Project Report
Copy Catch Project – Team 203

Hari Prasath Sivanandam, Anju James, Karan Sharma, Keerthi Pendyala

Submitted to Prof. Jose Annunziato
Department of CCIS

CS5500- Managing Software Development
Department of CCIS
Northeastern University

Introduction and Overview

Plagiarism is a major challenge faced by instructors in academic institutions. Academicians in computer science have perceived the necessity to check for plagiarism in the programming assignments submitted by students to assist the grading procedures. The chances of submissions being a copy or modified version from other submissions makes plagiarism check a necessity in academic institutions. The fact that, the process of checking for plagiarism among student submissions manually can be a time consuming practice, calls for an automated system to support the academicians. Hence the goal of the project was to automate the plagiarism check and facilitate the academicians with a tool or application to help detect plagiarism.

Plagiarism of source code can be performed in several ways. It includes moving methods around in the same file but to different locations of the file. A plagiarized file may also include similar fragments from other persons code but slightly altered by renaming variables, names of classes and methods, or adapting the methods moderately by extracting sequences of statements into methods, etc. These copying techniques will be different depending on different programming languages and to detect them we would need different techniques based on the programming language we intend to target. The problem we faced was to develop an application to handle plagiarism detection for multi-file submissions. We read up literatures on syntactic similarity techniques, clone detection, ast comparison methods to give ourselves a clarity as to how to approach the problem.

The resolution to our problem was to develop an application that will allow the students and professors to register, facilitate students in the process of assignment submissions for the courses they have registered for, automate the plagiarism check, view the results visually and send alerts in email with the detailed reports to the concerned professor when plagiarism has been detected.

Overview of the Result

The plagiarism detection application we developed as a team project meets the basic functionalities and pre-requisites that were put forward by the client(professors). Our application offers the user with a good user-experience(UX) in interactions, look and feel. The product we built helps to support the faculty and students in an academic settings in process of assignment submissions and plagiarism checks. We completed every base expectations in all three sprints from environment setup to implementation of the functionalities.

The statistics are below:

Sprint Number	**Base Spectations**	**Stretch Expectations**
1	9/9	7/7

2	11/11	4/4
3	8/8	6/11

Testing coverage standards from SonarQube:

Criteria	**Grade**
Bugs	A
Vulnerability	A
Coverage	A - 86.8%
Code Smells	A
Code Duplications	A - 0.7%

Bug Hunt Effect

In Phase three of the project, the testers who were allotted our team for bug hunt helped us in locating the bugs or issues with our system which we had missed during our development process. The testers analysed the quality of our code and the functionalities that were in place in the candidate branch and helped us in the code clean up and refactoring process by pointing out the bugs that we unintentionally failed to address during our implementation process.

They created a total of 49 issue tickets in Jira, and we discussed and closed the issues that were already solved at the current stage of the project. The irrelevant and duplicate ones were closed and the relevant issues were moved from the backlog to the Sprint 3 issues in Jira.

Development Process

Phase A- Requirement Gathering and Use Cases

Once the problem was specified, the team set out to develop a plagiarism detection tool in an SDLC(Software Development Life Cycle) pattern.In the initial phase, we gathered the client requirements and after discussing the relevant use cases from the client requirements, we made user-interface mockups of the system to informally describe the behaviour of the system from client's perspective. We decided to target Python language for plagiarism detection and the following are the development infrastructure decisions we made as a team.

Application type: Web Application
Front-End: AngularJS 1.0, HTML, CSS, JQuery, Bootstrap
Back-End: Java, Spring Boot

Automated Testing: Junit
Version Control and Issue Tracking: NEU Github
Database: MySQL Relational Database System
Integration Environment: Amazon Web Services

Phase B - Object Oriented Design using UML

In the second phase of the project, we focussed on designing the architecture of the system based on the use cases and mockups we created in phase A. The major components of the system and the data structures that will represent or manipulate them were generated after discussions and reading up multiple literature on source code analysis and clone detection. A UML diagram was designed to guide us through the development process. The UML class diagram we developed reflected the relationship between our major components and data structures. We created Java interfaces for the components based on the UML diagram in this phase.

Phase C - Implementation, Testing and Refactoring

The real product implementation of the system started in phase three. Phase three was divided into three sprints.

Sprint 1

In the first sprint, we setup the basic environment to manage our development process. We made sure our work was being managed in Jira. We set up Jenkins to automate the software development process with continuous integration and SonarQube to continuously inspect the quality of our code to detect bugs, code smells and security vulnerabilities in Java programs that we wrote. Since we designed our application to be a webapp, we deployed it to a cloud environment in AWS. We attained basic functionalities of login and register and constructed AST for a single python files using ANTLR4 library. We implemented a single comparison strategy using edit distance for comparing the asts generated. What we struggled with the most was finding a suitable and efficient comparison strategy algorithm for comparing pythons asts.

Sprint 2

In the second sprint, we moved to the development mode. Backend functionalities was split between two team members and front-end features by the other two. We focussed on developing functionalities for letting a user login as a student/faculty/admin and letting the admin grant the role of faculty to a user registering in that role. The faculty and student UI displayed semesters with courses they registered for and assignments under them. The faculty was given the option to see the student submissions and initiate comparisons manually.

The students submitted their assignments in github and the github url of the submission folder was submitted in our system. We used Jgit library to clone and access the contents in source code folders. The comparisons for all student submissions under a single assignment was supported by our system. We used Jplag library to aid us in plagiarism detection. JPlag takes as input a set of programs, compares these programs pairwise (computing for each pair a total similarity value and a set of similarity regions), and provides as output a set of HTML pages that allow for exploring and understanding the similarities found in detail. JPlag works by converting each program into a stream of canonical tokens and then trying to cover one such token string by substrings taken from the other (string tiling).

We detected the submissions that committed plagiarism based on a threshold set by the faculty. If the maximum similarity score we obtained in the comparison exceeded the threshold, we highlighted the compared submissions as plagiarized and the faculty was given the option to view the report and viewed the files side by side with the similar parts of the code highlighted in same color. We made sure that the system send the email with plagiarised report link to the concerned faculty once the comparison was over. What we struggled with the most was finding a good api library that would aid us in comparing multiple submissions in files and folders and computed a legit score based on similarity tokens. We completed the base and stretch expectations of the sprint. However we struggled to attain the minimum test coverage of 85%.

Sprint 3

In the third sprint, our application was fully developed to cater to the requirements put forward initially by the client. We attained the base expectations that said any faculty can login, select a semester and CRUD courses and assignments. We focussed on enhancing the performance of the system by incrementally comparing the submissions. The comparisons were scheduled to happen for every 10 minutes, so that the plagiarism check is automated and the faculty does not need to login and initiate comparison manually. All the submission comparisons which were detected of plagiarism were made available for the faculty to inspect when he logged in to the system under “View Violations” option. The same report link was also mailed to the faculty upon plagiarism detection. Our system maintained stats for all the submissions and faculty was given the option to view all the submissions from the students. We made sure we attained the base expectations and most of the stretch expectations in this sprint. Time constraint was a limitation for letting us finish all the stretches before the sprint review, however we made sure we exceeded the test coverage above 85% this time. We also focussed to eliminate the code smells from our source code and refactor the code based on suggestions from the sonarQube to attain maximum code quality.

Retrospective on the Project

On looking back, the project was structured very well in such a way, that it enabled students to experience a real software development life cycle in its true sense. From requirements gathering to use cases in phase A to implementing, testing, refactoring in the final phase, the team experienced the real process of software development. A collaborative effort by our team helped us obtain the result we intended to at the end.

What we didn’t like about the project was that the expectations were posted for section 2 only two days before the sprint review. There was no clarity as to which project expectations [Section 1 or Section 2] were to be followed up until sprint 2.

References:

Jplag Documentation, http://page.mi.fu-berlin.de/prechelt/Biblio/jplagTR.pdf