Throughout this testing and debugging project I used a variety of methods to find faults and quality issues with the code. The faults I found are listed in the section bellow.

**Faults I found within the Module.java class include:**

* I changed the value of **public** **final** **static** **int** ***MSC\_MODULE\_PASS\_MARK*** = 40; //50 from
* Changed sem2Credits = sem2; from sem1 to sem2
* Added Sem1Credits to **return** sem1Credits + sem2Credits + sem3Credits;
* Changed the sumLevel7Credits in the checkModuleSelection() method by replacing getSemesterOneCredits(); with getTotalCredits();

**Quality error I found within the ModuleRecord.java classis as follows:**

* There is a lack of comments compared to the other classes which doesn’t outline the purpose of the code

**Faults I found within the Studetnt.java class include:**

* I changed all of the **public** **static** **int** to **public** **final** **static** **int** ***MAX\_NUMBER\_OF\_MODULES*** = 13; adding final to them as they will not be changed for good practice
* I commented out int sumOtherCredits = 0; as it is a redundant variable, and it doesn’t fit with the quality of variable names in the rest of the code
* The first for loop in the checkModuleSelection() method creates and off by one error as it starts from position 1 int the array insteat of location 0 in the line **for** (**int** i = 1; i < ***MAX\_NUMBER\_OF\_MODULES***; i++) {so I corrected this fault.
* I placed the projectModule = moduleRecords[i].getModule().getTotalCredits()

>= ***MINIMUM\_PROJECT\_CREDITS*** as the clause of an if statement and subsequent lines where the result of the Boolean expression

* I removed (sumOtherCredits > 0) as this will never be true as the values is always set to zero
* I changed some of the comments in the getQualification() method as the reduced the quality of the code such as // clever clogs was changed to //MSC with Distinction and other comment within the method labelled the qualification
* In the same method the return statement for **return** Qualification.***MSC\_WITH\_DISTINCTION***; and **return** Qualification.***MSC\_WITH\_MERIT***; were in the wrong position I swapped these statements so the correct qualification would be returned at the appropriate time
* public boolean checkAllModulesPassed() { method is labelled above as private but the method is public so I corrected this fault
* I added get and set methods for ModuleRecord in order to be able to access a private array
* moduleRecords[i].recordMark(i); changed to moduleRecords[i].recordMark(mark);

**Faults I found within the reference library visualisation\_out.jar include:**

* The Graph displays only 7 bars when there are 8 modules
* The summary table in the console displays the module in a different order to which they are tested in, this could lead to confusion when interpreting the graph

**Methods of Debugging**

**Visual Inspection**

I utilised visual inspection as a method of gaining overall understanding of the code along with the original specification supplied. Whilst reviewing the code I spotted several faults which were clear without having to used test classes. This approach was a static as I examined the code without running it, this occurred for the majority of the errors found by this method. This is also a white box method of debugging as it requires knowledge of the internal workings of the code. This method utilised problem solving techniques to see not just faults in the code but quality issues according to good practice we have learnt about in lectures such as good descriptive comments of the code to help any user understand so that the code is useful and meaningful equally.

**JUnit Testing**

This method was useful in highlighting potential errors within the code as you could take individual methods and test them independently of the full programme. I tried to implement a Junit test to see whether I could add a module to the array. This proved difficult because of the private nature of the array. I added get and set methods to enable me to access locations in the array but this was not successful. With further development and understanding of Junit testing I do think it could be useful for the testing and debugging process as you can take an in depth look into one method to identify a weak link in a long chain of sequential methods and classes.

**Black Box Testing**

This method was used in the visualisation\_out.jar class as I did not have access to the code, instead had test modules and an example student to evaluate any faults within the library in which the code was contained. I concluded that the errors in my class were the missing bar on the graph and the scrabbled order of modules in the table. I changed the marks for the modules so that there were 4 modules that passed and they were all different values, only three green bars were displayed on the graph which highlighted that one of the modules were missing. By setting three values of the pass modules to the same value and identifying which bar was not being displayed. This turned out to be the csc8001 module. By setting module marks to be cascading downwards to see a clear pattern which revealed that the graph suited the order of the modules written in the test however the table in the console showed that the was in the wrong order. This will lead to confusion as it interpreting the graph is difficult as the bars are in different places to the table.