1 Problem Overview

Plagiarism in source code is a widespread problem and it usually goes unnoticed due to the lack of sophisticated detection systems to find different types of plagiarisms in source codes. For example, there are several scenarios in which any two students copy their work from one another through different code modifications or transformation such as renaming the variables, methods or class, insert or delete comments or change method signatures. At times, students collaborated and developed a code without permission.
All such activities are unethical and needed a mechanism to be caught. Such questionable scenarios generally go unnoticed as an instructor would have to manually check each possible source code pairs and check which of the two pairs were plagiarized. The task would be even more time consuming and cumbersome if the student copied a piece of code from another student of a different section or from a student of a previous semester. In this case, the combinations to check would be unwieldy for any instructor or teaching assistant to carry out.
There is already a broad array of tools such as MOSS and YAP3 that fulfill such a requirement, however such tools compare any two current submissions, without comprehensively referencing past submissions of previous semesters. Also, such systems did not allow a faculty to set a cutoff threshold beyond which the submission would be categorized as a case of plagiarism. Further, there is no mechanism available at hand which can automatically notify the student when his or her submission was caught in a case of plagiarism.

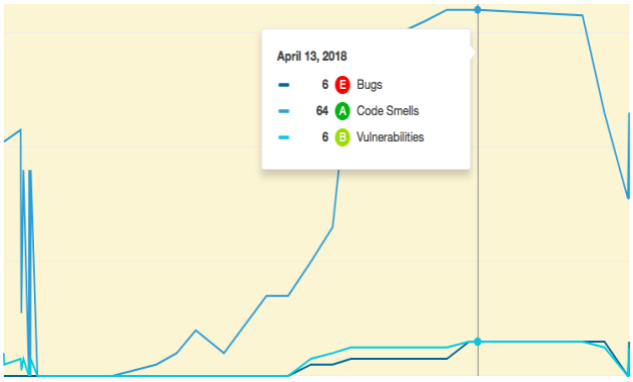
Consequently, there is an underlying need for an automated comparison and detection system which can compare and detect plagiarism between any two submissions of students across different semesters and sections. Such an automated plagiarism detection system would be able to perform a multi file comparison between any two submissions across different sections of the same course and the same assignments from the previous semesters. The comparison should also be performed incase a student submits a github link to his directory for that assignment. The system should have a mechanism to visualize the results with the similar code highlighted. Also, such a system should allow the faculty to set a minimum level of plagiarism threshold below which the cases would not be shown.
With these issues underlined, team-208 developed a web application to detect plagiarism between any given programs written in python

2 Overview of the result

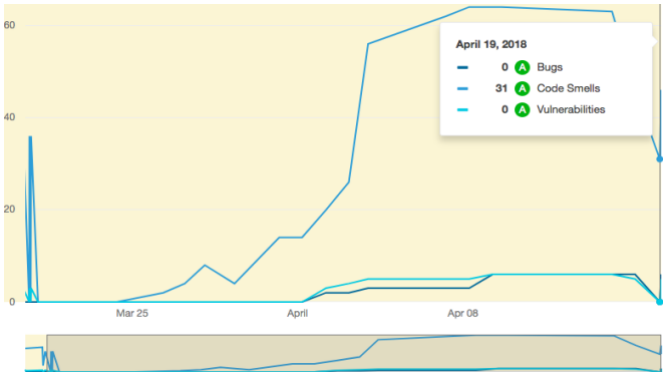
Quality:

The quality of the code improved over the course of time as can be seen from the following graphs taken from Sonar Cube

Status as on April 13,2018



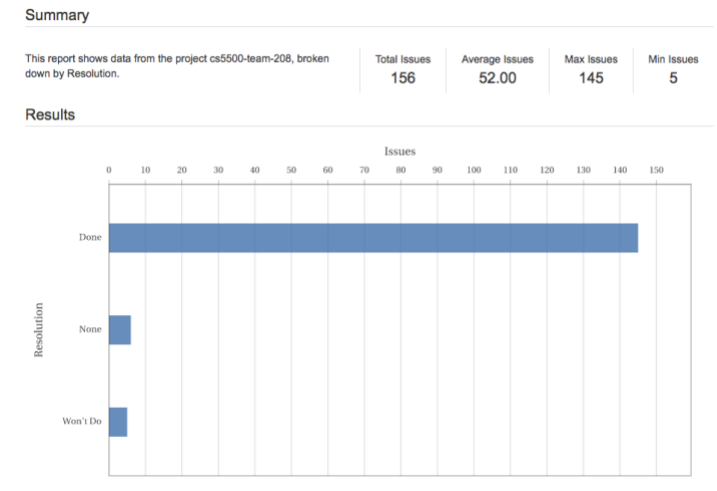
Status as on April 19, 2018



Report Results page:

Completion

There were a total of 156 issues which were resolved:



3 Team's development process

The project was split over 3 phases-Phase A- B – C, where every phase was stretched over 2 weeks

Phase A: ▪ (Requirements Analysis and Project Planning Phase)

The first phase involved project planning activities such as deciding the target programming language for which the plagiarism would be detected. After a careful study of the syntax of different programming languages such as racket, python, java and C++, python was chosen as the language of choice as it was found that it was a preferred choice for many students to write code in and it had a clear and readable syntax. Also, Python compiles to bytecode which is easier for tracking. The requirements for the system were gathered through discussion with the professors and teaching assistants of the course. Based on the collected requirements a set of *15 high level use* cases were extracted. It was initially identified that the professor and the teaching assistants would be the prime users of the system. The professor was the prime user of the system and preferred least possible overhead of tasks. However, after a careful re-evaluation and brainstorming activities with the professor Jose over hangouts, the Teaching Assistants were no longer identified as a separate user of the system and no special features were allocated to them.

Thus, the system had 3 primary actors- The student, The professor and The Admin.

CHANGE TRACKER 1: A track of changed activities

CATEGORY	Proposed	Actual	Comment
Actors	2 Primary actors- • Professor • Teaching assistant	3 Primary Actors • Professor • Student • Admin	Requirement Change
Use cases	The professor wants to view reports generated by the Teaching assistant.	Use case for Teaching Assistant was scrapped	Requirement Change
Use cases	The actor wants to view files sorted or filtered per matched percentage	The use case is scrapped	Not Required

Table 1 changelog for phase A

Phase B: ▪ (Requirements Analysis and Project Planning Phase)

The goal of Phase B was to design and develop the architecture of the system.

Based on the requirement analysis and the use cases finalized in phase A, the components, data structures, interfaces, the database models for the overall system were designed and developed. The initial system was developed in Spring- MVC framework.

It was the first and obvious choice as it was the traditional way for implementing the model-view-controller design pattern for a web application in java

Due to the lack of domain knowledge, available online tutorials and user manuals for successfully implementing a web application in Spring MVC, the system was redesigned in Spring Boot framework of the Spring frameworks. Spring MVC hindered the pace of development of the project as it involved an extensive set of configurations, which caused a substantial overhead. On the contrary Spring Boot involved fewer configurations and dependencies. Thus, the initially proposed data structures, interfaces and database models were developed in the Spring Boot framework.

Phase C: ▪ (Requirements Analysis and Project Planning Phase)

Phase C was split up in 3 sprints where each sprint had different priorities for implementing the functional requirements.

Sprint 1:
From sprint 1 onwards, all the tasks were assigned on JIRA and smart commits were followed. The comparison strategy and the algorithm was discussed and the high-level interfaces for the comparison algorithm was implemented. Sample programs in python were written and all the programs to be compared were parsed using ANTLR-4 and converted into token strings. The Zhang sasha-algorithm proposed to compute the edit distance between any two token strings as a measure of similarity between 2 programs. The expectations were fulfilled by the team.

Sprint 2:
Sprint 2, involved implementing the comparison algorithm. However, due to the change in the requirements, instead of implementing the Zhang-Sasha Edit distance Algorithm, JPLAG was used in the backend to compare programs written in Python 3. 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.
Using JPLAG, additional features and stretches to the overall scope of the project were added and were fulfilled by the team. Also, smart commits were not being followed which started with this sprint onwards. Master was also made protected.

New Requirements

1	Students can submit an assignment multiple times. System maintains stats for every submission.
2	Any faculty should be able to login and view multiple submissions
3	Any student can login, select a semester, course and assignment and submit
4	Faculty should be able to set a minimum threshold for plagiarism
5	Student should be able to submit assignments only to the registered courses.

Sprint 3:
Sprint 3 involved completing the stretches and the following stretches were completed successfully by the team
Stretch Goals:

1	Email/text message with plagiarism notice includes link to report that shows students, files, and comparisons of interest.
2	Link only works if user is logged in
3	Email/text message with plagiarism notice can be shared with relevant students, and other faculty. Link only works if user is logged in
4	Login/register with at least one social network account such as Google, Facebook, LinkedIn, etc.
5	Compare against previous semesters, faculty can choose how far back to compare. Defaults to none.
6	Supports both file/folder/zip upload AND github submission
7	Performance enhancement using autoscaling, throtling, etc.
8	Compare at least another language other than Python.
9	Stretch (even bigger): uses more than one comparison strategy and computes an overall score using a weighted polynomial function.
10	Compare against different sections of the same course.

4 Retrospective

Learnings:

- Application of strict agile process
- Requirements Elicitation techniques and Requirement Planning
- Usage of tools such as JIRA, JENKINS, SONARCUBE, AWS,s3, Postman, Spring Boot
- Team management

What can be improved:

- Better Project Structure
- Be allowed to choose team mates so that time is not lost in building rapports and kick starting the project.

Best

The best experiences involved:

- Learning the best coding practices such as the usage of loggers, comments
- Project Planning
- Working in an actual production level environment

Worst

- Need better overview of the project problem at hand