2018

Northeastern University
College of Computer & Information Sciences
Managing Software Development

Sherlock

A PLAGIARISM DETECTION TOOL

TEAM 211
[HARSH MEHTA, SANDIP SAMAL, SHANTANU KAWLEKAR, SURAJ NAIR]

UNDER THE GUIDANCE OF PROF. JOSE ANNUNZIATO

ABSTRACT

Source code plagiarism is a severe problem in academia. In academia programming assignments are used to evaluate students in programming courses. Therefore, checking programming assignments for plagiarism is essential. If a course consists of a large number of students, it is impractical to check each assignment by a human inspector. Thus, it is essential to have automated tools to assist detection of plagiarism in programming assignments. To detect plagiarism on programming course, the plagiarism detection method based on Abstract Syntax Tree (AST) is proposed.

INTRODUCTION

Program design is a very import part in the process of teaching and practice for computer specialty in colleges. With the development of information technology, especially the development of the Internet, information becomes more and more available. On the courses of programming, students can communicate with each other conveniently through Internet. In various web forums, there are a lot of experienced developers introducing their experiences or putting forward their problems. These are undoubtedly precious wealth for programming beginners. However, such resources also provide great convenience for plagiarists. In this project, we use the AST as a code plagiarism detection mode and MOSS as a standard for plagiarism detection. MOSS (for a Measure of Software Similarity) is an automatic system for determining the similarity of programs. To date, the main application of MOSS has been in detecting plagiarism in programming classes. Since its development in 1994, Moss has been very effective in this role. The algorithm behind moss is significant improvement over other cheating detection algorithms. AST as the similarity detection model is mainly based on the following considerations: AST is an intermediate data structure in the procedure compilation or interpretation process, the syntax tree not only reflects the structural information but also retains the attributes of the source program. This can provide a more accurate and comprehensive information for plagiarism detection. First, the corresponding lexical and syntax parser are generated according to Python grammar files, and the source codes are converted to AST; second, the similarities are calculated by using a sequence matching method. We collect sample data with variety of outputs generated based on AST and MOSS for training. Using this training data, we predict the plagiarism score using Linear Regression. Our application is a web-based MAS (MySQL, AngularJS, Spring-Boot) stack application with three user roles namely Admin, Student, Instructor. Instructors can add a course and make it available for students to register the course for a semester. Instructors can add assignment for a course they are involved in. The target language is set by the instructor for an assignment. We currently support plagiarism detection in two languages: Python, Java. Students can register for a course that is available for a semester. Students submit the assignment and on submission, we check if any submissions have similar code. A report is generated which can be viewed by instructor. In case of any plagiarism that is detected, a mail is sent to the instructor and students involved in plagiarism.

DEVELOPMENT PROCESS

As a team, we followed Agile methodology as a process. Agile software development describes an approach to software development under which requirements and solutions evolve through the collaborative effort of self-organizing and cross-functional teams. During the process, we realized the importance of agile. It helped us in the following ways.

- Better decision making: As a team, we made sure that we meet or arrange a call every day to update each other with the status of our work as well as plan for the future. Any shortcomings were discussed in the call.
- Honesty: We were honest to each other within the team in terms of the requirements. One learning here for us was when we work in a team there will be disagreements, but these were healthy disagreements as all of us had the same goal in mind that is to make our project one of the best.
- Transparency: Following agile helped us avoid misunderstandings and the need for massive corrections at the last minute.

During the course of the project, we had to take several collective decisions. One such decision was during the start of our development, we had agreed upon using JSP as the front end. WE managed couple of sprints with the same. We realized during the development that we needed a change in our front-end technology to make it more user friendly. We did not take the right call with the technology stack initially. We collectively took a decision to change our front-end technology to AngularJS as this was identified as a strength in our team. We acted immediately to change our stack and before Sprint 3 we had successfully implemented all the features and stretches. This was only possible through sheer communication and accountability. This was another learning for us of why Agile as a process is so helpful.

Our intent was to develop a system that was ready for the real-world and we understood that a good UI/UX is very important for the same. Thus, we dedicated a lot of time to develop a great User Experience for the users. We also have our portal ready to be used by any faculty at Northeastern as our system supports university sign on using Northeastern Email and automatically identifies instructors as well as students. With regards to the functionalities, we allow students to submit assignments posted by instructor for a course. On every submission we run incremental checks for plagiarism. Our instructors can see a detailed report showing lines copied, side by side comparison of code with copied lines highlighted, and plagiarism scores.

Primarily, we planned to only implement MOSS for plagiarism detection. But, post Sprint 2, we planned to have a fallback in case there was an unavailability issue with MOSS API. Thus, we implemented Linear Regression using MOSS as the standard, and Edit Distance and Total Lines Copied as the features.

The biggest difficulty in following agile was that it is assumed that every team member, by default, are skilled, disciplined, and willing to self-organize. The real world isn't so. People with different skills, cultures, and social frameworks are assembled for a short time to produce a project.

RESULTS

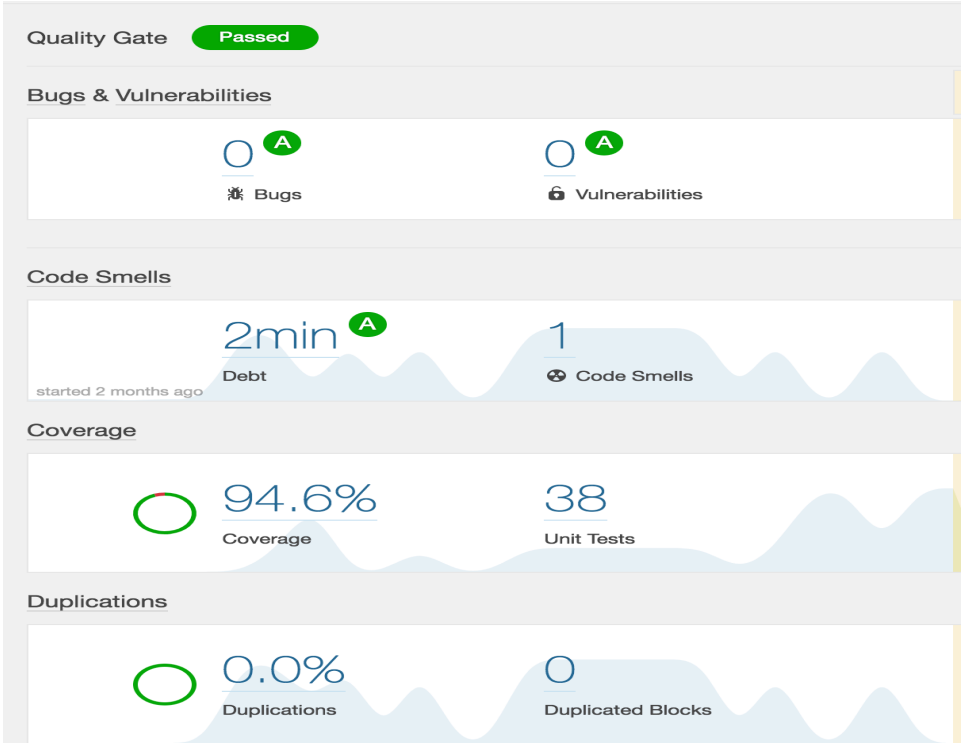
1) Algorithm:

File 1	File 2	Regression Score (Linear Regression)	Plagiarism score (MOSS)
Bubblesort1.py	Bubblesort2.py	86%	93%
lcs1.py	lcs2.py	68.7%	77.8%
check1.py	check2.py	97%	100%

- We can observe here that MOSS output gives more accurate results and thus is used as a standard for running Linear Regression.
- MOSS handles indentation and tab spaces during check and thus making it more accurate than all the existing algorithms.
- check1.py and check2.py are exactly same files and thus MOSS gives an output of 100%.

2) SonarQube: Provides the capability to not only show health of an application but also to highlight newly introduced issues. With a Quality Gate in place, one can fix the leak and therefore improve code quality systematically. The quality of the code was successfully monitored and maintained throughout during each sprint.

   Shown below is an image which describes the statistics of our SonarQube application.



3) JIRA: We used JIRA for issue tracking across each sprint. Each spring we have addressed the reported issues and closed several tickets.

LEARNINGS

Managing Software Development course taught us many industry standards related to design and testing and also how a modern agile team works. It is rightly said that the strength of the team is each individual member and the strength of each member is the team; following agile made us believe in the same. From the beginning, our communication with each other was always intact. There were several important decisions to take amidst the development and of course there were disagreements; but these were handled professionally by being honest to the team and working to achieve same target that is to make our project as efficient as possible.

We used several tools for making the project hassle free. With the integration of source control, Jenkins, and JIRA, we could relate issue to code to build/deployment/environment. This provided useful intertwined information quickly which avoided manual communication and tracking.
In terms of programming, we collectively understood the use of design patterns. We have used design patterns like Singleton and Strategy design patterns to make our code more flexible and less cluttered. We followed Model–view–controller (MVC) architectural pattern for developing user interfaces that divides an application into three interconnected parts.
For testing, we performed Whitebox and Blackbox testing were conducted on our code to make it as robust as possible. During the course of project, we did a lot of pair programming as a result we were more efficient and created less development errors.