# Software Engineering Group 1

# Reflective Essay

Introduction

The premise of the project was to develop an application for a customer using an agile methodology and software engineering tools. Through a number of deliverables, the application was iterated from a simple location sent to a server, to an app with the capabilities of showing a house price heatmap, and crime data for a given area. We used a variety of software engineering tools, including Trello for task distribution and tracking, Git for version control and Slack for development communications.

Good Feature

On the whole, the most useful tool for our Software Engineering process was probably Git, and the Github website. Git allows for easy version control and maintenance of a complex workspace of code which is constantly changing, as is the case in an Agile environment, and we used it extensively, not least because our group’s circumstances meant that most of our work was done ‘remotely’. Due to various commitments of each team member, we found it difficult to meet face to face a lot of the time, and so Git kept us up to date (for the most part, however as mentioned below, we failed to commit frequently enough at times) on one another’s work, allowing us to see the latest things that somebody has done.

Our methodology on Git was to merge work into Master only after at least two of us had reviewed and approved of it, be that via a quick demo of the app front-end, a look through the code on smaller commits, or a run through of the test suite on the back-end. We attempted to use code reviews as much as possible, and this linked in with our use of Trello, as we would move a task card from ‘In Progress’ to ‘In Review’ in Trello when ready, and then the team would be aware that a feature is ready for checking, which we could discuss. The code reviews we did gave us some level of knowledge sharing, and we felt their benefits by feeling more comfortable with both the client and server sides of the code, allowing those of us who were mainly working on the back-end to jump in and help a bit on the front-end and vice versa, and were mostly combined with the pull requests on Github, which we used to make sure we weren’t merging any code without due confirmation.

Along with our policy on when to merge to master, we decided our branching strategy would be to use a new branch for each fairly large feature, such as ‘basic-node-server’ for the initial server setup, or ‘Crimedata’ for the crime data overlay feature, since we agreed that this was the most intuitive for all of us, and with the potential for there to be smaller sub-branches from each of these features as necessary. Commitment to this, and to trying to ensure we wouldn’t run into any big merge conflict problems, enabled us to have quite a good level of modularity within our code, wherein separate features wouldn’t be directly in conflict with others, allowing people to work on separate features in the same overall files without causing chaos, and thankfully we only ran into one merge conflict problem throughout the project, which we had to pick apart using the Source Tree Git software.

Additionally, Git’s utility was further enhanced by our integrating it with Slack. This meant that we could see when anybody had committed some work, created a new branch or opened a pull request, and thus effectively gave us notifications of work progress, which meant that another team member could go in and check progress, or would know that he can start working on his tasks, and so on. It aided us in keeping up to date with all the complexities of a many-branched Github repository.

If we had had more time, we would have also been able to complete our linking of Github to Jenkins, the auto-build and deployment service, enabling us to automatically build and test the project upon each new commit, as is desirable in real production environments, and even receiving notifications in Slack and on Github about the build status of each branch.

Bad feature

One of the software engineering features that proved to be problematic was our choice of development approach and work distribution. Unfortunately, due to two of our members being away during the first weeks of term, the initiation phase of working out team dynamics and practices was delayed until the more complex tasks started, and we didn’t quite foresee how important it would be to prepare this comprehensively. However, we still maintained communication during this time and the early tasks didn’t need a working unit to complete. Upon meeting and discussing our individual strengths, there was a clear division of knowledge and skillsets – which led to us choosing a divide and conquer approach to the projects. Due to the fact everybody was comfortable with different aspects of the problem, at the time we felt this was a safe approach to take. However, just two weeks from the major deliverable hand in, one of our members which had been working on the backend fell ill and was absent for just over a week. On top of this, we were also going through the process of upgrading our server as it was take too long to process client requests when they were fetching house price data. The situation bottlenecked the development of project because we were unable to access the server to implement our node js solution or set up important tools like Jenkins and Docker. Although we did manage to develop a working solution, our choice of team structure lead to us not meeting the original deadline and skipping some software engineering practices like continuous integration, deployment and automated testing tools that we now understand are essential to good software engineering.

Reflecting on the module, this could have been avoided with better team communication and the use of better suited software engineering practices like paired programming and mentoring. We had a single point of failure in a core part of our project which was a risk which we failed to identify in the planning phase and looking back, this could’ve happened in other sections of our project as well. If we were to start this term over again, or tackle a similar problem now, we would scrap our initial decision to divide the work and invest in teamwork, communication and paired programming. Spending time sharing knowledge between members and discussing solutions means that everyone understands more about the project which puts the team in a stronger position to deal with conflicts like the one we have experienced. Furthermore, a greater commitment to the Agile methodology of small and frequent commits of work done would have helped alleviate this problem, since there would always have been some up to date code for another team member to work on if another member became unresponsive for a while.