REFLECTIVE ESSAY GROUP 4

Project Context

The outcome of our Advanced Software Engineering project was to produce a cross platform application that displayed house price data in the form of a heatmap within a radius relative to the current location of a user. The project was undertaken by six students within a period of four months. The team adopted a flat hierarchy where the team had one team leader who was the main point of contact with the client and the remainder of the group are subordinate to the team leader. The nature of this project as with every other software product started off with a high-level overview of the software's expectation, and with the accommodation for feature modifications as hinted by the client. As the project progressed with the client, regular meetings in form of the various iterations of the software being developed was demoed to the user and feedback for modifications were collected and implemented, with each set of requirements increasing in complexity.

Positive Outcomes

As a team we were able to explore a lot of different technologies during the process of the project. This was achieved through running spikes whenever there was a need for the introduction of a new technology. Spikes are when the team choose one or two technologies that we potentially want to use and aim to produce a proof of concept in a set amount of time. We then have a meeting and produce the findings and democratically vote on what we feel like the best option is. This meant that we were sure that we are picking the best option for the outcome we are aiming to achieve. It also meant we were aware of the limitations of each technology such that it allowed for better planning when deciding our approach.

Pair programming was explored during certain sections of the project. Where two people from the team were able to work on a piece of code or part of the project. This provided advantages such as knowledge sharing, as it was an opportunity for individuals who are more competent on one area to shadow or mentor someone else. It also brought the advantage of saving time as the job that would have been done by one person is now done by two people, it also means that errors or redundancies are likely to be spotted much sooner, because as one person is coding the other is analysing what is being done and can suggest improvements. The team also achieved the idea of knowledge sharing in another format known as mob programming, this was achieved using Zoom, where the one member of the team shared their screen and other members in the call would work together to edit and make changes to the code.

Throughout the duration of the project, every member of the group was working remotely, as each person had their own machines and predominantly worked from home. As a team it never caused an issue as we were able to maintain communication between all members daily through our instant messaging application of choice: Slack. It enabled for there to be clear communication, so no one felt isolated from the project. As a team we ensured that meeting timings were negotiated and communicated clearly through the chat, with most meetings being held over a video messaging application called 'Zoom'. Slack also allowed us to get instant feedback on any issues or questions we have had about various elements of the project. The application allowed us to link the external application e.g. GitHub, Travis CI and Hooks etc. A separate channel was created where the team was notified about new pull requests or the status of build tests, these notifications were set up to automatically trigger.

Code reviews were implemented in an ad hoc manor however they proved to work effectively. The team implemented a GitHub feature which meant that any pull request had to be reviewed by at least one member of the team before it was committed to master. Additionally, it is not possible to directly commit to the master as it is protected. No one, person oversaw reviewing a pull request giving any member of the team to review and approve a request. We aimed to follow a set of guidelines set out by Google for code reviews, some guidelines included: code is well designed, UI changes are sensible, code is not more complicated than necessary etc. This was very effective as it allowed for the reviewer to pass any comments or improvements, they left necessary, it was the responsibility of the original author to implement these changes. We aimed to have pull requests reviewed and committed as soon as possible, to mitigate the written code becoming outdated. This was aided with the implementation of GitHooks which notified the team through Slack.

Continuous Integration (CI) and Continuous Delivery (CD)

We use CI/CD in our project to enable us to deliver software faster in a continuous and repeatable manner. The process has two stages: continuous integration and continuous delivery.

In the CI stage, we run all tests from our application, the react-native application and from the platform specific applications: Android and iOS. Tests are run automatically by the CI server (Travis CI) when a new pull request is raised on GitHub. This acts as a gateway, if the tests pass then the pull request can be reviewed and merged into master. In the CD stage we have defined build conditions that will create an Android application and release it for beta testing. The above processes are automated and are triggered automatically by the CI/CD server based on external interactions (GitHub hooks for pull requests and merges to master) or based on internal interactions (different Boolean flags enabled or disabled on the CI/CD server).

Our CI/CD pipeline is described using a configuration file called `.travis.yml`. This configuration file describes how the virtual machine should be configured, what software should be installed and which steps to be executed. All interactions in the CI/CD server run in virtual machines that are booted using the operating system required for the application: Ubuntu Linux for Node.js and Android and Mac OS for iOS versions of the application.

The first stage is to describe what base image you want for your virtual machine and the CI/CD server will provision it for you. The next stage is to describe what software you need to have installed on that machine to run, test and build your application. For example, Node.js and react-native are always required to be installed. Furthermore, for Android requires Java8 installed while for iOS requires XCode. All these tools are installed by the CI/CD server based on the configuration file.

Last stage is to describe which steps should be followed to execute your tasks. For example, in the test stage, we describe in the configuration file how to run (which commands to be triggered) our tests for each of the three platforms. One requires running JavaScript tests using Jest, another one requires running Java tests using Gradle and the last one requires running Objective-C tests. The CI/CD server allows us to describe build stages and jobs. Jobs that belong to the same build stage will run in parallel, like the test jobs we have. They are independent and allow us to have a fast test stage when they run in parallel. Build stages run sequential, based on the order in which they are defined in the configuration file. This allows us to define our CI/CD pipeline sequentially: first compile and test our code, then version, build and release for beta testing. The latest merge to master and last build stage is to tag, version, build and release to production.

Besides our CI/CD server, we use other helper tools for distributing our application. We use Fastlane to distribute our application to the Firebase App Distribution for beta testing and to Google Play Store and Apple Store for alpha and production releases. Fastlane allows us to use a single tool to deploy to both application stores (Google and Apple). We also use Firebase and a suite of their services for app

distribution for beta testing, monitoring our application's performance and getting notified via issues if we exceed certain predefined thresholds. We also use it for monitoring our application's error rate based on release version such that we can monitor any new bugs that were introduced with new releases. We are aware of the code quality and code degradation in time and we use several tools to prevent and monitor code quality. One of the is ESLint for local development such that our entire team is aligned to the same set of standards in writing code. Another tool is SonarCloud that will run code analysis for each pull request. If the quality of the new code does not meet the requirements, we defined in the quality gate, then this pull request cannot be merged into master until the code improvements happen.

Our team is pro-active in security and we put a lot of effort in protecting ourselves and our clients for these possible attacks. We use Snyk to scan our repository and all our pull requests for software vulnerabilities. Snyk produces a report that we use to upgrade our libraries such that we reduce the number of vulnerabilities we have and preventing new vulnerabilities being added to the master branch by scanning each pull request. We also encrypt our secrets and do not put them in our public GitHub repository. They are encrypted in the CI/CD server and decrypted on the server only when they are needed. We also watch GitHub and Google notification for possible key leaks in our repository.

We try to follow the best practices in industry in terms of software development. We continuously evolve and improve our processes and we try to capture as much knowledge as possible into our README file. We are a team and we help each other in developing the best value we can for our customer and in becoming better professionals and better humans.

Negative Outcomes

One of the main issues we had as a group were the meetings that we had. A lot of them were done using an online tool called Zoom, which allowed us to video call as a group. This seemed like a good idea as it allowed actions like screen share allowing us to demonstrate solutions to the rest of the group. However, the main issue was that people had different hardware and internet speeds, meaning we spent a lot of time configuring hardware. Zoom was also prone to a latency issues meaning we needed a constant fast internet speed as mentioned. Meaning there were a lot of times when multiple people were talking above each other. Overall, this was a good tool for demonstrating code etc. however it did cause a lot of inefficiencies as a lot of time was wasted when communicating and meeting timings were not used efficiently.

An outcome of the project was to produce an application capable of running on iOS and particularly on iPhone. During development we used an iPhone emulator and we not able to deliver to the Apple app store like we able to do for the Android version. The main determining factor was the cost of $99 involved to become an Apple developer compared to the Google Play Store that incurred no cost. However, the Play Store enrolment took some time to gain approval the cost was zero and hence we focused development more on Android. If the team were able to gain funding from an external source then it may have been possible to focus equally on Android and Apple development, which is something that needs to be considered in the future.

We are an inclusive team and we care for people with different skills and different levels of skills. We failed in making all our team members actively contribute to our project. Due to different reasons they did not participate in our activities or in the development of the project. We tried to make them active participants and we failed. This is an area where we need to improve in the future by aiming to improve communication in the early stages of the project, ensuring that we aim to find where they fit best in the team's structure and culture of working.