**Reflective Essay**

**ASE-Group-1**

**Team Background**

The team was composed of five members, with Agne joining the team at the end of Task 2. No members of the team were acquainted before the project. The team has a mixed age range and software engineering background, as listed:

* *Stephen Dixon* (53) – Currently studying Advanced Computer Science MSc. Has been a Software Manager, Senior Business Analyst and Senior Software Developer. 26 years Empress 4GL experience and 1-year Java, Java Script.
* *Gerardo Juárez* (26) – Currently studying Evolutionary and Adaptive Systems MSc, Previously studied Computers Systems Engineering BSc. Work experience in software development (3 years), previous experience in C#.
* *Agne Kovalkovaite* (23) – Currently studying Advanced Computer Science MSc part-time while working as a Systems Administrator intern at the University of Sussex. Previously studied Computing for Multimedia BSc. Previous experience in software development from BSc degree
* *Ho Kan, Leung* (23) – Currently studying Computing with Digital Media MSc. Previously studied Accounting and Finance BSc. No previous experience in Software Engineering or Computer Science, but relevant experience in Android Application Development and C#.
* *Joshua Weeden* (22) – Currently studying Computing for Digital Media MSc. Previously studied Computer Science and AI BSc. No previous work experience in software engineering, but relevant experience in Android Application Development.

**Project Approach**

The project was approached with the mentality that each member would have a chance to try each aspect of development in software engineering, be it programming, back-end or team leadership. Each member has a different background with varied skills, so it was thought by mixing members in different roles skills could be shared and learnt. A couple of members had a small amount of experience in Android Development, but every member wanted to reach an average level of competency.

**Helpful Features of the Group’s Software Engineering Process**

A positive in the process was the number of assisting software engineering aids that were deployed, and how these both helped to improve communication, and also the quality of the application. All in all, the project ran rather efficiently, with the meeting of deadlines being associated with the practices described below.

**Project Management Software**

Project management software (Glip) proved to be the saving grace of the engineering process. Part of the software, a simple task list, editable and available for all members to see online managed to rectify any communication issues (as described later, project management software had basic tasks uploaded for Task 2, but was only thoroughly used from the beginning of Task 4 onwards). After devising tasks on the basic step-by-step plan they could be ticked off to signify the task’s completion, keeping the team updated. The team met consistently, but from the project it can be seen how if development is carried out remotely the use of team management software is essential to keep everyone up-to-date and using their time effectively. Although the functionality of this task list may sound small and insignificant, after reading the issues had with project management the weight and importance of such simple functionality should hopefully be apparent.

**Use of the Agile Process**

The Agile process was selected by virtue of having short deadlines, and size of the task – it was an intuitive fit. The timescale immediately eliminated the use of Test-Driven Development; as no team member had written tests before, the code of the application would have to be written around potentially incorrect or badly written tests, and therefore this approach would be too timely to maintain. Furthermore, due to time restrictions there would not be time to fully complete each stage of development before signing off and moving on to the next (i.e. the Waterfall process). There was not enough work to have five members working on separate aspects of each stage, and compiling it together would overcomplicate the process and not be time-efficient.

Within the Agile process continuous communication is crucial. Due to the unpredictability of development all members must be up-to-date with progress so they can continually redirect development as allowances have to be made. Communication is a pillar of Agile process so problems with communication can create problems in the process, and this is where project management software proved to be instrumental.

Before the comprehensive use of project management software in Task 4, during the initial meeting a vague plan was verbally described in which systems would be implemented and how they would connect to create the final application. Without being able to update the team on progress, members would take initiative and push forward in specific areas of development, often making decisions on software or functionality without consulting the group. Therefore, the development process could be labelled as each team member conducting a sprint, and then bringing these features together in the days before the deadline – but not in a well-managed way. Members of the group were unaware of the progress and additions other members have made, but each team member’s development would output what was required and be patched into the system.

With Glip, at the beginning of each new task the specification and architecture were considered, and requirements engineering took place to formulate a list of tasks which were allocated to team members. The plan was not always concise, as software such as databases were chosen with no prior experience of their functionality, and consequently had to updated to make way for new functionality in later tasks e.g. updated the database from Kinvey (database size too small) to Azure (total database size allowed is 2GB, but max 1GB per database – too small) to DynamoDB (not a relational database, as was realised would be required for Task 3) to AWS RDBMS (using MySQL, fulfilled size requirements and is relational). Of course other issues typical of the Agile process due to its adaptability were encountered along the way, with time spent on ideas that were not workable, (Task 2 application was going to have a service inside, passing a JSON directly into the database) but Glip helped to minimise the issues that arose from this.

**Team Member Roles**

Although not exclusively assigned, each member can be seen to have fallen into a role that was most suited to their abilities. This allowed for tasks to be assigned to members who would either already have the expertise to complete it, or the necessary background knowledge to complete the task with some minor research. It was fortunate that the team had a varied knowledge that covered most of the required skills, and this allowed for teaching of skills between team members through peer programming. Tasks listed in Glip could immediately be assigned to the members in relevant roles, and those more general or relating to new technologies picked up by the remaining members.

**Testing**

Once testing had been introduced in the lectures during Task 4 a push was made to implement tests, specifically in relation to the testing of postcode inputs. In each test 100 postcodes were randomly generated using all possible combinations of a postcode, and parsed through the postcode validation method using regular expressions to check they were correct. Any postcodes that failed to pass were returned, and the postcode checker method could be updated accordingly.

This was one test that highlighted errors in the application, and changes were made from this test to reduce the number of bugs a user could encounter. More time should have been dedicated to tests, specifically looking at code coverage and working to improve this figure.

An unaccounted mistake (due to the ever-changing nature of the Agile process) was made in choosing Travis as a CI tool for application testing – Travis is slow to deploy when compared to similar technologies such as Greenhouse, which would have been of greater use to the project, being more straight forward and far easier to set up. Travis also does not currently have proper support for Android emulators, which means currently a dated ARM emulator is required to run and takes ~20mins to build each time an adjustment is uploaded to GitHub. The code is analysed each time a commit is made to GitHub, and as GitHub was side-lined until later tasks (explained later), it would have been easier to runs the tests locally.

The use of CI tools to test on different platforms to that of the coder, run incredibly long tests etc. was not overly necessary to this project. Although not implemented properly within this group, now with an understanding of the software development process it can be seen how testing tools would be useful in future projects.

**Problematic Features of the Group’s Software Engineering Process**

No disastrous mistakes were made during the project in regards to the software engineering process, but with foresight a large number of practices could have been conducted in better fashion, and with more efficiency. As explained below, the majority of issues stemmed from a rotating leadership position in the group.

**Team Leader Rotation / Team Management**

A major issue was had with the management and communication within the group. The idea was formulated and proposed in the Project Plan to rotate the Team leader of the group every few weeks, or upon the changing of tasks. Given that the project was being conducted on a university course and not in the real-world, this decision was taken from an educational standpoint of giving every member some experience in leading a team. It was understood that from a real-world standpoint this would most likely not be a productive approach due to a lack of consistency, and the swapping of leader mid-way through a task, but the extent of issues this would cause was not anticipated.

Giving each member the opportunity to lead initially sounded like good experience for the real-world, but did not work in practice. The first leader was the most experienced member of the team, and as such was exemplary in the role. This made it difficult to hand over responsibility when the time came, and as such the position of team leader was not defined or allocated to an individual as such from Task 3 onwards in the project. It was clear that even though the team leader did not officially change, if the team leader were swapped this could lead to competing opinions on the path development should take. Each member had differing opinions on technologies to use, and features to implement – and this could cause conflict in the group when there is no leader to judge and make a decision for the team.

From the course of the issues that arose helped to highlight the necessity of a consistent leader whose role is to maintain an element of continuity, and keep development on a trajectory where each task builds upon previously implemented features towards an end goal. Without stable management throughout, there was perhaps a slight loss of momentum as each task was a handful of features individually designed by team members and then patched together and integrated before each deadline. Every task that was handed in did not suffer in quality from this lack of leadership, but perhaps could have been completed in a smoother way.

Perhaps the swapping of leaders may not be an objectively bad practice in all circumstances, but rather within this project it was handled in a manner that exacerbated the problems that it can cause. Placing the most experienced member in charge of the simplest task was conceivably a misstep in terms of leaving those less experienced worried about managing a team at a more stressful time, and therefore no one stepped up. The first team leader had the project under control, and was the eldest and most experienced. This deters younger members from taking the position, with the contention that it would feel odd for those less experienced to direct someone who knows better. A suggestion for making leader rotation work would be to order leaders should in order of least confident and experienced, to most, following how the tasks get progressively harder and workload increases. Of course, this would be beneficial in helping rotation to work, but professionally it seems unqualified as a concept to benefit the development of software.

As detailed in the sub-sections below, it will be made clear that the majority of issues stemmed from a lack in leadership, and as a result a lack of communication. The abandonment of swapping leaders led to a lack of a distinct leader in Task 3 where there was no explicit leadership, and consequently delegation and management of tasks was ill-conducted.

**Project Management & Communication**

Without structure or leadership, an in-the-small approach mentality was adapted by each team member, whereby each person was trying to hold the entire program structure in their head, rather than splitting the operation between members. This was best illustrated in Task 2, where over a weekend four members of the team had individually written their own working version of the program from scratch, and the best elements had to be combined into a single application. Up to Task 4, team members assigned themselves an aspect of the program to work on, and worked down this route with little supervision. Had project management software have been fully integrated and used during these tasks, each member would consistently be updated with progress on the application, and therefore not repeat the same work.

A number of factors contributed to the problem of management and communication:

* Towards the end of the second task, a final member joined the group and a change in project management software had to be made from Taiga to Glip due to a limit on the amount of possible members in a free Taiga project. This delayed the integration and use of management software, meaning all communication took place through Facebook group messages only.
  + Once the software was in use, and despite having no clear leader, from Task 4 management processes as simple as an online task list proved to improve communication and the allocation of tasks.
* A similar communication issue was had with GitHub, which fuelled the lack of interaction over ongoing development. Throughout the project every team member had issues with the use of GitHub, specifically committing new code - once or twice code was accidently lost by pulling from GitHub. New code was not committed until the final upload for a task, and therefore there was not consistent, up-to-date version of the program accessible, and consequently members would overlap and work on the same functionality of a program, or not have the program to work on. Members would only update when they had finished a task, not when they had started on it.
  + This problem could have been rectified had a quick group training session of been held, ensuring all members had a complete understanding of committing new updates.
* Facebook Group Chat was used to communicate, with no obvious problems being seen with this during the project. Better, more efficient software could have been used though.
  + Although the Facebook Chat was used to an advantage, other software could have been used with more benefit to the team. The task list was essentially the only used feature of Glip, along with its integration displaying notifications from GitHub and Travis. Through the use of Slack instead of Facebook group chats, GitHub, Travis CI, AWS and Glip could have been integrated into one communication channel, keeping every member’s updated on every aspect of the project in a single screen.

It could be argued that without the several scrums that occurred for each task development would have suffered drastically due the lack of communication about the status of development. Being in constant personal communication reduced the need for management software, but those who could not attend a meeting were left behind and not updated fully until the next meeting. This is not to say that the team was not in constant communication – a successful policy was introduced in the project plan that every member must reply to communications planning meetings regardless of whether they can attend or not – but that it was hard to be aware of the exact progress of development outside of being in a meeting. Project management software rectified this situation.

**Additional Suggestions for Improvement**

A large number of suggestions essay to improve the process were made throughout the essay, but not all were listed. Those that were not relevant to be mentioned above have been bullet pointed here:

* The team should have begun writing unit tests (not just paper tests) for the application earlier in the process, specifically Task 3. Testing and removing bugs at the end of Task 4 meant that bugs created in Task 3 could have been built upon and harder to remove. Members of the team should have been assigned to testing earlier in the process.
* In the first guest lecture with Alan Donohue, it was suggested that a time estimate for each task should be decided upon by the team. This practice would manage the time spent by each member’s more effectively, giving members an effective timescale, and highlights that there are issues that should be resolved if someone is taking longer than anticipated on a single task. This is of course harder to implement outside of a working environment, where everyone is not solely dedicating all their time to the project but have different timetables and multiple projects to work on.
* Maybe not from an efficiency standpoint, but from an educational standpoint it would have been nice to rotate the roles each member had within the project. Each member became specialised in their area of the project, (be it programming or back-end support) and remained working on that area for the entire project (it is of course noted that the swapping of roles could create issues as it did with team leader rotation). Every member wanted to write a portion of code in at least one task, but the short deadlines and Agile process did not permit for this.
* Ensure that as code is written it is well commented so that other team members can download from GitHub, read and understand without a verbal explanation from the programmer.
* More thorough and regular code reviews would have benefitted those who had no direct input in writing code, allowing them to keep their aspects of the project as relevant as possible. Even pair programming between a programmer and a member in another role of the team would help to minimise the communication gap between different roles.
* With a lesson given to ensure every member has an understanding of how to operate it, ensure and enforce that every team member uses GitHub and commits regularly with appropriate messages.
* Written in the original project plan was the idea that the last 5 mins of each meeting would be used to discuss any issues had with development or group. This was never upheld, but would have been extremely good method to allow members to vocalise their concerns (perhaps solving the issue of team leadership).

**Summary**

Reflected by the large amount of suggestions for improvement throughout the essay, and the section reflecting on the problematic aspects of the process, the project certainly was a learning process. It makes sense that without any real leadership maintained throughout the project, maintaining communication became even more crucial than it would have been in a standard software engineering group project, and therefore Glip was essential in helping uphold this in the final tasks. A lesson was learnt and there is no chance a rotating leadership role would be implemented,

In review the importance of a thorough project plan seems like it should have been considered more. A lot of time was spent planning the peer assessment, but not on conflict resolution or other issues such as a lack of communication. What if members stop talking, or a meeting does not resolve the issues that are had? These were not accounted for, and in retrospect these all seem like plausible situations. Despite the peer marking criteria being fairly rigorous in ensuring every member participates, every member agreed it had little impact on their motivation and was forgotten about.

In regards to functioning as a team and the social aspect of software engineering, it was amicable throughout the process. Our initial plan Peer Assessment Criteria stated each member would have to fulfil certain criteria to attain marks out of the pot at the end of the project. Despite many members not achieving all the criteria laid out, these were not unfulfilled due to a lack of motivation, but rather unforeseen personal circumstances such as illness and work. Peer assessment marks are still to be worked out, and will be sent via email.

Overall, the process was a learning curve, and subsequently mistakes were made and lessons learnt from them. Restarting the process again now, with the knowledge gathered would smooth out the process significantly, and allow for more precautions to be made in case of disputes between team members, or issues with software. Communication and management have been found to be essential along with testing, so both of these elements of the software engineering process would be prioritised from the beginning to help guarantee a smoother development process.

**References**

1. Mobile Backend as a Service (mBaaS) for the Enterprise | Kinvey. *Enterprise Mobile Backend as a Service | Kinvey*. 2016. Available at: https://www.kinvey.com/. Accessed December 1, 2016.

2. Microsoft Azure: Cloud Computing Platform and Services. *Azuremicrosoftcom*. 2016. Available at: https://azure.microsoft.com/en-gb/. Accessed December 1, 2016.

3. Amazon DynamoDB Documentation. *Amazon Web Services, Inc*. 2016. Available at: https://aws.amazon.com/documentation/dynamodb/. Accessed December 1, 2016.

4. What Is Amazon Relational Database Service (Amazon RDS)? - Amazon Relational Database Service. *Docsawsamazoncom*. 2016. Available at: http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Welcome.html. Accessed December 5, 2016.

5. Travis CI - Test and Deploy Your Code with Confidence. *Travis-ciorg*. 2016. Available at: https://travis-ci.org/. Accessed December 2, 2016.

6. Firebase Test Lab for Android  |  Firebase. *Firebase*. 2016. Available at: https://firebase.google.com/docs/test-lab/. Accessed December 1, 2016.

7. Cloud-Based IT Systems Monitoring. *Anturis*. 2016. Available at: https://anturis.com/. Accessed December 2, 2016.

8. Taiga.io. *Taiga - Love your projects*. 2016. Available at: https://taiga.io/. Accessed December 2, 2016.

9. Glip - Team Productivity. *Glipcom*. 2016. Available at: https://glip.com/. Accessed December 2, 2016.

10. Facebook - Group Chat. *Facebook*. 2016. Available at: https://en-gb.facebook.com/. Accessed December 1, 2016.

11. Build software better, together. *GitHub*. 2016. Available at: https://github.com/. Accessed December 2, 2016.

12. Download Android Studio and SDK Tools | Android Studio. *Developerandroidcom*. 2016. Available at: https://developer.android.com/studio/index.html. Accessed December 2, 2016.

13. Eclipse - The Eclipse Foundation. 2016. Available at: https://eclipse.org/. Accessed December 2, 2016.

14. MySQL. *Mysqlcom*. 2016. Available at: https://www.mysql.com/. Accessed December 2, 2016.

15. SQLite Home Page. *Sqliteorg*. 2016. Available at: https://sqlite.org/. Accessed December 2, 2016.

16. Project A. Apache Tomcat® - Welcome!. *Tomcatapacheorg*. 2016. Available at: http://tomcat.apache.org/. Accessed December 1, 2016.

17. Discovery Education Coding - Espresso Testing. *Discovery Education UK*. 2016. Available at: http://www.espressocoding.co.uk/. Accessed December 1, 2016.

18. Mockito framework site. *Sitemockitoorg*. 2016. Available at: http://site.mockito.org/. Accessed December 3, 2016.

19. JUnit. *Junitorg*. 2016. Available at: http://junit.org/. Accessed December 3, 2016.

20. Any Developer, Any App, Any Platform | Microsoft Visual Studio. *Visual Studio*. 2016. Available at: https://www.visualstudio.com/. Accessed December 2, 2016.