
PROJECT ON

# PLAGIARISM DETECTOR

**Team- 108**

CS 5500: Managing Software Development

Northeastern University - Spring 2018

Prof. Michael Weintraub

By:

APOORVA NAGARAJ

DISHA SULE

TEJA GUMMALLA

SIDDHESH SALGAONKAR

# OVERVIEW OF THE PROBLEM

Student Plagiarism is a major problem faced in academic institutions. Institutions are spending a lot of time and resources to combat this issue. Some institutions employ the use of text-matching software to check a student's work against other submitted work or materials on the internet. This might be effective for essays and textual solutions but is not very effect in detecting plagiarism in source codes. There are different scenarios where two or more students produce different versions of the same code, one version is derived from another by either using copy-then-modify technique, or moving around blocks of code or splitting a block of code into multiple blocks. Text-matching software fails to detect such instances of plagiarism. Hence institutions need a robust tool to detect plagiarism in code.

The systems that may be present to check for user code plagiarism are not exactly user friendly and may not have a setting to tweak for different algorithms. A good plagiarism detector application should be easy and intuitive to use, should provide what the results mean and show where the plagiarism has occurred. We have in this application attempted to cover these and many more objectives.

The project aims to create an interactive web-based application dedicated to detecting plagiarism in C language code. The application should look beyond textual differences and combat the below plagiarism:

1. **Exact Match** - Directly copying the code from another source without any modifications. No changes in original and duplicated code.
2. **Lexical Changes** - Involves only lexical changes which do not affect the structure of the program. like changing variable/ method name, adding/ removing blank lines or comments.
3. **Split Code Files** - Functionality is split across different files.
4. **Simple Syntactic Structuring** - Involves syntactic and structural changes that slightly affect the syntax or structure of the program like changing the order of the operand, adding redundant statements to change the overall appearance of the code.
5. **Complex Syntactic Structuring** - Involves syntactic and structural changes that significantly affect the syntax or structure of the program replacing expressions with equivalent expressions and modifying independent code.

# OVERVIEW OF THE RESULTS

Base Expectations met:

1. The application has a Login feature to log into the application using the registered username and password.
2. User can upload either two submission files or multiple files in two submission folders to compare against each other.
3. The application successfully detects the all 5 types of plagiarism
4. The application offers two on-demand AST based sophisticated strategies to detect plagiarism:
    a. Longest Common Subsequence algorithm
    b. Needleman-Wunsch algorithm
5. The application logs user activity in rolling log files and logs errors in error files and send a mail to the admin with the stack trace of the error.
6. The application displays user statistics of the number of plagiarism detection cases run and system status.
7. The application has a fully functional and user-friendly user interface

Stretch Expectations met:

1. The user can upload multiple submission folders having multiple files to compare against each other.
2. The application offers a weighted combination of Longest Common Subsequence algorithm and Needleman–Wunsch algorithm where the user can adjust the weights using a slider.
3. The application offers another weighted combination of Longest Common Subsequence algorithm and Needleman–Wunsch algorithm where the weights are trained using Stanford's MOSS as the benchmark.

Interesting features added:

1. A user can register themselves on the application.
2. Application has session management and encrypted password.
3. Application only takes C files as input.
4. The application allows users to download the results of their comparison in an excel format or pdf format.
5. The application allows users to directly enter two pieces of code on a textual interface provided in our comparison module.

SonarQube Statistics:

| | | | |
|---|---|---|---|
| **Coverage** | 85.0% | **Unit Tests** | 39 |
| **Bugs and Vulnerabilities** | 0 | **Security Rating** | A |
| **Code smells** | 0 | **Maintainability Rating** | A |
| **Duplicate Code** | 0 | **Reliability Rating** | A |

Backlog Statistics:

- Bugs Raised: 105
- Bugs fixed: 91
- Bugs in Backlog: 14

# OVERVIEW OF THE TEAM'S DEVELOPMENT PROCESS

**The TEAM's development process involved a flow of procedures as stated below:**

- The team worked in sprints. Every sprint began with a Sprint planning meeting.
- We reviewed all the requirements together as a team. The requirements were then converted to tasks.
- We used Jira tickets to keep track of the tasks.
- The priority of the tasks where decided and assigned to team members.
- Each member then expanded his/her tasks into subtasks and decided a passing criterion for it.
- The progress of the tasks was tracked using smart commits in Jira.
- The tasks were marked complete only after testing the module
- We had group code reviews where the team members reviewed each other's code and pointed out bugs or suggested improvements.
- Before every sprint review with the TA, we went over all the work and made sure it was ready for the demo
- After the sprint review we incorporated the review comments in our code.
- The team had an efficient development process and this helped in having a smooth workflow.
- The team had sprint planning meetings before the sprint began where everyone pitched in their ideas. Hence a lot of novel and interesting ideas came up.
- The tasks were assigned in these sprint meetings hence every team member could get sufficient clarity on the task.
- When in doubt we always had open discussions. Such discussions facilitated knowledge transfer and helped us catch many bugs and issues at early stages of development.
- The team also did pair-programming for many tasks. This was another effective way to share knowledge and combat any hitches. Two heads are better than one!
- The team grew as a unit and helped each other in their tasks
- All the team decisions were taken unanimously.
- Whenever there were disagreements, they were handled in a very professional manner.
- GitHub was helpful in keeping track of the versions of code and parallel development using branches
- Jira helped the team keep track of all the tasks.
- Jenkins for continuous integration and Pull Request approval for master made sure only good code was pushed to the master branch. All the Junit tests were executed and code was promoted only when those tests passed.
- SonarQube made sure we could only push tested and quality code to the master
- Slack integration of Jenkins helped notify all the members of build and failures. Also, Slack proved efficient for daily tasks updates.

**What didn't work?**

- The team had planned to add some extra features for user's like 'Forgot Password' or Admin authenticating the user, i.e. if a registered user is indeed a faculty. But due to time constraints and other priority tasks we could not complete this functionality.

# Retrospective View

**What the team liked best:**

- The course of the project introduced us to a lot of new technologies
- We were introduced to industry practices of delivering quality software
- Through the project, we all worked on various aspects such as front-end implementation, backend, core algorithm, setup and deployment.
- With the sprint requirements not being too rigid, we were free to choose the design and implementation strategies of our project.

**What the team liked worst:**

- The timeline for the deliveries was quite hectic. Phase A and B were completed in the first half of the semester and we did not quite estimate the time we would have to implement these mock-ups through the rest of the semester.

- The implementation of the project is concentrated in the second half of the semester, some more time to improve a few features and implement additional ones would allow make the app complete.

**What we learnt:**

- Working in a team can be challenging task, especially when there are multiple aspects to the implementation. To deliver a project that is beyond the scope of one person's implementation, we learnt to collaborate, divide the tasks among the team members, and integrate our work.

- We learnt to follow good software engineering practices such as working with Jira for subtasks, continuous integration with Jenkins, quality code development using SonarQube and SonarLint analyzers.

**What could be better:**

- The development of the project is quite a large task in itself. The delivery of the final project using reports and presentations is comprehensive, however it might be nicer to have a final review with our TA or a professor.

- An interactive demo of our project would give a channel for the TA's or professor to ask questions or clarifications, about our implementation, strategies etc. We could also receive some feedback on our work, what we did right and what we could improve.