# Reflection document

## 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 most of the time has worked together in a single group room.

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. In other words, everyone had a good understanding of the code base.

## Documentation Reflection

## 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 entire 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 organized 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 majorly 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 this was to do it manually (we also made several major refactoring’s 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 test, 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.

Why y no only use UDP?

Two week sprinst.

# Reflection document

## 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 most of the time has worked together in a single group room.

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. In other words, everyone had a good understanding of the code base.

## Documentation Reflection

## 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 entire 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 organized 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 majorly 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 this was to do it manually (we also made several major refactoring’s 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 test, 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.

## Hindsight

Why y no only use UDP?

Two week sprinst.