Slack++

CS5500 Managing Software Development

Team 207

Venkatesh Koka

Prajakta Rodrigues

Omar Tuffaha

Sean Ylescupidez

Table of Contents

Project Goals

The goals of Slack++ were collected based on a backlog of product requirements. The product backlog requested a lot of specific features for this instant-messaging app including the ability to make group chats, recall a sent message, and having special government wiretapping privileges. Essentially, Team 207 came up with a major goal for this product: designing a flexible and robust application that allows users to message each other and perform a series of actions relating to messaging one another. However, this generalized goal was broken down into four main goals that were divided up into four different sprints.

Sprint 1 possessed a variety of major goals that aimed to kickstart the project. Specifically, the team sought to develop a deep understanding on the product that the client requested, understand the starter Prattle code, create a backlog of tickets in Jira, and add technical features that will increase the team's organization in the future. To understand the client's desired product, the team spoke with the client again, created a use case diagram, and fleshed out a list of functional and nonfunctional requirements. Then the team took the list of requirements and transformed them into a work backlog in Jira for tracking. After Jira was completely set up, the team also introduced both smart commits and build status notifications on GitHub to further increase the organization of the team. Team 207 spent several hours going through the given Prattle code and videos to understand the code. The team wrote unit tests to increase the line coverage to 85% and branch coverage to 80% throughout their process of learning the Prattle code.

Sprint 2's goal was to completely set up the rest of the development environment and implement the foundational architecture of sending messages in a group and a dm. This included setting up a local database, setting up a production-level deployment to AWS, and setting up a production database. This sprint also included creating a user login and password. These goals were set in place as a major foundation for the rest of the project.

With all of the development environments configured and the foundational architecture set up, Sprint 3's goal took the team into implementing the core functionality of regulating group chats and user relationships. In terms of regulating group chats, this sprint incorporated adding moderatorship, group invites, and removing users from groups. With regard to user relationships, this sprint introduced friend requests and notifications.

Sprint 4 tied the rest of our project goals together. Sprint 4's goal was to add the wiretapping privilege that is dedicated to governmental users along with adding extra user and group functionalities. This included adding group chat passwords, changing moderators for a group, deleting groups, searching for users on the application, recalling a sent message, translating messages, and adding parental controls. This sprint was one of the best in terms of sprint goals because it addressed a lot of the different requested features from the given product backlog.

Overall, the project goals were all successfully achieved because the team implemented a fully functional instant-messaging application that allowed users to message each other in direct messages and in groups, interact with one another through a series of commands and navigate through the application's features with a help command. The application also sent notifications when appropriate and offered the government a special wiretapping feature, as requested.

Results

Completion

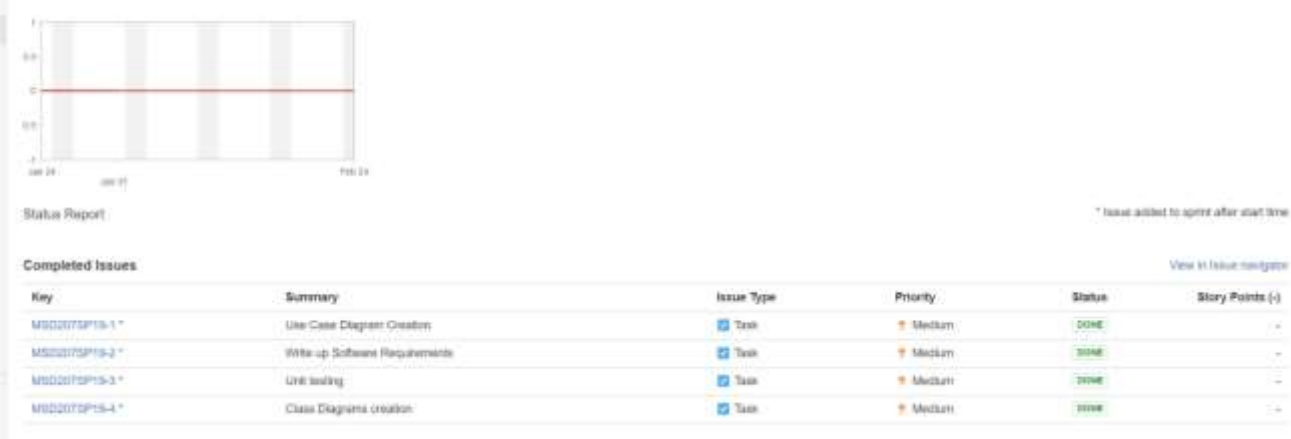
The project as a whole was very successful. Team 207 defined success as the following: implementing a fully-functional instant-message application that allows users to perform a series of core and unique functions while also being able to interact with one another. This section of the report will address how the team's results mimic the team's definition of success.

In terms of a fully-functional instant-messaging application, Slack++ allows users to login to their account with their username and password. If the user does not have an account, it will allow them to create one with a specified username and password. Slack++ enables users to send and receive direct messages to and from any of their friends. It also provides users the option to create group chats, making them the admin/moderator of that group. Messages can be sent and received from different groups and direct messages at the same time, but they will only show up if the user is on the corresponding message window. Still, a fully-functional instant-messaging app should notify a user when they receive messages, especially if they're from another message window. Slack++ accommodates this by sending out notifications to users when they receive messages and other important notices (which will be addressed later). With the described features of logging in to a profile, sending and receiving messages from direct messages and group chats, and receiving notifications, it is clear that Slack++ is a fully-functional application.

Still, a fully-functional application may not be interesting if there are not a series of extra features added to it. Part of Team 207's definition of success is to allow users to perform a series of core and unique functions while also being to able to interact with one another. Here is a list of core features that were implemented within Slack++: inviting users to group chats, removing users from group chats, adding, removing, and changing ownership of a group chat, deleting a group chat, checking which friends are active on the application, searching for users on the entire application, setting a user's status to do not disturb, checking recent notifications, and checking group invites. Notifications were added for invitations to group chats, the deletion of a group chat, and the changing of ownership within a group.

In terms of unique functions, Slack++ accommodates the special governmental privilege of wiretapping a user. This means that the government can retrieve a full history of sent and received messages from and to a user, even if the messages have been deleted and even if the user was part of a deleted group. Slack++ also allows users to translate a message with a language of their choice, supporting over 100 languages. Next, Slack++ enables a user to recall a sent message within a group chat or within a direct message. Last, the team implemented parental controls to block out swear words, which is applicable when a young user is using Slack++.

To allow users to interact with each other, users are capable of sending one another friend requests, which is paired up with a notification to the desired user. Once two users are friends with each other, they are allowed to direct message one another and view one another's availability status on the app. Please see Figures 1-4 for statistics relating to the backlog throughout the project.



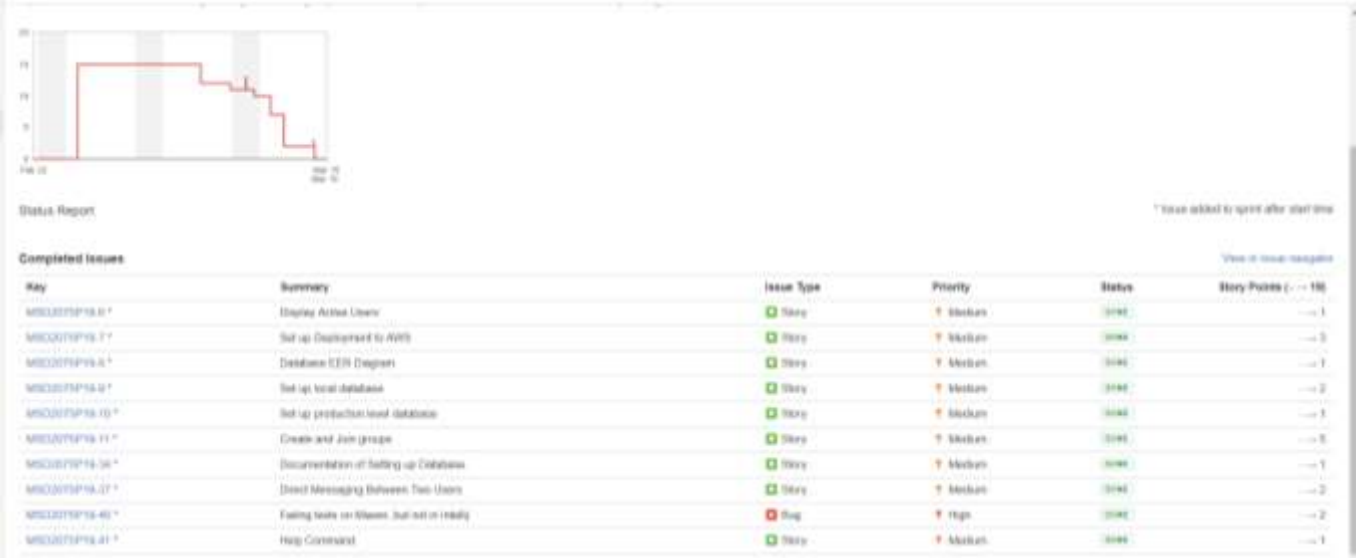
Status Report

Completed Issues

Key	Summary	Issue Type	Priority	Status	Story Points (-)
MSD20TSP1S-1 *	Use Case Diagram Creation	Task	Medium	DONE	-
MSD20TSP1S-2 *	Write up Software Requirements	Task	Medium	DONE	-
MSD20TSP1S-3 *	Unit testing	Task	Medium	DONE	-
MSD20TSP1S-4 *	Class Diagrams creation	Task	Medium	DONE	-

Figure 1: Displays the backlog statistics for Sprint 1.

Note that the team was still learning how to use Jira during this Sprint and spent more time setting up kickstarting the project than doing any development. Based on that, there were only 4 major stories and the team did not point them, but they were all finished.



Status Report

Completed Issues

Key	Summary	Issue Type	Priority	Status	Story Points (- → 19)
MSD20TSP1S-6 *	Display Active Users	Story	Medium	DONE	→ 1
MSD20TSP1S-7 *	Set up Deployment to AWS	Story	Medium	DONE	→ 3
MSD20TSP1S-8 *	Database EER Diagram	Story	Medium	DONE	→ 1
MSD20TSP1S-9 *	Set up local database	Story	Medium	DONE	→ 2
MSD20TSP1S-10 *	Set up production level database	Story	Medium	DONE	→ 1
MSD20TSP1S-11 *	Create and Join groups	Story	Medium	DONE	→ 5
MSD20TSP1S-36 *	Documentation of Setting up Database	Story	Medium	DONE	→ 1
MSD20TSP1S-37 *	Direct Messaging Between Two Users	Story	Medium	DONE	→ 2
MSD20TSP1S-40 *	Failing tests on Maven, but not in ready	Bug	High	DONE	→ 2
MSD20TSP1S-41 *	Help Command	Story	Medium	DONE	→ 1

Figure 2: Displays the backlog statistics for Sprint 2.

Shown above are the tickets for implementing the foundational architecture of Slack++ along with finishing up the development environment by creating a local and production level database. Note that the stories were pointed for this sprint. There were some jumps in the burndown chart that is shown in the top left of Figure 2 because the team finished some work early and decided to pull more work into the sprint backlog. Also, there is a long horizontal slope in the burndown because Spring Break occurred right in the middle of Sprint 3. Still, all of the work was completed.

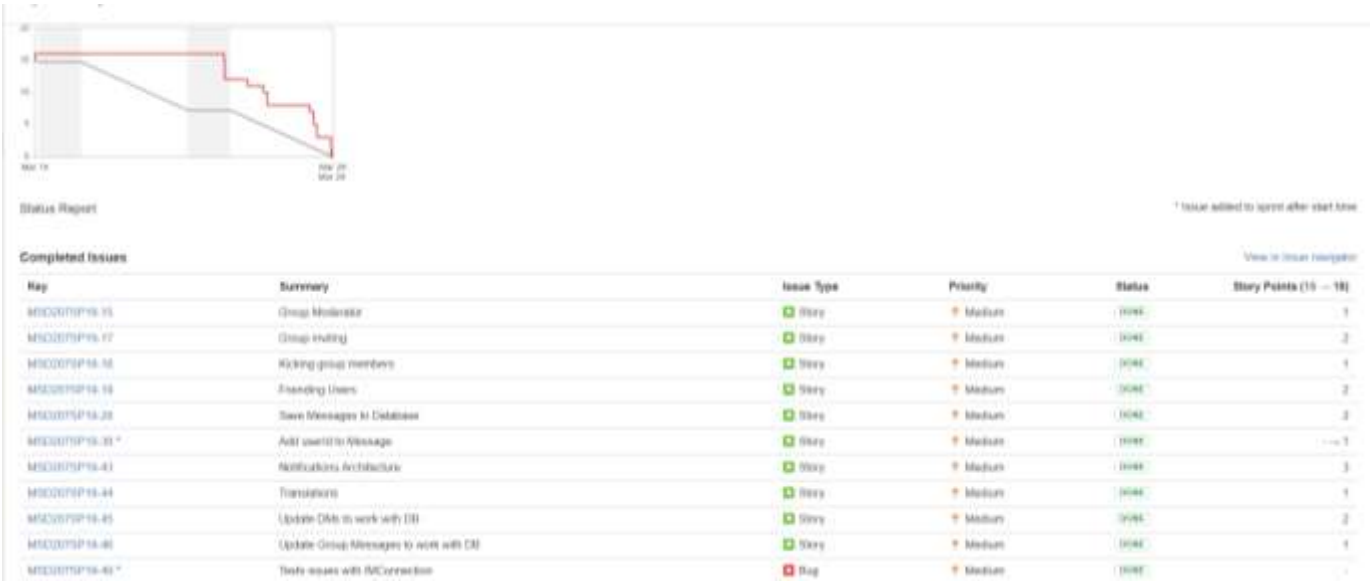


Figure 3: Displays the backlog statistics for Sprint 3.

Shown above are the tickets for implementing the core architecture of group inviting, notifications, friend requests, and other special commands. Although the burndown chart shows slow progress throughout this sprint, it is important to note that burndown charts do not keep track of ticket progress, they only keep track of ticket completion. Therefore, it is very normal for the chart to take a steep dip at the end of the sprint. All the sprint backlog tickets were completed.

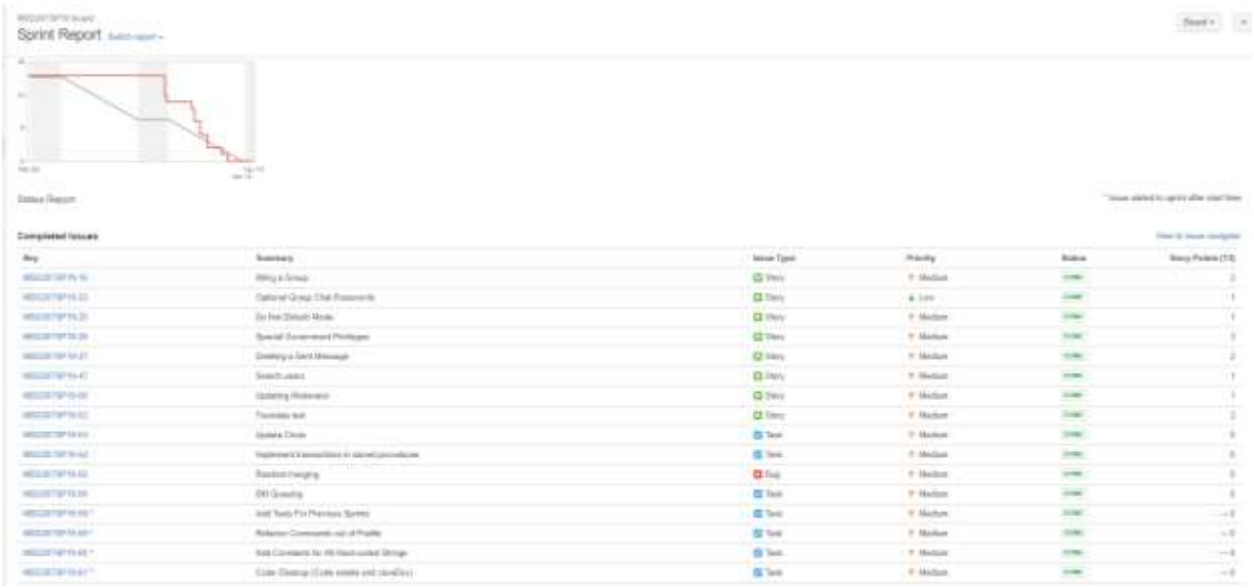


Figure 4: Displays the backlog statistics for Sprint 4.

Shown above are the tickets that wrapped up the project in Sprint 4. Notice how all of the tickets were completed, resulting in an empty product backlog.

Quality

In terms of quality, Team 207's results were very sufficient. According to SonarQube's metrics, Slack++ maintained about 85% code coverage with over 400 unit tests. There were only 30 code smells and only 1.9% code duplications. However, many of these code smells and code duplications were uncontrollable due to the legacy code. Similarly, a lot of exceptions and branches were not realistically testable from Dr. Jump's code. Please see Figure 5 below for exact metrics in terms of quality.

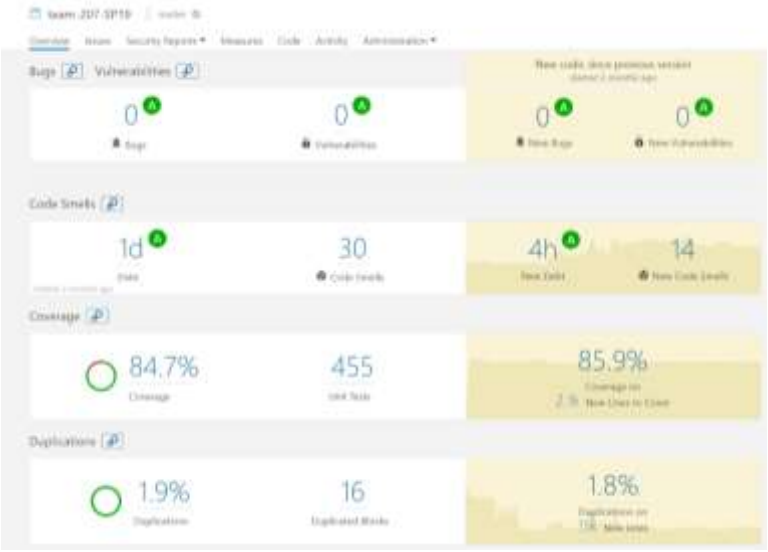


Figure 5: Shown above are quality metrics for Team 207's Slack++'s software.

Developmental Process

Team 207 possessed a very structured and organized developmental process to help ensure success throughout the project.

Sprint Planning

After the end of every sprint demo, the team would immediately meet for the following sprint's planning. This included the creation of new issues, the story pointing of new issues, and pulling tickets from the product backlog into the sprint backlog. This way, the sprint backlog was finalized before the start of the subsequent sprint and every team member was aware what was expected of them. Each team member was informally assigned different tickets to divide up the work evenly based on the team members' different skill sets and time budget.

In terms of story pointing during Sprint Planning, the team used PlanITPoker as a software to anonymously point stories.

Meetings

The team had routine meetings that occurred twice a week. These meetings were every Tuesday during our TA's office hours in case we needed help. Even if we didn't need help, we would meet at that time and discuss progress and see if a team member would need any guidance. We also met every Thursday, which was useful every two weeks before the Sprint Demo. There, we would quality assure the tickets that were considered done in unison with other tickets that were considered done. We essentially performed mini integration testing on Thursdays. We also met after each Sprint Demo for a Sprint Planning. The team also organized plenty of random meetings as well where every team member always attended. These random meetings were in addition to the routine meetings and were organized for important discussions.

Daily Standups

The team participated in daily standup meetings on Slack with a channel specifically dedicated to the standups. There, the team updated one another with the work that was completed in the previous 24 hours and the work that each team member planned to complete within the next 24 hours. Any potential blockers were also brought up during the daily standup meetings. These were especially useful in keeping the team honest to their work and keeping them aware of what they were expected to complete.

What Worked Well

Here is a list of qualities that went well for Team 207:

- The team worked well together and put in a lot of effort to achieve each sprint goal.
- The team met after each demo to plan the subsequent sprint, keeping them organized and ready for the upcoming weeks.
- The team was present at every meeting on time, which added a lot of trust and credibility for one another.

- The team was willing to meet at random times and meet very often for long periods of time, allowing the team to achieve sprint goals on time.
- The team gave detailed reviews of each other's work and required a peer review prior to merging into any branch, ensuring good work quality.
- The team was good at adding test coverage for new stories, maintaining good code quality.
- The team was flexible working on different tasks and pitching in to complete the sprint goals on time.
- The team was organized and in agreement about the technologies that were used (Slack, Jira, GitHub).
- The team was good at working on tasks early, leaving room to achieve more than was promised.

What Didn't Work Well

- The team was not accurate at pointing work because stories were usually underestimated or overestimated.
- The team started off with light quality assurance, which slowed progress later on in the sprints.

What Was/Might Be Done to Address Those Issues

In terms of the first team issue, the only thing realistic solution is to gain more experience pointing stories. Hence, having more sprints and taking this software further can address this issue.

Luckily, the team was proactive about the quality assurance issue by the end of the second sprint. This issue was mitigated by allocating more time for quality assurance in person as a team. The team met twice a week and set aside time during each meeting to test each ticket together.

Retrospective

Best Parts of the Project

Team 207's favorite part of developing Slack++ was the freedom to design the software in the way that they chose. Although the team was given a backlog of product requirements, the actual implementation of requirements was left to the team. This freedom to build a software from the ground up was very new and enjoyable to each team member. Another part of the project that the team really enjoyed was overseeing the whole project. There was very little guidance from the instructor and the TAs about what needed to be done. This management and responsibility were unique and fun for the team throughout the project.

Worst Parts of the Project

On the contrary, Team 207 did not enjoy some aspects of the project. For instance, the team did not enjoy going through the given Prattle code to understand it. Creating unit tests for code that did not belong to the team was slightly frustrating and difficult at times. Some features were just not testable. Still, the team agrees that it was something that needed to be done considering the time frame and scope of the project along with the fact that software developers almost always have to work with code at a company that they did not write. Another thing that the team did not enjoy was the retrospectives. Although it was nice to learn about retrospectives, it seemed like there was too much to write about in terms of the guidelines that were posted on the course website.

What the Team Learned

The team learned a great deal throughout the life cycle of Slack++. Some team members were unfamiliar with specific technologies and were able to learn them as a direct result of this project. For instance, Omar was not familiar with MySQL, JDBC, or anything to do with databases. He needed a lot of assistance in the beginning of the project when his tickets required some database and query knowledge. By the end of the project, he was successful enough to complete all of his database and query-related tickets on his own. Similarly, Koka was not familiar with Java programming, but this changed throughout the span of the project as he was able to write his own code and unit tests with JUnit.

From a nontechnical perspective, Team 207 learned a tremendous amount about software development and maintaining a software. For example, the team learned how to use Jira for requirements tracking, Jenkins for continuous integration, and AWS to host a server and database. With regard to software development, the entire team became very comfortable working in Agile/Scrum with daily stand-up meetings, bi-weekly sprint planning meetings, bi-weekly demos, and bi-weekly retrospectives.

Suggestions for Improvement

There really is not much that can be done in terms of the given Prattle code. The unfavorable part of the given starter code was having to go through the code, clean up code smells, add unit tests, and understand it as a whole. The given Prattle videos were very helpful and should definitely stay for future semesters. It was also nice that the professor allocated a little bit of time during one lecture to go through the code one more time. Still, it would be nice if the Prattle code

contained a lot less code smells and perhaps a few unit tests to begin with, just to make it a smoother transition for the students.

With regards to the retrospective, it would be nicer if each retrospective had a specific purpose to it and therefore required a lot less information to discuss. For instance, it might be helpful if the first retrospective was about reflecting on organizing a team, initial thoughts on Jira and SonarQube, and becoming familiar with Prattle. The middle two retrospectives could have been more geared towards design decisions and how the team was working with one another. The last retrospective could have been focused on the decisions that were made to wrap up the project and what things were prioritized to make the deadline and why.