**PROJECT GOALS**

A one-page summary of the project goals.

**RESULT**

A one-page overview of the result, with additional pages as necessary presenting completion claims supported by backlog statistics and quality claims supported by testing data.

**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:
   1. 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.
   2. Good: Usage of Jira tool so that we were able to access and monitor our own progress as well as team member’s progress.
   3. 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:
   1. The main thing that we learnt which is true for almost all such projects is “*Constant communication is the key*”.
   2. 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.
   3. 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.
   4. 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.
   5. Time Management is very important, and tools like Jira helps in that. Divide the work properly and deliver.
   6. 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.