Project: Slick Communications

CS 5500 Managing Software Development

Final Report

Team 111

Xia Yang

Pratik Devikar

Farha Binte Jawed

Table of Contents

1.0 Project Goals

The goal of the project is to enhance and test the new communications server of CCIS so that it delivers a slick set of communication services and scales to high demand. The project aims at developing the "server" side. While the legacy code provided the basic messaging capabilities, it lacked following features that were included in this project to make the system operational and useful:

1. **Notion of users and groups to the system:** It enhances user and group management by including basic create, read, update, and delete capabilities. All the entities persist in the LDAP server. Users have the option to login to system with username and password where password is encrypted to ensure system security.
2. **Broadcasting messages to individuals and groups:** Messages can be directed to an individual. If it is a group message, it provides a reply-all function that allows broadcasting the sent message to all members of the group.
3. **Message persistence and search:** All messages sent to individual and groups are stored in the LDAP server. Messages can be searched based on sender, receiver and timestamp.
4. **Message queuing:** Messages are queued for offline users. Queued messages are delivered to the targeted user when they come online. These messages are delivered in the order they were sent based on time.
5. **Duplicating messages:** Messages can be duplicated and forwarded to an agency. It is done by creating a read-only channel. No one involved in the original communication can determine if their communications are being sent to the agency. Messages forwarded to an agency are wrapped with sender and receiver's IP address maintaining CALEA[1] compliance.
6. **Recall messages:** Messages can be recalled by creating a recall message by message sender. Recalled messages are delivered to clients and not delivered as part of message backlog delivery.
7. **Parental control of messages:** It provides a parental control feature for online and offline messages which looks for words and phrases with vulgar content. Inappropriate messages are blocked and saved in the LDAP server with a flag.

In terms of environment, the project aimed at fulfilling the following goals:

1. Test coverage: More than 85% code coverage is achieved
2. Integration environment: The system is deployed to cloud environment using AWS
3. Slack integration: PR from github and failure notices from Jenkins are sent to Slack
4. Git integration: Smart commits in Git are enabled
5. Issue tracking: jira.ccs.neu.edu is used to track issues

[1] https://www.eff.org/pages/calea-faq#2

2.0 Overview of Results

The team accomplished almost all of their goals of the project of creating a chat system with plethora of functionalities. Provided an acceptable user interface for the clients, they can now send messages to other users as well as to groups of users and can avail a lot of messaging functionalities.

Success was measured on the following factors:

1) **Number of functionalities implemented**
   Since the first sprint itself, the team implemented all of the features that were asked including the environment settings. For the first sprint, our task was to get the legacy code running. Being familiar with the code and network programming, we picked up the pace for sprint 2 and sprint 3. Sprint 2 included implementing features such as CRUD (create, read, update,delete) operations for users & groups and provided a component for inter-user and group communication by sending and receiving messages. In Sprint 3, we carried out more of the complex tasks such as delivering messages to offline users, duplicating messages for an agency and added a parental control.

2) **Quality of the implemented functionalities and the overall process**
   The main goal of our team was to conduct the process of software development smartly. Our team most often used smart commits for Git and managed our work on Jira.
   As opposed to the popular databases out there, our team went on to lesser-known but highly efficient LDAP server for storing our data. Also instead of implementing just the explicitly stated requirements, we also went ahead and added some more extension to them. For example, we provided a way to search messages by not only the name of the sender and receiver but also the timestamp.

Completion claims by backlog statistics:

The result of implementing the features with each of the sprint was done carefully with the help of the sprint issues and backlog which were well-maintained in Jira.
According to the Control Flow Diagram in Jira, the first sprint expectations in which we had to test the legacy code and get it running took the most time because the of the complexity in understanding the basic requirements of the Prattle and ClientRunnable. However, we picked up the pace for 2nd and 3rd sprint.
According to our ‘Created vs Resolved Issues’ report we completed 34 out of the created 37 issues (92%).

According to the Pie Chart report, the following are the % of time taken by each sprint:

Sprint	Percentage of time taken(%)
None	29
Sprint 1	13
Sprint 2	35
Sprint 3	21

Quality claims by testing statistics:

Testing was one of the essential components of this project and we made sure our code is well tested. Every new functionality created was tested using JUnit and we made sure that each unit created and the overall testing coverage both had more than 85% test coverage. In the end, our test coverage was 86.6%. The rest 13.4% code was not covered because it is rather difficult to test some of the exceptions.

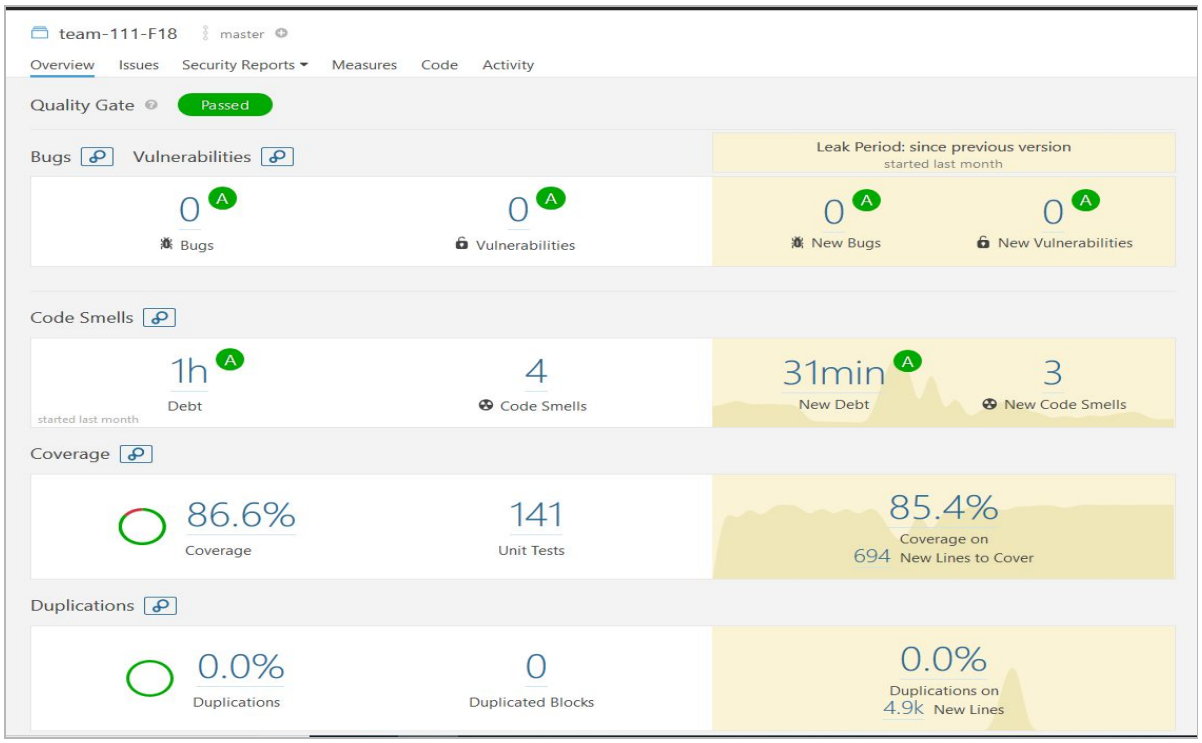

Fig: SonarQube report after our final commit

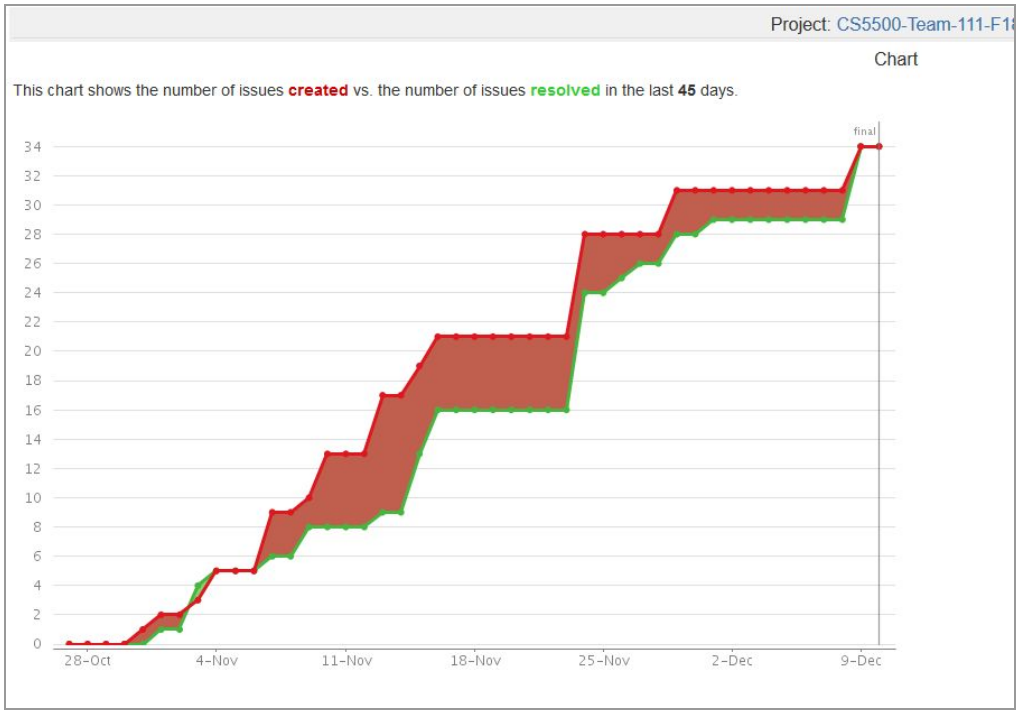

Fig: Graph of issues Created vs Resolved

3.0 Development Process

What works:

Generally speaking, we are able to deliver an instant message application with the following functions:

1) Creating users and groups: Persist users and groups in LDAP. User can login with username and password. The password is encrypted.
2) Sending message to users and groups: Message will be delivered instantly to online users. For the offline users, we will queue up the messages and send to the user in order once the user comes back online.
3) Host the server on AWS.
4) Recall message: Messages can be recalled by creating a recall message by message sender.
5) Parental control: Detects bad words and blocks the message.
6) Enabling agent for message duplication: Messages can be duplicated and forwarded to an agency. The duplicated messages will be wrapped with the IP of both sender and receiver.

We developed all the functions mentioned above with the help of the following:

1) Jira: Assigns tickets and smart commits.
2) SonarQube: Test coverage and code style check.
3) Jenkins: Automate the pipeline of build, test, and deploy.
4) Github: Version control tool to help us collaborate.
5) Slack webhook integration: Send notifications to group channel.

What doesn't:

Although we have accomplished most of the functionalities, due to the fact that we are a 3-member group, there are some defects that we should put more effort on:

1) Testing of CRUD LDAP: Right now we are testing the part using mockito. This is the common practice for testing database related functions, however, as the TA suggested, we should also set up a testing environment like MongoLab, on the cloud service and test the actual creating and deleting methods. This has been done because of the following issues.
2) LDAP Server: We failed to connect to the cloud LDAP server. One of the team members installed Open LDAP on his AWS machine but was not able to connect to it. Firewall issue might be the cause of it. Unlike RDS or MongoDB, we did not find much documentations on LDAP, nor did we have enough developers to tackle it. But since we have tested all the functionalities on local LDAP, everything should work as expected once we are able to connect to the LDAP server.
3) Message persistent: At this point we persist messages on LDAP the same way to did to Users and Groups. After talking to the professor, we realized that this is a dumb idea, because of inefficiency. But it's too late for us to change at that point. Should we given another chance, we think a better way (depends on the amount of users that we plan to have) would be a combination of relational database and file storage.

4.0 Retrospective

What the team liked best

1. The team learned a lot about Java socket, multi-threading, and of course LDAP.
2. Apart from the technical details, making design decisions is also fun. Sometime it needs the whole team gather together and brainstorm the pros and cons, sometimes individuals come up with new ideas and justify it to the team. Sometimes the team made the right decision, and the following implementations will be smooth. However, that is not always

the case. In case of bad design decisions, (in our case, use LDAP to persist message) refactoring is a necessary process.
3. The experience of solving a real-world problem as a team instead of doing tiny-toy like assignment as individual.

What the team liked worst

1. One of the real problems was the port conflicts and time-out issues with Jenkins.
2. Setting up LDAP on aws.
3. Very few online tutorials were available for LDAP database that actually worked.

What the team learnt

1. This project taught us many industry standards related to design and testing. Moreover, we learnt how an agile framework works.
2. The team learned how to use new tools such as Jira, Jenkins and SonarQube. All of these tools are widely used in the industry and may turn useful in our careers.
3. As the legacy code was based on Java networking, we learned about the basics of Java networking. Also use of LDAP server as a storage base helped us get familiarity with this concept to some extent.
4. The team learned how to come together as a group and work proactively over time. In sprint 1, the team could barely meet the minimum requirement without stretches. However, in subsequent sprints, the team learned how to prioritize and distribute tasks for prolific outcome. We used a number of tools to enhance our workflow including efficient use of Jira, smart commits on Git, and integration of Jenkins and Github with Slack.

What needs to be changed in the course

1. You can share our implemented code to the next batch of students and ask them to implement a GUI or add some more features on the server side instead of an entire new project.
2. Give the students an option to develop the project in other Programming languages as well.