Scalability Report

Originally when this program was initially produced it had only considered the constraint of 20 people in 5 teams. Some aspects of this program are scalable. This program uses a number of data structures, including HashMaps, TreeMaps,HashSets and ArrayLists which can dynamically expand regardless of the number of objects are inputted into the collection.

Also, the GUI uses the MVC model of GUI design. In this model is one of the models that can control the flow of information from model and view. In this program the model handles all of the business logic. Any changes from the view will inform the controller which updates the model. And the model notifies the view. This way there is greater separation of the UI from the model. This is more maintainable since the UI often changes faster than the business logic. This way any changes in the GUI will not interfere with the business logic in the model, making it appropriate for maintainability and scalability.

However, some aspects of this program cannot scale well due to a number of coding limitations. Firstly, for shortlisting projects. It does the job of shortlisting the top 5 projects. However, in generateTopProjects(Map<String,Integer> projPrefMap) method in the MenuOptions class, the top number of projects was hard coded as 5. In reality, if there was the more projects and more students, the top number of projects would change according to the number of students. In retrospect the top number of projects would dynamically be processed by int totalNumberStudents/4.

The GUI at the moment can only sustain at maximum 5 teams. The graphs are dynamically populated by the data imported from the model, by iterating through the projects collection. However, the part that isn't dynamic is populating the list views which show the members in each project. This is because in the GUI design it already has a set number of 5 list views for each project. Potentially, I could've actually written the code to use multiple views for each projects and another view for the statistics to allow for greater flexibility.

Also, the sort algorithm performs alright for a small number of students. However, the current algorithm isn't scalable. This algorithm relies on iterating through almost every single pair of students within a hardcoded limit of swaps (30). In this program the threshold SD will be calculated based on input of projects with their allocation of students allocated by the user. It goes through a nested for loop to select students for swaps and compares if the total SD (sum of preference SD, average skill SD and skills gap SD) after the swap is lower than the threshold SD. For just 20 students it works alright, but this method will have a combinatorial explosion if hundreds of students needed to be allocated to other projects. This is due to having a linear search of (O)n complexity and a run time of n^2 In retrospect, instead of a linear search, I would attempt to do something like a binary search or a merge sort which have a log(n) complexity and much more efficient.