MONASH University
Information Technology

Advanced Programming

FIT3140

Assignment 5 Final Report

Team 29 – Matthew Ready

Name	Student ID
Matthew Ready	25121987

Table of contents

Topic 1: Agile Software Development Practices

When developing our morse code decoder project, we adopted several agile processes, including Test Driven Development, using a Backlog, and Refactoring, among others.

For this project, it was critical that a few of our user stories were properly unit tested. After writing the code for a basic implementation, we wrote some basic unit tests to ensure that the code was in working order. However, this is certainly not Test Driven Development; the tests should have been written first and then the code should have been written to match the test specification. We realised our mistake only after the first iteration was complete. In iteration 2, we alleviated this by writing our tests prior to any code for new functionality. This is far closer to real test driven development. Further improvements could have been made by enforcing stricter rules and test coverage by ensuring that new code is only written if an appropriate test exists that requires the code to exist.

We also made good use of a project backlog. We converted all required functionalities into user stories and stored them onto a Trello board. Using T-shirt sizing, we approximated the required amount of time to complete each story relative to the other stories. This allowed us to fairly divide workload between the team members. Trello was very useful in seeing the state of the project iteration and ensuring that all required tasks made it into our submission on time.

The final major aspect of Agile adopted was refactoring. As the project grew, things needed to be renamed and moved around to maintain a neat and readable codebase. For example, a class named `"DotDashInterpreter"` was eventually renamed to `"ShortLongInterpreter"` to closer match the specification. Adjusting things in this way improves the nonfunctional aspects of the code (removing code smells) while leaving the functional aspects alone. It's very important to make sure that refactoring is not forgotten and is frequently done throughout the development of the software. Without it, code will quickly become highly resistant to change and very unreadable.

Due to the small size of the team, the requirements and the academic environment, many Agile techniques were not used. For example, having a scrum daily meeting was totally infeasible. We did not have nearly enough time for both of us to get together to talk about the assignment amongst all our other units. We also did not manage to perform any sort of retrospective of the work we did in the first iteration before commencing the second one.

Topic 2: Working in teams

This assignment was completed in teams of 2 people. Most of the communication was performed online, using Facebook and Trello. Facebook was used to check up on each other in real time (for asking questions), while Trello was used to make sure we both had a clear idea of how the tasks were delegated and how the project was moving forwards. We also met during our allocated tutorial classes and discussed how we would approach the tasks there. It would have been better to meet more frequently offline, as when you meet offline it's easy to discuss all facets of the project, including helping each other out when a technical issue is encountered.

Allocation of tasks was performed by choosing the user stories we each wanted to implement, and discussing whenever there was a conflict. We would also make sure to assign the card to someone on Trello. This way, we could remember easily who was responsible for each task. The task assignment was mostly acceptable, but could have been handled a little better by agreeing on tasks a bit earlier. Another problem that we encountered was that we occasionally misunderstood the specification and had to adjust the tasks and reallocate them. We would be better served to thoroughly discuss the user stories we're implementing and breaking them down more appropriately before allocating them.

In a small company, these practises would become more difficult. First of all, agreeing on who will do each task when there are only two people is very easy. With several, it becomes a challenge to make sure that each person is satisfied with their workload and are working on an aspect of the project that they are well suited to. Using the planning game to estimate user story workload would be an excellent approach to ensuring each team member agrees upon the amount of work they are getting. The approach to communicating would also need to be changed. Having simple facebook messages is not going to work in a small company. Each person in the team needs to have some idea of the tasks that others are working on. It's at this time that the daily scrum meetings become very important. You would also be able to physically talk with people every day of the week in a small company, so digital messaging would not be quite as important. Finally, Trello's task board will likely become full of very unrelated tasks that separate people are working on if we don't adopt any strategies to alleviate this. One idea would be to use separate Trello boards, but you could even adopt an entirely new service that is well suited for Agile development, such as Rally.

Topic 3: Design

Our morse code application is designed to be as extensible as reasonably possible. We have divided our application up into several loosely coupled components. The code that determines whether or not a signal is long or short has no dependencies on the code that interprets morse code. This makes it easy to adjust the way signals enter the morse code interpreter, and helps facilitate easy testing. It also means that if you wish to implement a different kind of morse code interpreter (say, one for Japanese morse code), you don't need to rewrite the code that determines which motions are long and short. We also abstract away the specific hardware that is used: our software can be used with a virtual hardware device (where the user just presses enter to stop and start motion) or with the real Arduino device. This means that any new hardware device can be added without having to worry about how the motion start and motion end events are translated.

Maintaining the code should also be relatively easy. Since all classes are cohesive and loosely coupled, it's fair to say that small changes in one class won't break unrelated functionality elsewhere. However, dealing with issues relating to client-server communication can be a little troubling because a third server (firebase) is involved. In this case, it is best to use the firebase web tools to see if the problem exists on the server or client side. We have also written several unit tests to ensure that the code works under several conditions, both normal and abnormal. This will help to ensure that when a bug is introduced it will quickly be found and won't be left for a user to discover accidentally. If were to make the design of our code easier to maintain, I would first go about adding additional tests for all parts of the code that remain untested.

The visual design of the application turned out to be quite decent. We have nice looking animated switches, and a decent looking user interface. If we were to change it for younger children, we would probably go about adding more colours and larger buttons. Since our user interface is entirely written in HTML and SASS, it makes it easy to easy to adjust how anything looks without affecting functionality. Using Sass was a great choice to maintain the readability of the stylesheets and also allow easy extension and modification.