Team 103

Project Report
CS5500 - Managing Software Development
Fall 2018

Project Goals

The main goal of the project was to extend the basic communication server provided as the part of the legacy code. As a part of the project resources, the team was provided with a basic client and server and the goal was to expand them and enhance their functionalities. The client and server provided at this point of time only had the basic capabilities. The client could send a message which would be received by the server. The server would then broadcast this message to all the active clients. The project goal was to enhance these server capabilities in order to make it more robust and to support more advanced communication.

Apart from this, another goal of the project was to learn and have an experience working with legacy codes. Working and developing on the code developed by someone else is something a developer would have to deal with in the industry and thus the project intended on giving the students that experience. The legacy code provided for the project first had to be understood thoroughly and cleaned before it could be extended. The primary goal, thus, was to get the legacy base code up and running. The team had to write test cases for the existing code in order to get it running. Some of the code also had to be changed in order to make it testable. Thus, working with the legacy code was also an important goal and learning of the project.

The project also intended to make the students learn and experience Agile Software Development practice which is followed extensively in the industry. For this, the project followed Scrum, which is an incremental and iterative version of Agile Software Development. The project had three two-week sprints which were followed by 30-45 minutes of sprint review.

One of the other project goals also was to involve in a pure socket communication practice and thus avoid using any external infrastructure such as HTTP. The project involved working with a database too and integrate it with the system. Though this too had to be done without involving HTTP, thus to limit the learnings to Java Software Development and to refrain from drifting towards Web Development. Thus, one of the goals of the project was to learn software development using Java and socket programming.

The project also intended on using Continuous Integration in the project. Continuous Integration is important in any software development project and is followed in industry, and thus the project was designed with the goal of using and experiencing the same.

Learning to use a version control system properly and effectively was also an important goal of the project.

Other project goals included learning to use a task management system, which makes working in a team easy and effective. It makes project progress tracking and task division easy. Use of such a task management system was enforced for the project since that is something which would be needed while working in an industry too.

To conclude, though the apparent project goal was to enhance the server functionalities, the project had other goals related to the software development practices followed in the industry.

Results

The give legacy code that just provided the basic functionality of sending messages to all the users (broadcasting) is now completely revamped and made it usable and deliverable. The team worked together dividing the functionalities equally and following Agile software development principles. Following this agile framework, the legacy code is made modular, testable, and easily extendable. In addition to the basic functionality of broadcasting, the system now has a notion of users. The system is now supported with a database that stores users and their information. After adding the notion of users, the notion of a group was created and the functionality to send messages to a group has been provided.

Sending images, videos, audio along with text is common these days. The legacy system did not support this. In this project, the team added support to send such MIME type messages to other users of the system.

The system is enhanced to support many real-world scenarios. The system now stores all the messages wrapped with its timestamp, sender, receiver and IP address. The functionality to retrieve these stored messages has been provided. The old system was not smart to save messages sent to the user who is currently offline. Because of this, all the messages that are sent to a user who is offline were lost. This project fixed this major issue and provided support for delivering messages to the user when he comes online. Some complex functionality such as undo sending a message (recalling a message that is not delivered) has been provided.

Not all messages are equal, some contain abusive or sensitive content that a user of this system might not want to see. Calling it parental control, the system also provides functionality to toggle parental control for a particular user. If on, any incoming and outgoing abusive message will be filtered.

With these features, the legacy code that was provided is made much more sophisticated and ready for real-world use. During the course of adding new features to the existing system, the team learned to understand someone else’s code and how to make it modular and testable.

In an agile setting, the team learned the importance of sprint review, daily standup, and moreover how vital team communication is. The development environment made use of the best practices, that included an automated Jenkins pipeline to test every commit and alert the developer about the quality of their developed code.

Maintaining backlogs was done using Jira. The team created a Jira ticket before starting to work on any issue or task. This provided a clear understanding of the state of development and prevented task conflicts. Additionally, Git was used as a version control system that made collaboration easy, allows the team to revert back to stable systems and prevent code conflicts.

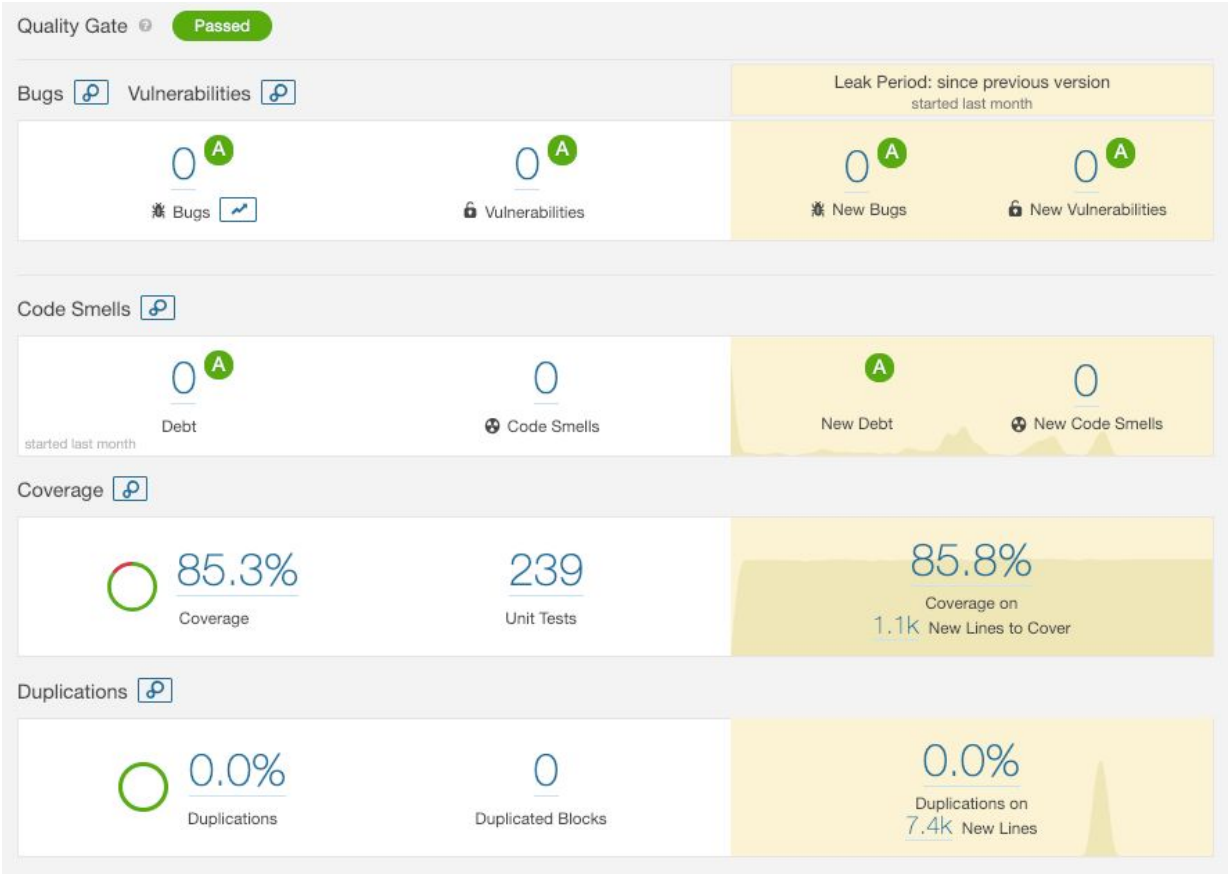
The implemented system is reliable and has been thoroughly tested. Both black-box, as well as white-box testing, has been performed.

Strict line coverage of at least 85% has been followed throughout the development. The submitted code has 90.0% line coverage. Additionally, strict condition coverage of at least 50% has been followed throughout the development. The submitted code has 75.8% condition coverage. Overall, there are 239 test cases which cover all of the functions and tasks. All 239 test cases pass and none of them fail or have any error.

The code is fully maintainable, as the complete code follows standard Java syntax and there are no code smells in the application. Each function follows the principal of one function one task and so the overall cognitive complexity of the code has been low (no function has cognitive complexity over 15). Also, every function has been provided with appropriate Javadoc comments making it well-readable and understandable.

It is also strictly followed that there are no code duplicates, each duplicating code has been extracted into an individual function and re-used wherever needed. That's why the code duplication is 0%. Additionally, there is high code reliability as there are 0 bugs and high security as there are 0 vulnerabilities present in the code.

All of these statistics were automatically calculated using SonarQube which was integrated into the continuous integration pipeline that was built using Jenkins.



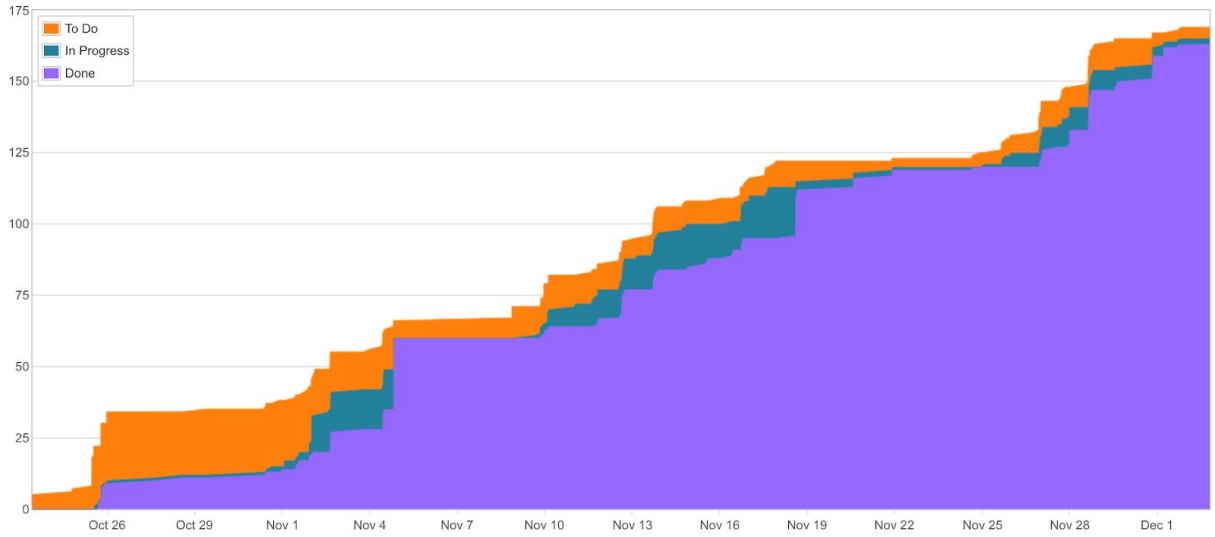
SonarQube Quality Analysis

Jira was used to track project backlog. All the tasks and issues were managed using Jira tickets which were created by every team member. Smart commits were integrated to automatically map GitHub branches with Jira tickets which played important role in task tracking.

Overall there were 181 tickets created on Jira for this project. Out of which 30 were completed during sprint 1. In sprint 2, 49 issues were completed. In sprint 3, 45 issues were completed and 2 issues were left incomplete.

The complete sprint backlog report is available here,

Sprint 1 Backlog
Sprint 2 Backlog
Sprint 3 Backlog



Cumulative Flow Diagram

The cumulative flow diagram represents the team's work during all the three sprints. The graph plots the creation, updating, and completion of tickets for the project.

Project Code Frequency

Team's Development Process

In the first sprint, the team spent time understanding the existing legacy codebase, making it modular and splitting it into testable blocks. Since the code became modular, it became easily testable. The team spent time writing the test cases for the existing legacy.

Both black-box testing and white-box testing was performed. In black-box testing, the team came up with standard scenarios that are expected from the Prattle application. Based on the functional specs the black box test cases were designed.

Considering the white-box testing, the team made sure that the code achieves at least 85% test coverage and 50% condition coverage. With these quality standards, the legacy codebase became well-stable and less prone to errors.

On concluding testing, the legacy code base was modular. Each module was independent and was highly cohesive, which made it easy to expand and introduce additional features. With such a codebase in hand, the team worked on adding the basic login and registration functionality using username and password.

Initially, a file-based system was used to store user data (username and password) on the Prattle. Testing was performed on the newly implemented functionality and then the basic functionality to update a user's username or password was implemented.

In the second sprint, the team primarily worked on adding features to the code-base to make it more functional. The notion of a "user group" was added to the system. Functionality to create a group, add members to a group, remove a member from a group, rename a group, and delete a group was implemented on the Prattle side.

We made a major design decision to change the file-based system that stored user data to a database system using MongoDB. We made this design decision to help the Prattle application perform well in the highly scaled environment.

The login functionality that was implemented in the previous sprint was improved by providing two levels of encryption. In the first level, the password that was sent from the client was encrypted on the client and then decrypted on the server as it was received. This provided us with end-to-end encryption on the channel transfers. The user data that was stored in the database was again encrypted using standard encryption algorithms.

With user and group notions, the team then implemented the functionality to send and receive messages to individual and send and receive messages to groups. Sending messages to a group sent the message to every member of the group.

The MongoDB database that the Prattle application used was moved from the local system to a database-as-a-service provider MLAB. This made the database handling and the database infrastructure management easy.

All the messages that were sent to an individual or to the group were stored in the database with sender's and receiver's name, timestamp. Thus, providing message persistence functionality. In addition to sending plain text messages, MIME type messages support was also added. This included sending images, videos, and audio to any user or group.

In the third sprint, the team worked on further enhancements to existing features and making the system ready for real-world scenarios. The work from the last sprint did not deliver messages to a user if he is not online and all the messages that were sent during the period any user is online were lost. This was a major drawback. The team worked towards implementing an offline queue for every user that stored messages if the user was not online. The messages were unloaded from the queue and sent to that user once he comes online. This solved a major issue of offline message delivery. Which was important in real-world usage of the Prattle system. Additionally, functionality to recall any message that was not sent to the user was provided.

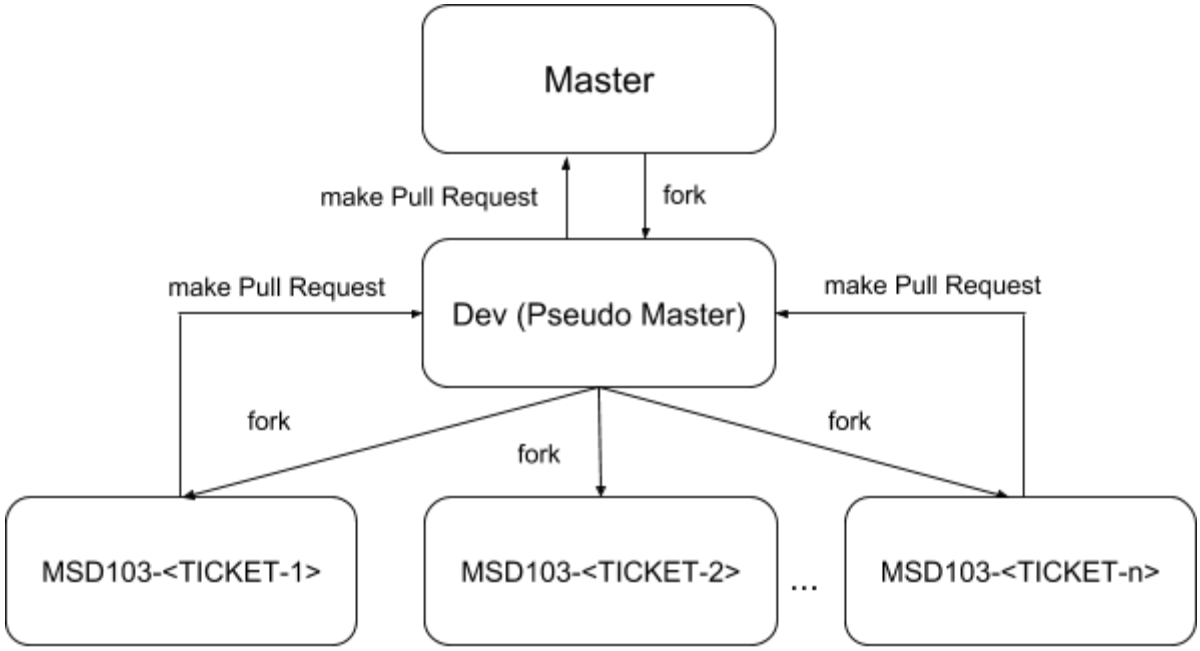
All the messages that were stored in the database during the previous sprint were not wrapped with the sender and receiver IP address. This functionality was achieved getting a user's IP address every time he logs in or registers and the IP address was updated in the database.

The team then worked on an important functionality of Parental control to prevent abusive messages. Functionality was provided to toggle parental control for each user. Every user in the database was updated with parental control "ON" or "OFF". As a message arrives from the client it is then checked with a pre-stored list of abusive words. If any word in the message matches with any word in the list and if the user has set parental control on, then the message is filtered, otherwise not.

The team provided a feature for any agent who has the subpoena to monitor messages between two users or a group in the time frame specified in the subpoena. Such a subpoena user is created by Prattle administrator and the agency is provided with the username and password. If an agency user is online, then all the messages between two users or a group are first sent to that agency user, before sending to the receiver.

Lastly, a feature to search for messages based on username, group name or timestamp has been provided. This concluded sprint 3. The team has not implemented stress testing using JMeter and as a performance upgrade, the team wanted to store all the media messages to Amazon S3 Buckets, which is not implemented as well.

GitHub was used to maintain version control and the team strictly followed the standard git conventions. Each Jira ticket was implemented as a branch and the person who is assigned to the task on Jira, owned that branch. The team followed a custom workflow,



Team 103 - Git Workflow

After the first sprint, the team had the working code in the master branch. The team then followed this Git workflow to maintain version control and prevent any conflicts. After analyzing the first sprint, the team came up with this custom workflow where a dev branch was forked from the master branch and was then treated as a pseudo-master branch thereafter. After a ticket was created in Jira, a new branch was forked from the dev branch with that name. After the functionality of each ticket was completed, tests were written and then a Pull Request was made to dev. Once the entire functionality of the sprint was completed, the dev was merged to master by making a Pull Request.

Retrospective

The part of the project that the team liked the most is the Scrum-based development. The team felt Scrum is a really effective development technique as it allows for continuous development and error rectification. It also helps since it allows for the project tasks to be properly divided among sprints, leaving no room for the last-day rush.

The project served as a good exercise to learn Java Programming along with other Software Development aspects. The project not only covered pure and qualitative programming, but it

also taught other Software Development practices that are followed in the industry and needed to make a project successful.

The team learned to code qualitatively in Java, which includes writing proper Javadocs and adequate testing. As the project included socket programming, the team learned how socket programming works and how to implement it in Java.

One of the most important learnings of the project was to work with Legacy code. As the goal of the project was to build on the top of the legacy code, the team had to first get the legacy code up and running. For this, the team had to write test cases for the existing code. As this code was based on socket programming, some of the code was not testable. These code parts had to be altered in order to make it testable. Thus, the project taught to work on the existing code and to develop upon that.

The project also taught and gave an experience of working in teams. The teams were formed by the instructors as opposed to giving that flexibility to the students. This created a situation similar to real-life where people don't get to choose their own teams and still have to adjust with them. This imparted an essential learning of communicating effectively with the team members, the division of work, etc.

Using Git was also one of the important learnings from the project. Using Git properly is important when there are more than one developers working on a project, which is the case most of the times. Using Git, the team can keep a track of the versions and manage the code and project effectively.

The project also taught using JIRA for issue tracking. JIRA is a very useful tool that is used for keeping a track of sprint goals and backlogs. It also helps in work division among the team members. Integrating JIRA with Git, the team can have the branch and commits corresponding to each ticket which helps in progress tracking.

The project taught using Jenkins for Continuous Integration and verifying code quality after each commit. When there is more than one person coding, it is important to keep a track of the code quality since the code should be allowed to be merged into master only if it meets the quality requirements such as no errors, minimum coverage requirement, etc. Jenkins help in achieving this and the project taught how this process actually works.

Agile Software Development using Scrum was also one of the important learnings of the project. Scrum is a very effective technique for managing Software Development projects and is used extensively. The project consisted of three sprints, each of which was 2 weeks long and were followed by sprint reviews.

Thus, to sum it all up, the project did a good task in teaching Java Programming and also included all the important aspects of Managing a Software Development project.

One of the shortcomings of the course though was its planning. The course, when designed initially was a lot heavy and difficult to manage with other courses. There were many weeks where the students had to spend about 40 hours on the course and that became difficult to manage. Though this issue was taken care of later on, one of the solutions can be to think from the student's perspective while designing the course structure, especially homework and their respective deadlines, keeping in mind that the students have other courses too, to work on.