Post Mortem Report Gab

This course has taught us how to work in a team following a developing strategy. Scrum was the agile development strategy used for developing the application gab. The practices used were as follows: backlog, behavior-driven development, information radiators, pair programming, planning poker, scrum events, user stories, and velocity tracking.

The development process could be split into three steps, these are: Planning, Development and Documentation. The time spent on the different steps of the project is as follows;

Planning: The first two weeks of the course was spent on planning the application, first of all we thought up a concept of what we wanted the application to become, initially we wanted a real time bus map that would track all the busses. So we made user stories and sketches for this idea. However we would later change this idea since we thought that our new application gab was a better idea. We chose to change it after spending some time at the concept workshop, where we realised that a social application would be a more unique solution. Hence we had to make new user stories and sketches, these were created at the concept workshop where we also wrote our vision for the application. The first two weeks were also spent on knowledge acquisition, learning the android framework and the steps required to successfully build an application. In total we spent 10 hours on the workshop per person and around 10 hours per person to learn the android

framework. We spent around 20 hours generating the concept for the application. Since this was a joint operation, all the members of the group spent about the same number of hours on these steps.

Development: When the electricity framework became available we started writing the application, first we setup the backend, a mysql server running on a raspberry pi. After this we wrote the basic framework for the application (the graphical structure). After this we started building the different parts of the application. We would later have to redo the basic framework since we upgraded our application to use the android design library, this made for an easier implementation of newer functions. In total we spent around 15 hours on the backend, 10 hours on the graphical framework and around 70 hours on the different parts on the application. These hours were distributed over the course of four weeks by 5 programmers. The backend was mainly implemented by Oskar Jedvert, while the rest of the team worked on other activities, similarly the framework was mainly written by Tobias Alldén.

Documentation: Roughly 1-2 hours was spent by the entire development team creating the concept prototype description and the continued development documents; which were written after a meeting. About 20 hours has been spent on writing the rest of the documentation later on in the project. The time spent writing these has been largely dominated by who has been available, making Allden, Jedvert and Ström the top contributors spending approximately 5-6 hours each and Hall and Boking clocking in somewhere around 3 hours each.

As mentioned earlier we used backlog, behavior-driven development, information radiators, pair programming, planning poker, scrum events, user stories, velocity tracking, and for each of these practices we will now answer the following questions:

What was the advantage of this technique based on your experience in this assignment?

1. What was the disadvantage of this technique based on your experience in this assignment?
2. How efficient was the technique given the time it took to use?
3. In which situations would you use this technique in a future project?
4. In which situations would you not use this technique in a future project?
5. If you had the practice/technique in a part of the project and not the entire project, how was using it compared to not using it?

Backlog:

1. Easy to get an overview of what can be implemented and what is left to complete on the project.
2. Breaking down tasks is an art unto itself, meaning it takes experience to be able to do it efficiently and correctly.
3. It definitely was very efficient. Since it takes so little time to implement and it is pretty essential for any work involving more than one person.
4. All projects which can be broken down into smaller parts or that the project is so trivial that making the backlog would take a large part of the development time.
5. If the project is very simple.

Behaviour-driven development:

1. Refocuses the mind to only focus on what brings value to the project/customer.
2. Needs a steady link of communication with the customer.

3. Communicating with the customer was somewhat hard and the development took quite a hit because of it. We would do it again, perhaps better next time.
4. All projects which are not trivial.
5. All projects which are trivial.

Information radiators:

1. Easy to see what can be worked on and what is being worked on.
2. Can be stressful if the project falls behind schedule and a lot of cards clutter the radiator.
3. Every time a backlog is used.
4. When a backlog is not used.

Pair programming:

1. Easier to catch errors, easier to make sound logical choices, and produces overall better code.
2. Time spent pair programming could possibly be spent individually programming, meaning it is not optimal if the task at hand is simple.
3. As long as both the developers are focused it can be a great improvement in the code quality.
4. At the beginning of a project to see/setup so all developers can develop without problems, when there is too few tasks to complete, for complex tasks or task which knowledge from multiple developers are needed (such as when integrating).
5. Situations where it is more effective having two programmers work on different things than on one because of a low error to code ratio.

Planning poker:

1. Lets us have an overview of the project's timeline, i.e what can be done before the final deadline and what has to be left for future development.
2. The estimates are only as accurate as the predictions are, meaning domain knowledge is needed to have even a chance of giving a proper estimation. It is also hard trying to estimate knowledge acquisition (how many hours do i need to know about this?) or time spent fixing technical errors. Planning this way is broken but there is yet no better alternative.
3. Definitely useful since the entire team gets to help with the estimation.
4. All projects which are not trivial.
5. All projects which are trivial enough that the entire planning can be contained in memory.

Scrum events:

1. Gives a plan of attack for how to tackle the project and helps the team progress. If the development process is going slowly for some reason it’s easy to pick this up as to why and possibly fix the problem.
2. Takes quite a bit of work.
3. Definitely helps as long as all team members speak up during the meetings and there is follow up on the things brought up.
4. In projects with more than two members in the development team.
5. Scrum events should not be used in trivial projects with a small team but benefits all other projects.

User stories:

1. Makes it easier for customer, product owner, and development team to know how the software will be used.
2. Takes time.
3. Since the use stories were done fast, they were definitely worth the time spent. Helps with focusing the development on what brings real-world value to the application.
4. Projects where there is very few user stories.
5. All projects with more than one person working on it and all projects which is not trivial.

Velocity tracking:

1. Gives a sense of how much of the project is finished.
2. If the estimates are off the velocity tracking will be off.
3. A 10 out of 10. Takes very little time to implement and gives overview.
4. Projects without user stories.
5. Projects with user stories.

As with every project there were things that worked well and things that did not work as well. The workshop we went to gave results and definitely helped us define our projects concept. The development velocity increased during the project as people more and more got the knack for how to contribute to the team. We ran into some problems that made us think we would be unable to meet the deadline but we still managed to complete the project. As for the things that did not go as well**:** The project lacked involvement in the daily scrum which were to be managed through a slack chat where the team could talk about what they have been doing during the day, what they were going to work on tomorrow, and any eventual problems keeping the developer stuck. The chat was rarely used which contributed largely with project members being stuck on trying to solve the same problem for a long time. Next time it would be better to *require* the daily scrum or make the meeting in person instead.

Gab consists of quite a few APIs and tools in relation to the size of the project so a lot of time has been spent on knowledge acquisition which in turn made it much harder to estimate what time a feature or story would take.

In a project collaboration is key, there were a few things we could have done differently. One such example is the communication, which was sparse in the beginning of the project and that the project was not setup correctly which inhibited some of the developers from progressing. All this was done much better after some time spent working on the project as the communication flowed much more freely.

In hindsight we understood that we should have had a workshop “setup” period where all the developers would be working at the same time at the same place (to

make sure everything is setup properly) and that the initial coding should have been done with pair programming, for a solid code foundation. A “build server” would also be nice to have as to not have the classic problem “It compiles on *my* machine”. In a project with sufficient time or a project that puts of emphasis on correct code, it would also be nice and educational to have code reviews on each big commit.

There were a few workshops along the way, such as the “Konceptstuga” and other workshops concerning things as the framework and the busses, however, we only went to the “Konceptstuga” workshop. Before the “Konceptstuga” we talked about what we wanted to gain from going, as well as what we needed to prepare. The team representatives present at the event were the ones who had the time to be there. Feedback from stakeholders were discussed internally within the team and the reflections were shared with student supervisors for further feedback until we felt satisfied.

If there is one thing we would do differently next time we have the opportunity to go to a workshop, it would be making sure at least *someone* is available. We would also have discussed early as to which workshops would be relevant for our project.

What we are going to bring with us from this project is that following up on problems brought up during meetings is important and so is making sure that everyone is comfortable contributing to the project.