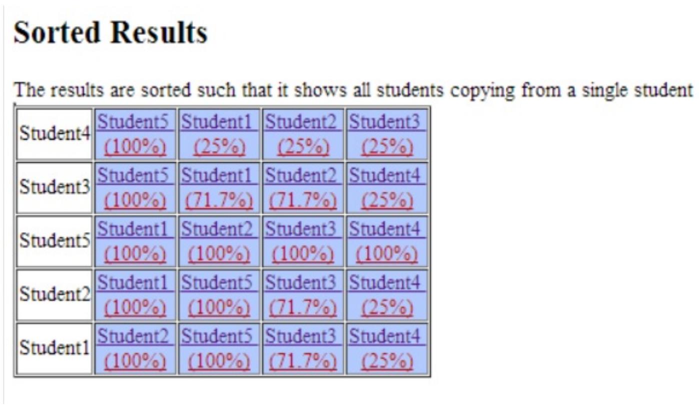
Figure 4.7: Plagiarism detection results

4.3 Changes to Plaggie2

We discussed in Section 4.1, how Plaggie2 compares submissions, generates similarity values and publishes reports and also how it uses a configuration file. In this Section we will discuss what changes were made to Plaggie2 to support the Web-CAT add-on.

Firstly, Plaggie2 is a standalone Java application, which requires user commands in order to execute. However as Web-CAT will call Plaggie2 as a background process, we have removed all these user dependent actions. Next as we have discussed in Section 4.1 how Plaggie2 uses a configuration file for settings and how the Plagiarism add-on in Web-CAT provides instructors the option to use their own parameters as configuration values. In order to support this feature of Web-CAT add-on, we made changes to Plaggie2 so that the add-on can pass the configuration parameters to Plaggie2. Next at the end of the operation, Plaggie2 has been configured to generate HTML pages to show the result of the plagiarism check. We made three changes to the HTML report generation process provided by the SimpleHtmlSubmissionReportGenerator class(Figure 4.1), these changes will help instructors find important information quickly than the default implementation.

1. Organising the results:
   In Section 2.2 we discussed that the HTML report file created by Plaggie, will print out the overall results of the check, where similarity between every students pairs is presented as a list. This way of displaying data is acceptable for small classes. But if the number of students submissions are large, then the instructor will find it difficult and time consuming to analyse the report. Therefore we have added a Section "Sorted Results" to the results page. This can been seen in Figure 4.8) and for every student this Section will show all other students who have similar submissions. In case of large number of submissions, this sorting makes the instructors job easier.



Sorted Results

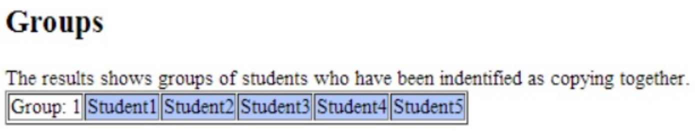
The results are sorted such that it shows all students copying from a single student

Student4	Student5 (100%)	Student1 (25%)	Student2 (25%)	Student3 (25%)
Student3	Student5 (100%)	Student1 (71.7%)	Student2 (71.7%)	Student4 (25%)
Student5	Student1 (100%)	Student2 (100%)	Student3 (100%)	Student4 (100%)
Student2	Student1 (100%)	Student5 (100%)	Student3 (71.7%)	Student4 (25%)
Student1	Student2 (100%)	Student5 (100%)	Student3 (71.7%)	Student4 (25%)

Figure 4.8: Sorted Results

2. Groups of cheaters:
   We added sorted results to organize the results in a more user friendly manner. However we discussed in Section 2.2 that none of the tools Jplag, MOSS or Plaggie solves the problem of finding groups of cheaters, for example: if Student A's submission has similarity with Student B's submission and Student B's submission has similarity with student C's submission. Then it is likely they have plagiarized the assignment together. In order to solve this problem we have introduced the "Groups of Cheaters" Section in the HTML report. This Section will show all the different groups of students(ref figure fig:Groups Of Cheaters) who have similarity between each other. This Section helps the instructor. In order to create the "Groups of Cheaters" section, we used the disjoint set data structure [Tar83]. A disjoint data

set data structure allows us to keep track of a set of elements in our case students by partitioning them into disjoint subsets. Then we can find the various groups of cheaters by putting each student in different partitions based on their program similarity.



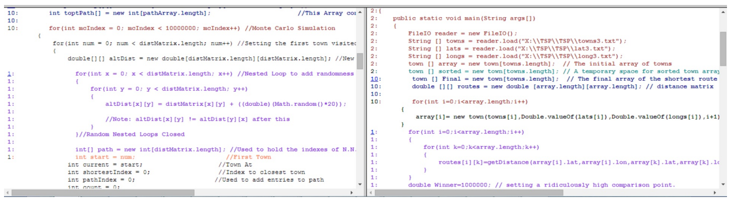
Groups

The results shows groups of students who have been indentified as copying together.

Group: 1	Student1	Student2	Student3	Student4	Student5

Figure 4.9: Groups Of Cheaters

3. Color coding the matches:
   In Section 2.2 we discussed how Plaggie provides detection results for pairs of submission that has been flagged as being similar. The similarity value link can be selected to go the detection results page, where we can see the values for SubmissionSimilarityValueA, SubmissionSimilarityValueB and SubmissionSimlarity. In addition, we can also see the Sections of the code and the tokens, the tool has marked as being similar. This page is helpful for the instructor, as it provides detailed information on the similarity between submissions. However if multiple Sections in the code have been marked as being similar, we found it visually difficult to find those matches. Therefore we have color coded the matches between the files, where the same color is used to mark Sections of code which are similar in both submissions. This can be seen in Figure fig:ColorCoding.



```
10:      int toptPath[] = new int[pathArray.length];                //This Array co
10:
10:      for(int mcIndex = 0; mcIndex < 10000000; mcIndex++) //Monte Carlo Simulation
       {
        for(int num = 0; num < distMatrix.length; num++) //Setting the first town visite
        {
          double[][] altDist = new double[distMatrix.length][distMatrix.length]; //New

1:           for(int x = 0; x < distMatrix.length; x++) //Nested Loop to add randomness
1:           {
1:             for(int y = 0; y < distMatrix.length; y++)
1:             {
1:               altDist[x][y] = distMatrix[x][y] + ((double)(Math.random()*20));
1:
1:               //Note: altDist[x][y] != altDist[y][x] after this
1:             }
1:           }//Random Nested Loops Closed
1:
1:           int[] path = new int[distMatrix.length]; //Used to hold the indexes of N.N.
1:           int start = num;                          //First Town
           int current = start;                       //Town At
           int shortestIndex = 0;                     //Index to closest town
           int pathIndex = 0;                        //Used to add entries to path
           int count = 0;
```

```
2:{
2:    public static void main(String args[])
2:    {
2:        FileIO reader = new FileIO();
2:        String [] towns = reader.load("X:\\TSP\\TSP\\towns3.txt");
2:        String [] lats = reader.load("X:\\TSP\\TSP\\lat3.txt");
2:        String [] longs = reader.load("X:\\TSP\\TSP\\long3.txt");
2:        town [] array = new town[towns.length];  // The initial array of towns
2:        town [] sorted = new town[towns.length]; // A temporary space for sorted town array
10:        town [] Final = new town[towns.length];  // The final array of the shortest route
10:        double [][] routes = new double [array.length][array.length]; // distance matrix
10:
10:        for(int i=0;i<array.length;i++)
      {
         array[i]= new town(towns[i],Double.valueOf(lats[i]),Double.valueOf(longs[i]),i+1)
      }
1:        for(int i=0;i<array.length;i++)
1:        {
1:            for(int k=0;k<array.length;k++)
1:            {
1:                routes[i][k]=getDistance(array[i].lat,array[i].lon,array[k].lat,array[k].lo
1:            }
1:        }
1:        double Winner=1000000; // setting a ridiculously high comparison point.
```

Figure 4.10: Color Coding Detection Results For Submission Pairs

4.4 Web-CAT Plagiarism detection Add-on Installation

Web-CAT supports extensibility by providing administrators the ability to update subsystems. The core functionality of subsystems are packaged inside java archive (JAR) files. Only the administrators of Web-CAT have access to the "Manage Subsystems" page, from where newer version of subsystems can be downloaded from the Web-CAT website and installed. When the administrator downloads a new version of a subsystem the JAR file of that subsystem is placed in the "Pending Updates" folder of Web-CAT, and when the application restarts the subsystem is installed on the server. By using this extensibility feature of Web-CAT, interested users can install the plagiarism detection add-on on their instance of Web-CAT. However they need to put the JAR file of our modified grader subsystem into their "Pending Updates" folder, update the database tables where we have made changes and restart their instance of Web-CAT. We assume here that only the administrator has folder and database access in the Web-CAT server.

4.5 Summary

In this chapter we discussed the design and implementation of Plaggie. Furthermore we provided arguments for selecting the basecode of Plaggie2 over Plaggie, for integrating with Web-CAT to create the plagiarism detection add-on. Then we discussed our changes to Web-CAT, describing in detail how the add-on works. Next we discussed our changes to Plaggie, where we focused on our changes to the HTML detection results. Finally we presented how our add-on could we installed on existing instances of Web-CAT.

Chapter 5

Evaluation

In this chapter we discuss how we have evaluated our implementation. Firstly we will apply sensitivity analysis on JPlag, MOSS and Plaggie2, this will help us determine how sensitive the tools are to different type of modifications. By performing the sensitivity analysis, we will get an idea of how the tools will perform for real submissions. Then we will run actual submissions on the plagiarism detection tools JPlag, MOSS and standalone version of Plaggie2. We perform this test to compare the detection results of the various tools with each other, but most importantly to identify the weaknesses of Plaggie2. Next we will perform GUI testing with Selenium on Web-CAT with the plagiarism detection add-on. We perform the GUI tests to validate that our add-on works and also that we have not introduced any faults in the existing system. Finally we perform usability tests with faculty members on Web-CAT with the plagiarism detection add-on. As plagiarism detection is a complex process with different metrics and values involved, it is important to know if the intended users find our add-on usable. In addition, the usability tests will also help us determine obstacles in the system that can be analysed to form future changes.

5.1 Sensitivity Analysis of Plaggie2, JPlag and MOSS

In Section 4.1 we used sensitivity tests to evaluate the performance of standalone versions of Plaggie2 and Plaggie on different kinds of modifications to programs. We observerd that the peformance of both tools were the same. We will perform the same sensitivity tests in order to compare the performance of JPlag, Plaggie2 and MOSS. For testing we used the same set of files from the earlier test, where version 0 represented the original file and versions 1 to 15 represent the modifications made to the original version. In addition, tokenization based tools Jplag and Plaggie2 require parameters, minimum match length and minimum submission similarity value. We used 9 for minimum match length and 50% for minimum submission similarity value as these values were proved to give optimal set of results by the study performed by Malpohl *et al* [PMP00] for JPlag. The results of our tests are shown in table 5.1 and graphically represented in the figure 5.1. We wont discuss the drop in similarity for MOSS in detail, as its implementation is hidden. However we will discuss the drop in similarity for JPlag and Plaggie2 as their implementation are nearly identical and known to us. Now we will discuss the significance of the sensitivity test results.

Version	Description	JPlag	Plaggie2	MOSS
0	Original version	100	100	100
1	Translated comments and minor layout changes	100	100	74
2	Move 25% of the methods	93.9	82.1	89
3	Move 50% of the methods	96.6	94.3	92
4	Move 100% of the methods	92	81.6	85
5	Move 50% of the class attributes	97.4	94.3	96
6	Move 100% of all class attributes	100	100	93
7	Refactored GUI code	92	86.9	92
8	Changed Imports	100	100	98
9	Changed GUI text and colors	100	100	99
10	Rename all classes	100	100	99
11	Rename all variables	100	100	99
12	Rename all methods	100	100	99
13	Eclipse - Clean up function - use 'this' qualifier for field and method access	100	100	89
14	Elipse - Clean up function: - use modifier final where possible - use blocks for if/while/for/do - use parentheses around conditions	100	100	54.5
15	Eclipse - Generate getters and setters	81.9	69.7	88.5

Table 5.1: Semantics preserving modifications applied to version 0 and similarity results for JPlag, Plaggie2 and MOSS

1. We found that the modifications 8, 9, 10, 11 and 12 have no significant effect on the results of all three tools. Therefore the tools are not sensitive towards this type of changes.

2. In version 1, where we only added comments and changed the layouts slightly, the results of tokenization and comparison tools JPlag and Plaggie2 are not affected.

3. In version 2, 3 and 4, where we moved 25%, 50% and 100% of the methods from their original position the results drop for all 3 tools. For JPlag and Plaggie2 the moving of the methods will cause the sequence of the tokens to be changed and if the number of tokens in the method is less than the minimum match length, then greedy string tilling algorithm will not find a match for that method.

4. In version 5 and 6, we moved 50% and 100% of all the class attributes from their original position. When we moved all of the class attributes, similarity values of JPlag and Plaggie2 are not affected. However when we moved half of the attributes we see a drop in similarity, this again is related to the shifting of tokens and the number of tokens that was shifted being less than the minimum match length.

5. In version 7, we shifted parts of the program which provides GUI functionality. The similarity values for JPlag and Plaggie2 has dropped and we found that the tokens have shifted and the number of shifted tokens is less than the minimum match

length. Therefore the greedy string tilling algorithm cannot find the match for the shifted section of code.

6. In version 13, we added the 'this' qualifier for all access of variables and methods. The similarity values for JPlag and Plaggie2 were unaffected, as both ignore the 'this' qualifier. However this modification is successful in fooling MOSS.

7. In version 14, we added blocks for if/while/for/do statements and used the final modifier for parameters. While JPlag and Plaggie2 were unaffected by these modifications, the similarity values for MOSS dropped significantly.

8. In version 15, we added getters and setters for all class attributes. In case of JPlag and Plaggie2, the getters and setters have introduced new tokens, which disrupts the original sequence of tokens. Therefore the similarity drops for Jplag and Plaggie2 as the greedy string tilling algorithm cannot find matches. However it worth noting that MOSS performs better than the other tools for this modification.

We found that tokenization based tools JPlag and Plaggie2 performed better than MOSS for most modifications. However the minimum match length of tokenization based tools plays a significant role in whether the modifications are detected or not. If the minimum match length is larger than the reordered code, then the modification will be successful in fooling the tokenization based tools.

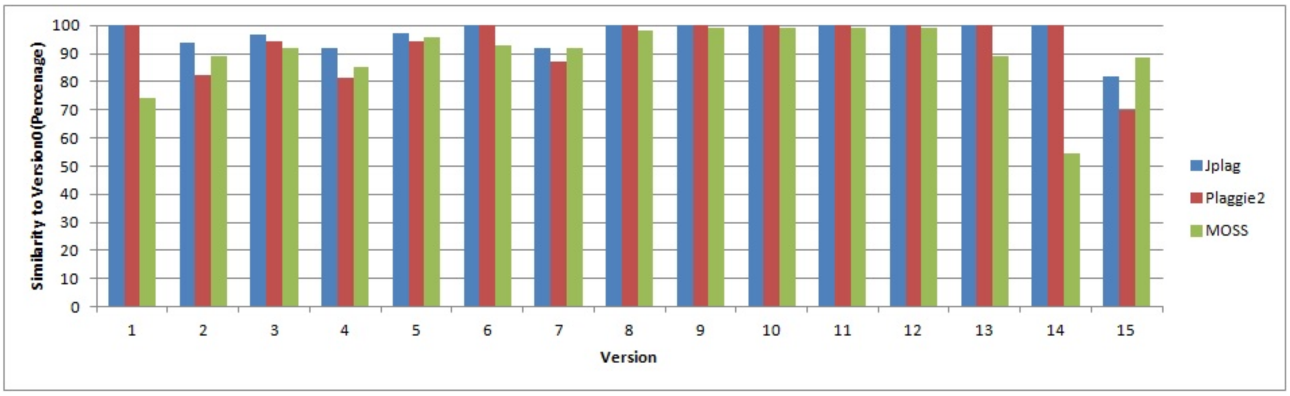


Figure 5.1: Sensitivity Test Results For JPlag, Plaggie2 and MOSS

5.2 Plagiarism Results for CS211

In order to truly analyse the performance of the standalone version of Plaggie2, we need to use real assignment submissions to test it and compare its result with the results of other plagiarism detection tools like Jplag and MOSS. For this We will use real assignments as they will have good distribution of programs and program similarities and are also more likely to expose faults or weaknesses of Plaggie2. Subsequently we will compare the detection results of Plaggie2 with results of other tools.

We do not expect all three tools to give the same similarity value for pairs, as all three use different algorithms. However, we expect the pairs that are flagged as similar by JPlag should also show up in the results of Plaggie2, as both use nearly similar algorithms. We also expect that any pairs that show up with high values in one tool must also show up with similar values in other two tools. If that is not the case then such situations are of interest to us, as they will help determine the weakness or differences of the tools involved

In order to perform our test, we asked lecturers in the Computer Science department in NUIM to provide us with their students assignment submissions. We looked at three modules, Introduction To Programming (CS141), Introduction To Object-Orientated Programming (CS142) and Algorithms & Data Structures 2 (CS211). After we looked at complexity and instructions for assignments in CS141 and CS142, we found that the instructions for those exercises were very specific. As both of these modules were introductory, the instructions for assignments were very detailed and included details on what classes to write, functions to create and fields to use. Therefore the assignment submissions for CS141 and CS142 could not be used for our study, as they would have very similar submissions. As for CS211, it is an algorithms and data structure module and students are expected to come up with very different approaches to their solutions. This module is comprised of students from a number of courses, including:

- MH203 - Bsc in Computer Science and Software Engineering
- MH140 - Bsc in Computer Science Software Engineering (Arts)
- MH214 - Bsc in Computational Thinking
- MHG54 - Higher Diploma In Information Technology
- MH211 - Bsc in Multimedia, Mobile Web Development

The course MHG54 is a postgraduate course whereas the rest are undergraduate courses. The students in CS211 were expected to submit a final project as part of their assessment. The final project was a traveling salesman problem, and required students to submit a Java algorithm to find the shortest route that visited all towns in a close loop, given the GPS co-ordinates of 80 towns in Ireland. The students were asked to submit the assignment in two groups and on two different dates, and were warned before hand that a plagiarism detection tool would be used on their submissions. On the first date students from MH214 and MHG54 submitted their assignments and on the second date students from MH211, MH203 and MH140 submitted their work. We were given permission by the lecturer to download the submissions from the NUIM moodle website [1]. 59 students submitted assignments from the MH214 and MHG54 courses as part of group 1, likewise 73 students from MH211, MH203 and MH140 courses submitted assignments as group 2. Two students have submitted the assignment in C# and Matlab programming language which cannot be checked by Plaggie2 so we did not use them in our test. All other submission were in Java and the details of the submissions are given in 5.2.

After downloading the submissions from Moodle, we found that there were 2 folders for each group and each folder contained all submitted files from students of that group. However, in most cases there were multiple files for an individual student. So we created folders for each student and placed the students files in that folder. The plagiarism

[1]https://2014.moodle.nuim.ie/

Group	Courses	Number of Submissions(Java)	Number of Submissions(Except Java)
1	MH214 and MHG54	59	1 (C#)
2	MH211, MH203 and MH140	73	1 (Matlab)
	Total	132	2

Table 5.2: CS211 Final Assignment Submission Overview

detection tools like Plaggie2, JPlag and MOSS can then use the folders to identify between files of each student.

JPlag and Plaggie2 are similar in their implementation. Both tools require parameters to operate, the most important ones are, minimum match length and minimum submission similarity which have been discussed in Chapter 2 and 4. In the paper by Malpohl *et al* [PMP00], they ran JPlag with multiple real world and artificial assignments. They concluded that for JPlag a minimum match length of 9 and minimum submission similarity value of 50% will give optimal set of results for real assignments. We will use these parameters for running tests on JPlag and Plaggie2. As the selected parameters have been tested and selected for JPlag, we will change the token length to 5 to evaluate how the results are affected for the two tools. In contrast MOSS does not require these parameters as it uses a winnowing process described in section 2.2.

After inspecting the students submissions we also found that some students wrote their own code to read information from files, others used the same Java file "FileIO.java" to read and write text file. The students used this file to read in the GPS co-ordinates from text files, which was part of the assignment. However using this file did not constitute as plagiarism, therefore we also set the tools to ignore code from this file in their comparisons. For Plaggie2, we set the name of the file to be ignored and it will not use this file for comparison. For JPlag and MOSS we used the option to set the base file and code from within that file will not be used in any of the comparisons. Next we found that some students who did not use a file reader were hard coding [2] the GPS co-ordinates in their code. The hard coding included around 80 lines of code, and would likely cause many submissions to be similar.

We then ran the submissions for both groups on the three detection tools. For JPlag and Plaggie we set the minimum match length of 9 and then 5 while keeping the submission similarity value of 50%. We did not have to set any such parameters for MOSS. After running the tool on our assignments, we realized that using the submission similarity value of 50% returned many genuine cases and cases where students had used similar approach to hard code the GPS co-ordinates. However increasing the submission similarity value to 75% returned results that looked like plagiarism. Therefore we used the submission similarity value of 75% in our tests and for any pair of students that one of the tools reports a similarity value of 75% or more, we will check the similarity values of that pair for the other tools too.

Now we will summarise the results after we ran the submissions for both groups in Plaggie2, JPlag and MOSS.

[2]http://en.wikipedia.org/wiki/Hard_coding

Pair	JPlag	Plaggie2	MOSS	
A	94.1	93.8	92	Plagiarism
B	90.7	98.8	69	Plagiarism
C	81.2	82	83.5	Similar
D	81.9	75.5	65	Plagiarism

Table 5.3: Similarity values for Group 1 with minimum match length 9 and minimum similarity value 75%

5.2.1 Results for Student Group 1

First we submitted the assignments of group 1 to the three tools, setting the minimum match length to 9 and submission similarity value of 50%. The results from this submission are shown in Table 5.3 and graphically represented in the Figure 5.2. 4 pairs of students were flagged as having similar programs. On inspecting the flagged sections of code, we found that pair A, B and D were indeed real instances of plagiarism. However, although pair C looked similar it could not be considered plagiarism as both were hard coding the GPS coordinates in their programs. For all 4 pairs, similarity values for Plaggie2 and JPlag look similar, but the similarity values for MOSS is different. We propose that this is the case as MOSS uses a different algorithm to JPlag and Plaggie2.

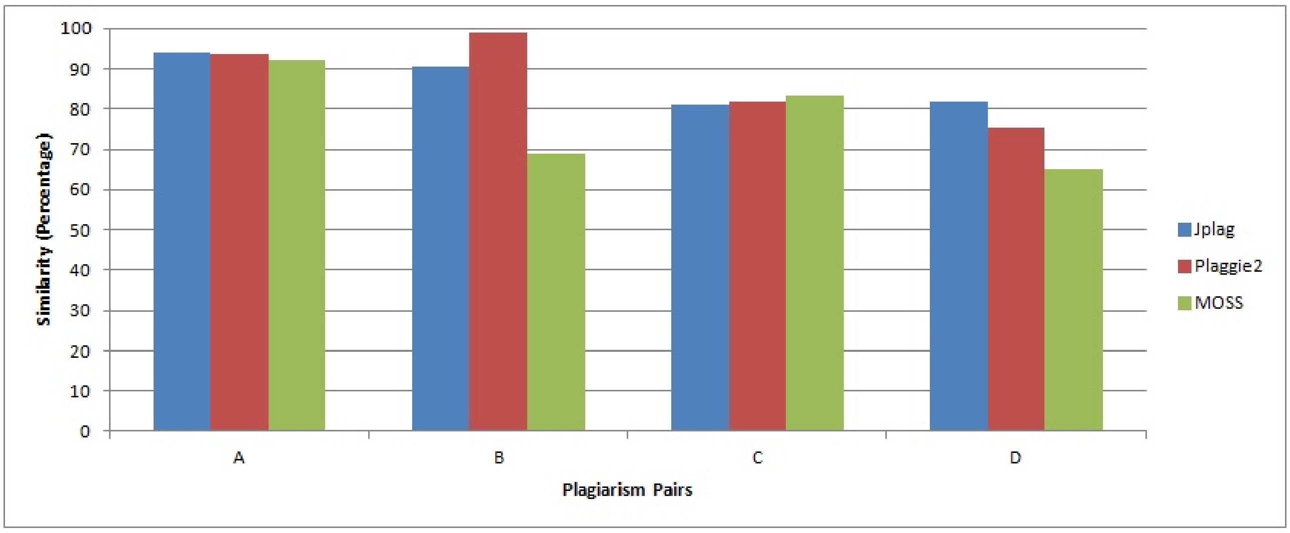


Figure 5.2: Similarity values for Group 1 with minimum match length 9 and minimum similarity value 75

Next we changed the minimum match length for JPlag and Plaggie2 to 5, and ran the submissions again on the 2 tools. However we did not have to rerun the submissions for MOSS as it does not need such parameters. We expect the previously identified pairs to appear for this run as well. The result from this submission are shown in Table 5.4 and graphically represented in Figure 5.3.

The pairs A, B, C and D do appear in the results for this run as well. In addition, more pairs have been flagged as being similar, but on closer inspection we could not identify any newer cases of plagiarism. Pairs E, G, H, I, J, K look interesting as those

Pair	JPlag	Plaggie2	MOSS	
A	96.3	96.8	92	Plagiarism
B	93.5	98.8	69	Plagiarism
C	90.4	85.4	83.5	Similar
D	87.1	89.1	65	Plagiarism
E	79.5	42.7	2.5	Similar
F	79.2	69.7	65.5	Similar
G	78.2	57.7	47.5	Similar
H	78	59.4	52	Similar
I	77.2	38.7	1	Similar
J	76.2	64.1	33.5	Similar
K	75.1	27.8	29.5	Similar
L	75	81.5	72	Similar

Table 5.4: Similarity values for Group 1 with minimum match length 5 and minimum similarity value 75%

pairs have very different similarity values for each tools.

1. For pairs F, G, H and J, there is a noticeable difference between the results for JPlag and Plaggie2. This difference is due to the manner in which Plaggie2 computes similarity value, which has been discussed in Section 4.1. Plaggie2 computes 2 values in the range between 0 and 1, which are called SimilarityA and SimilarityB. The two values represent the similarity of submission A with respect to submission B and similarity of submission B with respect to submissions A respectively. Finally the 2 values are multiplied to compute the similarity value of two files. Therefore Plaggie2 return high similarity values only for pairs that have high SimilarityA and SimilarityB values. For example, if the values for SimilarityA and SimilarityB are 0.80 and 0.80, when represented as percentage are 80% and 80%, the computed similarity value then is 0.64 which converted to percentage is 64%. However this implementation won't cause any similarity pairs to be skipped in the results for Plaggie2 as higher values of SimilarityA and SimilarityB can be seen in the results.

2. For pairs, E, I and K, there is significant difference between the results for all 3 tools. First we will discuss why there is such a difference in results for Plaggie2. The difference is again due to the manner in which Plaggie2 computes similarity. If Plaggie2 finds that for a pair of students, 2 pair of files are similar (as represented in table 5.5) it will then compute similarityA and similarityB by using formula 4.3, 4.4 and 4.5, where the tool takes the average value of the file similarities. Therefore if there is a similarity pair with lower similarity values, then it will result in a lower similarityA and similarityB value for the pair. Thus the total similarity value is also reduced considerably. However such pairs won't be skipped by Plaggie2, as the column maximum similarity value for the pair will be shown in the results page.

   Now we will discuss the significant difference in result for pairs E and I for MOSS compared to the other 2 tools. After inspecting the results of MOSS, we observed that JPlag and Plaggie2 had marked the sections of code as similar, where the

Student1	SimilarityA	Student2	SimilarityB
Student1_File1.java	0.817	Student2_File1.java	0.792
Student1_File2.java	0.318	Student2_File2.java	0.583
Average	0.5675	Average	0.6875

Table 5.5: Plaggie2 Similarity Values

Pair	JPlag	Plaggie2	MOSS	
M	94.5	92.9	91	Files too small
N	84.4	84.5	80.5	Plagiarism
O	90.2	92.1	69.5	Plagiarism
P	75.2	64.4	78.5	Files too small

Table 5.6: Similarity values for Group 2 with minimum match length 9 and minimum similarity value 75%

2 students had hard coded the GPS co-ordinates in their code. But MOSS did not flag this section as being similar, as the 2 students used different a number of parameters in the constructor to create the GPS object.

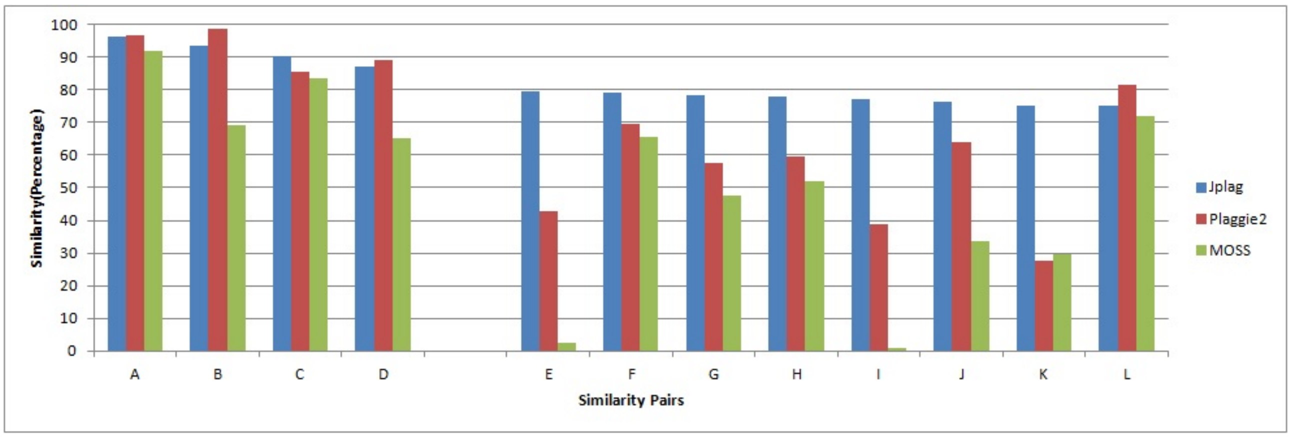


Figure 5.3: Similarity values for Group 1 with minimum match length 5 and minimum similarity value 75

5.2.2 Results for Student Group 2

We then submitted the assignments of group 2 to the three tools, setting the minimum match length to 9 and submission similarity value to 75%. The results from this submission are shown in Table 5.6 and graphically represented in the Figure 5.4. Four pairs of students were flagged as having similar programs.

After inspecting the results, we found that for pairs M and P, the files submitted were very small and students had submitted incomplete assignments. Therefore they could not be considered as plagiarism. However, pairs N and O were identified as plagiarism. We found no significant difference between the similarity values in the results to be discussed.

Pair	JPlag	Plaggie2	MOSS	
M	90.9	92.9	91	Files too small
N	92.2	93.2	80.5	Plagiarism
O	90.2	92.1	69.5	Plagiarism
P	79.5	72.4	78.5	Files too small

Table 5.7: Similarity values for Group 2 with minimum match length 5 and minimum similarity value 75%

As for group 1, we changed the minimum match length for JPlag and Plaggie2 to 5, and ran the submissions on the tools again. The results are shown in table 5.7 and graphically represented in the figure 5.5.

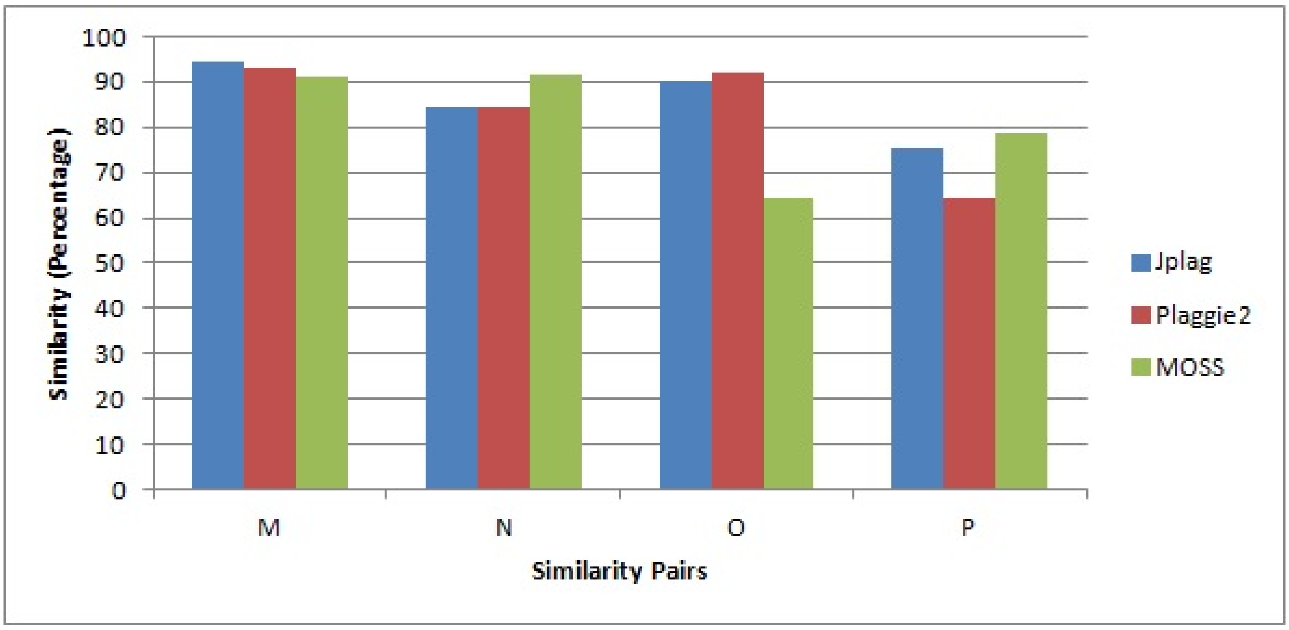


Figure 5.4: Similarity values for Group 2 with minimum match length 9 and minimum similarity value 75

The tools again showed the very same 4 pairs as having similar programs. Here too we did not find any significant differences that needed to be discussed.

5.2.3 Results for Students of Group 1 and Group 2

After running the three tools on submissions of Group 1 and Group 2, we identified 5 pairs involving 10 students as being involved in plagiarism. However we also suspected that there might be cases where students within these groups may also have copied work from each other. Therefore we combined all the submissions into a single folder and ran the combined submissions for the three tools. We followed the previous approach of using token length 9 and 5. Then we analysed the results and found no such instances where students had copied between the groups. Also we found no substantial differences in the results that should be further investigated.

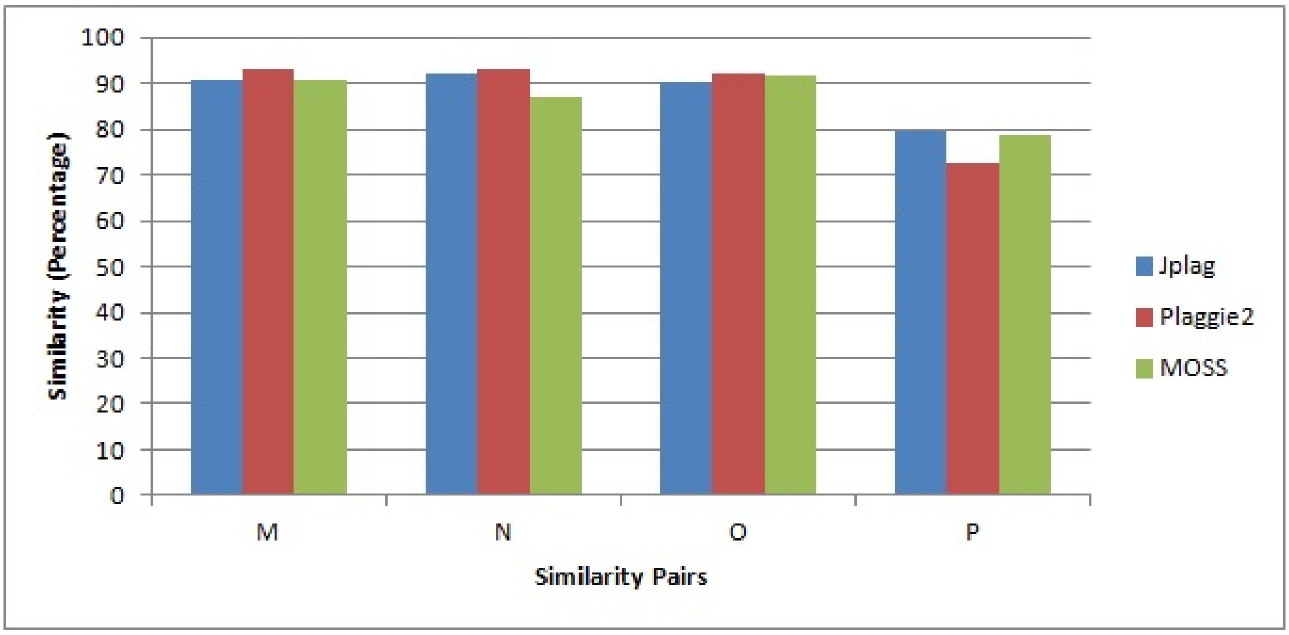


Figure 5.5: Similarity values for Group 2 with minimum match length 5 and minimum similarity value 75

5.2.4 Plagiarism Inquiry

We then notified the lecturer of this module about our findings regarding the students involved in plagiarism, and the lecturer initiated a formal inquiry where the identified students were asked to state why their submissions had been flagged. From the 5 pairs identified, within pair A one of the students admitted to copying from his friend and the student was given no marks for this assignment while the other student involved was not penalized. For pairs B, D and N the students admitted to having worked together, but stated that they had understood each part of the program and could defend it, therefore none of the students were penalized. Finally in last pair O, the students admitted to working together but the lecturer deemed that the code they had shared did not contribute significantly to the assignment so they were not penalised either.

5.2.5 Groups of Cheaters

In Section 4.3 we discussed how we had extended Plaggie2 to also report groups of students who had similar programs and that it would help us detect additional students who were involved in plagiarism. Now we will discuss the effectiveness of adding this feature to Plaggie2. We gathered the relevant similarity information from the results pages of both the submission groups 1 and 2. We used the results of our Plaggie2 run with the minimum match length of 5 as it has more detection pairs. The results are displayed in the Table 5.8.

We see that for submission group 1, there are 4 similarity groups, similarity group 2 has the maximum number of students while similarity groups 1, 3 and 4 have between 2 and 3 students. The similarity pairs we identified earlier are also listed, as they can be used to find other students copying from them. As similarity group 2 has a large number

Submission Group	Similarity Group	Number of Students	All identified pairs	Identified plagiarism pairs
1	1	2	K	
1	2	34	A, B, C, D, E, F, G, H, I, J, L	A, B, D
1	3	3		
1	4	2		
2	1	2	N	N
2	2	2		
2	3	44	M, O, P	O

Table 5.8: Similarity Groups for Submission Groups 1 and 2

of students, we cannot retrieve any information from it. However, as similarity groups 1, 3 and 4 are small we looked into the submissions of the students in those groups, as they may contain instances of plagiarism that we might have missed. Similarity group 1 contains the similarity pair K which we have already identified as being similar but not as plagiarism. Next we looked at similarity groups 3 and 4, and in both groups we saw the programs as being similar but they cannot be called plagiarism.

For submission group 2, we have 3 similarity groups, similarity group 3 has 44 students which is the maximum for that submission group. Similarity groups 1 and 2 contain 2 students each. Here too we investigated the smaller similarity groups 1 and 2 for instances of plagiarism. Similarity group 1 contains the pair N, which has already been identified as plagiarism while similarity group 2 had similar code but cannot be called plagiarism.

Therefore our added feature of showing similarity groups was not very helpful for the results of CS211 final assignment. For both submission groups, one similarity group contains a large number of students as many students are hard coding the GPS coordinates into their code. This has caused many of the students to have similar code and our grouping has put most of them in a single similarity group.

5.2.6 Summary

In this section we compared the results of Plaggie2 against the tools JPlag and MOSS by using real world assignment submissions. We found that for very similar programs, JPlag and Plaggie2 results are very similar, but in most cases the results of MOSS is different from the two as it uses a different algorithm. Next we also realised that for some pairs, Plaggie2 reports similarity values that are lower than that of JPlag. It is not that Plaggie2 is missing these pairs, it is due to the manner in which Plaggie2 computes similarity. Therefore when we inspect the results of Plaggie2, we cannot just rely on the similarity values, we must also look at the values SimilarityA, SimilarityB and Maximum file similarity. We subsequently evaluated our added feature of showing groups of students with similar submissions, and found that it did not provide any additional information for assignments of CS211. In addition, our results were able to capture 5 pairs of students that were positively identified as being involved in plagiarism. Finally we can conclude that Plaggie2 has acceptable detection performance and can be used to provide plagiarism detection functionality on Web-CAT.

5.3 GUI Testing on Web-CAT with Plagiarism Add-on

In section 4.2 we discussed how we were able to extend Web-CAT with Plaggie2 and provide a plagiarism detection add-on. As Web-CAT is a GUI based web application, it is important that we perform GUI tests after our modifications. In this section we will discuss how we tested Web-CAT with plagiarism detection add-on using GUI testing tool Selenium discussed in section 3.5.

Web-CAT is a complex web application which consists of large number of GUI elements and operations that can be performed. However, we will not test every operation that can be performed on Web-CAT as that would take enormous amount of time and effort. Instead we will focus our GUI tests on operations and GUI elements that we have added or changed as part of the modifications to Web-CAT. Now we will discuss the significance and results of the GUI tests performed on the modified version of Web-CAT

5.3.1 Web-CAT Assignment Submission

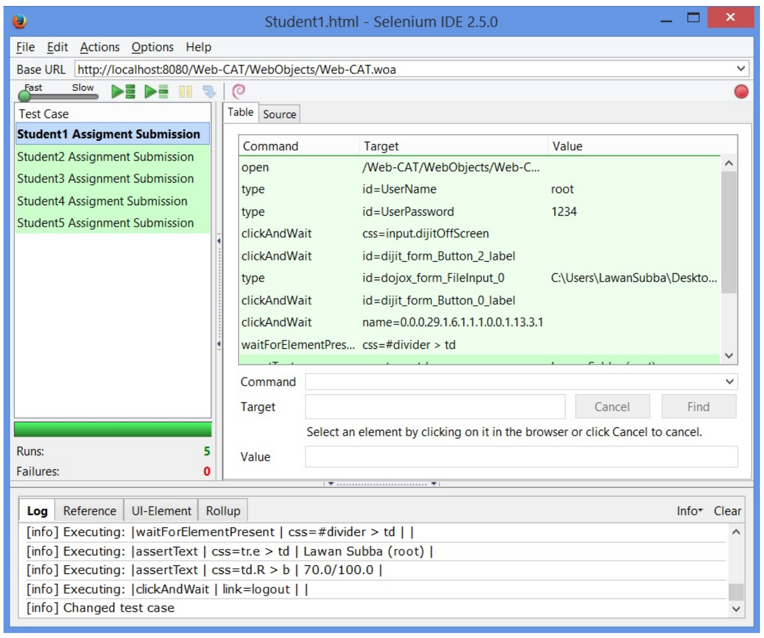
We discussed in Section {sec:Changes to Web-CAT how we modified the assignment submission process in Web-CAT, such that instead of deleting all submissions after evaluation. Web-CAT will keep backup of the submissions for every assignment in a single folder. With this test scenario we are testing that no bugs have been introduced into the assignment submission process and students can submit assignments to Web-CAT and those assignments will be graded.

In order to generate a selenium test case, first we created a dummy assignment called "Polymorphism in Java" and used the Selenium IDE to record a single student logging into Web-CAT and submitting an assignment. Next we used that test case to create 4 more test case to simulate 5 students logging into Web-CAT, submitting their assignments and being graded. Then we ran the tests on Selenium IDE, the results of this test scenario are displayed in Figure 5.6

From results panel of Selenium IDE, we can see that all 5 of the test cases have succeeded. Each test case represents a student successfully submitting their submission and receiving grades. This means our test was able to validate that the assignment submission process does work.

5.3.2 Web-CAT Assignment Submission and Plagiarism Detection

Earlier we tested the scenario that students could successfully submit assignments. In this test scenario, we will test that the plagiarism detection add-on can be used by an instructor to check for plagiarism and detection results will be returned. In addition, we will use the default parameters of the plagiarism detection add-on to run the check. Like in our earlier scenario, we generated a test case by using the Selenium IDE to record


Student1.html - Selenium IDE 2.5.0

Base URL: http://localhost:8080/Web-CAT/WebObjects/Web-CAT.woa

Test Case
Student1 Assigment Submission
Student2 Assignment Submission
Student3 Assignment Submission
Student4 Assigment Submission
Student5 Assignment Submission

Runs: 5
Failures: 0

Command	Target	Value
open	/Web-CAT/WebObjects/Web-C...	
type	id=UserName	root
type	id=UserPassword	1234
clickAndWait	css=input.dijitOffScreen	
clickAndWait	id=dijit_form_Button_2_label	
type	id=dojox_form_FileInput_0	C:\Users\LawanSubba\Deskto...
clickAndWait	id=dijit_form_Button_0_label	
clickAndWait	name=0.0.0.29.1.6.1.1.1.0.0.1.13.3.1	
waitForElementPres...	css=#divider > td	

Select an element by clicking on it in the browser or click Cancel to cancel.

```
[info] Executing: |waitForElementPresent | css=#divider > td | |
[info] Executing: |assertText | css=tr.e > td | Lawan Subba (root) |
[info] Executing: |assertText | css=td.R > b | 70.0/100.0 |
[info] Executing: |clickAndWait | link=logout | |
[info] Changed test case
```

Figure 5.6: Selenium Tests Overview For Web-CAT Assignment Submission

an instructor logging into Web-CAT and running the plagiarism detection add-on. Next we combined this test case with the student test cases of our earlier scenario. Therefore we have 6 test cases, which represents 5 students sequentially logging into Web-CAT and submitting assignments and finally an instructor logging in and using the plagiarism detection add-on to check for plagiarism in the students submissions. The results of the test scenario are displayed in Figure 5.7.

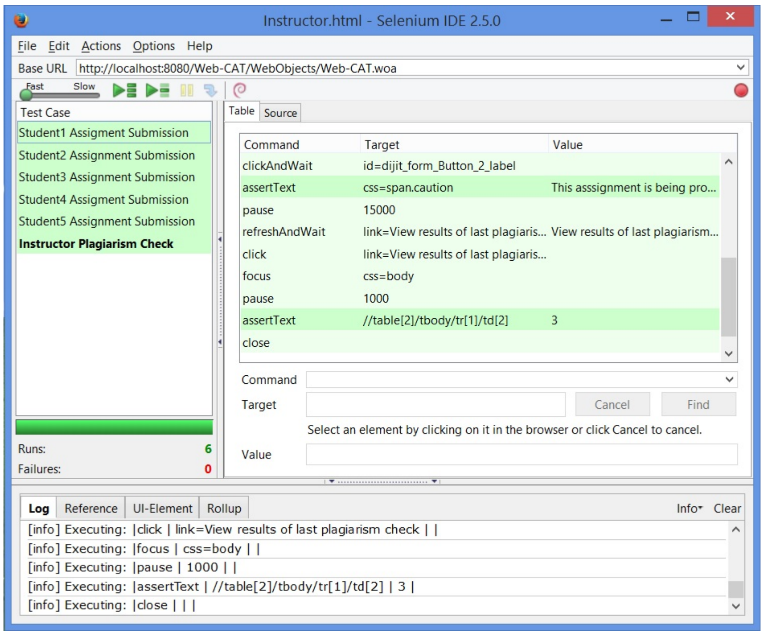
From results panel of Selenium IDE, we can see that all 6 of the test cases have succeeded. This means our test was able to validate that the plagiarism detection add-on works and also returns results.

5.3.3 Invalid Parameters For Plagiarism Detection Add-on

In Section 4.2, we discussed how we had set allowed ranges for parameters on the plagiarism detection add-on. If the instructor attempts to pass parameters outside of these allowed ranges, the add-on will not allow the instructor to run the plagiarism check. In this test scenario, we will test that invalid values are identified by the add-on and plagiarism check is prevented. We will use the instructor test case we created earlier in the test scenario "Web-CAT Assignment Submission and Plagiarism Detection" and add invalid values for the parameters instead of using the default parameters. The results of the test scenario are displayed in Figure 5.8. From the results panel of Selenium IDE, we can see that the test case has failed, which means our add-on has detected the invalid values and prevented the check.

5.3.4 Summary

We performed various Selenium based GUI tests on the modified version of Web-CAT. We were able to verify that assignment submission functionality of Web-CAT works correctly.


Instructor.html - Selenium IDE 2.5.0
File Edit Actions Options Help
Fast Slow
Table Source
Command
Target
Cancel Find
Value
Log Reference UI-Element Rollup Info Clear

Base URL http://localhost:8080/Web-CAT/WebObjects/Web-CAT.woa

Test Case
Student1 Assigment Submission
Student2 Assigment Submission
Student3 Assigment Submission
Student4 Assigment Submission
Student5 Assigment Submission
Instructor Plagiarism Check

Runs: 6
Failures: 0

Command	Target	Value
clickAndWait	id=dijit_form_Button_2_label	
assertText	css=span.caution	This assignment is being pro...
pause	15000	
refreshAndWait	link=View results of last plagiaris...	View results of last plagiarism...
click	link=View results of last plagiaris...	
focus	css=body	
pause	1000	
assertText	//table[2]/tbody/tr[1]/td[2]	3
close		

Select an element by clicking on it in the browser or click Cancel to cancel.

```
[info] Executing: |click | link=View results of last plagiarism check | |
[info] Executing: |focus | css=body | |
[info] Executing: |pause | 1000 | |
[info] Executing: |assertText | //table[2]/tbody/tr[1]/td[2] | 3 |
[info] Executing: |close | | |
```

Figure 5.7: Selenium Tests Overview For Web-CAT Assignment Submission and Plagiarism Detection

Next we also verified that the plagiarism detection add-on can detect invalid parameters and also works correctly. However we were only able to perform these tests on Mozilla Firefox as we ran out of time.

5.4 Usability Testing on Web-CAT with Plagiarism Add-on

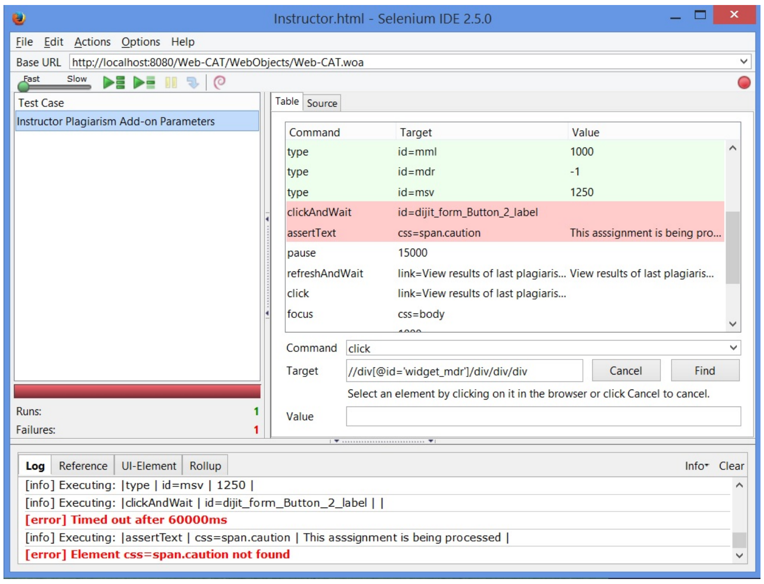
After successfully adding the Plagiarism detection add-on to Web-CAT, we want to find out the effectiveness of our add-on. The goals of this study are to:

1. Evaluate the overall effectiveness of our add-on for 2 user groups, the Lecturers and Teaching assistants in the Computer Science department at NUIM.
2. Identify the major obstacles that prevent the users from utilising the add-on.
3. At the end the users will also be given, the Plagiarism detection results of other tools and will be asked to compare the effectiveness of our detection results.

5.4.1 Research questions

In addition, this study will try to answer these questions:

1. How easily can the users find the Plagiarism Detection add-on?
2. The plagiarism detection tool requires some input from the users in order to execute. Do the users easily understand the purpose of the parameters they are passing?


Instructor.html - Selenium IDE 2.5.0
File Edit Actions Options Help
Fast Slow
Table Source
Log Reference UI-Element Rollup Info Clear

Base URL http://localhost:8080/Web-CAT/WebObjects/Web-CAT.woa

Test Case: Instructor Plagiarism Add-on Parameters

Command	Target	Value
type	id=mml	1000
type	id=mdr	-1
type	id=msv	1250
clickAndWait	id=dijit_form_Button_2_label	
assertText	css=span.caution	This asssignment is being pro...
pause	15000	
refreshAndWait	link=View results of last plagiaris...	View results of last plagiaris...
click	link=View results of last plagiaris...	
focus	css=body	

Command: click
Target: //div[@id='widget_mdr']/div/div/div
Select an element by clicking on it in the browser or click Cancel to cancel.
Value:

Runs: 1
Failures: 1

```
[info] Executing: |type | id=msv | 1250 |
[info] Executing: |clickAndWait | id=dijit_form_Button_2_label | |
[error] Timed out after 60000ms
[info] Executing: |assertText | css=span.caution | This asssignment is being processed |
[error] Element css=span.caution not found
```

Figure 5.8: Selenium Test Overview For Changed Plagiarism Detection Add-on Parameters

3. After executing a plagiarism detection run, can the users easily find the results?
4. After accessing the results of the plagiarism detection run, are the users able to identify the students involved in plagiarism?
5. From the results can the users identify the sections of code that the tool has identified as instances of plagiarism.
6. For any pair of students plagiarising, lets say Student A and Student B two values are computed. The values indicate the portion of Student B's code found in Student A' submission and the portion of Student A's code found in Student B's submission. Using these two values a similarity score is calculated for that pair. Can the users understand this calculation from the results?
7. In some cases Student B may copy code from Student A and Student C may copy code from student B. Therefore Student A, Student B and Student C fall under a single group of plagiarisers. Can the users identify the various groups of cheaters involved?
8. In some cases, the tool will not detect instances of Plagiarism. This can either occur as there are no plagiarism in the submissions, or the user might need to change the parameters passed to the tools. Can the user relate back to this requirement?
9. What kind of obstacles do the users face while trying to run the tool on the assignments put up on Web-CAT?

At the end of the session, we will have the following quantitative data:

1. Errors in finding the add-on and results: We will know if the user is able to find our add-on and its generated results or not.

2. Errors while setting configuration parameters: The parameters passed in order to run the tool will determine the outcome of the results. Here we will be able to identify what types of errors users make while passing configuration parameters to the tool.
3. Errors in the documentation: At various sections of the results and the add-on itself we have added documentation to support the users understanding of the tool. However, in some circumstances it might offer inadequate or contradicting information. We will be able to identify those cases.

We will also have qualitative data

1. Debriefing interviews and asking participants to fill questionnaires after the tests will allow us to find out what the users liked and disliked about our system. We will also be able to analyze the sections where they struggled or our system was not able to provide sufficient information which gives us meaningful insight and exactly what to change in the next iteration.

5.4.2 Location and setup

In order to conduct the testing sessions, I requested appointments with willing Lecturers and Teaching assistants. For these appointments a Windows laptop was used which had our modified version of Web-CAT with the Plagiarism add-on, MySQL and Google Chrome. On the laptop I asked them to run Web-CAT and perform the tasks. I was present in the same room as they tried to perform the various tasks set for them and I had a data sheet to take notes from the sessions.

5.4.3 Recruiting participants

For this survey we chose 2 user groups of Lecturers and Teaching Assistants from the Computer Science Department at NUIM. These are the user groups that are responsible for conducting and teaching the programming courses. They possess the most insight into programming assignments and student submissions. Also these are the user groups that will most likely use Web-CAT and its plagiarism detection add-on. The characteristics of the participants are outlined in the following table.

Characteristics	Number of participants
Participant Type	
Lecturers	5
Teaching Assistants	5
Total number of participants	10

Table 3: Participant data for the usability test

5.4.4 Methodology

In order to conduct the usability study, we chose the within-subject design where the two groups selected are asked to perform the same sequence of tasks. After that we collected quantitative data documenting our success and failure as well as qualitative data about users experience on the Web-CAT add-on. The breakdown of the testing session is given below.

1. Pre-test arrangements:

    - As all the participants will be working on the same machine for the environment was prepared prior to commencing any testing. We had to create simulated assignments and their submissions on Web-CAT, so that it would be able to provide answers to our research questions.

    - After each survey, plagiarism results with their files will be created and tables in the database will be filled up depending upon the testing involved. Therefore we had to clean up all the old files and data in the tables in order for next survey.

2. Discussion(10 Minutes):

    - We discussed whether the participant has previously had any experience using Web-CAT or any other automated assessment tool. We also discussed whether or not they had previously used any type of plagiarism detection tool.
    - We also discussed the significance of their participation in the survey and how their feedback would direct the direction of our research.
    - We outlined to the participants what kind of role we as moderators would be playing during the survey. We had to highlight that the emphasis of the survey was on the participants experience with the system and that it was not to judge the participants learning abilities or cognitive skills.
    - At the time of writing, NUIM is not actively using Web-CAT as an automated assessment tool. Therefore before they started on the tasks, we gave them a brief overview and walk-through of an instance of Web-CAT. However we did not focus on the Plagiarism Add-on as its usability is one of our research questions.

3. Tasks (30 minutes):
   Each of the participant was asked to perform the same sequence of tasks designed to answer the research questions of the usability study. The details of the task are detailed in table 5.9

4. Post-test debriefing (10 minutes)

    - In this section we asked the participants to fill in a questionnaire with broad questions to collect preference and other qualitative data. We also wanted to know whether they would consider using such a system.
    - We also followed up on the various parts of the test where the participant had struggled.

5.4.5 Tasks

The same sequence of tasks was used for each participant and the tasks were designed to allow us to explore our research questions set in Section 5.4.1. The scenario for the tasks was that the participant was using Web-CAT to support two different programming courses. Furthermore the university had recently added Plagiarism detection add-on to the Web-CAT. Now as a member of the faculty the participant had to use the add-on and find instances of plagiarism on the submissions. For every task that the users performs without our help we will get 1 marks otherwise we get 0.

The breakdown of the tasks are described in the Table 5.10.

5.4.6 Results

After the usability tests with participants, 3 of which were lecturers and 3 of which were teaching assistants, we realised that almost all of them were struggling in the same tasks. Therefore we stopped our tests as per principle of heuristic principle[3], there was no point in asking more people to take the test and struggle in the same tasks. We had to fix the identified problems in our system first before we could proceed more with the usability testing. The results of our usability tests with the tasks are summarised in Table 5.10.

We will now discuss the tasks which have a success rate of less that 50%, as those tasks represent significant failures in the design of our system.

1. For task 2 which is one of the more complex parts of the system, the success rate is 16%. This is where the user has to set the parameters for the add-on before it can start checking the assignment submissions for plagiarism. In some cases, participants were able to explain some of the parameters, but we considered a task as being successfully completed only if the participants could explain all the parameters, as even misunderstanding one parameter can significantly alter the results. We intervened here and explained the significance of the parameters to the participants when they got confused. After watching the participants suffer in this task, we realised that we need to provide even more extensive explanations next to the parameters.

2. For task 3, the success rate is 0%. After the participants run the add-on, they have to refresh the "View Submissions" page in order to check if the results have been returned from by the add-on. However all the participants could not anticipate this and kept staring at the page, and we had to intervene and help them. Here we realised that we need to set a message in the page asking the participant to refresh the page to see the result.

3. Task 6 is one of the more complex parts of our system, where the user is expected to view the results and interpret the significance of the various metrics. None of the participant could explain all the metrics successfully. We again intervened and had to explain the results. From this task, we realised that we had to add even more extensive explanations to the results as well.

[3]http://en.wikipedia.org/wiki/Heuristic

	Tasks	Success Criteria
1	After successfully logging in the participant has to find the plagiarism add-on	They go into the assignments detail page and are able to show it to the moderator.
2	The Plagiarism add-on requires some configuration parameters to execute. The parameters are set by default. But does the participant understand the significance of each parameter.	After starting the add-on, the participants is able to define the role of each parameter.
3	After running the plagiarism tool for a particular assignment. Can the participants detect in which part of the page the result is located?	The participant is able to show the results on the page to the moderator.
4	After opening the results detect the students involved in plagiarism.	The participant is able to tell the moderators the students the tool has flagged as plagiarizers.
5	For each pair of students codes flagged as plagiarism detect the portions the add-on has flagged as plagiarism.	The participant is able to show the sections flagged as plagiarism to the moderator.
6	Also for each pair of students codes flagged as plagiarism, the tool calculates a similarity percentage. There are 3 percentage values, similarity of code A to B, similarity of code B to A and total similarity between A and B. What do you understand by the 3 values	The participant has no difficulty explaining the meaning of the 3 values in relation to the code.
7	Detect groups of students involved in plagiarism.	The participant is able to show the moderator the groups of plagiarizers.
8	Why is that some results show no detection of plagiarism. What do you think can change this?	The add-on is only set to display detection results above 50%. But the participant should see that there are pairs detected below 50% from the statistics section of the results. After detecting this scenario, the participant should change the configuration parameters and rerun the tool.

Table 5.9: Tasks for participants in Usability Test

Task Number	Total Points	Lecturer	Teaching Assistant	Success(Percentage)
1	6	3	1	66%
2	6	1	0	16%
3	6	0	0	0%
4	6	3	3	100%
5	6	3	3	100%
6	6	0	0	0%
7	6	3	3	100%
8	6	1	1	33%

Table 5.10: Task Summary For Usability Test

4. For task 8, we wanted to see if the users would be able to realise that the configuration parameters they had used had resulted in no similarity pairs being detected. Some of the users(33%) were able to understand this, but most suffered in this task. We intervened here also and explained the situation when the participant could not understand it. From this task, we realised that we needed to add instructions in the results page asking the user to check the statistics section and his configuration parameters when the add-on returned no similarity pairs.

5.4.7 Questionnaire Summary

At the end of the usability testing, we gave each participants a questionnaire to fill up where we asked them 4 different qualitative questions related to our system. We will now present the questions and discuss the responses from that questionnaire.
Question 1: Overall the plagiarism add-on was easy to use.

Strongly Disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree
0	0	0	3	3

Why or why not?

1. It took a while to find some features but overall I found the system intuitive.
2. The system provides clear instructions and overall a good user interface
3. I found it easy to navigate and operate.

Question 2: I was able to understand the results of the add-on

Strongly Disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree
0	0	1	4	1

What part of the system was confusing for you?

1. The system was difficult at first but it took a few tries before I got used to the different data being reported.
2. I could not properly interpret the results, I think more help and demo files should be added.
3. Some of the features are difficult to grasp, for example: SimilarityA and SimilarityB. However it makes sense after the developers explanation.
4. It was very difficult to understand the similarity between A and B.

Question 3: I was satisfied in the manner in which the results are displayed

Strongly Disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree
0	0	0	4	2

What are your suggestions for the add-on.

1. It would be better to also color code the tokens in the results. And also add more sophisticated matching where it would also catch the moving of methods.

2. The system provides too much information in the results, maybe it is a better option to provide parts of the results only if the user wants to see it rather than see all in one go.

3. The user interface is not very friendly, much work needs to be done.

Question 4: I am willing to use such a system in the future for plagiarism detection

Strongly Disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree
0	0	0	2	4

Why or why not?

1. It would provide great assistance to correction, grading and giving feedback to students particularly in the early years of programming courses.

2. It is useful, and I may use it if the tool is improved to be more user friendly.

3. Yes I would use it as automatic plagiarism detection can greatly enhance staff productivity.

4. For large groups of students this would be useful. The teaching assistants cant catch it in the labs because the students sit apart.

5.4.8 Summary

After analysing the results and responses from our usability testing, we found that most participants found the add-on to be very useful. However due to the complex nature of plagiarism detection, most participants struggled to understand some of the features of the add-on. In order to help them in this problem, we need to add help files and more instructions in the different parts of the add-on. An even better option would be give a department wide presentation to the lecturers and teaching assistants on how to use the add-on. Finally the participants have also suggested additional features which can be considered for future work.

5.5 Summary

In this section, we performed sensitivity tests on the 3 tools JPlag, Plaggie2 and MOSS and the results of the sensitivity tests showed that JPlag and Plaggie2 performed better than MOSS for various kinds of modifications. Therefore the test vindicated our decision to use the codebase of Plaggie2 to extend Web-CAT and provide Plagiarism detection functionality. In addition, we also tested the 3 tools with actual student submissions and the results from that test showed that JPlag and Plaggie2 returned similar results for submissions that were very similar. However the results of MOSS varied from the results of JPlag and Plaggie2, we assume this occurs as it uses a different algorithm. Next we performed Selenium based GUI tests on the modified version of Web-CAT and the GUI tests verified the tool for our test scenarios. Finally we conducted usability tests on Web-CAT with plagiarism detection support with members of faculty of the Computer Science department at NUIM. Most of the participants in the study found the add-on helpful and also gave suggestions for future enhancements.

5.5.1 Threats to Validity

In this section, we will discuss the threats to validity of our results and limitations of our solution.

1. Testing of Web-CAT and Plaggie2 Code:
   Both Web-CAT and Plaggie2 contain large volume of code, which are distributed across multiple files. Testing all the code in the codebase of both applications would be tedious and require great effort. Therefore we have only tested parts of Web-CAT and Plaggie2 where we have made modifications. We assume that the original authors of these tools have conducted tests to verify and validate their applications.

2. Availability of a single assignment:
   In order to compare and evaluate the performance of plagiarism detection tools JPlag, Plaggie2 and MOSS, we only used submissions for one assignment in the CS211 course in Section 5.2. However, we realize that a single assignment is too small a dataset to expose all the weaknesses of a detection tool. Therefore we cannot say with certainty that Plaggie2 is infallible, however it showed promising results for our dataset when compared to JPlag and MOSS.

3. Web-CAT installation Conflict:
   In order to install our add-on to their instance of Web-CAT, users are provided a JAR file(Section 4.4) containing the modified grader subsystem. This new JAR file will override the existing grader subsystem in the current instance of Web-CAT. But if that user has already made modifications to his/her grader subsystem. Then modifications will be overridden by our subsystem. For such cases, our modified version of Web-CAT needs to be installed in a SVN repository so that users can merge their modifications with ours.

4. Time Constraints:
   The time taken by Plaggie2 to generate detection results is dependent upon a number of different factors such as number of files, number of submissions, number of tokens in the Java file and parameters used for execution. Therefore considering all these factors and testing Plaggie2 for time constraints would be very time consuming. In our test runs, we found that the maximum time it took for a very large set of assignments was 30 minutes. Therefore we assume that time constraints and performance wont be a problem on dedicated servers where Web-CAT is normally hosted. However we suspect that this assumption may not always hold true.

5. Vulnerabilities of Plaggie:

Chapter 6

Conclusions

We discussed the general problem of plagiarism and focused on software plagiarism and how it was detrimental to the student's learning process. We also discussed the benefits of using Web-CAT with its test driven development and how despite of its popularity it did not have plagiarism detection functionality. We started this study with a goal to create a plagiarism detection tool from scratch that we would later integrate with Web-CAT. However during literature review we found that an open source detection tool Plaggie already had features we were looking to develop in our own plagiarism detection tool. In addition, we also found a newer version of Plaggie called Plaggie2 and when we performed a comparison of the tools and found that Plaggie2 had more benefits in terms of future changes.

However in order to determine whether using the codebase of Plaggie2 was an option, we performed various performance based tests on it and also compared its results with other peer reviewed tools like JPlag ang MOSS. Although our test data was small consisting of submissions for only one assignment, Plaggie2 performed well and was able to identify all plagiarism pairs. In fact all the tools were able to identify programs when they were very similar, but the results varied when the files were not so similar. Therefore we see a requirement to perform more test on Plaggie2 with more data sets.

However, we were successful in integrating Plaggie2 with Web-CAT by implementing a plagiarism detection add-on on Web-CAT. We are the first to provide this functionality and we believe existing users of Web-CAT will find it beneficial.

Next we also realized that, automated plagiarism detection tools are not 100% accurate. They do require the user to analyze the results and check for validity. Furthermore using our add-on we can only identify pairs of similar programs, we wont know who copied from whom. Finally after running the tool on real assignment submissions of NUIM, we were able to verify that software plagiarism is not such a big problem for that particular course, as only a fraction of students were involved in plagiarism.

6.0.2 Future Work

At the moment Plaggie only supports Java 1.5, this seems to be sufficient for Web-CAT as it also only supports grading of Java 1.5 assignments. However we see great benefit in adding support to Java 1.7 in the standalone version of Plaggie. The latest version of JavaParser already supports Java 1.7. Therefore we only need to make changes to certain classes of Plaggie in order to make it support Java 1.7.

Although Jplag and Plaggie are very similar, they differ in the fact that JPlag has implemented different optimizations[PMP00] to the tokenization and greedy string tilling approach. As we ran out time for this thesis, we could not dedicate much time to those optimizations. However it would be interesting to see what kind of optimizations we could perform on Plaggie by taking concepts from Jplag.

Also as Web-CAT is an open source project, we put up a question in the Web-CAT forum asking if any of the Web-CAT users would be interested in using our plagiarism detection add-on. Two of the users have expressed interest, the first is a tutor from the University of Basque Country in Spain. He plans to use the add-on for an object-oriented programming class which contains 75 students. The next person to show interest is also a tutor from the Lynbrook High School in San Jose, California. In his institution, there are around 260 computer science students distributed across multiple courses and they have a high volume of Web-CAT usage. With further testing and evaluation we would like to release our add-on as an open source project so that people can use it with their Web-CAT instances.

Bibliography

[A$^+$05] Alex Aiken et al. Moss: A system for detecting software plagiarism. *University of California–Berkeley. See www. cs. berkeley. edu/aiken/moss. html*, 9, 2005.

[App] Apple. Webobjects overview. In *https://developer.apple.com*.

[App98] Andrew W Appel. *Modern compiler implementation in ML*. Cambridge university press, 1998.

[ASR06] Aleksi Ahtiainen, Sami Surakka, and Mikko Rahikainen. Plaggie: Gnu-licensed source code plagiarism detection engine for java exercises. In *Proceedings of the 6th Baltic Sea Conference on Computing Education Research: Koli Calling 2006*, Baltic Sea '06, pages 141–142, New York, NY, USA, 2006. ACM.

[Bec03] Kent Beck. *Test-driven development: by example*. Addison-Wesley Professional, 2003.

[CJ06] Georgina Cosma and MS Joy. Source-code plagiarism: A uk academic perspective. 2006.

[CML01] Fintan Culwin, Anna MacLeod, and Thomas Lancaster. Source code plagiarism in uk hecomputing schools, issues, attitudes and tools. 2001.

[DH05] Charlie Daly and Jane Horgan. A technique for detecting plagiarism in computer code. *The Computer Journal*, 48(6):662–666, 2005.

[Edw03] Stephen H Edwards. Improving student performance by evaluating how well students test their own programs. *Journal on Educational Resources in Computing (JERIC)*, 3(3):1, 2003.

[Edw14] Stephen H. Edwards. Web-cat. 2014.

[Gru06] Dick Grune. The software and text similarity tester sim, 2006.

[Hag06] Jurriaan Hage. Programmeerplagiaatdetectie met marble. technical report uu-cs-2006-062. In *Technical Report UU-CS-2006-062*, 2006.

[HRvV11] Jurriaan Hage, Peter Rademaker, and Nikè van Vugt. Plagiarism detection for java: A tool comparison. In *Computer Science Education Research Conference*, CSERC '11, pages 33–46, Open Univ., Heerlen, The Netherlands, The Netherlands, 2011. Open Universiteit, Heerlen.

[IAKS10] Petri Ihantola, Tuukka Ahoniemi, Ville Karavirta, and Otto Seppälä. Review of recent systems for automatic assessment of programming assignments. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, pages 86–93. ACM, 2010.

[JCYS11] Mike Joy, Georgina Cosma, JY Yau, and Jane Sinclair. Source code plagiarismâ"a student perspective. *Education, IEEE Transactions on*, 54(1):125–132, 2011.

[Jon01] Edward L Jones. Metrics based plagarism monitoring. In *Journal of Computing Sciences in Colleges*, volume 16, pages 253–261. Consortium for Computing Sciences in Colleges, 2001.

[Kat] Beata Katuscakova. Detekcia podobnosti zdrojovych kodov. Undergraduate thesis, Univerzita Pavla Jozefa Safarika Prirodovedecka Fakulta.

[LC04] Thomas Lancaster and Fintan Culwin. A comparison of source code plagiarism detection engines. *Computer Science Education*, 14(2):101–112, 2004.

[MG05] Stephen Marshall and Maryanne Garry. How well do students really understand plagiarism. In *Proceedings of the 22nd annual conference of the Australasian Society for Computers in Learning in Tertiary Education (ASCILITE)*, pages 457–467, 2005.

[Nie03] Jakob Nielsen. Usability 101: Introduction to usability, 2003.

[NUI] Nuimpolicy. 2014.

[Pla14] Plagiarism. http://www.oed.com/. In *Oxford English Dictionary*, 2014.

[PMP00] Lutz Prechelt, Guido Malpohl, and Michael Phlippsen. Jplag: Finding plagiarisms among a set of programs. 2000.

[RC08] Jeffrey Rubin and Dana Chisnell. *Handbook of usability testing: howto plan, design, and conduct effective tests*. John Wiley & Sons, 2008.

[Sha03] Anuj Shah. *Web-CAT: A Web-based Center for Automated Testing*. PhD thesis, Virginia Polytechnic Institute and State University, 2003.

[Tar83] Robert Endre Tarjan. *Data structures and network algorithms*, volume 14. SIAM, 1983.

[TD] Matt Morse Jean Ostrem Kelly Toshach Terry Donoghue, Katie McCormick. Webobjects developers guide.

[Wha90] Geoff Whale. Identification of program similarity in large populations. *The Computer Journal*, 33(2):140–146, 1990.

[Wis06] Michael Wise. Yap3: Improved detection of similarities in computer program and other texts.sigcseb: Sigcse bulletin (acm special interest group on computer science education). 2006.