Managing Software Development Project Report - Team 108

I. Project Goals

The project goals were determined in Sprint 1 based on the product backlog, the resources and the available time to develop the product. The Project Product Backlog contained various features and product ideas. As a team, we first prioritized the features to develop in our product based on how essential each of those features were as an integral part of our product and if it would be beneficial for the users to have them in this phase of the product development. We listed such features into a priority list and communicated with the client to discuss our queries regarding some of the features. We distributed the development, documentation and quality assurance tasks into the 4 available sprints and documented them into each sprint goals and commitment list. Each sprint goal underwent further refinement after constantly being in touch with the client and understanding his perspective and accommodating it in the sprint commitments.

The project goals are as listed below:

1. Review product backlog and generate a list of priority features that would be a part of the product.
2. Familiarize and setup the initial development and testing environment. Also, familiarize with the prattle code base.
3. Familiarize and setup a Jira board to log all the developer tasks and help the scrum master monitor the tasks and development progress.
4. Create UML diagram to understand the project structure and draw out the use case diagrams to understand client-feature interaction.
5. Distribute the features among different sprints and achieve all the goals mentioned in them each sprint.
6. Write unit tests and achieve 85% code coverage every sprint. Perform integration tests to check for bugs. Maintain a detailed java doc for all methods and classes.
7. Report bugs to developer so that the bug can be fixed before the sprint release.
8. Generate SonarQube reports to monitor code smells and remove them.
9. Complete the following feature development (dubbed as core functionality for our product) and any development related to the said broader topic by the end of the sprint 3:
    a. Register users to the application, create a profile and store their data in the database. Authentication for logging in. Encrypting user email id and password before storing it into database.
    b. Updating profile information and changing passwords. Manage users.
    c. Setting up groups to provide platform for users to text all users part of a group. Adding a moderator feature. Manage groups.
    d. Private messaging between users. Messaging to a group of users that are already a part of a group. Manage messages. Invite other users to the group
    e. Looking up other users or group. Following other users and managing a user's circle.
    f. Improving user interaction with prattle using custom exceptions.
10. Everyday scrum meetings with the team to discuss the development activities and any issues that needed to be looked at.
11. The final sprint will focus on refactorization and debugging and thorough integration tests and cleaning up the code base to get code base ready for handing over process.

II. Result

We were able to achieve all the core functionality that we prioritized from the Project Product Backlog. These features, we believe, are important since these would be what any messaging/chat application server owner will desire to have built in their product at a basic level considering what is available in the market today.

These core functionalities we implemented in our product are:

1. Users will be able to create and manage their profiles. User registration and logging in falls under this functionality.
2. Proper encryption system is in place for password and emails. A modified JPA attribute converter is used to encrypt the data before storing it into the database.
3. Users have the option to hide their profile. If so, when another user looks them up, they will not be able to view hidden user's information.
4. Users will be able to create and join groups. They can invite other users to the same group.
5. Users creating a group would be made the moderator of that group. The moderator has the power of adding other users to the group through a well-managed invitation system.
6. Moderator will be able to delete groups if required.
7. A well-managed chat functionality where users can chat with other users within groups and have a private chat with other users as well. ChatApp will have separate threads to distinguish between group chats and private chats.
8. The ability to show unread messages to the users whenever he/she logs in to a chatroom. It will help him/her distinguish between read and unread messages.
9. Users will be able to follow other users. They can also lookup their circle information i.e. the users they are following.
10. Users can search other users using the username.
11. Users can delete the messages such that the user may no longer see the message in their chat window, but the message is still stored in the database for future feature enhancements.
12. Show helpful error messages to the users under non-desirable situations. We believe these messages will assist the user in taking meaningful actions based on the responses.

Each of the functionality along with their testing have a corresponding backlog ticket in the JIRA system. Please find attached a report of the JIRA backlog for supporting the claim in the presentation folder.

Regarding the code base, we tried and implemented proper design principles and architecture patterns, namely, MVC architecture, Factory, Observer and Singleton patterns wherever it was applicable. Every component of the system, starting from handling the user request to the database transaction, are well separated with the help of component layers. All the service layer classes were enforced to have single instances with the help of singleton design pattern to keep things simple. To make the code understandable, every java file has been provided with proper Javadoc to explain the utility of each class and methods associated with them.

In order to achieve proper quality measures, we have tried to cover all the necessary system tests including intensive integration testing, as well as 385 Junit tests that have a code coverage of 86%. Code smells have been eliminated after generating SonarQube at the end of each sprint. Please find the SonarQube report attached in the presentation folder.

As an add-on, we are going to deliver a simple CLI application which can provide users the ability to communicate with the chat application server. Having said so, using JSON as our protocol for communication, the server can be made to work with other form of clients through http proxies with minimalistic effort. The system demo video demonstrates on how to access and use this simple CLI app.

III. Development Process

Our development process overall was relatively smooth. We started off each sprint by creating out Sprint Goals and Commitments document. At first, we weren't sure how much to include in this document and how much would be considered 'over committing' and how much is considered 'under committing'. As students we are very used to having an outline and being told to complete all the tasks by a given date. We haven't had a lot of practice creating our own outline to follow so having this opportunity was a unique and challenging experience. Our first couple sprints were quiet feature heavy and required a lot of time-consuming work. We learned we had overestimated how much work and effort some tasks require despite appearing to be relatively 'simple'. We didn't have a sprint where we felt we 'under committed', but we did figure out how to make a reasonable list where we weren't losing sleep each sprint.

After figuring out how to structure and plan a sprint with a balanced list of commitments, we would delegate tasks and assign Jira tickets for the same. This process was democratic for us. We were open about how much time each of us thought a specific task would take and how much time we had given our course load/workload. This was helpful because sometimes tasks depended on other tasks being done, so the tasks that were important to get done earlier in the sprint were assigned to people with a light workload at the start of sprint while more dependent tasks were assigned to people with a lighter workload at the end of a sprint. This was another neat feature based on the structure of the class. Since sprints weren't a weekly commitment it allowed us to have flexibility when assigning tasks and their priority list.

Talking to the product team was another really important and valuable resource we used each sprint. Rob, being our TA, was especially helpful. He always asked our Sprint Goals and Commitments were submitted a day early simply so we could get feedback from him as early in the sprint as possible. Of course, he didn't require us to do this, but it was especially nice knowing if we did submit it early, he would give us a response sometime Monday and we could adjust our Goals/Commitments based on his comments. This allowed us to get an earlier start on our sprints. Instead of waiting for his feedback on Tuesday we had his feedback and our final draft Monday night. On Tuesday we could start developing knowing we were headed in the right direction. Rob also was always willing to meet and was extremely responsive via Slack and email. When we had a question about the direction we were heading, we could rest easy knowing he would reply to us very quickly. This eased our minds, especially early, when we were trying to figure out what directions to go and how much flexibility we had when developing our app. Because we weren't told to develop the app in a specific way, we had to figure out how we were going to approach development. Other groups probably approached the similar tasks in a very different ways, but whenever we weren't sure what to do, we asked Rob. A good example of this is when we decided to use Hibernate as JPA. We could have used JPQL but decided to use the Hibernate library instead to simplify some of the query writing. We asked Rob if this was okay and he went on to ask the professor where we quickly received our response. At first, we were told it isn't allowed, but after some clarification with how we plan to use JPA we were given the approval. This was extremely helpful as we approached each sprint knowing there is a degree of flexibility with what technologies we use, and if we ever had a question, we knew it would be quickly answered.

After figuring out our direction and technologies, we began development. This process was tricky as we quickly figured out how much work we had in front of us. We always started with the tasks that had the most dependencies and then completed tasks with less dependencies later in the sprint. This worked for the most part. It made a lot of sense, but sometimes we were left with a big project with no dependencies at the end of the sprint. The thought was if we got everything done on our other tasks, we could focus on adding this brand-new feature last. This led to a couple long nights, but in the end, we never missed a feature. We could

have handled this better by assigning a small team to start work on these larger features earlier in the sprint. Maybe two people, one working on the client and the other on the service layers. The small mistake we possibly made was separating our team into two groups and never adjusting the groups. We had two teams, one team focused on the client and the other team focused on the service layers. There were two people on the client team and three on the service layers. This led to great separation of roles as we had people performing tasks on code, they were very familiar with. It also allowed each individual to work on a specific task and layer. Having three people work on the service layer was a bit much though. There was plenty of work to do, but we could have taken one or two of these individuals and put them on a 'new feature' team. This way their primary focus would be developing the new features and ideas. The other one to two people would be left with refactoring the code and testing the existing code base (Operate as a Quality Insurance individual). We know engineers shouldn't test their own code, yet we ended up having our engineers test and refactor their own code a lot. If we had other people checking the existing code base it could have led us to finding bugs earlier in the development process.

At the end of each sprint we dedicated the last day to getting ready for our presentation. What we learned in the first two sprints is we can't rely on the last hour before the presentation to get ready for the presentation. It requires more work to be ready to present to our client. We had a couple of unexpected bugs show up when we presented to our client in the first sprint or two. He gave us the suggestion to set aside the entirety of Friday morning to testing and trying to break our system in any way. Typically, the bugs we found were very minor quick fixes, but it doesn't look good when your system breaks, and the client is there to see it. In our later sprints we managed our time better and were able to dedicate Thursday night/Friday morning to trying to break our code base in any way possible and fixing any minor issues that arise. I think this really shined in our last sprint as we came to Rob with the cleanest code we ever had. We knew exactly what to show him/who was showing him each feature. We knew the different ways a user could try to break the system and how our system handled these tasks fluidly bug free.

A small, yet resourceful part of our development process was the way we utilized GitHub. We decided to create a branch for each sprint called 'develop/sprint X'. This branch was the 'one branch to rule them all'. This meant everyone created PR's to the branch and created branches off this branch. This way we had all the up to date code in one area. It also allowed us to merge un tested code. This sounds dangerous, but there is a reason! Sometimes teammates need help testing code and ensuring it integrates into our system seamlessly. Having this branch active meant we could merge code into the branch that otherwise wouldn't merge to master. Other people could access your code and test your code rigorously. Once we decided the code was thoroughly tested then and only then did, we create a PR to merge the develop branch to master. This ensured everything merged to master was in a state we wanted and was the most up to date system. This process was defined after we had many merge conflicts and problems merging to master. This process helped teammates to easily test another teammates code if it was needed. This also created a single branch everyone branched from, allowing far less merge conflicts.

Overall our sprints were managed better as we progressed through the semester. Having many sprints was very nice. The only thing we wish is we had even more sprints to work on development. Maybe have each sprint's focus coincide with the lecture as well so we could employ the concepts we learned in class dynamically in each sprint. While we, of course, have a few areas to improve (as does anyone) we were very happy with the way we were able to adjust and learn from our mistakes.

IV. Retrospective

The project was a unique and challenging portion of the CS5500 course. Every team member had to put in about 15-20 hours work per week. The team spent a considerable amount of time together for planning, development, testing and merging the code. It is very important for a team to be in sync for a project to be successful. To implement this ground rule, we discussed and voiced out all our opinions during team meetings/daily scrum meetings and resolved the issue. As a team we all worked and submitted all our tasks within the time we were allotted which helped in working on any dependent tasks in the latter part of the sprint.

We got a considerable amount of help from Rob regarding the design and this helped in structuring our code and developing the important features in a timely manner. As a team we initially thought about developing the front end for the application and to design an end-to-end application, but we weighed this against other features that needed to be implemented and decided we did not have enough time to work on it.

Every sprint brought a unique challenge and it was an all-round development experience to get involved in this initial phase of product development which is a rare opportunity in a real scenario.

- What did the team like best?

  We loved implementing new features and testing the features to see they worked as expected. Every sprint had an exciting beginning since we were eager to get coding and collaborate with other developers in the team to produce the result we desired. We also liked implementing the different design patterns that we learnt in the class and to have a solid understanding of the patterns helped us achieve the design we wish to see in the code base. As a team, we liked discussing ideas on various ways we could implement different features of our project and then work using the agreed implementation.

- What did the team like the worst?

  We lacked the ability to estimate the number of tasks that we could implement within a sprint and how much time each of those tasks required. This got us into a lot of trouble during Sprint 2 where we committed to a lot of development features and then we spent considerable amount of time to get them all coded and ready. This left us with very little time to add new unit test cases that would have bumped up our code coverage and let us push the code base into the master sooner. We learnt this the hard way, but we are slowly improving our estimation skills.
  We also did not like fixing merge conflicts since initial merges took considerable amount of time trying to fix the conflicts.

- What did the team learn?

  The most important aspect we learnt from this is that working as a team towards a goal helps us reach the goal quicker. We also learnt considerable number of new things like code patterns, styles, different implementation logic from our peers. Agile methodology is a sought-out method of product development that is used extensively in many companies. This project helped us gain scrum development experience and get used to a fast-paced development environment to produce significant code features at the end of two weeks. We also discussed and reimagined developing a product considering both the developer and the product owner perspectives. We learnt to use various continuous integration tools and development aides like Jenkins, Jira, Junit, Mockito, JPA, smart commits and GitHub processes. We learnt to better estimate our tasks and we eliminated

major merge conflicts by using GitHub pull requests for every new feature. We learnt the importance of testing early in the development phase which helped us avoid unnecessary detours in the development phase. We also learnt to do pair programming to debug and perform integration tests.

- What needs to change in the course to support great experience?

  The course is well designed to cater to the needs of students who have other subjects as well to concentrate on. The course helped us have a project experience under our belt as well as to showcase it in our resume/portfolios and have a virtual development environment experience using agile methodology. If there was less weightage on feature implementation, this would help us dedicate one of the sprints to develop a front end to visualize all the awesome features of the application. The creative output from each team would probably be a good add on to see at the end of the course work.