Northeastern University
CS5500-Section 02 – Managing Software Development
Spring 2018, Namrata Bilurkar, Parshva Shah, Praveen Singh

Plagiarism Detector

Overview of the problem:

1. Detecting the plagiarized code in student submissions for a coding assignment is of most concern to educational institutions.
2. Most of the existing methods for detection of plagiarized document compute the similarity of two documents by performing just text comparisons.
3. However, there are many types of code plagiarism where text is changed but the underlying code is actually the same; like addition or deletion of redundant elements, changing comments, changing the names of the variables and functions, reordering the code, rewriting loops, can make the code appear different while still being the same etc.
4. It becomes tedious for the Course staff (Instructor and Teaching Assistants) to assess files/projects of all the students of the class manually and find students whose assignment submissions were very similar.
5. We propose a system that will not only consider textual content while performing comparison, but, will also consider the structure of the code segments by comparing Abstract Syntax Trees (ASTs) of the code submitted.
6. The course staff never expects 0% similarity, as the base code could be similar in almost all the students submissions, thus, there is a provision in our system for the course staff to set a threshold, above which the submissions would be tagged as plagiarised.
7. For this project, we have limited the programming language of assignment submissions to be just Python, but, in future development of the project, any programming language whose AST can be generated can be supported by this system and incorporated into it easily.

Overview of the result:

Following functional expectations are completed:

1. As the project is a web application, login and registration is provided.
2. More than one strategy has been provided for comparison and they are on demand.
3. Each strategy reports similarity score of the submissions and the similar lines of code are highlighted.
4. System provides support for multiple file submission.
5. System logs user activity.
6. SMTP trap for error logs, where all the errors in the production are reported to the user(course staff) via email.

7. Comparison of all submissions in the system is provided on demand.
8. A CRON job has been set in place in the system which runs every night and reports to the course staff via email if plagiarism is detected over the threshold.
9. Usage statistics and system status are displayed to the course staff.

Following environment expectations are met:

1. Each pull request must be approved in order to merge the branch into master.
2. Project is a Maven project.
3. Jira is used for issue tracking.
4. Smart commits are used in Github.
5. Jenkins is used for continuous integration.
6. Pull Requests are notified to the entire development team by email and Slack.
7. SonarQube is used for continuous code quality.
8. Web application is hosted on AWS.

We were able to achieve a code coverage of 95% (Source: SonarQube) and consistently maintain A grade on sonar dashboard.

Testing included mocking each layer that was not part of the layer being currently tested and testing each layer on its own. Logging and email features were disabled for tests to avoid memory issues and email spam.

About 70 Jira issues were raised by testers which were addressed within 12 hours by adding comments, assigning issue level and marking tickets as duplicates if same issue had been reported by multiple testers.

To hit success, the following modules can be improvised:

1. OAuth login like Google login can be supported for faster registrations and logins
2. Login sessions can be maintained for providing proper access control
3. Support for providing Github project URL instead of Github file URL can be added
4. A new type of user - Administrator can be added for administering the system and performing roles like changing roles of users, for e.g., changing a student to TA (course staff).
5. Support for submissions in additional languages like Java, C++, etc can be added to the system.
6. Support for course staff to add/delete assignments can be added.
7. Plagiarism can be found among submissions across sections, semesters, and languages.

An overview of the team's development process:

Things that worked:

1. We used Agile Software Development methodology for the development of our project.
2. We used Github for version control, Jenkins for continuous integration, Jira for issue tracking, SonarQube for continuous quality control, Slack for performing daily standups and AWS for deploying the web application and maintaining database.
3. Entire development was done in sprints of two weeks. Sprint reviews after each sprint were very helpful in receiving a feedback on the work performed and whether the expectations were met or not.
4. Smart commits were used in the project due to which Jira issues were moved on the Jira board automatically between To Do, In Progress, PR Submitted and Closed.
5. Slack was used for addressing issues and getting status updates from all the teammates.
6. Direct merging of branch to master was blocked. Thus, after implementing a feature, a pull request had to be submitted, and there would always be a team member who would review the PR before merging that feature into the master branch.
7. The branch could not be merged with the master until all three Jenkin runs of branch, head and merge were successful. Only after that, the new feature branch could be merged. The feature branch was deleted after merging to avoid keeping stale branches on the repository.
8. At each pull request and merging of a branch, all the teammates were notified about it via both slack and email.
9. SonarLint was used on each developers system to ensure code quality.
10. We consistently maintained Grade A code quality on SonarQube over the entire course of the project.
11. JUnits were used to write test cases for all the Java files. Only the tests for the same layer were written. Functionalities of upper/lower layers were mocked using Mockito.
12. Swagger-UI was used for API documentation.
13. We, as a team, coordinated with each other quite well for meeting and fixing issues, conducting peer reviews and assigning tasks for next sprint.

Things that did not work:

1. Occasionally, we used to miss stand-ups on Slack, due to which, issues were not addressed on that very day and updates were given a bit late.
2. Initially, we had planned to design user interface using React.js, but, due to lack of sufficient knowledge and experience in React.js and frontend development among the team members we switched to AngularJS for the development of UI.
3. We started out the project with the aim of using a third party API for performing plagiarism detection and obtaining similarity scores, in which case, we had to focus more on UI. On later stage, after realizing that our team lacked the time and resources in UI development, we switched to the approach of implementing comparison strategies on our own.

4. The expectations for each sprint was for that of a team of four members, and since we were short of a team member we missed out on meeting one or two requirements which cost us a lot of points.

Retrospective:

The best part of the project that the team liked was touching each aspect of software development, right from the role of Product Managers of identifying requirements, role of developers in writing and testing the code, role of DevOps in maintaining and keeping the infrastructure up and running all the time and automating where needed to keep the development fast paced.
Also, the team came across and learned new tools like Jira, Jenkins, SonarQube which would surely be of great advantage while working in a corporate environment where similar tools are used.
While using Jira, the team learned how to create and prioritize issues, tracking bugs and issues, finding bugs and reporting the same, responding to bugs raised by other testers and linking duplicate issues.
The team also solved SonarLint issues and always tried to keep the code quality to A grade and code coverage above 90% because of SonarQube restrictions, which helped keep the team in check by ensuring high quality of code.

The worst part was maintaining Jenkins which used to go out of memory all the time and especially when everyone was working simultaneously, but at the same time it gave us a great learning experience. Also it made the developers life easier by automating all the builds and deploying the code finally to AWS without any human involvement.

Course improvement suggestions:

1. Teams with a smaller size should have a different sprint expectation than the other teams as due to lack of resources and the expectations being the same as of teams with more people, we tend to lose grades.

2. Also students poll should be taken for front end and back end developers and accordingly assign them to teams so that each team ends up with at least one backend developer and one front-end developer.