Final Report

Md Istiaque Al Jobayer (27149323)

Agile Practices

For assignment 5, we adopted the agile practices of sprint planning, sprint review, and daily meetings. The basic process per iteration was as follows:

1. Begin sprint with a sprint planning session to decide on tasks for the sprint
2. Do daily meetings to track progress
3. At the end of the sprint, review progress and effectiveness of process, and discuss enhancements

We saw some nice improvements to our workflow when we implemented the above process. For starters, having planned out tasks in advance ensured that we had a very good idea of what was involved in the work and could effectively divide the work. Daily meetings were quite helpful in keeping track of any blocks in the work. Compared to our prior assignments, we noticed that we were more up-to-date on the state of our project since we were forced to share concerns and progress reports every day. We didn't find much use for the sprint review at the end, but that is to be expected since we only had two iterations to go through. Our first iteration's review ended with no major changes to the process.

The academic setting forced us to keep our process simple since both of us had to deal with quite a bit of workload apart from this unit's. While we kept to good software engineering practices such as keeping code modular and following the single responsibility principle, we found that a full adoption of all agile practices was overkill for this assignment given the limited time window within which to see any of the long-term benefits.

In retrospect, there is a lot of benefit to the agile techniques since it keeps a nice balance between specification/design work vs implementation work. For a fast-moving project such as this (weekly due dates), it was quite ideal in keeping us on track and keeping both members on the same page. Our implementation of the process was kept light on purpose, but one area of improvement is coordinating integration of different features. Our current approach was to do pair-programming most of the time, meaning that we progressed at half the speed of parallel work.

Working in Teams

Our approach to teamwork involved managing communications via WhatsApp and Trello boards. We used WhatsApp to keep ourselves updated with daily progress and Trello boards to track our user stories and tasks for each sprint. WhatsApp was very effective as an alternative to daily-meetings when both of us were starved for time and simply wanted to keep ourselves updated on the progress.

We divided tasks based on their size so that every member had roughly equal work. It's a fairly straightforward approach that shouldn't need much explaining. We simply divide the tasks based on their sizes into roughly equal sized assignments -- collections of tasks assigned to a member. There wasn't much to complain here since both of us have to balance this work with other responsibilities.

The practices we applied would not carry over nicely into a bigger team, if only because of the fact that our communication used an instant messaging service not designed for scale and Trello is not very suitable for use in agile processes. We would require a proper project management software to ensure proper task tracking and team collaboration.

Design

Our approach to design involved a mix of functional and object-oriented techniques to manage complexity. Due to time constraints and other responsibilities, we did not manage to build the best possible architecture, but our design is relatively modular and, with a refactoring phase, should be ready for further enhancements on top of the core features.

Our design places most of the logic on the server-side, with the client simply acting as a thin view over the server's state. As a result, we have a reduced need for clients to be fast machines with a lot of memory. The client has access to these three important states in the server:

1. Signal sequence from the motion sensor
2. Motion sequence from the classifier in the decoder
3. Character sequence from the decoded message

Our decoder process is designed to be on-line so that it assumes an infinite stream of signals, as is the case when a motion sensor is polled indefinitely. It commits to a decoding decision only when the choice is unique, otherwise it retains the state for future signals, which might force a choice.

Given this architecture, we have cleanly separated the different concerns within our application. As a consequence, it is quite easy to pinpoint bugs in the code to a certain software component.

Our current user-experience is very primitive and, as such, would require much work before it is ready for public consumption (e.g. teaching young children about codes). Although it offers some basic control over the server, it is lacking in its presentation of the server state. Future improvements should be focused on displaying server results in a user-friendly format (e.g. visualizations).