Plagiarism Detection System

CS 5500 – Managing Software Development

Northeastern University
College of Computer and Information Science

Team Members:

Frenia Pinto

Vaibhav Shukla

Vaibhav Rangarajan

Northeastern University
CS5500– Managing Software Development
Spring 2018, Jose Annunziato and Michael Weintraub

PROJECT REPORT

Problem Overview:

Unfortunately, plagiarism is a major problem at many schools.

We have considered the specific problem where two or more people produce different versions of the same code.

The goal of this project is to design, implement, test, and evaluate an application to help instructors detect situations where two or more students submit similar solutions to an assignment, in which one version derives from another version through one or more behaviour-preserving transformations such as renaming variables, methods or moving methods to another location in same file etc.

Scope:

Our plagiarism detector works on programs written in python language and the code should have been uploaded to a “github repository”. The system is a web application which uses “HTML, CSS, Bootstrap and JavaScript”. The majority of implementation of plagiarism detection algorithm is implemented in “Java” and the system used “postgresql” database.

The detector detects plagiarism beyond just textual diff and considers more sophisticated transformations such as but limited to:

- ❖ renaming of variables
- ❖ extracting code into functions
- ❖ moving code around

The detector also handles multi-file projects, where files with similar content have different names.

Result Overview:

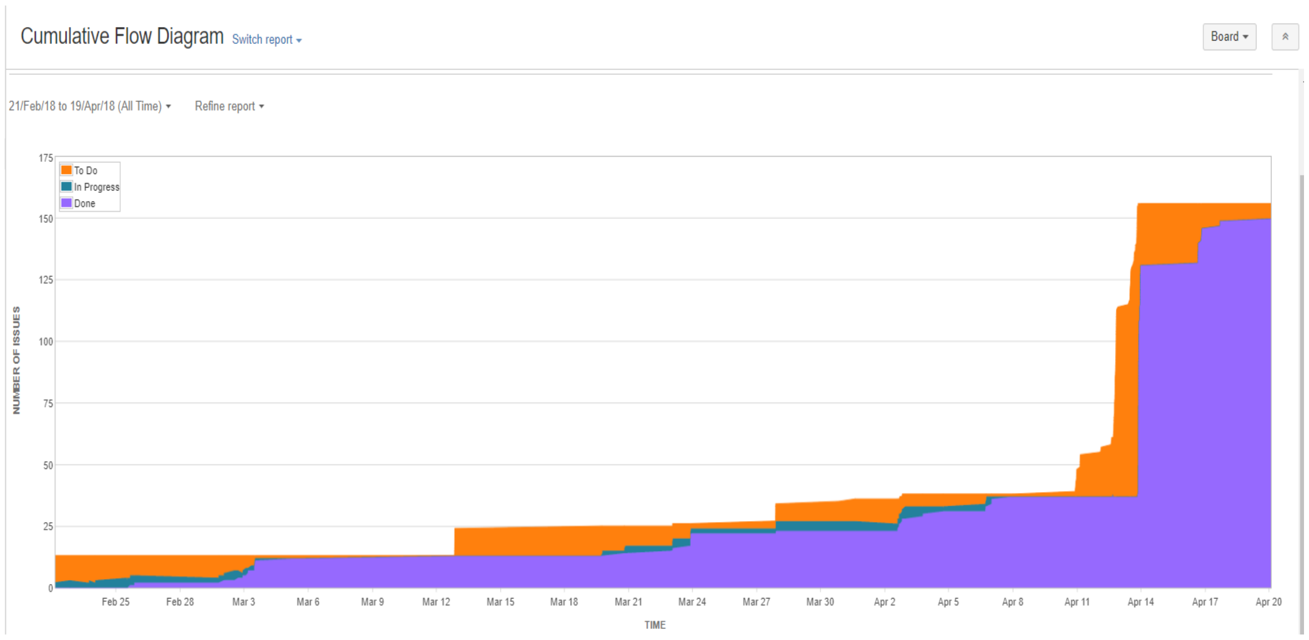
The system accomplishes all goals covering the base and stretch expectations. The system has been tested rigorously to check for different kinds of plagiarism as defined in the scope.

Implementation Assertion:

The development of functionalities can be confirmed with the Jira backlog;

- Fully functional UI (CS110-22)
- Plagiarism computed using more than one comparison strategy and using a weighted polynomial function (CS110-17)
- System logs are recorded (CS110-23)
- As part of environment,
  - Jenkins runs on all pull request submissions (CS110-10)
  - Generate system quality report using SonarQube (CS110-11)
  - Project using maven structure (CS110-1)
- System trains the plagiarism function using Stanford's MOSS as performance standard (CS110-18)
- System displays usage statistics of the plagiarism detection system(CS110-29)

Jira Cumulative burn down chart:



Quality Assertion:

The quality of the system can be measured from the test cases and also the SonarQube quality gate.

Link to our system's SonarQube report: http://128.31.25.108:9000/dashboard/index/edu.neu.msd:plagiarismdetector

- The system has passed the SonarQube quality gate with test coverage of 90.2% and 51 unit tests.
- All the Java programs are supported with Javadoc and function description.
- The system has received a reliability, security and maintainability rating of 'A' in SonarQube.
- The system is tested with test cases of various scenarios such as full, partial and no plagiarism and all essential part of the algorithms have been exercised and also the services have been tested with mockito.
- All the bugs raised by the testers have been fixed.

Development Process Overview:

- Our team followed a modified version of agile using scrum. We had stand-ups in person whenever feasible and when not we had slack-ups to update the status and progress.
- The TDD(Test Driven Development) approach was used to implement many modules of our system.

Project Roadmap:

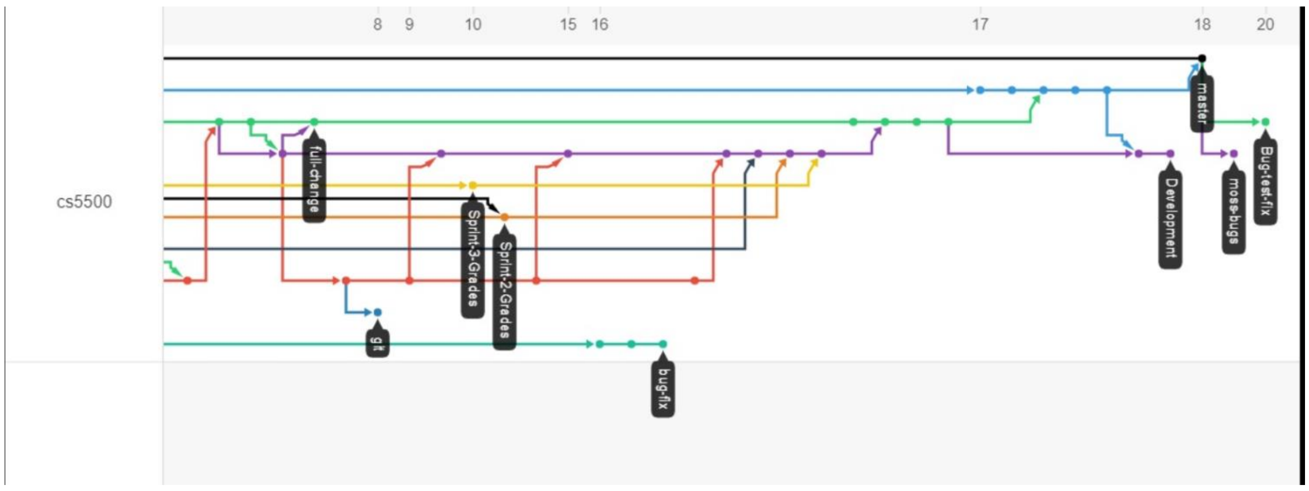
- The project was carried out in 3 phases;
  - Phase A:
    - Requirements were identified.
    - Use cases were designed.
    - System mock-up of UI was designed
  - Phase B:
    - UML diagrams were drawn.
    - Java interfaces were developed.
  - Phase C:
    - Implementation of system.
    - Testing and documentation

- Phase C (Implementation) – Divided into 3 sprints with sprint overview
    - Sprint 1:
        - Development of 2 use cases with basic login functionality
        - AST generation and Maven project structure setup
        - Environment setup.
    - Sprint 2:
        - Comparison strategies development
        - Working UI for comparison and plagiarism detection
        - System logs
    - Sprint 3:
        - Fully functional UI
        - More than one comparison strategy with a weighted polynomial function and learning based on Stanford's MOSS.
        - Usage statistics of system.

GitHub and Quality process:

- The team used Github for version control and a separate branch was created for each feature implementation.
- Once each development of feature branch is unit tested by the developer, it was promoted to a Development branch.
- After a particular complete functionality has been developed, it was merged into the Development branch to avoid redundant code being merged directly to master.
- Upon passing of all test cases a pull request was created to merge the Development with master.
- For each pull request, Jenkins runs a set of tests to make sure there are no conflicts.
- The final stage of Jenkins is the SonarQube quality gate which assesses the system quality based on test coverage, bugs, vulnerabilities etc. and allows the code to be merged to master only when it passes the SonarQube quality gate.
- Once the code is merged to master, the code is deployed to AWS.

Git working on Branch:



Jira for Project Management:

- ❖ The team used Jira to manage tasks assigned for each sprint.
- ❖ For each sprint the expectations would be framed as a ticket in Jira and a developer would be assigned to it.
- ❖ Once the development for a particular ticket is done, it will be closed and hence all progress of the team can be tracked through Jira.
- ❖ All the bugs and backlogs can also be tracked through the team's Jira page.

What worked?

- ➢ All the requirements for the sprints were met and the team was going in a good pace showing progress. All base and stretch expectations were met.

What didn't work?

- ➢ The team had some environment memory issues which were later resolved with the help of peer students and discussion on Piazza.

Brief Retrospective:

- The team had a lot of learning out of the course. Despite the three of us having work experience we had an opportunity to work on all latest project management tools and software.
- We got a better understanding of all theoretical Software development concepts by implementing it practically in a project.
- The course load was a bit heavy considering only 3 members in our team.
- We enjoyed working in a team and also had a good learning from each other.
- We did a mock trial of the system by each of us writing a code and the system gave pretty accurate results.
- The tagline of the course that, not only good programming is enough to quantify a software good but there are various quality measures that have to be followed, is instilled in each of us.