Plagiarism Detector

Team-111

Amritansh Tripathi

Gautam Baghel

Veera Venkata Sasanka Uppu

Xin Li

Managing Software Development

Spring 2018

Northeastern University

College of Computer and Information Science

Overview

Plagiarism in a code is extremely common these days. Under the guise of shared resources, students justify to themselves and the authorities for copying another people's work. Plagiarism in texts, papers and written works are relatively easier to detect as they're strings, and since the problem has been around for a while there are many solutions for this problem. The problem becomes harder for codes. It is not only due to the large and sudden influx of codes in the past two decades but also changing few parameters in a code could make same codes look drastically different.

So, does that mean that plagiarism doesn't exist in the world of computer programming? Is it anyone's game to simply take and use other's code?

This is not the case; just like in the world of literary plagiarism, there is a fine line between lawful use and plagiarism in a code.

The Brown Daily Herald(http://www.browndailyherald.com/2010/11/01/computer-science-tops-in-academic-violations-last-year/) in 2010 published a piece that revealed their computer science department to have the highest level of plagiarism within the university among all other majors. Even though usually the Computer Science department has the highest percentage of plagiarism, the measurement may be inflated due to the department's frequent use of MOSS (Measure of Software Similarity) to detect instances of duplicate code. MOSS is a free solution (developed by a Stanford professor in 1994) that enables a list of files to be compared and duplicate code to be highlighted.

While researching the problem we came across multiple ways someone can plagiarize a code. Either the variable names are changed for example from "var x= 10;" to "var k = 10;" or order of the code is changed (for something which doesn't make difference in execution) or the code is simply copy pasted without proper citation or the program is distributed into several smaller files. Other problems may include system or settings file which remain common in every project work.

Writing for the humanities and sciences builds on layers and previous frameworks just like code. Authors frequently take concepts, styles and ideas from one another. However, citing sources and providing proper attribution is the difference between borrowing and plagiarism. With both coding and literature – technology can be utilized to detect exact instances of duplicate content to help us draw that fine line.

As we can observe from the cases stated above that a simple string comparison is not sufficient to detect similarities between codes. A more complex strategy needs to be implemented. The problem which persists on codes is that every code gets parsed and converted to machine codes so the solution may lie in the node generation and syntax tree which is being generated. For a more complex solution we will try to connect the string comparison algorithms and abstract syntax tree generating algorithm to form a more complex and better analysis.

To test the accuracy of our system the reference point would be Measure of Software Similarity (MOSS) for us to determine which algorithm implementation works best.

Development process

Sprint 1

During Sprint 1 and on initial meetings it was hard for us to gauge each other's skill set and areas of expertise. Since nobody was officially notified as the group leader we prepared to pose the question to ourselves as to how we wanted to proceed. We laid down the foundations of what we know and what we don't. We had a sense that everyone has hard time mentioning their weak spots so we just focused on the strength instead of weaknesses. Initial mode of communication that we decided was weekly meetings, twice every week to be precise. This didn't pan out as expected as everyone had different schedules and schedule changed every week. Best mode we could find was to do skype calls through which we could share our screens if need be. In terms of troubleshooting if someone needed help TeamViewer also proved to be an indispensable tool.

Sprint 2

Going forward in sprint two, we had our work/code spaces set. We made sure that everyone was comfortable with the segment of code they chose. Playing on strengths they were the core algorithm, frontend, backend and Jenkins and pipeline. Although we didn't restrict ourselves just to the areas we primarily worked on, we frequently crossed over to other segments of code but dividing the problem this way gave us the advantage of a person maintaining the code base clear. In simpler terms, we had a "go-to" guy for those segments of code. We were aware that we couldn't restrict each member to just one segment because most modules of the code worked in tandem with each other. For example, when determining what users will see on the frontend it was indicative that the relevant information was to be produced and delivered from the backend. Second example would be setting down a common layout/format for each implemented algorithm. We needed to define an interface that every algorithm could follow to implement different strategies. This required Modelling the whole structure of the project with what user needs to see, what the backend needed to store and what information was logged during an unexpected behavior. This required high levels of communication skills, some expertise of design skill and fair level of coding skills.

Sprint 3

Consequently, during sprint three the website was in fair shape. We had a working front end, a solid Jenkins pipeline, a solid and complex algorithm and fair knowledge and familiarity with the development process. Communication was easier than earlier sprints so we just had single small skype sessions. Most discussions were done in the slack channel. The emphasis was mainly on testing the code and delivering a quality product. We had been following test driven design so everyone had their own test cases but we also decided to cross check each other and write test cases for each other without any indication of who wrote the test case. This helped us achieve unbiased cross checks and test for potential faults in our system. After unit testing came integration testing and system testing. We had to work together to create test cases to check whether modules worked well when combined. For example, to check whether analysis between each file is accurately done mock requests were sent and the generated report was compared from the end results to make sure that the controller modules, uploader modules and the comparer modules were working in tandem with one another. This required good communication and splitting the team further into teams of two.

Final debug

During the final segment of our project we received issue tickets raised in JIRA for either a feature request or a bug report. We scheduled two meetings to distributed and assign each ticket to the member working on the issued module. We had to distinguish the feature tickets raised from ones which were feasible with the time constraints and project requirements and one which had to be future enhancements. Since the production pipeline was already set everyone worked on their tickets and resolved issues.

Results

The Plagiarism detection system has been tested on various edge cases as shown below:

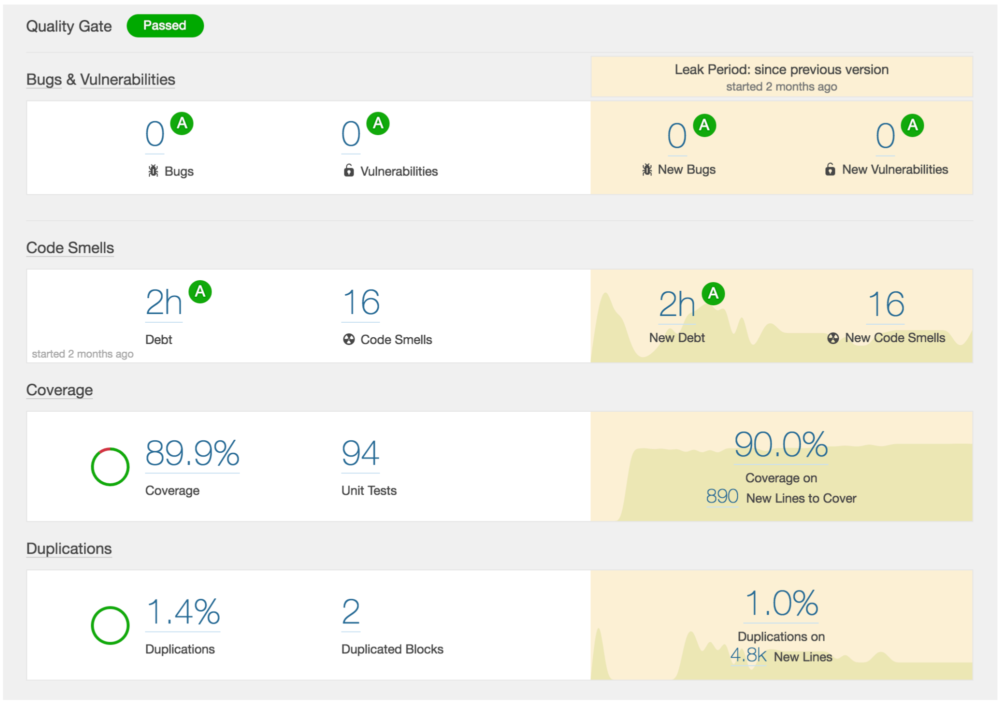
File 1 description	**File 2 description w.r.t to File 1**	**Algo - LCS String**	**Algo – LCS Ast**	**Algo - Jaccard**	**Algo - TokenEdit distance**	**MOSS OUTPUT**	**Machine Learning Model**
Bubble sort code	Exactly same as File 1	100.0	100.0	100.0	100.0	96.0	100.0
Detailed Bubble sort code with comments	Bubble sort code without comments and variable names changes	40.0	99.07	55.56	94.96	76.0	80.66
Bubble sort code	Completely different code	3.52	9.50	0.94	11.31	0.0	7.22
Bubble sort code	Very small python code	0.0	1.62	0.0	2.43	0.0	1.37

As we can see the system predict exact same files and nearly same files very accurately by using different techniques. LCS with AST and Token Edit distance performed to be better algorithms and have more higher influence / weights in the machine learning model. In the third and fourth scenario when Bubble sort is tested with different code files, there is a slight noise caught from the algorithms which was observed due to same variable names/small functions etc. and MOSS is able to correctly eliminate that.

Below table is a breakdown of number of tickets resolved for each sprint. During sprint 1 and sprint 2 the time spent on tickets was more as it was involved in building more things from scratch. During sprint 3 some of the bug tickets have been resolved and other minor works have been dealt successfully in implementing an end-to-end system as per requirements. Post sprint all the backlog items have been cleared successfully.

Sprint Information	Sprint 1	Sprint 2	Sprint 3	Backlog with duplicates
Number of tickets resolved	11	11	39	198

The quality of the code has been successfully maintained throughout each sprint. There was unit testing as integration testing for each module and below is the detailed report by SonarQube.


Quality Gate Passed

Bugs & Vulnerabilities

Leak Period: since previous version
started 2 months ago

0 A Bugs
0 A Vulnerabilities
0 A New Bugs
0 A New Vulnerabilities

Code Smells

2h A Debt
16 Code Smells
started 2 months ago
2h A New Debt
16 New Code Smells

Coverage

89.9% Coverage
94 Unit Tests
90.0% Coverage on 890 New Lines to Cover

Duplications

1.4% Duplications
2 Duplicated Blocks
1.0% Duplications on 4.8k New Lines

Project Retrospective

Things we love about our project:

First of all, we managed to finish most of the requirements we set at the beginning of the project. During the entire process of development, for every sprint, we delivered something usable and deployable. It may not work very well, or working under certain restriction, but the basic sprint backlog is achieved. And we always have interface to show the user.

Secondly, we used the design patterns we learnt from the class. We have drawn out the UML, Class Diagram for our product and implemented them on JAVA interface codes. What's important is that we are still using the design pattern we made during phase A and phase B, which indicates that we did a pretty good job with the design.

Meanwhile, we do like that fact that we managed to implement 4 algorithms and be able to show the line by line comparison. Also, we made a pretty decent weighted algorithm to combine the algorithms we have implemented.

What's more, in the end, we start to enjoy using Jenkins to thoroughly test our platform and kept our code quality by using SonarQube running through our codes, which makes our project deployable every time we commit something new.

Problems:

We once had a hard time to set up Jenkins server to continuously integrate our system. None of us has any experience with Webhook for pull request. We firstly used pull-request builder plugin for our Jenkins. It worked for a long time, then we changed it to Github multi-branch project to cope with more complex requests. In the middle of sprint 3, when we were trying to upgrade the Jenkins, we lost the privilege of being an admin for our system, which cost us a long time to get the Jenkins back on.

Meanwhile, we also had some hard time with Implementing complex algorithms to detect various forms of plagiarism. We used a lot of algorithms, but we definitely should have paid more attention on at least one of them to get a better result.

Also, we do have some difficulties to implement the user and files management system, because we require the user to have certain folder structure to upload the file. With that being said, we spent a lot of time figuring out how to store folder information and file information in our system. However, the structure is still not as good as we expected.

Citations

Bakshi, Tony. "Computer science tops in academic violations last year" The Brown Daily Herald. 1 November 2010. http://www.browndailyherald.com/mobile/computer-science-tops-in-academic-violations-last-year-1.2388353

Jannett Perry. "Plagiarism in Code" Nov 5, 2010 http://www.ithenticate.com/plagiarism-detection-blog/bid/52952/Plagiarism-in-Code#.WtkNPJch1EY