Reflections and Lessons Learnt

Overall in this assignment I have learnt a lot about the basic OOP software design and development. A major thing that I have learnt is refactoring code. Previously I didn't have a solid framework to actually learn how to edit and make presentable, maintainable code. Now I at least have an understanding of how to do that.

Also, I've learnt more about the SOLID principles in design. On reflection some of my classes and methods (especially the GUI controller) break the dependency inversion principle. It required a number of private instance variables of other utility classes like Standard Deviation, Sort Algorithm and MenuOptions. In hindsight, I could've used some of the classes into interfaces.

In retrospect the GUI design pattern could've benefited from further separation of the model from the view. If I were to revise this program, I've would've implemented a MVVM style. This is where the model and the view only communicate changes, data binding and commands through a view model. This would clearly separate the user interface from the business logic. Also, the GUI could've used a singleton pattern to ensure that only one instance of the GUI controller was used, but can potentially control multiple views using the

Also, I had learnt how to use advanced data structures. In previous studies I had no understanding of how to select certain objects or data from collections. However I have since learnt how to use HashMaps which were very useful for getting a certain object from a collection. Also, there are many qualities of other data structures that were particularly useful, like TreeMaps for populating the projects list views in order, HashSets to ensure only unique values are inputted. In future these data structures will be useful in other future Java software design projects.

Some of the issues that I had initially encountered where incorporating threads into the Java FX GUI. When I had initially built the GUI I hadn't accounted for the incorporation of the thread. It was only later that I thought it would be great to incorporate a thread for the back-end analysis while the auto team allocation algorithm would work in the background. However, I kept running into Java Fx Thread Exception. In a future development, I could've incorporated multithreading to prevent that issue.

In hindsight, it would've been more practical to read and write information from databases and serialised files. Databases are more organised, and serialised files can store information about objects independent of programming language.

Also,in hindsight, I could've utilised a better and more computationally efficient algorithm to automatically sort the teams. I had initially tried to compare the students' grade with the skills requirements of the projects. However, this had resulted in all the best students with high marks being allocated into one group. I could've used another approach by making an algorithm where the it picks the worst student of all the teams and swaps it with the best student out of all teams and see if it helps with the standard deviation. At least with this approach it wouldn't be attempting to sort almost every possible student leading to combinatorial explosion.