Reflective Essay

ASE Group 2

This project saw a software engineering team of size 6 look to produce an application that visualised house price data across the UK. The application was to run on mobile devices and utilise a client-server model. During the process of creating the application it was pertinent that the team utilises good software engineering practice and automated tools to ensure the delivery of the application adhered to a number of short deadlines. To aid in the delegation of responsibility for tasks, as well as accounting for team members strengths and weaknesses, a number of sub teams were introduced. This saw the team split based on the mobile platforms (Android and iOS), the backend, design and DevOps. Movement of members within these teams was somewhat fluid based on the need for resources in the different areas. This pragmatism caused resource allocation in terms of people hours changing somewhat dependent on the needs of the deliverable at hand.

In light of this, the process for each of the deliverables followed a set structure. The AGILE method was utilised throughout, splitting each deliverable into a number of smaller sprints. The start of each deliverable saw the first sprint dedicated to requirements analysis, design and any research that may be required to meet these requirements. With the requirements analysis performed first, a weekly meeting then allowed for tasks to be allocated amongst members of respective teams and pointed. Tracking of these tasks was performed using GitHub issues and an automated Kanban Board (waffle.io) to keep each sub-team updated with each others progress. The use of a Kanban Board allowed the lifecycle of an issue to be easily visualised.

The lifecycle of an issue was consistent across all deliverables. An issue was assigned to a team member based on their role within the team. A branch was created following a specific branch nomenclature pertaining to this issue. The team member would work locally using git, committing changes locally as progress was made. This allowed for local testing to be performed to prevent time being lost in waiting for the extended test suite built for continuous integration testing to be run. A commit to the remote repository was performed when substantial progress was made or the issue deemed complete to allow for code review and integration testing. A commit to a remote branch triggered a Travis Continuous Integration tool build to ensure that any changes could be safely integrated into the code base without causing failure in any other components of the application. Once an issue was deemed complete by the assignee a pull request was created to allow for code review. Reviewees were assigned within subteams to allow for effective code review across the different languages used. This allowed for feedback on code quality alongside the raising of concerns regarding integration that had been flagged by Travis. A branch for an issue was finally merged once all reviewees had approved the request. Frontend design changes were not merged without communication between the front end teams to ensure that the application design was consistent across platform with the exception of platform specific conventions.

A feature of the the team's software engineering process that was especially helpful throughout the project was the use of the Discord application. Discord provided open communication channels for the duration of the project, a set-up that became increasingly pertinent as our meetings dropped off towards the end of the project (this will be addressed in due course). The project plan split the team into smaller sub-groups of operations that were reflected in the set-up of a number of different Discord channels: General, iOS, Android, Backend. In addition to these chat channels, a GitHub webhook and a Travis webhook were implemented, provided automated alerts for every member of the team. This helped the team keep track of Travis build jobs as they passed or failed (ensuring merges would not introduce further issues into our code base).

The use of Discord aided the team in keeping all communication in one place. Dedicated chat channels allowed members of each sub-team to discuss issues/features without disturbing the workflow of other members of the greater team while still providing a record of conversation that any member of the team can access at all times for reference if necessary. These dedicated channels provided a means by which the mentors (as defined within the project plan) could share their knowledge and assist those as part of their sub-teams. This became particularly useful for later deliverables when DevOps was introduced. When working within Travis CI with 3 separate languages, a clear idea of who to ask regarding the build processes for Android, NodeJS and Swift was invaluable in saving time and providing a loose form of peer programming.

Having a place in which any team member could enter in general discussion also provided a straight forward means of suggesting of proposing new features or improvements. As a result of this no overly ambitious features were attempted as each suggestion could be discussed in length remotely, ensuring that all deliverables were met on time. These suggestions could then be translated into GitHub issues to give a more formal means by which to track them. In saying this, the use of remote communication proved not to be a direct replacement for physical meetings, a concern that is addressed in the latter parts of this essay.

The integration of automated alerts in the form of the before-mentioned GitHub and Travis webhooks within the same application provided an easy process by which two of the most important tools could be monitored. Notifications were received for commits, merges and most importantly pull requests to ensure that code reviews were performed within a timely manner and that pull requests that were ready to merge a useful feature into the master branch were not sat idle waiting for approval for several days. Further alerts were introduced within the final deliverable from FireBase Crashlytics and Papertrail. To utilise Discord more effectively these alerts could have been forwarded onto Discord to provide a central hub for all of the alerts.

In general, the team were happy with the application produced as part of the project. The features that were proposed throughout were met in a timely manner and all team members

contributed within their roles. Adhering to a well defined software engineering process throughout the course of the project certainly contributed positively to the team's ability to deliver a feature rich application.

While the general demeanor of the group was positive and the project came to completion with minimal friction, there were some areas that could be improved. More notably, our organizational skills could have been better. We were using waffle.io to track our issues, we perhaps relied on it more heavily than we should have. Initially, our sprint meetings were mostly about issue ideas and development, with an aside to describe the relationship between the issues. This was a long and cumbersome process that was abandoned in the latter half of the project, due to all members having adopted a workflow where it felt unnecessary to do so. Such a shift was detrimental to the overall transparency of the work - we could all be working on features, and have them all be on waffle, but we would not know the exact nature because not everyone would communicate to the group what they were working on.

On the note of issue tracking, we had made an attempt to use sprints as a cyclical structure to our workflow. We attempted to assign tasks to specific sprints and have a retrospective at the end of each of them. The goal of this was to have regular intervals in which we would provide feedback (positive or negative) to each member of the group, and have a quantifiable manner in which we could track our progress instead of seeing an endless list of issues that had yet to be resolved. Unfortunately we only managed to have a single retrospective, and it was for our first attempt at a sprint. We came to the conclusion that the reason that we were unable to stick to the format is because the amount of work was skewed to different subgroups each week. Because of that, providing feedback through a retrospective in a regular fashion proved itself to be difficult since there could not be any interoperable feedback. This is also the reason that sprints were difficult to stick to, since the workload was skewed at different times, it would not make sense to fit their tasks to an arbitrary one week time period.

To mitigate the issue of transparency, we did use Discord to have everybody solve their problems through cooperation and update everybody on potential features we could add, but this did not remove the problem of members not knowing what other member were doing in greater detail. More frequent meetings could have been the solution to this since we were only meeting up once a week as a whole group during the Thursday lab session. They would not have had to been long or extensive meetings, it could have been over coffee or lunch, just enough to update each other on progress and roadblocks. This could also mitigate the eventuality where a group member feels like they are working more or less than other members.

The detrimental effects of this lack of transparency and decline in meeting hours were most visible in the team's presentation for the Task 4 deliverable. On reflection, the root cause of the adhoc/unprofessional nature of this presentation was the time spent preparing for the presentation and the lack of communal knowledge regarding exactly what some of the completed tasks actually entailed. This led to group members simply interjecting in the presentation to input the somewhat varied knowledge there was among the smaller teams

regarding the application as a whole. More positively, the team reflected heavily after this presentation and worked hard within a short time frame to ensure that, along with additional features, the final deliverable would be presented in a clear, concise and informative manner. This was achieved through extended communication regarding what was being performed and why, outside of our automated issue tracking. Moreover, a meeting was scheduled in which the team properly discussed the structure of the presentation alongside what would be demonstrated, by who and what value this brought to the presentation. This allowed the group to present in a logical order, making the presentation considerably more coherent in comparison to the ad hoc nature of the previous presentation.

It was interesting that these issues came up in the first place, since our measures of using Waffle.io and Discord were put in place to avoid these problems. It was also an issue that was not particularly sudden, so it was something that had grown over the weeks as we noticed that communication quality was faltering. It proves that there is no substitute for more frequent vocal communication.

On a more positive note, we can successfully say that we did not attempt to implement a feature that was too ambitious. Anything that members committed to vocally was generally completed in timely manner.

In conclusion, the team took a number of positives from the project as a whole. All work that was assigned was completed in reasonable time and there was little to no friction amongst group members throughout the process. A number of automated tools were studied and used and the team's use of mentoring has enriched the skills of a number of group members. That being said, the group has learned a lot regarding communication, organisation and the value of face to face meetings. Unfortunately, it was to the detriment of the project that this lesson was learned leaving the group with plenty to reflect on after the Task 4 deliverable. Nevertheless, given the speed of reflections and actions taken to resolve this issue and mitigate these problems in the future, if the group were to go on to perform further work the overall quality of this work and the software engineering process adhered to would certainly be of a higher quality.