# Managing Software Development (CS5500)

# Project Report – Prattle Chat Application

Team number : 209
Team members:

1) Akhilesh Hegde
2) Elavazhagan Sethuraman
3) Varsha Muroor Rao
4) Vinayakaram Nagasubramanian

GitHub repository URL**:** https://github.ccs.neu.edu/cs5500/team-209-SP19

## Contents

## Overview of project goals

We decided on the sprint goals at the beginning of each sprint after discussions on the scope and duration of the sprint to assess the feasibility of the goals. Below we provide a breakdown of the goals we set and achieved during the course of the project.

**Sprint-1**

1) We spent a significant portion of the time on understanding the codebase we inherited, which we used as the foundation of this project. To help with this, we generated a model of the code base in UML format.
2) We then developed the model incorporating new client requirements to have a holistic understanding of the product and the components we will require en route to developing and incorporating new features into the product.
3) Good software development practices were put in place by setting up development environment, agile and source control conventions we would follow. This involved proper integrations between JIRA, Jenkins, GitHub, Slack and SonarQube, which automated and regulated our process to maintain good code quality.
4) For a working product, we focused on coverage for the codebase and having a steady version of the existing functionality tested and passing quality gates.

**Sprint-2**

1) We deployed a first stable build of the server passing quality gates by means of automated deployment upon merge to master.
2) We implemented abstractions we modeled in the first sprint for the application model.
3) Integrated an authentication mechanism to support the notion of users and have a minimal layer of security.
4) Adapted the chat room functionality to operate as a direct messaging service between two registered users.

**Sprint-3**

1) Implemented group chat functionality and abstractions to handle member roles in groups.
2) Implemented (soft) deletion of messages.
3) Added functionality for users to request to be added to a group.
4) Added moderator operations such as adding and removing members from a group and changing a user's role to be a member/moderator.
5) Implemented fetching messages received by users when they're offline.
6) Implemented message forwarding to another user using the message ID.
7) Implemented info commands to get all group names, all registered users and all users in a user's circle.

8) End-to-end message encryption implemented with a simple cipher based encryption solution.

**Sprint-4**

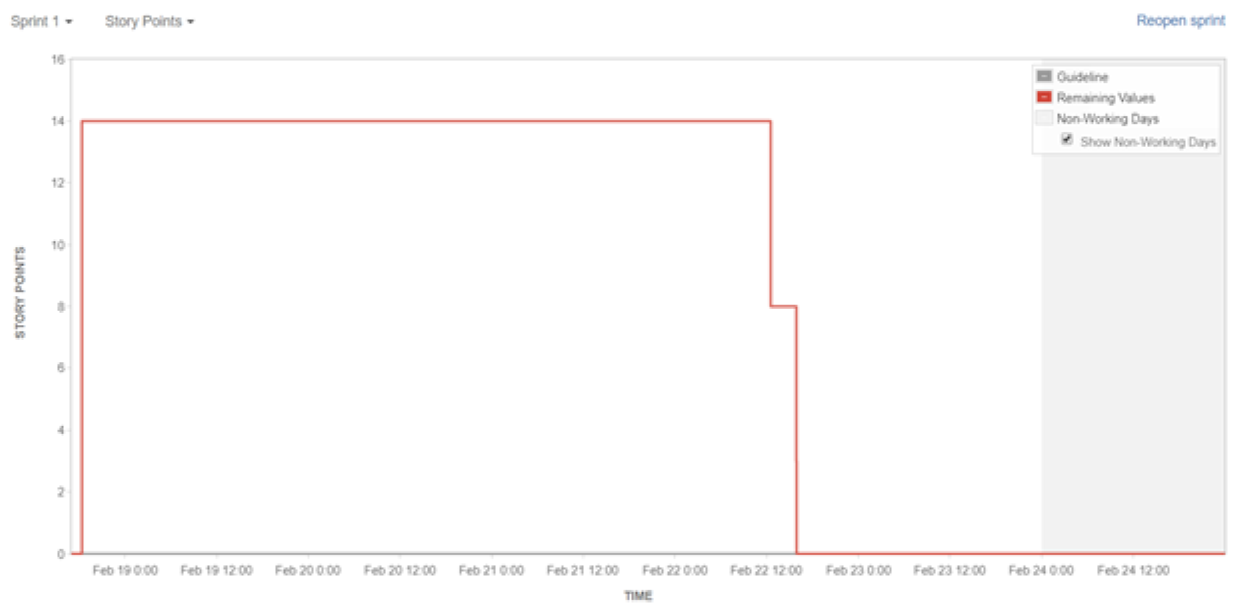
1) Allow users to mark themselves invisible and not appear in global search results.
2) Allow group moderators to make toggle groups between public and private.
3) Add an operation to allow messaging a subset of members in a group.
4) Allow sending messages that time out after a specified number of minutes.
5) View old messages sent in a direct or group chat window in batches.
6) Provide external APIs to fetch messages sent or received by a user to abide by CALEA requirements for the product.
7) Track the origin of forwarded messages to see who originally sent it.
8) Allow toggling text filtering to use a censoring engine to censor received messages.
9) Allow user to toggle the language in which they receive messages. The received messages are translated in real-time when a user receives it.

## Results

The team was able to achieve all the goals as described in the previous section throughout the duration of 8 weeks. In this section, we discuss about the backlog statistics for each sprint and the quality reports as given by SonarQube.

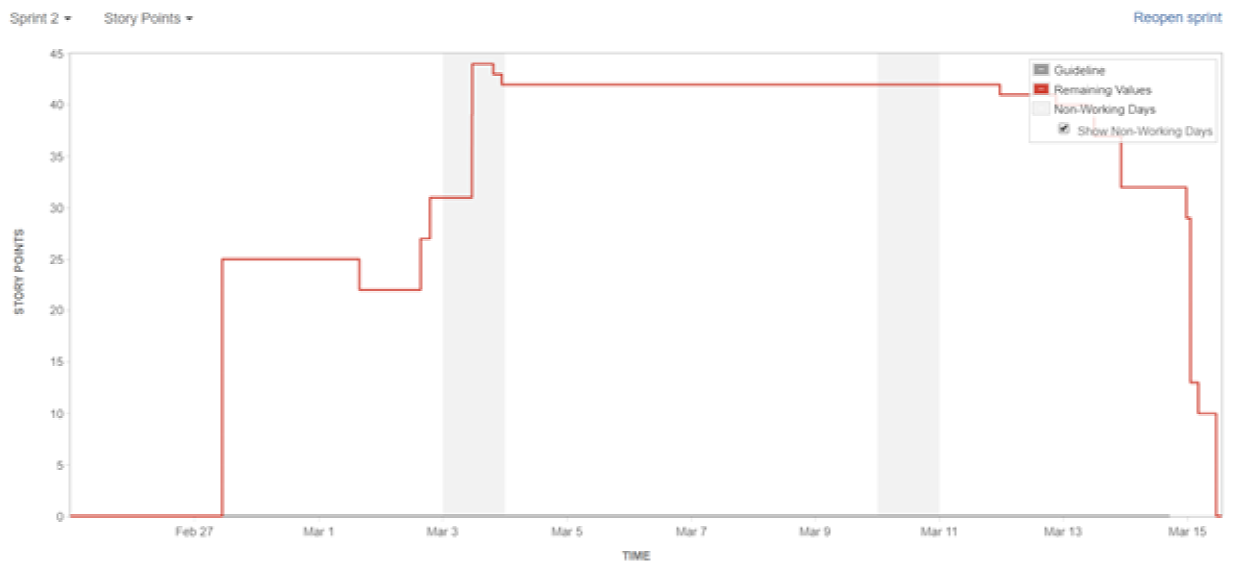
### Statistics from JIRA

**Sprint 1**



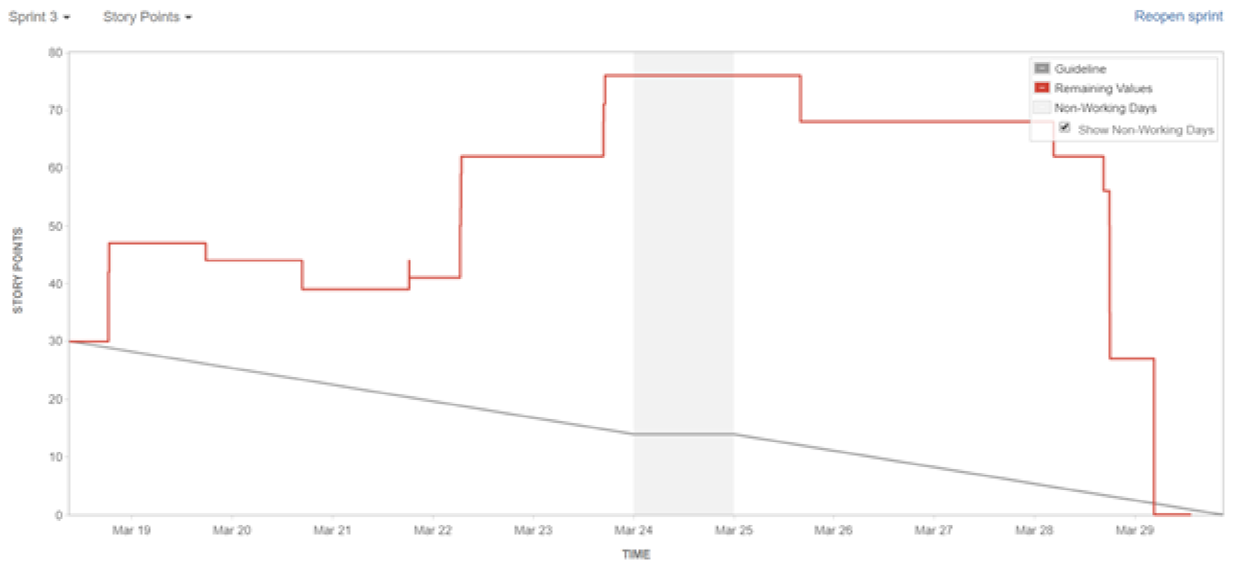
The above graph is the Burndown Chart for sprint 1. In the first sprint, the team had not created a lot of items in the backlog, since the focus of the first sprint was mainly on attaining the test coverage. Hence the number of story points is comparatively lesser than in the other sprints.

## Sprint 2



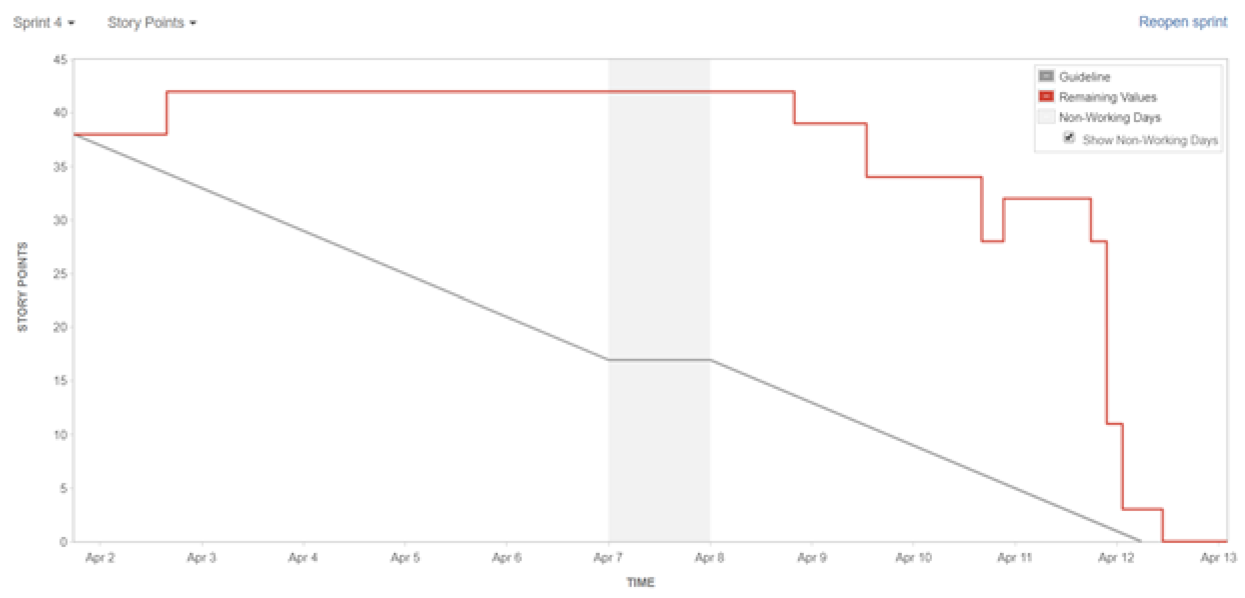
From sprint 2 onwards, the team started pacing up and was able to take up a good number of functionalities to implement, and hence the number of story points are greater. In the beginning of the second week of the sprint, more items were added to the backlog. The team was able to complete all the items in the backlog by the end of the sprint.

## Sprint 3



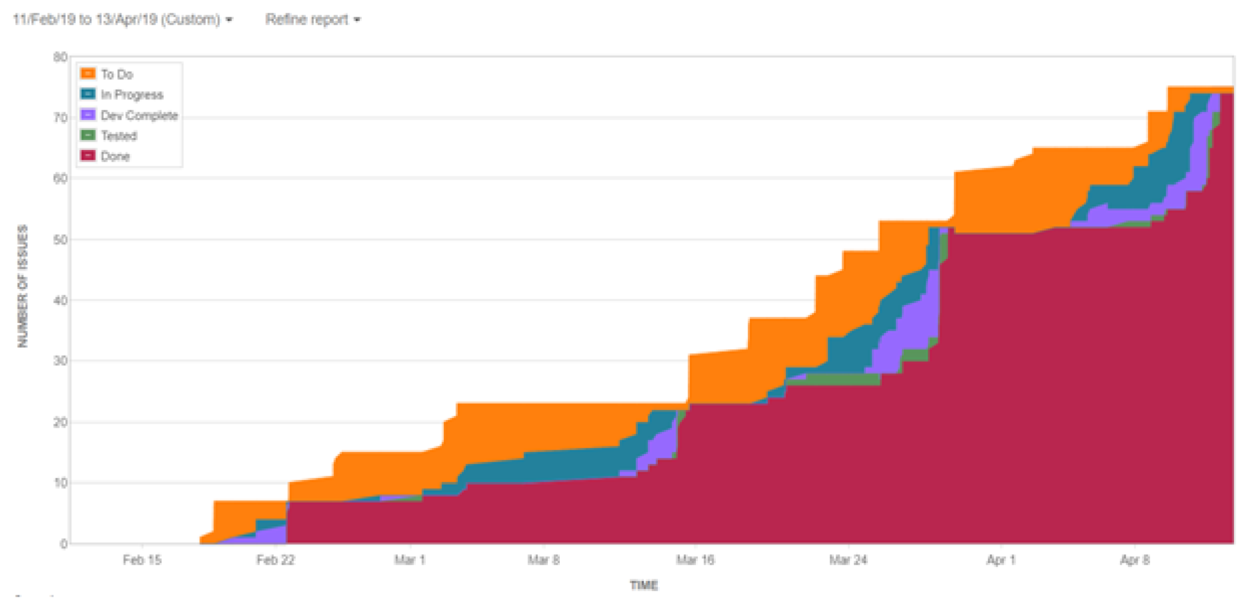
In sprint 3, the team aimed at completing most of the functionalities from the product backlog document. In the beginning of the second week of the sprint, after receiving feedback from the client, the team added more items to the backlog and was successfully able to move all the items to completion stage.

## Sprint 4



The fourth sprint was very similar to the third sprint, since the team attempted to cover as many functionalities as possible and was successfully able to do so.

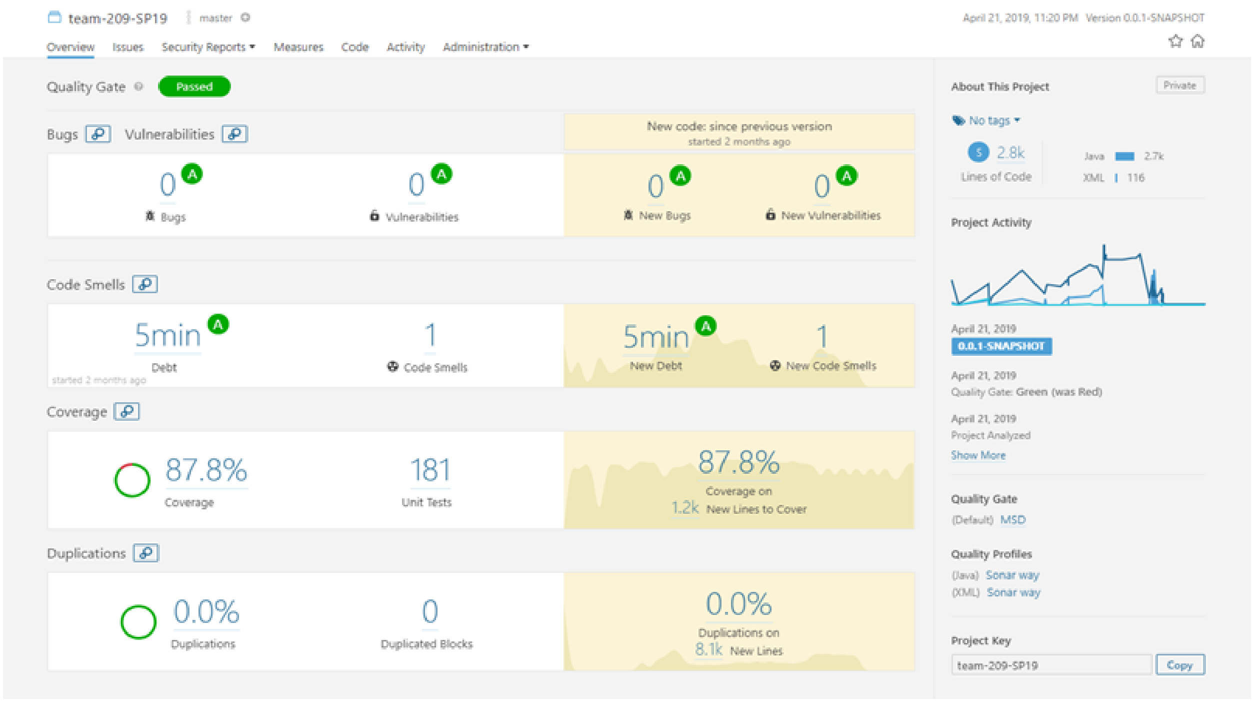
## Cumulative Flow Diagram – For the period from Feb 11 to April 12



This graph depicts the Cumulative Flow Diagram for the entire duration of the project. All the lines grow almost parallelly indicating that the number of incoming items and the number of items moving to completion are almost equal to one another.
We can roughly divide this into four phases, corresponding to each sprint. We can make the following observations from this diagram:

- To Do: The vertical spikes in to do indicate the start of each sprint, when we identify and add items from the backlog into the next sprint depending on the sprint scope.
- Done: Similar to the spikes in To Do items, the curve for Done steadily increases as the sprint progresses and close to the end of the sprint, all the sprint items are moved into Done.
- In Progress, Dev Complete and Tested: These three buckets are used as transitional buckets for the user stories and issues in the sprint. The curves for these steadily increase throughout the duration of the sprint and flattens out leading into the spike for the done items. These stages demonstrate that we made steady progress throughout each sprint and JIRA was used effectively for issue tracking which helped reflect the state of each sprint accurately.

## Quality claims



team-209-SP19 master

April 21, 2019, 11:20 PM Version 0.0.1-SNAPSHOT

Quality Gate: Passed

Bugs | Vulnerabilities
0 A Bugs | 0 A Vulnerabilities

New code: since previous version, started 2 months ago
0 A New Bugs | 0 A New Vulnerabilities

Code Smells
5min A Debt | 1 Code Smells
started 2 months ago
5min A New Debt | 1 New Code Smells

Coverage
87.8% Coverage | 181 Unit Tests
87.8% Coverage on 1.2k New Lines to Cover

Duplications
0.0% Duplications | 0 Duplicated Blocks
0.0% Duplications on 8.1k New Lines

About This Project
2.8k Lines of Code
Java 2.7k
XML 116

Project Activity
April 21, 2019
0.0.1-SNAPSHOT
April 21, 2019
Quality Gate: Green (was Red)
April 21, 2019
Project Analyzed

Quality Gate
(Default) MSD

Quality Profiles
(Java) Sonar way
(XML) Sonar way

Project Key
team-209-SP19

The above picture shows the quality metrics as taken from SonarQube at the end of the fourth sprint. There were 0 bugs, 0.0% code duplications, and about 87.8% code coverage. During the initial sprints, the team was able to achieve the minimum required coverage to pass the quality gates, but as we progressed, we were able to boost the test coverage to more than 85%.

## Development process

As we followed agile methodology with a total of 4 sprints for this project, we had to work at a consistent pace. Each backlog item was attributed with a story point that emphasized its importance. Eventually, that helped us in prioritizing tasks for each sprint. The backlogs were divided into sub tasks. Hence, progress was effectively monitored throughout the sprints.

Although the components were developed independently, since they would be integrated later as one single product, it was necessary to ensure that all the team members are in agreement and in sync with each other. This required a good amount of discussion and pair programming which was a major factor in the development process. The process ensured that basic functionalities are completed and working. Later, additional features that improve usability and provide rich and convenient experiences were added. One key part of the development was to have frequent meetings with the client to evaluate and verify the requirements which was given high importance and was done throughout the process.

The project is categorized into three major components:

1) Persistence
2) Server
3) Client

The developers were working on the respective parts and were responsible for the sanity of that component. The development process ensured that any change in a component is thoroughly tested and pushed to the branch. Peer review was mandated before considering the changes as stable and getting them into master. This ensured that the master branch is always stable for new changes to be updated and used.

Each sprint was two week long and following were the major practices that were followed throughout each sprint.

1. Discuss as a team and decide the work-items for the respective sprint
2. Update JIRA backlog based on the decided work-items
3. Evaluate and discuss the requirements and priorities with the Client (Week I)
4. Re-prioritize the work-items based on Client's inputs
5. Start working on the assigned work-items.
6. Mandatory Team meeting - week I
7. Impromptu meetings if necessary
8. Pair programming wherever necessary
9. Peer Code review and Sanity testing
10. Client discussion (Week II)
11. Mandatory Team meeting - week II
12. Integration Testing and Functionality testing
13. Deployment
14. Presentation to the client

We also configured an automated slack bot, that sought daily standup inputs and published in to the team's slack channel at 12 PM every day so as to be better aware of the team's overall progress. To make the development process smoother and more communicative, the slack channel was integrated with GitHub to notify update in git repository and Jenkins reports. SonarQube was monitored after commits to ensure code quality.
For each sprint, a dedicated development branch was created to ensure that we have an isolated copy of the work environment. Also, following pair programming approach helped us in faster fixation of issues. While integrating different layers for each feature, we all collaborated to prevent potential system breakdowns. After each such integration, all features were verified with both functional and structural tests. The system is thus claimed to be consistent.
Our initial plan to persist all messages in a simple file system appeared to pose a lot of futuristic burden as the project evolves. Hence, we decided to have database setup for persistence. That decision was proved to be worthy when we had to seamlessly use joins and other complicated querying mechanisms to aggregate data.

## Retrospective of the project

All the sprint goals were developed, tested and deployed as per plan. It was an excellent teamwork with equivalent amount of work from all the four team members. That was crucial for getting the project done within the stipulated time. During the last two sprints, robustness of the project was evident as we could pinpoint the issues using legacy test cases. All of us possessed deep understanding of our respective features.
With this project, we learnt the importance of discussions before coding our ideas immediately. As we spent more time in interpreting the requirements together, each of us had a new perception to implement those requirements. Discussing the pros and cons of each approach saved us from a lot of potential time consumption. And those discussions also ensured each team member is well informed on the approach that every feature is intended to be implemented with. More importantly, we had the space to disagree with each other during the discussions. That is an indicator of how well the team's cause was prioritized above individual's interests. Discussions in setting up the persistence layer was another challenging aspect where each of us had to justify our choice. The collective consent on all these confirmed that we were always on track.
Although all the team members have prior professional experience, this project was very interesting for we had to take the complete ownership throughout the duration. Additionally, the individual responsibilities had a progressive effect on the overall flow of the team because of its small size.
Often times, we realized the importance of writing good test cases. There were occasions where we modified several features at different layers. The propagation of errors after such modifications in other layers were well caught as the test cases failed appropriately. Looking at those, we fine-tuned the application and updated the test cases.
Our choice of interpreting the application's context came handy except for features like messaging to a subset of group users, forwarding message to the group and the like. In such situations, instead

of coming with a parallel approach, we discussed and worked our way around that still fitted in to our team’s framework of conventions.

## Suggestions for course improvement

The Personal Sprint Retrospective write ups could be demarcated better in its scope. A questionnaire form would serve as a better alternative. For instance, how many tickets were completed? Was the stretch goal fulfilled? That would also help the graders to easily assess the team’s progress.

Until the last sprint, the teams were kept isolated from one another and we were less informed about others’ approach. Briefly discussing each team’s approach (at least in Piazza) would help us think better and potentially correct ourselves periodically. Since this is an open-ended project, understanding various approaches for the same problem would be a great learning experience.

More stress on the incorporation of design patterns, testing strategies and other core programming principles than on feature completion by both client and developers would certainly result in better fulfillment of this course’s purpose.