PROJECT GOALS

The project goals were decided based upon the existing code base and the product backlog provided to us. Based upon those, we decided our goals for each sprint which progressively became more advanced in terms of functionality. The skillset of team was also the decisive factor when it came to prioritize and deciding goals for the sprint. Based upon all these factors, we had set several goals for our project and we did achieve all of those. Setting project goals was kind of worked as a motivation for us to get the code done and features added to the product.

Deciding goals was not just a team decision, but time to time we consulted the client to decide what features they require the most or which are more important to them. Considering that we decided our goals. Deciding the goals provided us with the overall context of what the project is set to achieve and how it aligns with business goals.

During four sprints, we had prioritized our goals based upon the essential need during that sprint. That clearly defined what needs to be achieved for the client to see that the product is getting useful and the actual work is being done.

Starting from the first sprint, we had set our main sprint goal to achieving enough test coverage that will help us put our code to master branch on GitHub. The aim was to achieve it, so that it can give us a good stepping stone for the next sprints. As the sprint progressed, we added goals to our sprint which would provide some features to the product and eventually decrease the size of the backlog.

During the project there were various goals which were of interest of the team, stakeholders or clients. While deciding the goal we followed some steps which would allow us to make sure if that goal is really a goal or not.

- Final outcome: What will be the final outcome when the goal is achieved.
- Methodology: Which the best project management methodology will work best for the project deliverable?
- Resources: Who from the team is best suitable to work on the goal and what other resources are needed for that goal. The aim is to decide and prioritize the goals in a way where resource utilization is balanced.
- Evaluate: How will the result be evaluated. That is, how we will decide that the goal is achieved.
- Monitor: How will the project be monitored and controlled.

Based on the arguments and thoughts depicted above, we had several goals as to achieve test coverage to push code on master branch, user crud operations, user chat, group crud operations, group chat, group moderator operations, user follow, super user etc. for our sprints.

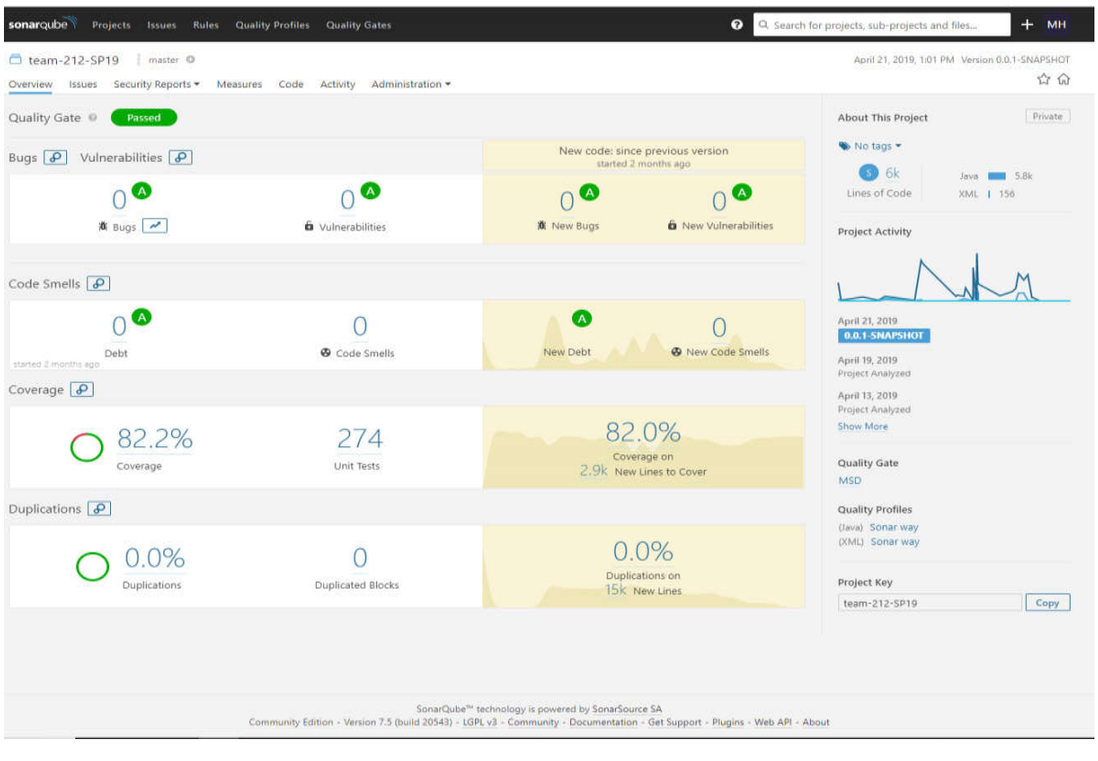
From start to finish deciding the project goals, we strictly followed above five principles which helped us decide and prioritize goals and meeting our goals easy.

RESULT

After four sprints and lot of development, the result that we have achieved for this product and project is outstanding. We have achieved all the goals that we were set out to achieve with the product that is completely functional, and client is also satisfied with the product that has been delivered.

We have followed not just good coding practices and design patterns, we also utilized product development methodologies, because of which, we are able to achieve the resultant product. Which is not just functional, but is

- Secure (We used BCrypt library for login and registration which uses Blowfish and is de-facto standard) and used salt and hash to store passwords (unlike Facebook ☺).
- Good coding standards
    - Our code used many design principles which lead to progressive addition of code in each sprint without much modifications. This is visible as in, for each smart commit against the ticket, one can monitor the progress of how new code was added without much modification. This can't be shown in one screenshot and can be seen as a bigger picture in our Jira tickets. We used patterns like
        - Factory pattern
        - Singleton pattern
        - Observer pattern
        - Command Pattern
    - We also adopted SOLID principles and paid attention to questions like
        - what if we change our front-end tomorrow? (All view updates take place from one file throughout the program)
        - What if we want to extend our communication protocol tomorrow? (Hence used JSON and made client from scratch)
        - What if we have many more options in the main menu? (Hence Command pattern)
- Good code quality
    - This is visible clearly in the final SonarQube report which has 0 code duplications, 0 code smells and 0 bugs or vulnerabilities in the thousands of lines of code. There are adequate tests in place as well.



This is a good achievement since the short amount of time that we had on our hands with the amount of work that we had put it. We completed and covered from the backlog:

- Users and Groups
  - CRUD operations on Users and Groups
  - Relationships and hierarchy associated with moderators and the allowed operations
  - Secure Login and registration using Blowfish (a symmetric key block cipher)
- Communicating
  - Follow and Unfollow users.
  - See which users are online/offline.
  - User to User messaging.
  - Group messaging.
  - Recall sent Messages.
  - Translate messages.
  - Read unread messages when offline / Saved messages ordered in the way received.
- The Government
  - Tap into user to user conversations.
  - Tap into group conversation for a particular group.
  - Tap into group conversation for all users for a particular group.
  - Add filters like specific dates to get messages from that duration only (applicable filter for all points above).
- Other Thoughts
  - Profanity filter

Sprint	Summary
MSD212SP Sprint 1	MSD212SP19-15 / Create sequence diagrams for the system
MSD212SP Sprint 1	Create sequence diagrams for the system
MSD212SP Sprint 1	Integrate Jacoco to pom
MSD212SP Sprint 1	Create a document to list out all the use cases
MSD212SP Sprint 1	MSD212SP19-11 / Decide various use cases for the system.
MSD212SP Sprint 1	Decide use cases for the system
MSD212SP Sprint 1	Create UML diagram for the initial system
MSD212SP Sprint 1	Junit : Give at least 85% line coverage
MSD212SP Sprint 1	JUnit client : Give at least 85% class coverage
MSD212SP Sprint 1	Junit Client : give at least 85% method coverage on client end
MSD212SP Sprint 1	Junit for Server

1–22 of 22

Sprint	Summary
MSD212SP Sprint 2	Remove some code smells and document code
MSD212SP Sprint 2	Initial Client Side POC extention
MSD212SP Sprint 2	Fixing Junit tests
MSD212SP Sprint 2	Junit Coverage for new codes
MSD212SP Sprint 2	Fixing previous tests
MSD212SP Sprint 2	Create a async listener using Observer pattern
MSD212SP Sprint 2	User to User chat client
MSD212SP Sprint 2	create client side communication authentication
MSD212SP Sprint 2	Testing deployment of prattle on AWS
MSD212SP Sprint 2	User to User chat server Functionality on Server
MSD212SP Sprint 2	Set client side socket communication layer
MSD212SP Sprint 2	setup communication message format
MSD212SP Sprint 2	Create View Delegate
MSD212SP Sprint 2	Research on the client side architecture
MSD212SP Sprint 2	Handle and Deploy our code on AWS
MSD212SP Sprint 2	Develop a queue to handle requests
MSD212SP Sprint 2	Extend the server side architecture
MSD212SP Sprint 2	Extend the client side
MSD212SP Sprint 2	Create the client side architecture
MSD212SP Sprint 2	Export DB Layer as a standalone Layer
MSD212SP Sprint 2	Create Database Layer
MSD212SP Sprint 2	Create Login Module

1–32 of 32

Sprint	Summary
MSD212SP Sprint 3	Deployed on AWS and tested
MSD212SP Sprint 3	Test Case for client side and server handlers
MSD212SP Sprint 3	Handle Update and Delete groups
MSD212SP Sprint 3	Rewrote all tests for login, registration, Client Handler, Unread Messages
MSD212SP Sprint 3	Group Chat in group
MSD212SP Sprint 3	Group chat delete users from group Client side
MSD212SP Sprint 3	Group Chat add users to group
MSD212SP Sprint 3	Handle list when user is chatting
MSD212SP Sprint 3	Unread messages delivery set on Jpa service and design
MSD212SP Sprint 3	MSD212SP19-50 / Search Group
MSD212SP Sprint 3	MSD212SP19-50 / Group Chat
MSD212SP Sprint 3	Handle Moderator Operations
MSD212SP Sprint 3	MSD212SP19-50 / Remove Users from group
MSD212SP Sprint 3	MSD212SP19-50 / Add users
MSD212SP Sprint 3	MSD212SP19-50 / Create group
MSD212SP Sprint 3	Show unread messages client
MSD212SP Sprint 3	Show unread messages server
MSD212SP Sprint 3	Show unread messages database
MSD212SP Sprint 3	Show all unread messages
MSD212SP Sprint 3	Handle default implementation in case of unknown issues
MSD212SP Sprint 3	Handle the threading issue on client side
MSD212SP Sprint 3	Group chat server side
MSD212SP Sprint 3	Group chat client Create Group
MSD212SP Sprint 3	group chat server side
MSD212SP Sprint 3	Handle server disconnect/ termination on client side
MSD212SP Sprint 3	Restructure database layer
MSD212SP Sprint 3	Change key id for chat table from int to long
MSD212SP Sprint 3	Remove ReplyTo or Discuss how to handle ReplyTo
MSD212SP Sprint 3	Add errorCode in models
MSD212SP Sprint 3	mapping between message type and model
MSD212SP Sprint 3	Handle Terminate / disconnected clients
MSD212SP Sprint 3	Loggers

1–21 of 21

Sprint	Summary
MSD212SP Sprint 4	Refactor database code and remove code smells.
MSD212SP Sprint 4	Test Coverage
MSD212SP Sprint 4	Writing test for SuperUser
MSD212SP Sprint 4	Rewrite JPAService Tests
MSD212SP Sprint 4	Comma separated users while adding in groups
MSD212SP Sprint 4	Add other language support for chat
MSD212SP Sprint 4	Cleanup new server side code for Super User
MSD212SP Sprint 4	User to User chat : Should not be able to chat with itself
MSD212SP Sprint 4	Remove all code smells from client side
MSD212SP Sprint 4	Remove all code smells on Databases side
MSD212SP Sprint 4	Remove all code smells if any - Server Side
MSD212SP Sprint 4	Test coverage
MSD212SP Sprint 4	Display the active users in following users list - client side
MSD212SP Sprint 4	Display the active users in following users list - server side
MSD212SP Sprint 4	Follow Users client side
MSD212SP Sprint 4	Follow Users server side
MSD212SP Sprint 4	Violence/ nudity filter server side
MSD212SP Sprint 4	Recall messages client side
MSD212SP Sprint 4	Message Recal server side
MSD212SP Sprint 4	Wire Tapping Client side
MSD212SP Sprint 4	Wire Tapping Server side

Another point which we want to mention here is, rather than using the existing client, we have created our own client which is very well structured and can easily be extended without much of a hassle. We have used JSON format to communicate between client and server. Hence, our code will be easier to scale and maintain as well.

In terms of team and individuals we have progressed well as well. As a team, we have become stronger, the comfort and confidence in working with one another has also increased and the concept of relying on teammates to get something done, is also the result of this project is. We have also realized each other's strong points and can approach the correct person based on the issue type.

The other side of this project is the managerial things that we learned. While working as a SCRUM master, we all acquired the skills of planning, prioritizing and managing the tasks while the product is under development, which is a result of working on the project.

TEAM'S DEVELOPMENT PROCESS

The team followed an agile development process and not waterfall design or extreme engineering like other teams in the course. Hence, the team got a chance to constantly iterate on the code and make it better. A lot of things which did not work on a specific sprint was re iterated with a different strategy to make it work. What really worked for the team was:

- The constant communication of the team really helped the team in making the progress. We met almost every day virtually at night around 10:30 pm and gave updates. We selected this time as we have work/classes/priorities which we had to attend during day. Hence, we can update every one of the progress we made at night. We used hangouts and team viewer for this.
- The constant communication also led to the process of helping and discussing whenever anyone is stuck or if anyone has a better way to do something. This was especially beneficial when we were discussing how to standardize how client talks to server. We ultimately used JSON and came up with a standard JSON communication pattern which was used throughout the course. This was especially useful later one when we were developing features at a faster pace as we did not have any issues due to our previously standardized communication format.
- The design patterns taught in the course and the principles like SOLID were implemented and hence our code was reusable and highly manageable. This led to easy addition of new features in every sprint and we had to refactor old code to just a minimum.

There were some issues which did not work for us like:

- The database code is not meant for Junit testing and was running locally but was failing on Jenkins. There was a lot of issues when we were trying to cover Database code and hence our code coverage on database end was low. We also had to iterate on Database code coverage due to randomly failing database code. To address this, we could have used database as an external dependency where we just called them, but we do not have to write Junit directly on database code.
- We wrote a lot of tests for the provided Client and then had to switch to our own client. This was no one's fault but was a planning issue on our end. A better knowledge of what needs to be done and better initial planning would have averted it.
- There were few standard threading issues that we faced due to the multiple design patterns that we used. We actually wanted to make the response from server to come in an asynchronous manner but since user had to wait due to synchronous connection between client and server via socket, this was a pseudo asynchronous connection. We also ran into threading issues with this as different thread were trying to update and access variables and leading to crash. We later on addressed this by accessing those certain areas in a synchronous manner and addressed this issue but this consumed a lot of our time and was a major blocker in some sense.

PROJECT RETROSEPCTIVE

The 3 main questions asked were:

1. What did the team like best/worst?
    a. Good: redesigning client side from scratch was a great experience as we got a free hand and were able to apply many design patterns that we learnt in lectures.
    b. Good: Usage of Jira tool so that we were able to access and monitor our own progress as well as team member's progress.
    c. Good: Having sprint reviews so that we can demo our work to TAs which was akin to showing a client the progress.
2. What did you learn?
    a. The main thing that we learnt which is true for almost all such projects is "*Constant communication is the key*".
    b. Another thing which we realized is, in a professional setting where we have to deliver what we claimed is, "Don't be over ambitious". It's better to not claim and deliver but it is bad to claim and not able to deliver.
    c. This brings me to my third point, "Do whatever you claim". So, be wary of whatever is being planned, plan early and give high importance to planning.
    d. Always have ambitious goals aka stretch goals to push yourself and does not harm if you don't deliver. This pushes developers to go for the extra mile and also shows the commitment towards the project.
    e. Time Management is very important, and tools like Jira helps in that. Divide the work properly and deliver.
    f. Look into previous sprint bugs and fix them as priority as the later we discover and fix the bug, the more the cost. Hence, try to find and fix bugs as soon as possible.
3. What needs to change in the course to support a great experience?

The course already has a great experience but since we are pushed to think and give feedback for an even better experience, these are points which might be considered based on resources and many other administrative factors. These are just suggestion and not all can be implemented at one point of time and are alternatives to one another.

- Managing Software Development was a great course and was a great experience in many senses.
- This is very far-fetched and might not be practical but if by some means we can work for real life clients and build whatever they require for free, that will be great. This can include working for NGOs, or small companies for free developing software like role-based access control, digitizing their portfolio etc. which they use for everyday work and are not their main software.
- We can also pick up ideas from open source and again push the code back to make them open source. This is true for our current project as well to certain extent but since we already have slack, our project won't be a popular option among people.
- A simpler thing would be to have a survey at the start of the course to access people's capabilities and divide them accordingly so that every team is balanced.
- Have real life code walks for assignments. This is again very far-fetched as the number of students are quite high. Hence, what can be done is make code walks optional and who ever opts in can get extra credit based on performance evaluation by TAs and also a certain minimum for opting in.
- I would also suggest having at least 1 assignment in different language other than Java. This can be highly beneficial because when we join a company, there might be many other languages at play where we have to jump, ramp up and solve bugs or work on minor issues. So, have one assignment based on Python, Swift, Kotlin or

C++ etc. which will push everyone to wear different hats and can be challenging in a very good way.
