Overview of the Problem

In a Utopian world, there are professors and students in a University and everyone performs their tasks with diligence. The graders, TAs are non-partial, there is no bias. However, that is far from the truth and the truth matters. The goal of the project is to design, implement, test, and evaluate an application to help instructors detect situations where two students submit plagiarized code as their submission for an assignment.

Scope (Features, Functionality, Performance)
To prevent students from copying material available on the web, there are systems in place. One is the plagiarism checker for checking code from computer science students' submissions. There are several plagiarism detection software systems in place like TurnItIn, MOSS and JPlag. However, each of these has some limitations. TurnItIn does not check code, MOSS is not usable without an account, Jplag requires using the Front end developed by the team that build JPlag. Our team's objective is to build a plagiarism checker that checks for the similarity between two input files/folders for python files that are submitted by students. The students can choose to submit a single file or multiple files in their submission and to build a UI that allows the professor to compare the submissions of students from the UI. The students need to be able to upload files to the UI. The professor must be able to see the results of the comparison between student submissions on the UI and get an email with the pdf of the report generated by the system.The team uses sonarqube for checking the quality of the code that is pushed to master. Sonarqube is integrated with Jenkins and the report is available at a specific URL every time code is pushed to the development branches and master. The team uses JIRA for issue tracking and Jenkins for continuous integration, GitHub for version control. The use of these systems is enforced as mandatory by the teaching team. We work in teams of four students. The team is decided by the instructors at the start of the semester.

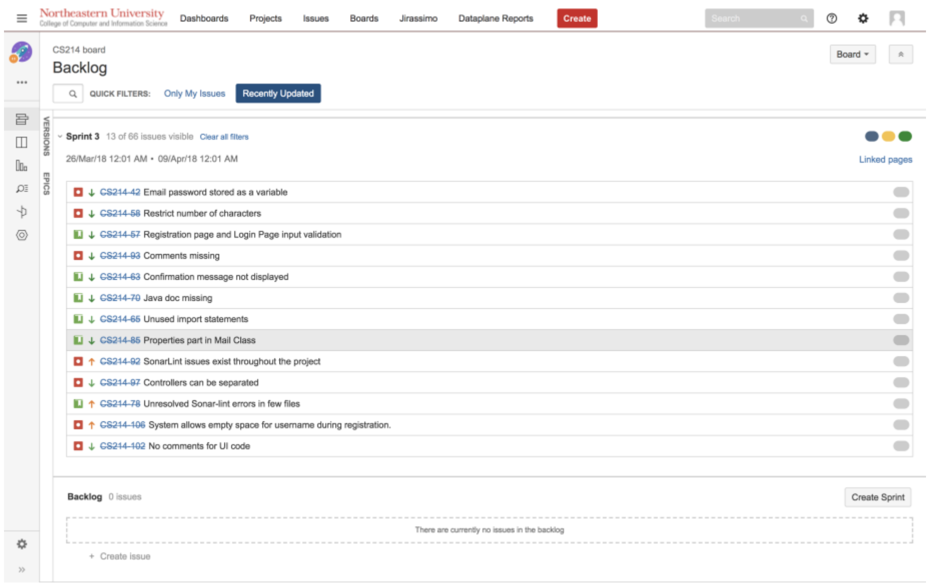
Cost (Resources, Budget)
The resources available for this project include the JIRA server provided the instructors, and AWS credits available to Northeastern students.

Time (Schedule, Deadlines)
The team uses agile methodology and works in three sprints of two weeks each with a sprint review with the TAs after every sprint to go over the progress of the team.

Phases
The project has three phases.
Phase A requires developing a set of use cases that informally describe the informally describe the use of the system. We also need to provide a mockup of the system in this phase.
Phase B requires creating UML diagrams to reflect the high level architecture of the system and java interfaces reflecting this design.
Phase C requires implementing the system in java.

Overview of the Result

The system effectively detects plagiarism between student submissions for python files.
The students can login to the UI and select the assignment they want to upload files to. They can upload single or multiple files for each assignment. The professor can login to the system and choose the comparison strategy he wants to implement and see the results page with details about which files are detected to be plagiarized.
From the JIRA dashboard it is clear that all backlogs have been effectively cleared.



CS214 board
Backlog

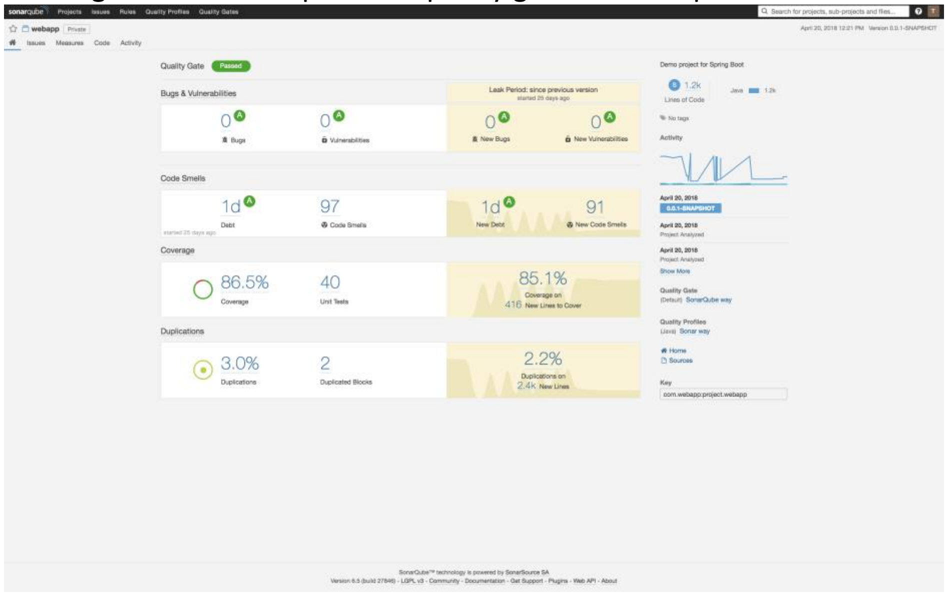
QUICK FILTERS: Only My Issues Recently Updated

Sprint 3 13 of 66 issues visible Clear all filters
26/Mar/18 12:01 AM • 09/Apr/18 12:01 AM

~~CS214-42~~ Email password stored as a variable
~~CS214-58~~ Restrict number of characters
~~CS214-57~~ Registration page and Login Page input validation
~~CS214-93~~ Comments missing
~~CS214-63~~ Confirmation message not displayed
~~CS214-70~~ Java doc missing
~~CS214-65~~ Unused import statements
~~CS214-85~~ Properties part in Mail Class
~~CS214-92~~ SonarLint issues exist throughout the project
~~CS214-97~~ Controllers can be separated
~~CS214-78~~ Unresolved Sonar-lint errors in few files
~~CS214-106~~ System allows empty space for username during registration.
~~CS214-102~~ No comments for UI code

Backlog 0 issues
There are currently no issues in the backlog

The quality of the code has satisfied the checks enforced by the teaching team. The results from the SonarQube report generated validate our claims. The entire code must have 85% code coverage to be able to pass the quality gate of sonarqube and we have achieved that.



April 20, 2018 12:21 PM Version 0.0.1-SNAPSHOT

Quality Gate Passed

Bugs & Vulnerabilities
0 A Bugs
0 A Vulnerabilities
Leak Period: since previous version
started 25 days ago
0 A New Bugs
0 A New Vulnerabilities

Code Smells
1d A Debt
97 Code Smells
started 25 days ago
1d A New Debt
91 New Code Smells

Coverage
86.5% Coverage
40 Unit Tests
85.1% Coverage on 416 New Lines to Cover

Duplications
3.0% Duplications
2 Duplicated Blocks
2.2% Duplications on 2.4k New Lines

Demo project for Spring Boot
1.2k Lines of Code
Java 1.2k
No tags
Activity
April 20, 2018
0.0.1-SNAPSHOT
April 20, 2018
Project Analyzed
April 20, 2018
Project Analyzed
Show More
Quality Gate
(Default) SonarQube way
Quality Profiles
(Java) Sonar way
Home
Sources
Key
com.webapp:project.webapp

SonarQube™ technology is powered by SonarSource SA
Version 6.5 (build 27846) - LGPL v3 - Community - Documentation - Get Support - Plugins - Web API - About

Overview of the team's development process

The team followed the Agile methodology for development of the software. The team conducted Daily Scrum Update on Slack where we answered
1.What we did the prior day.
2. What we are planning to do that day.
3. If we are stuck on something or blocked by someone else on the team.
In project meetings, we checked the status of the development with the members of the team.
In addition to project meetings we also had review meetings with the Teaching Assistants to go through the progress after every sprint.
Minutes of the meetings were recorded on Slack.

The steps required to be followed to successfully deploy new updates to the project included:
1] Creating an Issue on JIRA
2] Committing changes on the local development branch using smart commits
3] Pushing the changes to the local branch
4] Approving the pull request after status checks have passed to allow merging to master.
5] Checking the sonarqube report to check for any bugs/code smells/test coverage.

In the begging of the project we faced difficulties in setting up the complete environment. If someone was facing a problem we sat together as a team and tried to resolve the issues. For all issues related to Jenkins and Sonarqube setup, we would like to thank Alex Grob for his help throughout the semester. We troubled him a lot for setting up Jenkins.

In every meeting that we had as a team, we decided a time frame for completing new tasks that were allocated for that particular sprint.

A brief retrospective of the project

The real problem lay not in choosing which option to pick, but which options to consider for building the project. (technology stack, continuous integration, issue tracking, …)
The Instructors made this task easy for us when they made it mandatory for us to implement Jenkins, to automate checking the code that gets pushed to master. Master was required to be protected. Other than that, we faced the question of choosing to explore new ways to implement the project vs using the standard. The burglar problem came into existence again and we decided the number of times we should try robbing the bank, depending on whether our team members were professional thieves or newbies.

If there were tasks that were of low importance that were blocking tasks of high importance, then for that time the low importance tasks got the priority of the high importance tasks that they were blocking. This allowed us to make rapid changes in our plans, as expected in the sprints.
There were times in our project when we could not complete the tasks assigned to us. At those times we used the regret minimization framework where not performing tasks incurred some penalty. Our goal became to reduce the weight of the penalty that we incur at the end of the sprint cycle.

We also faced the problem of choosing to perform a small task when it was small vs waiting when we actually needed it. We made the choice as a team to not allow the problem to escalate to increasing proportions and complete the tasks when they were small.
Even after following all these precautions there were times when the tasks assigned to the team members could not be completed. In those situations, we practiced constraint relaxation to get over the tasks assigned to the team.

All in all, we learnt a lot about managing development. We used the Cobham Edmonds thesis several times to decide which problems were tractable vs which were not tractable for the team to complete on time.