Northeastern University
CS5500 – Managing Software Development
Spring 2018, Team-113 (Bhardwaj, Das, Lin, Monga)

Overview of the problem

The aim of the project is to design, develop, test and implement an application that facilitate instructors to detect plagiarism among students submitting assignments. The submitted assignments are to be compared based on textual similarities, moving similar functions or methods to another location in the same file, renaming variables, classes, and methods, extracting sequences of statements into methods, etc.

For the design phase our task was to analyze the given requirement, document our analysis to fixate on how we are going to implement our system. We also had to provide a timeline to our project and decide which programming language we are going to target. Next step was to figure out the use cases of our system to help us create the database, models and controllers in the backend. Finally, we were supposed to deliver a mockup UI to visualize the actions the user and demonstrate to the client, what the final UX/UI will be like.

During the development phase of sprint1, we were supposed to provide a basic algorithm for detecting plagiarism and setup libraries to parse our files. We also had to finalize the architecture of our project and set-up tools like Jenkins (for quality assessment), Jira, Sonarqube etc. These tools are supposed to help developers in the development process and provide a code base of specific quality measure. The application was supposed to function as a standalone system and was to be deployed to cloud environment like AWS.

During sprint 2, we were supposed to implement at least one sophisticated algorithm for detecting plagiarism and add more features in UI to make the system as user friendly as possible. We also had to follow industry standards and implement certain features to our system like logging, sending mail on encountering exceptions, etc. The system was also supposed accommodate scenario to perform comparisons for multiple files submitted by a student. It was also expected to implement a continuous integration environment.

During sprint 3, we had to optimize our algorithms, implement design patterns and fix bugs to make our system more reliable and fault tolerant. To make the algorithm results more reasonable, we were supposed to compare our plagiarism results against the results of an online plagiarism detection tool, MOSS. We were supposed to adjust weight of each algorithm accordingly. The requirement also included an option for the user to monitor system status, usage and view other statistics.

For the testing part, we had to provide enough test cases for the code coverage to test everything that was implemented. The test cases had to validate the algorithms, classes of controllers and services, and parsing mechanism.

Overview of the result

Our team completed every sprint expectation provided by the client, including the stretches. The team was able to implement a system where instructors can create courses, each course can have homework, and students can submit their assignment files to those homework. The instructors can add students to courses, run plagiarism detection on assignments submitted by the students for a homework, to check if there are any similarities. The instructors can also view and download a report highlighting the lines that were plagiarized. Instructors can also look at the statistics of various plagiarism checks performed. Admins can approve registration requests of instructors, monitor various system usage statistics such as, sessions of users over time, number of plagiarism checks performed over time, etc. Operations team can view and get notified of any change in the status of the system.

Please find below the backlog statistics of all the tickets generated and reported throughout the sprints.

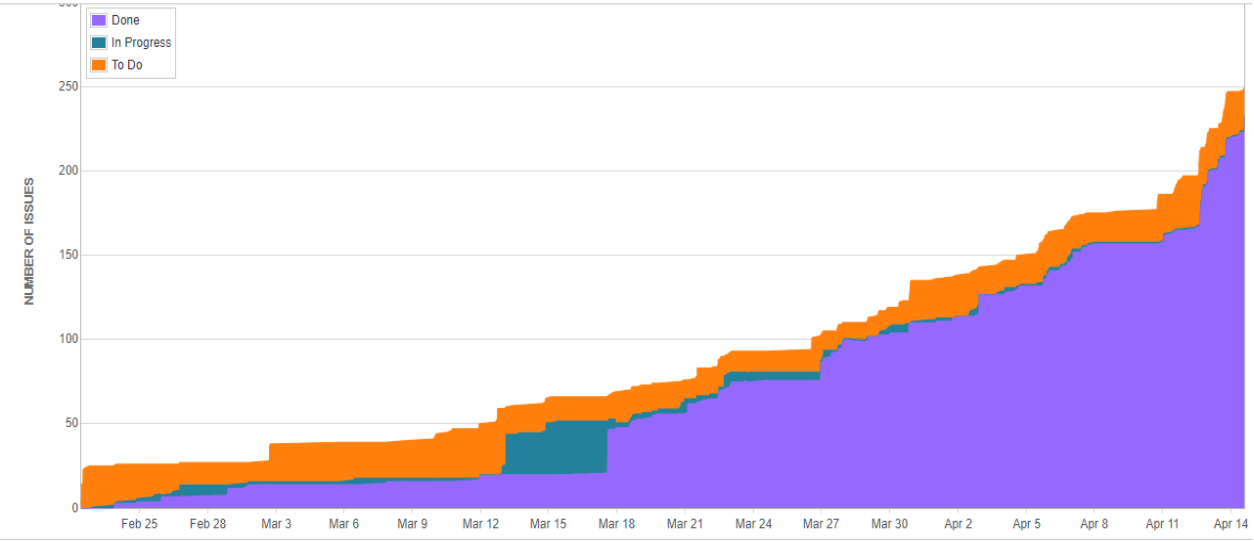

Figure: Details of Jira backlog issues with respect to date

The quality of the code is monitored at every stage to minimize the quality related issues. Sufficient number of test cases were written to ensure that all the code components are working correctly. Edge cases were tested for all the critical modules and the code was refactored to reduce code duplication as well as to increase reusability of each module. Various design patterns (like strategy pattern, factory pattern, singleton pattern) were implemented to facilitate in this process.

Please find below the Sonarqube report of the quality assessment of the application

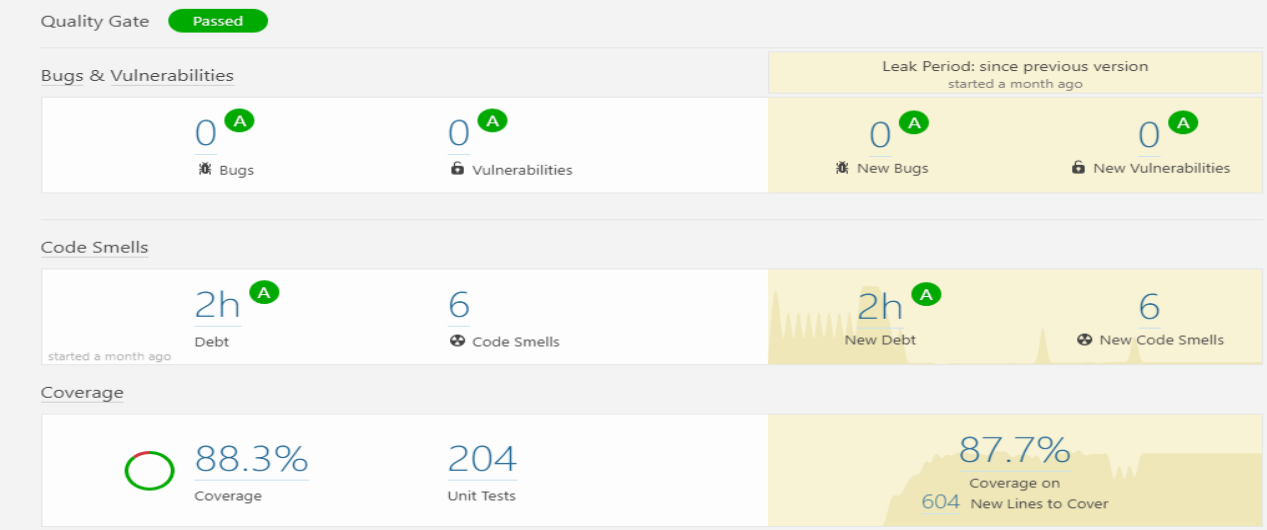

Figure: Sonarqube report of the quality assessment of the code

Development Process

The team followed the Agile development methodology for building the application system, for a span of 4 months lasting from January to April 2018. The entire development involved three sprints, each of 2 weeks. Every sprint consisted of a specific set of goals to be achieved, which resulted in addition of functionalities and features to the system. The project was divided into 3 phases. The initial two phases consisted of requirement analysis, research and system design. The last phase was, in turn, divided into 3 sprints and consisted to actual implementation of code.

Phase-A

In this phase, the team decided the schedule for the scrum meetings spanning the project duration. Every member's availability was discussed, and all the logistics decisions were made. A plan was outlined that set forward milestones for performing different tasks over the course of time. The target language for detecting the plagiarism was finalized and a set of use cases were defined for the system. The team also provided a mocked User Interface of the system to provide a basic understanding of the product it was going to deliver.

Phase-B

In this phase, the team researched some literature on plagiarism detection. It also gathered information for the libraries to use for parsing the source code, to form an Abstract Syntax Trees. The library used to parse the source code into ASTs was ANTLR. The team understood the process of determining plagiarism detection by following the hints and specifics provided by professor in class. After understanding how the product should work, the team created a UML class diagrams and Java Interfaces for the envisioned system. Four Comparison Strategies were decided to be implemented, which are as follows:

- Basic string matching, between two files.
- Comparison between number of identical nodes between two files
- Longest Common Subsequence
- Levinstein's Edit Distance

The strategies of basic string comparison, longest common subsequence and edit distance were hinted by the professor in class.

Phase-C

In this phase, the team switched to development mode and started working on different modules of the system. The phase consisted of three sprints, each lasting 2 weeks. Each sprint was followed by a sprint review with Teaching Assistant. The team did short scrum meetings throughout the weeks during the sprints and decided upon every team members tasks for the sprint. The team used Jira to manage and monitor the work being done by each member. The team utilized tools such as Git for version control, SonarQube for maintaining code quality and Jenkins for automation of tasks and continuous integration of code into master branch.

Sprint 1

This sprint started with setting up environments. This consisted of setting up personal

development environment, SonarQube and Jenkins integration for continuous monitoring of quality for the code being pushed to master branch, integration of Jira with Git repository for processing of smart commits. All the above tools were in turn integrated to slack to inform the team of any errors encountered during the processing. The team fulfilled all the sprint expectations, including the stretches. The team implemented a partial UI of the system, constructed an AST from a file, a basic comparison strategy was implemented.

Sprint 2

In this sprint, the team was tasked to implemented at least one sophisticated comparison strategy. The team was successful in implementing two strategies which were longest common subsequence and Levinstein's edit distance. The team also incorporated strategy pattern to accommodate different strategies. A weighted function was implemented that was trained through MOSS for calculating a more refined plagiarism percent. The components of user interface were developed which displayed the data provided by the backend. A student user was provided the functionality to upload multiple files multiple time. The system also logged activity of various tasks performed. All sprint expectations including the stretches were met.

Sprint 3

In this sprint, some additional features and functionalities were introduced such as snippet generation, admin approval for registration of a professor and system usage statistics. In addition to this some improvements and refactoring of the code was done, tests cases were expanded and sample homework submissions by students were created for testing. Any bugs or broken code detected was fixed. The system was also being prepared for final packaging for delivery.

After sprint 3, the team met a few times to discuss any tasks that needs attention and worked on bugs pointed out by the testers. The team worked on the final report and the final presentation of the project and packaged everything according to client's expectations for final delivery.

Retrospective of the project

There are many things that the team liked and appreciated in this project. First, by using Sonarqube, the team was able to get a constant feedback on the quality of code being developed and checked into the master branch. Because of this, it was easier for us to scale up our system and reuse specific components that were being duplicated throughout the system.

Secondly, we followed the agile development process which helped us to complete our project in a limited time. We learned the agile methodology which gave us a glimpse of the development process followed in the industry. Understanding and meeting the client expectations, building a system from scratch, encountering and fixing problems throughout the development process was helpful for getting accustomed to the code development work.

Then, we were supposed to implement a set of functionalities and features in each sprint. This helped us organize, set goals and work accordingly. Thus, we learned the art to divide tasks, manage time and resources to achieve the goal specified.

Finally, we really appreciated the in-person sprint reviews. It helped us figure out the areas of improvement and provided an alternate approach for encountering any existing problem.

However, we felt the slack communication was not very useful after Jenkins integration. It was easy to lose important messages in the pool of Jenkins alerts.

Overall, we learnt how to work in a team, follow agile methodology to build a product, rectify our errors in early stages and testing our application thoroughly. We also learned how to understand and comprehend client requirements, visualize what could work and go wrong in the development process, test an application. Finally, we learned how to deliver a system that adheres to industry standard, which not only can be enhanced by future developers, but also can be scaled up and reused.