Project Report

CS5500: Managing Software Development
Professor Michael Weintraub

23 April 2018

Team Members:
Aditi Kacheria
Mansi Jain
Joshua Dick
Rohan Gori

I. Summary of Project Goals

Our goals for the project are divided into the following categories:

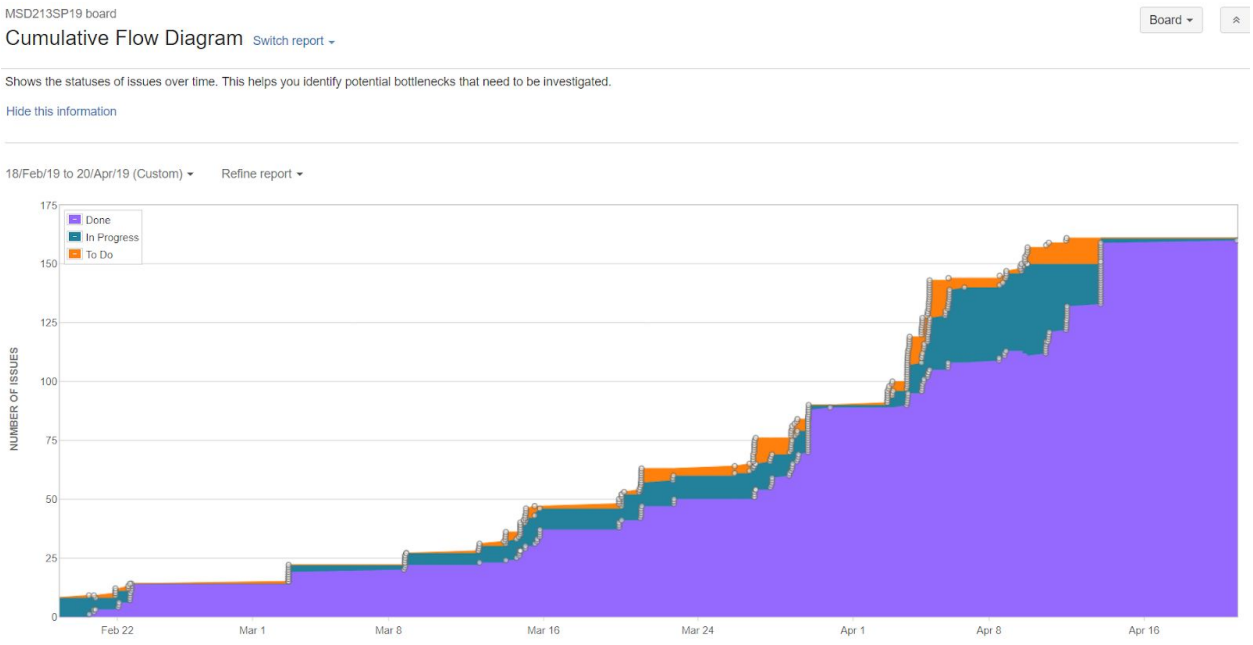
1. User can:
    a. Login/ Register
    b. Create their User Profile, view their profile, edit their profile
    c. Set status as DND (which will prevent them from receiving notifications)
    d. Follow/ Unfollow other users
    e. User can view their followers/ users that they are following
    f. Search for other users
    g. Deactivate/ Delete their account
2. Groups:
    a. Users can create groups
    b. Admin of the group can make another user an admin (multiple admins)
    c. Admin can remove users from groups
    d. Group members can view list of users in a group
    e. User can view all the groups that they are a part of
    f. Admin can set group restrictions to High or Low
    g. A member of the group can leave the group
    h. A member of the group can delete the group
    i. Members of the group get notified when another user is added/ leaves the group
3. Hashtags:
    a. Users can create a hashtag and post to a hashtag
    b. User will be notified of the activity on a Hashtag of the users they are following
4. Messages:
    a. User can send/ receive messages to another user/ group
    b. User can reply to a specific message
    c. User can forward a message
    d. User can set a message as a Secret (secret messages cannot be forwarded)
    e. User can delete/ recall a message
    f. Users can retrieve a chat/ a chain or thread of messages
5. CALEA compliance functions
    a. CIA user can wiretap/ untap a user
    b. CIA will get a live notification of any message that is sent or received by a wiretapped user
    c. CIA can retrieve all the communications of a wiretapped user
    d. CIA can view a list of wiretapped users
6. Caching of groups and chat ids for faster retrieval
7. Incorporation of Password hashing for increasing security
8. End to end message encryption
9. Deploy the system on AWS instance
10. Incorporating a help menu to facilitate ease of use for the users (while using the system's features).

II. Overview of Result

The final product delivered is a Slack like communication platform with advanced features as described in the Project Goals. The result of the development process is a low latency, highly scalable system with clean code and considerable code quality.

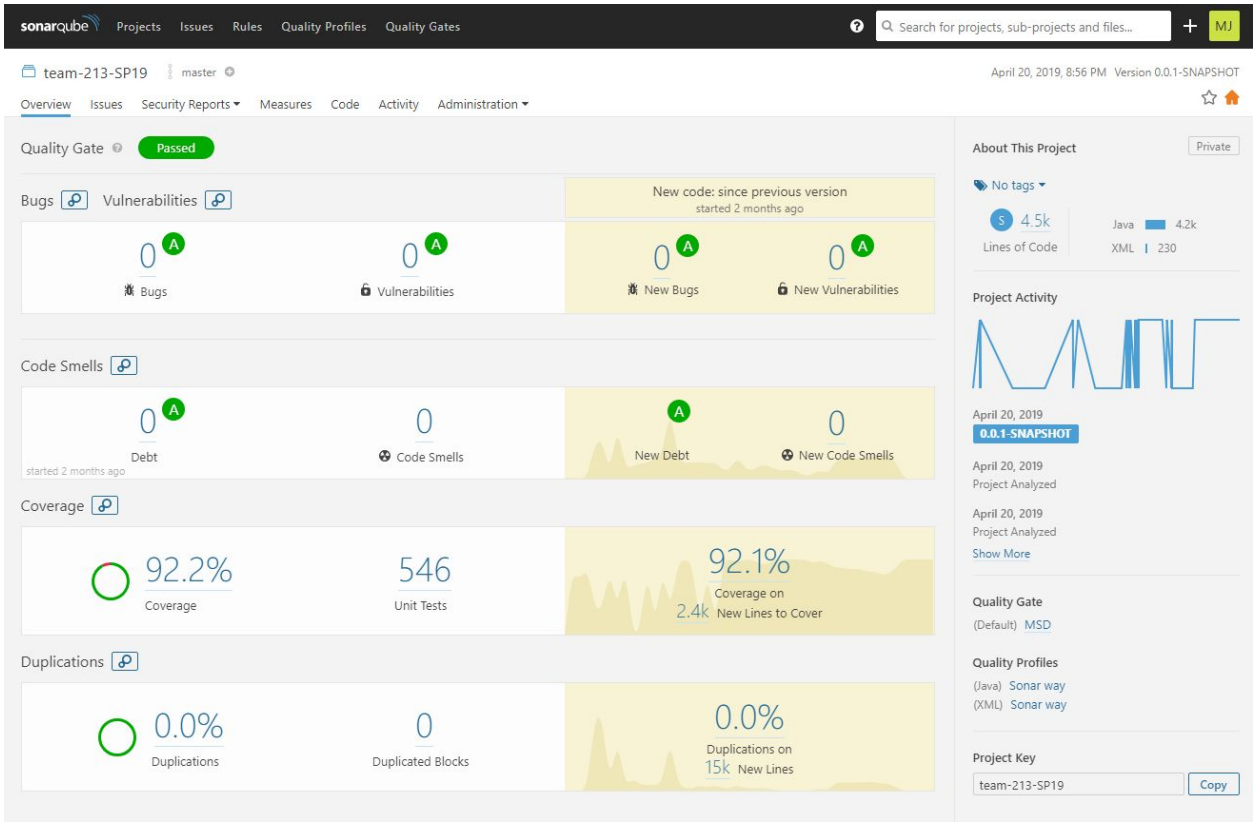
Backlog Statistics:

All of our goals have been achieved. We successfully delivered more than what was promised in each of the sprint goals. The cumulative flow diagram captures the project's progress in development over the course of four sprints providing evidence that all backlogs have been completed that were assigned to each member. As a team, we have been very active in tracking our work.

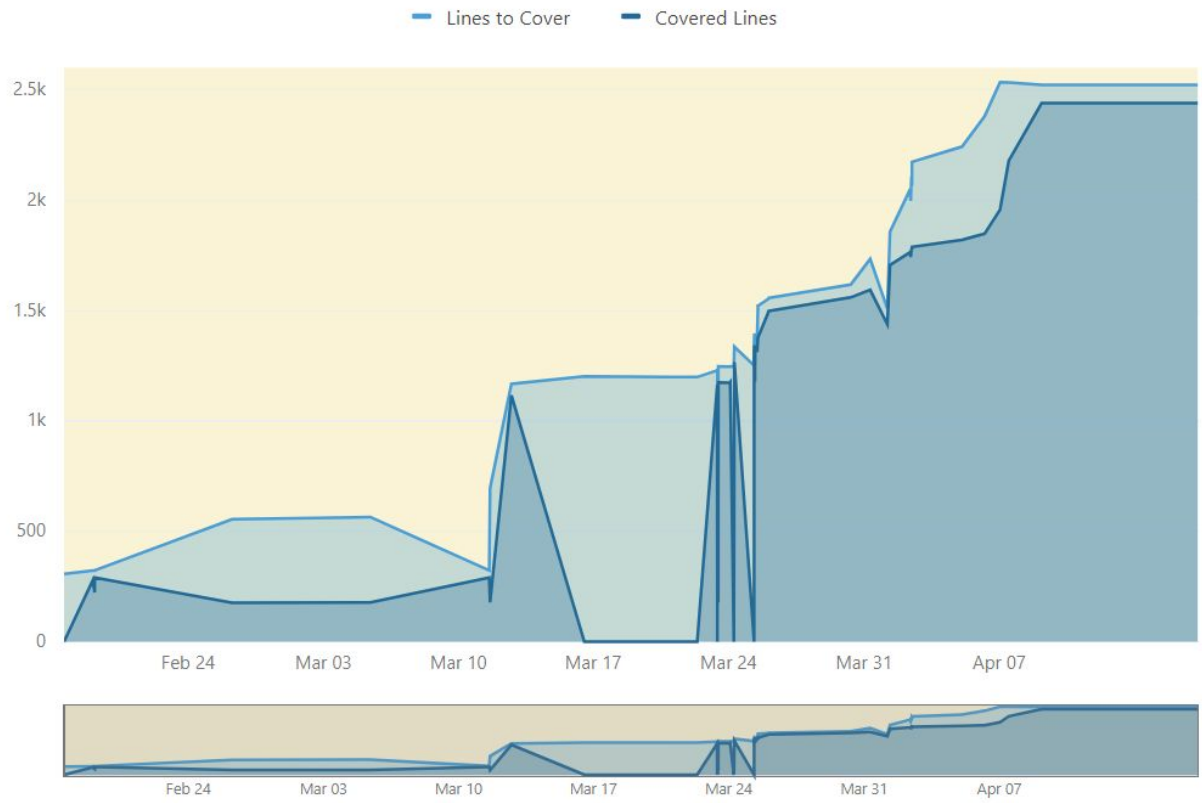


Quality Claims:

Overview:
As seen here, we currently have no bugs, vulnerabilities or code smells and have achieved over 92% code coverage.

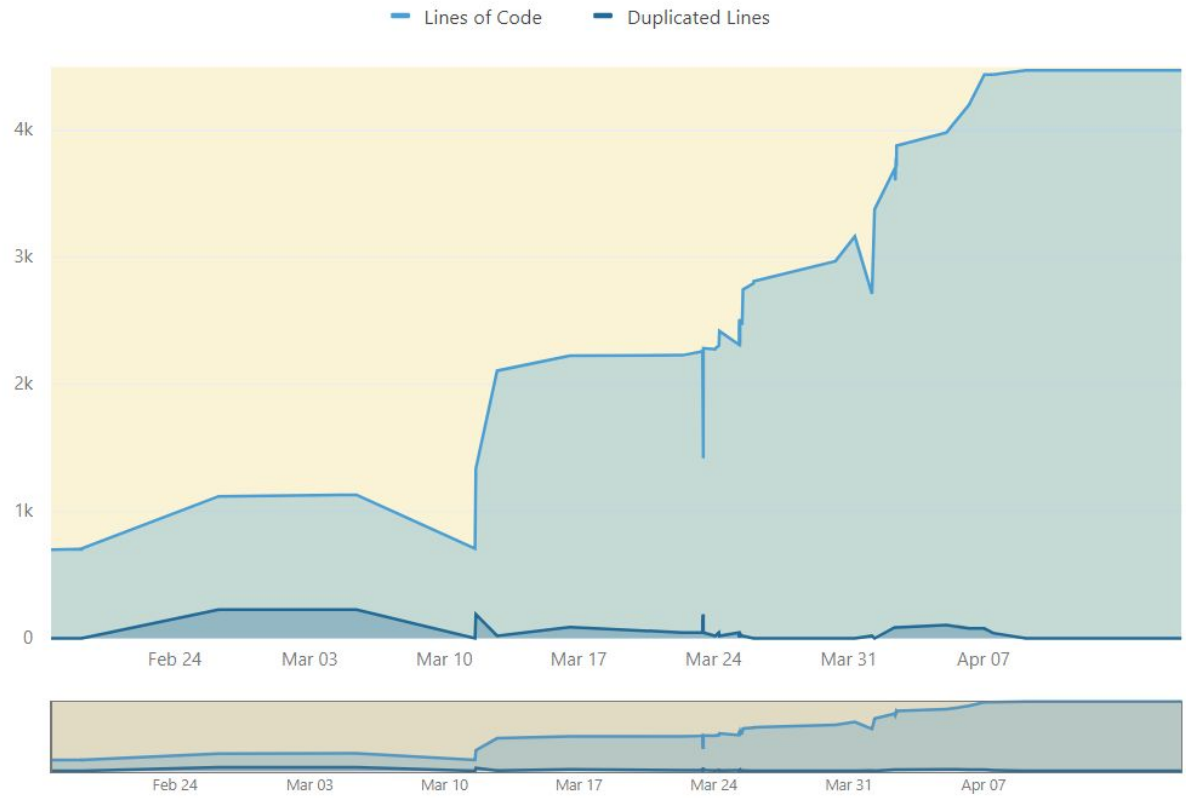


Code Coverage Report:



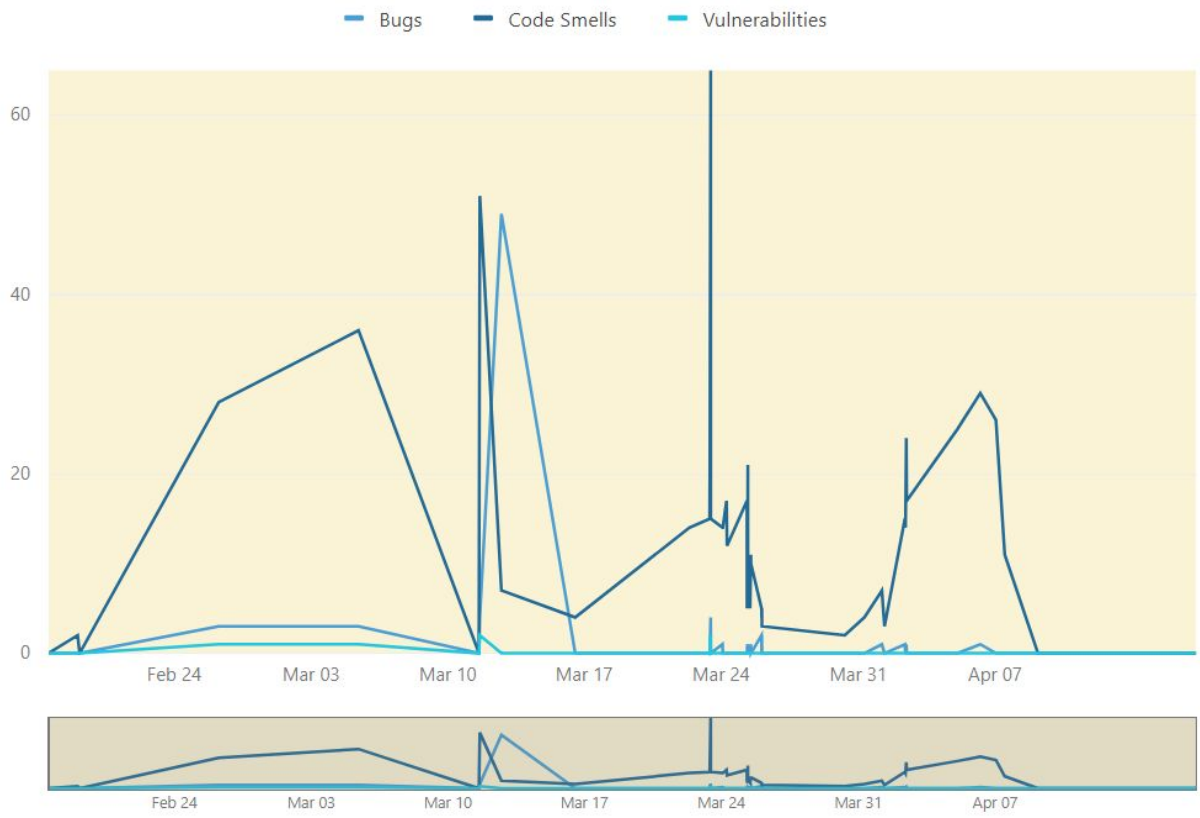
As shown in the report above, we have always been consistent with the quality assurance of our code in terms of code coverage. The spike and drop in the coverage result in between the graph was a result of migration of our test cases to use mock database objects instead of actual database connection.

Duplicate Code Report:



As delineated in the report, the level of duplication in our code was always minimal, and went down to zero at the end.

Issues Report



We managed to deliver our code with zero bugs, code smells and vulnerabilities. The spikes in the graph pertained to bugs corresponding to database connections and cognitive complexities, which we managed to eliminate completely by incorporating various design patterns in our code.

III. Team Development Process

We followed the Agile methodology for developing this software. We discussed our availabilities in the beginning and found that evening worked best for everyone for team meetings. We used Slack as our main communication platform.

Our team meetings were structured in the following way:

1. Meet up on the first day of the sprint. Scrum master leads the meeting. The team discusses the sprint goals pulled from the product backlog, suggestions from the client about the previous sprints, improvements to be done etc. We decide on the day and time to meet next.
2. Following meetings throughout the sprint, we started off deciding the goals for the meeting, then continued working on our tasks, discussing, troubleshooting and collaborating on ideas and features. Scrum master would monitor the team's progress and provide updates on any communication with the client/TA.
3. The sprint ended with a sprint review where the team demonstrated the product to the client, gained feedback on our performance and suggestions for improvements.
4. After the sprint review with the client, the team conducted a sprint retrospective meeting discussing what could have been done better in the current sprint, along with prospective process improvements for the future sprints.

Upon prior discussion about each team member's skill set, we decided to divide the work into two parts: the database logic (worked on by Aditi and Mansi) and the implementation logic on Prattle server side(worked on by Rohan and Josh).

Our team development process was structured in the following way:

1. Discuss what issues/features to work on.
2. Divide the work and coordinate on any dependent features.
3. Create our own development branch for the sprint.
4. Create jira tickets for individual work. (Epics, stories etc.)
5. Code, test, and work in pairs where required.
6. Review code for pull requests.
7. Check sonarqube reports for code smells, bugs, vulnerabilities and coverage and fix them respectively.

Overall, this methodology proved to benefit us in delivering a product at the end of every sprint that not only passed the quality gate, but also contained features which were above and beyond client expectations.

Changes made along the way based on what didn't work:

1. After the first two sprints, we made a conscious effort to maintain the quality gate. Instead of struggling to achieve a decent code coverage near the deadline, we decided to keep it significantly above.
2. To keep up the work pressure contained, we set up internal deadlines for deliverable features from sprint-2 onwards. This resulted into a hassle free delivery at the end of the sprint, since we were quite comfortable with our completed features.
3. From sprint - 2 onwards, we incorporated a feature-test rule, which mandated completion of the tests for a feature, the moment a feature was developed. This resulted in us passing the quality gate with every feature that we pushed on our sprint-branch.
4. From sprint - 1 onwards, we also mandated minimum 90% coverage for our code instead of the 85% coverage as specified by the client. This helped us in having an exhaustive regression pack, and also resulted in us delivering a flawless product at the end of the sprint.

IV. Retrospective

What did the team like best:

1. We were given the opportunity to build a potential competitor to slack with some really interesting features.
2. Teamwork. Each member made essential contributions to ensure successful completion of the project and worked well with as a team.

What did the team dislike:

1. The legacy code was not an easy task to deal with.
2. The heroku database was a pain to deal with when it started giving out max connections error.
3. The test coverage was quite difficult to achieve on the legacy code

What did we learn:

1. Using Mockito/Powermock was relatively new for us.
2. We learned and implemented a cache-based architecture for most of our features. Where we cached most of our static data from the database, and managed consistency between database and cache. This helped us deliver a low latency product.
3. Most importantly, we learned that completely relying on sprint design can result into delay of our end product. The team must incorporate internal deadlines within the sprint to achieve an end product by the specified deadline.
4. Also, at an early stage in our project development, we learnt that leaving code coverage for the end might be peril for the product and the team. The complete coverage guarantees(99%) that the feature works fine. Leaving coverage till the end might result into the team discovering new bugs by the end of the sprint, which might be difficult to fix by the deadline.