Reflection documentation

Team Reflection

The teamwork went smoothly. Everyone contributed and there were no large quarrels in the team. The reason for this is probably because we have worked together in a single group room most of the time.

The first weeks of the project we actively used pair programming, so that every decision was taken by at least two team members. Furthermore, we had code reviews at the end of each sprint. That way, everyone had a good understanding of the code base.

Documentation Reflection

To organize the scrum documentation we used Kunagi. This organized our sprint planning and backlog in a simple way. Each coder wrote documentation for their respective part of the code.

After that, the documentation was put together in Microsoft Word (by Joakim Johansson) in order to have a unified design.

Software Engineering Reflection

We are by no means opposed to using agile processes such as Scrum. However, having one week sprints in a project with only four team members was very inefficient. We lost the every Monday reviewing the last sprint, handling documents and planning the next one. In spite of this we at least are happy that we used Kunagi for handling our sprints, since it helped organize our scrum documentation.

Since we after a while realized that the process was not working, we gradually started to have longer sprints. Using waterfall wouldn't really have improved things either though.

We were essentially done with a rudimentary implementation after about three weeks; we could send sensor input, move mouse, etc. Subsequently, we spent our time refactoring and improving the code.

When we finally realized how to use branches (after about three weeks); the project got a lot easier to maintain (since now master always featured working code).

Coverage Reflection

Since the application was divided into two parts (C# server and android java client) and the only way we could find to easily test the interaction between them was to do it manually (we also made several major refactorizations of everything at all stages of the project), we did not do any unit testing until the project started to near its end.

When we actually started with unit testing, we tested everything that was easily testable (meaning everything not having a direct dependency on the android framework). Since android unit testing proved to be significantly harder than we originally anticipated (we had problems with the emulator and testing services) we gave up on this and focused on unit testing our core library and cleaning up our application.

A positive consequence of using non-android unit test was that we could easily analyze coverage using ECLEmma (only on our core library though).

Hindsight

We should have spent more time originally to learn how to unit test on android. A clearer introduction to how to do unit testing on android during a lecture would have been greatly appreciated.

We regret using TCP sockets as the basis of the network at the start of the project, since this could not be done for a real time system on a low performance network (such as the Eduroam/Nomad combination). Because of this, we lost a lot of time implementing and testing the implementation since we threw it away anyway in the end.

We have mixed feelings concerning our result since our controller application doesn't make use of our sensor library. This is mostly because we couldn't come up with something relevant to do with the sensor values. We tried using the accelerometer to control the mouse, however this was a pointless idea making it much harder to control than simply using touch input. The API can still send all the sensors, making it easy to implement in a different app (e.g. making an accelerometer game controller).

We feel that the project would have progressed better using longer sprints since the administrative work became too overwhelming with one week sprints. And since we spent most of our time in the same small room we didn't really need a scrum meeting to discuss issues.