Inf2C Software Engineering
Coursework 3 Project Log

Dylan Thinnes, s1737075

November 28, 2018

1 Overview

This report will go over three aspects of every day in which work was done:

- Plans for the day.
- Achieved results. This includes:
  - Items of work done for that day
  - The time spent for that day
- Reflections on improvements and obstacles.

The majority of this log is modeled after my commit history and personal recollection of events as I typed them up.

November 17th

1.1 Plans for the Day

We began the day by looking at the coursework specification. Most importantly, we sought to answer three main questions before actually doing any coding:

1. **What work is there to be done?**
   This was split into three categories:
   (a) Implementation of individual components
   Each component must be defined according mostly to the specification laid out in our class diagram. Appropriate types must be chosen for internal variables, the functions internal to the class must be written, and getters must be written for any private variables we wish to expose.
   (b) Gluing components together according to user stories
   This work resides mostly in AuctionHouseImp. It must implement each documented user story and, in doing so, tie together the individual internal functions of each related class to produce the user story's primary outcome.
   (c) Testing standalone components and relationships
   JUnit tests must finally be written to be sure that all of these user stories and the individual components behave correctly. This is especially important for ensuring functions abort appropriately when bad inputs are given.
2. **How will we divide the work to be done?**
   Generally, we resolved to split coding jobs as we went along. Mikey has more extensive experience with testing, so he would have a bias towards that. Simultaneously, I find general grunt coding less tedious, and would slightly bias myself towards that work.

3. **What tooling and external specification should we consider?**
   Our workflow would remain unchanged from our usual ones, using vim/Emacs, with git, latex, and bash utilities.

   We discussed the relative merits of switching over to a more accepted Netbeans or Eclipse-based workflow, but considered it tedious in comparison to writing our own scripts for common tasks and using our already powerful editors to write up everything.

   In the way of coding conventions, we read through the Oracle guidelines for coding styles. We found nothing we disagreed with, and resolved to follow it.

   We read some brief blog posts about JUnit, and the documentation of logger, to be sure we could troubleshoot as necessary and use them effectively.

After this, we decided to implement the skeletons of our structure, and to make workflow tweaks as necessary since this was the earliest stage.

1.2 Achieved Results

I continued the day by developing skeletons and proper class hierarchies of all of our components. This way, we have everything clearly named, preventing naming conflicts in our function and variable definitions between merges.

I created a 'compile' script, to compile the code before running it. This made typechecking very simple and immediate, so we knew that we had an error the second we typed it.

I modified our test script to pipe stderr to stdout - this would let us run grep, tail, and other shell utilities on our tests.

1.3 Reflection

Workflow was significantly improved so that we can identify type errors and failed tests quickly. This will be especially useful in the long run.

Also, most major skeletons were set up successfully, so we are prepared to begin our implementation with the confidence that we will not have to resolve variable naming conflicts, as everything has been determined here ahead of time.
Hours Spent: 7

2 November 18th

2.1 Plans for the Day

- Refine our class inheritance structure, especially for actors

- Ensure correct encapsulation
- Write a user story in AuctionHouseImp

2.2 Achieved Results

2.2.1 Refine class inheritance structure

After brief discussion, I reworked the hierarchy of our actors (Auctioneer, MemberOfPublic, Seller, Buyer).

Previously, all actors implemented the Actor interface, which required the "viewCatalogue" function. Instead, the Actor became a class unto itself and implemented viewCatalogue for all actors.

Some discussion was had about the inclusion of MemberOfPublic. Mikey argued that because it wasn't in the spec, we shouldn't include it. I argued that we should include it since it would serve as an example of the Actor class's usability as an anonymous, read-only actor. Mikey relented on the grounds that it wouldn't complicate tests or user stories.

2.2.2 Ensure correct encapsulation

In the way of ensuring correct encapsulation, I found many skeleton implementations where I had mistakenly allowed class attributes to be public where they needed to be private. I privatized those variables, and wrote some getters.

2.2.3 Implement a user story

I began work on the lot closing transaction system. Mikey also made progress on this - upon merging in his changes, I found that our code is very pleasingly similar.

In this endeavour, I also made sure to verify an auctioneer as existent for every lot opening/closing event.

2.3 Reflection

Our work from yesterday was very useful in quick iteration of the code design - typing let us catch most if not all bugs early.

The tests e were provided, and some of Mikey's beginning work with tests, was very useful for catching other bugs in the lot closing system. I think it is advantageous to have tests written ahead of time as code mutates.
Hours Spent: 6

3 November 23rd

3.1 Plans for the Day

- Track and fix bugs Enough logic has been written at this early point that it is important that we go over it to find potential runtime issues.

3.2 Achieved Results

I found a few bugs, especially in the payment transfer logic, that needed to be immediately fixed. Other logic in the way of verifying transfers and reacting to payment errors could be added in.

3.3 Reflection

It quickly became apparent that our modular design and our choice of workflows was an excellent choice for this project. Whenever we were writing code in the same parts of the codebase, merges would usually be automatic, thanks to git's merge system.

Also, the use of TODOs across the system did not introduce any fragmentation of priorities or knowledge - thanks to our environment, it was simple to write a quick grep command that would find any active TODOs in the system.

We clearly work well as a team. We easily transfer between parts of the codebase, can communicate issues quickly, and are comfortable with one another's code. This lets us extend and review one another's work rapidly.
Hours Spent: 3

4 November 24th

4.1 Plans for the Day

- Work together on report, esp. class diagram
- Catch up on latex transcription of my project log

4.2 Achieved Results

The report was finished piecemeal — initially we discussed the differences between our CW2 report and what we had actually implemented in CW3. Once we were clear on that subject, Mikey wrote up a skeleton set of titles. Then, he fleshed out paragraphs for the first two of three sections. We refrained from writing too much more — we wanted feedback first.

I also began transcribing what had up until then been a project log only in plaintext into latex.

4.3 Reflection

The deadline is fast approaching, and much of our implementation, especially on a macro level, is complete. So, it is both logical and important to begin our report.

Everything we wanted to achieve was achieved.

Hours Spent: 3

5 November 25th

5.1 Plans for the Day

Planned to finish off my side of reports.

5.2 Achieved Results

I edited Mikey's previous project report paragraphs and wrote up sections Mikey hadn't touched. Mostly, this included documenting structural decisions and improving explanations of differences between CW2 specification and CW3 implementation.

5.3 Reflection

Mikey and I have different writing styles — generally, he is more concise. I find it useful to run my sentences past him before committing them to git.
Hours Spent: 2

6 November 26th

6.1 Plans for the Day

Just look over the code, make sure that nothing is amiss.

6.2 Achieved Results

Main issue I could find was authorship attributions. I changed all instances of "@author djt" to be "@author Dylan Thinnes, Michael Andrejczuk" instead.

6.3 Reflection

Fewer issues are coming to the forefront than before. It is really a question of getting smaller details tidied away before the coursework deadline.

The lack of feedback at this point is a problem — we will have to wait until feedback comes out to make our final documentation and implementation edits.

Hours Spent: 1

7 November 28th

7.1 Plans for the Day

- Finish off all documentation in lieu of looming deadline
- Finish transcribing, compiling, and submitting my project log

7.2 Achieved Results

Today, I made no code changes.

All efforts were directed towards taking into account the feedback we got, and amending our class diagram appropriately.

I also finished transcribing my log, and final submissions were made. This took longer than I expected.

7.3 Reflection

This entire coursework was a useful exercise in reading a design prompt, making appropriate plans and specifications, and then seeing how those specifications can be well realized in a well-tested implementation.

Keeping a log, and as much documentation as we did, also put into scope how difficult the issue of documentation must be for real—life, large projects.

Finally, managing these issues a team, with a workflow and multiple editor environments, served as a useful lesson in mutual team dynamics.
Hours Spent: 3