For this assessment, we decided to stick with ​**two types of testing**​: White-box testing and Requirements testing. The purpose of the former is to thoroughly assess and examine the code’s syntax, semantics, inner workings and structure. The latter was introduced in order to provide a higher-level layer of testing that is lacking with the white-box testing method, and whose aim is to test whether the implementation of the features elicited in the requirements was fully achieved or not. The ​**reason for choosing strictly White-box** **testing** over a combination of White- and Black-box testing was based on the decision that, unlike in the pastassessment in which a sub-team was assigned strictly to testing, every team member would be involved in the test development process, thus eliminating the possibility of black-box testing in toto.

We decided to research and experiment with different testing methodologies that would suit better the purposes and scope of our project. Among all possible methods we considered, ​**Test Driven Development** seemed like an ideal method to adopt for our project. As explained in the paper by Janzen and Saiedian [1], Test Driven Development is an agile testing strategy that “requires writing automated tests prior to developing functional code in small, rapid iterations” [1]. Thus, TDD is a perfect fit for our project because not only does it provide a quick, precise and iterative approach to writing tests which is perfectly in line with the Scrum agile development method used by us, but it also encourages quick refactoring of code that allows the team to improve the code without causing major breaks or bugs whilst providing the necessary flexibility to adapt to changes in requirements.

Another new testing approach we implemented in this assessment was the use of -or, better, the formalisation of- ​**Peer testing**​.This ​involves reviewing the code of other team members to identify areas for improvement. Despite everyone in our team already having employed this method in multiple instances throughout the course of the last assessment, we decided to dive deeper into it and research appropriate literature to support our technique. This allowed us to formalise a practice that was already informally in use by all team members, providing us with a more systematic and technical method for Peer reviewing. We consequently found evidence that Peer testing improves the quality of a software product and also encourages steady progress “since the students need to show their work to peers before it is actually due for submission, they are required to begin working on it much earlier” [2].

We decided to stick with our tool of choice for automated unit testing adopted in assessment 2: JUnit. The advantage of automated testing in the context of the current assessment is that tests are reusable and are therefore more reliable and consistent than manual approaches to testing. [1] On top of that, since the whole team is involved in producing tests as well as working on the implementation of the features for assessment 3, employing automated unit testing allows the team to reuse tests and improve time efficiency.

1. D. Janzen, H. Saiedian, “Test-driven development concepts, taxonomy, and future direction”, Computer (Volume: 38 , Issue: 9 , Sept. 2005), 2005, [Online]. Available: https://ieeexplore.ieee.org/abstract/document/1510569. [Accessed: 3 December 2018]

1. N. Clark, “Peer testing in Software Engineering Projects”, ACE '04 Proceedings of the Sixth Australasian Conference on Computing Education - Volume 30 - Pages 41-48, 2004, [Online]. Available: https://dl.acm.org/citation.cfm?id=979974. [Accessed 3 December 2018].