Over the course of this project we noticed that we slowly began to refine previous code, by working together as a team we would improve previously bad code to a higher standard. This went so well that in the later weeks we didn't even have any Design Flaws. We also experienced the benefit of having proper code reviews, it often lets someone else catch your stupid choices or style errors.

We found out that our team of programmers works really well together. We always divided the work equally and we all made sure that our part got done. It is very satisfying to see how you can always see all of our names appear in the version control we used. Also, when one of our team members got sick for one week, we were still able to get all of the work done.

There is a magnificent difference between our Game 2 weeks in and now. At the 2-week point we had a game that only contained balls and a player, we have now added on menu's, multiplayer and power-ups. We even managed to make a level-editor which used drag- and drop mechanics to place objects. We do however think that a problem of our game was that adding new features became like adding whistles and bells to a bike, it was a worthwhile exercise, but did not necessarily make for a better product. We do however think it was a better choice to add for example a level-editor then a block which balls can bounce off. The block would probably improve the game more, but would be too easy to be a decent challenge.

Implementing more than 6 design-patterns over various weeks became a very useful exercise to make our code more complex. We found this extremely useful as evidence that you should not overuse design-patterns in your code. Especially being forced to place them in existing code made it clear that your should not force yourself to use design-patterns. But we also understand design-patterns can be very useful, our most important system, updating the drawing of objects, is done with the observer pattern after all.

We didn't quite like the exercise where we had to implement enhancements thought up by another group. We had a rather bad review, where some things were just complete and utter nonsense, and other things clearly showed they did not bother to read our code or had a very poor grasp of the english language. We were really happy that our TA in the end agreed with us that the suggested changes wouldn't benefit our code.

We also had a couple of arguments with our TA about the grading, because we thought his reviews sometimes were subjective and the improvements he mentioned were based on his opinion about what was good or not. We have had an argument over the use of PowerMockito for example, because we used it to mock the results of some static and private helper methods. However, our TA's review told us not to use it, because "PowerMockito is bad practice". When we asked why, he could not answer us, but refused to add the deducted points. Therefore we think that there should be clearer guidelines for the TA's and the groups about these kind of issues to prevent such arguments.

Furthermore, it would be nice for the groups to see the different points on which their code was graded. For example: when points are deducted because of checkstyle errors, it should be visible for the groups how much points per error are deducted, instead of only seeing the number of points given for the formatting category, as the points given for this category do not only depend on the number of checkstyle errors. In the case that points are deducted because of multiple factors, let's say checkstyle and pmd errors to continue our previous example, it is now clear for the group how much points are deducted for checkstyle and how much points are deducted for the PMD errors. When a group can not fix all the errors in time, they can select the type of error for which most points were deducted last time and fix these errors.

Eventually it became clear to us that "doing something for the sake of doing it" was a practice well used in this project. Yes, we had to implement design patterns in places where it wouldn't really benefit us, but eventually they did help us understand them better. Which is of course what the Software Engineering Methods course is for.

We used the coverage tool Codecov.io on recommendation from our TA. This due to cobertura being down at the time we wanted to implement it. Codecov serves a similar purpose. It is, however, **not** free on private repo's. Neither is Travis-CI. One of our team members had to sacrifice his only free private repo (on travis and codecov) a student gets in the student pack from github. We think there should be better solutions for this. Devhub was a very good alternative, which we used for the *Software Quality and Testing* course.