CMP1127M Programming and Data  
Structures: Assessment 2

Table of Contents

[Application Description 2](#_Toc449007066)

[Class Diagrams 3](#_Toc449007067)

[Use-Case Diagram 4](#_Toc449007068)

[Sequence Diagram 4](#_Toc449007069)

[Testing 5](#_Toc449007070)

[Video URL 13](#_Toc449007071)

[References 14](#_Toc449007072)

# Application Description

For this Assessment, I needed to design a desktop application named ‘modnote’ that functions as a ‘Module note taker’. Allowing the user to import text files with module information and add notes, which could contain text, images and links; these notes would then be added to the module, which can have many notes. The modules would also have assessments with due dates that should be shown to the user. The application I have created allows the user to create modules manually, as well as importing them from a text file; and also lets the user add their own assessments, which can either be an assignment (e.g. coursework), or a test. Notes may be created for each assessment too, allowing modules to have separate notes from its assessments notes.

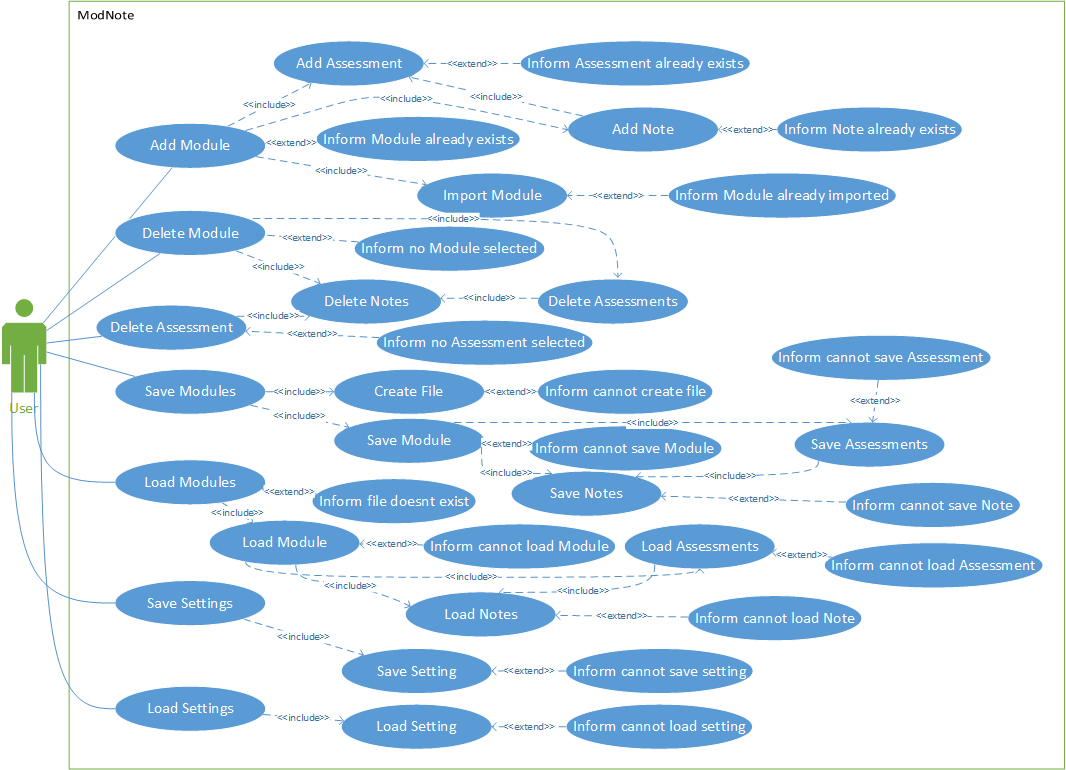
The user can delete modules, assessments and notes from the application; as well as edit them, allowing the user full control over the stored data. The application shows each module in a list with a tree format, showing each module as a top-level node with its assessments shown as sub nodes; the user can then double click on any of these nodes to bring up its details and notes.

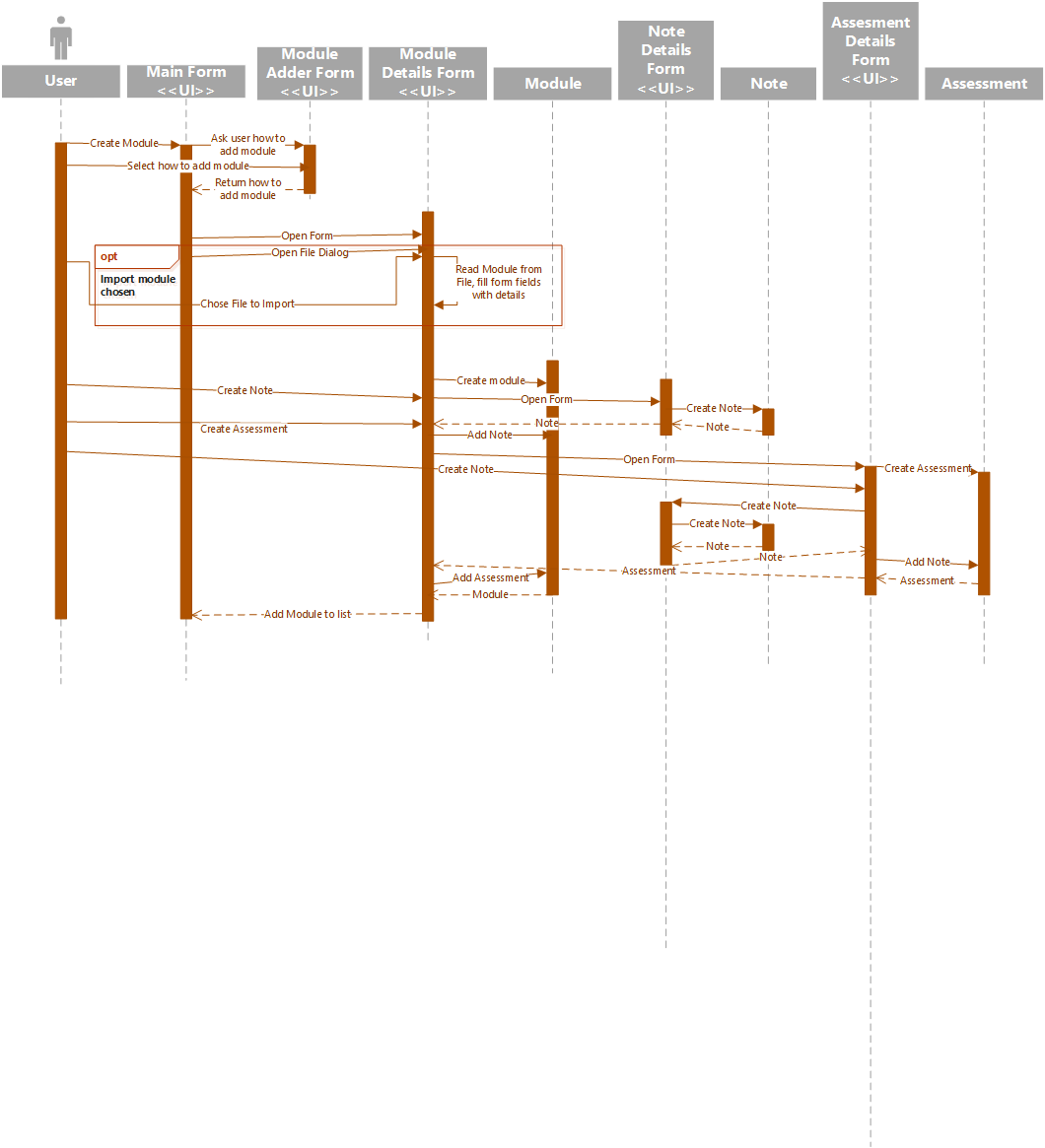
Assessments are also shown to the user in another list with a tree format, showing each assessment as a node, and these are shown in two different ways, the default way is a list of all the assessments with a due date within 2 weeks; the user can also change this to view all assessments, and sort the assessments according to due date (the amount of days left to complete). Assignments have a progress bar which allows the user to show how much they have completed, and tests have Room, seat and Duration textbox allowing the user to store have information about the test readily at hand. Notes allow the user to store text and also create links to website, or file locations and are opened by the appropriate program when clicked; notes also have images which can be added and then viewed in their full size in another window upon being clicked.

The application has settings which allow the user to select whether to auto load modules on starting the application, and choose what modules to load; on exiting the application the settings and modules are serialized into separate files xml files allowing them to be deserialized to the exact same state upon opening the application again. These saved settings and modules can also be loaded into the application at will by the user.

## Class Diagrams

As seen in the class diagram above, I chose to create interfaces for each class, with the IAssignment and ITest interface inheriting from the base IAssessment interface. I chose to do this because using interfaces allows ‘easier maintainability, makes your code base more scalable and makes code reuse much more accessible because implementation is separated from the interface. Interfaces add a plug and play like architecture’ (Mccutchen, 2010) which I thought would be useful in this program allowing adding more classes (e.g. an Exam class) later to be easier and not break any existing code structure.

****Use-Case Diagram

Sequence Diagram

Testing

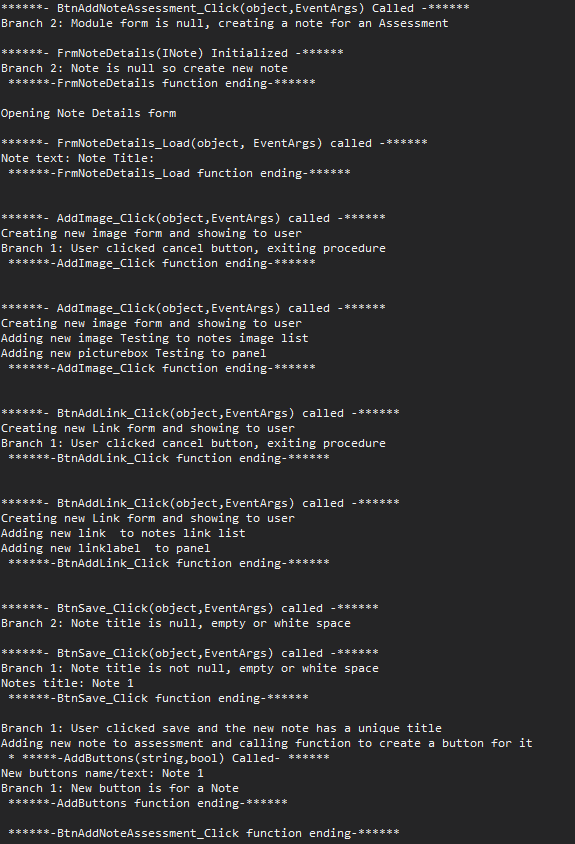
#### Black Box testing

|  |  |  |
| --- | --- | --- |
| Requirement | Pass/Fail | Screenshot |
| Import Module information from file | Pass |  |
| Modules can be deleted | Pass |  |
| Create Note for Module | Pass |  |
| Module can have multiple Notes | Pass |  |
| Assignments should show whether their due date has been reached | Pass |  |
| Modules displayed all together | Pass |  |
| Assessments shown according to due date | Pass |  |
| Assessments can be sorted according to due date | Pass |  |
| Create Note for Assessment | Pass |  |
| Assessments can have multiple Notes | Pass |  |
| Clicking a link in a Note opens the link in browser/File explorer | Pass |  |

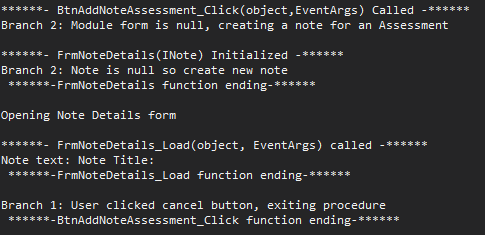
#### White Box testing

To test the method coverage, statement coverage and branch coverage of adding a note to an assessment, adding an assessment to a module, and adding a note to a module; I used Debug.Writeline() to show when a method was called, when statement was encountered and for every branch of each statement. I chose to test creating a note for a module/assessment and adding images/links and a title to the note because there are a lot of statements and branches that would allow for thorough testing. My results are shown below.

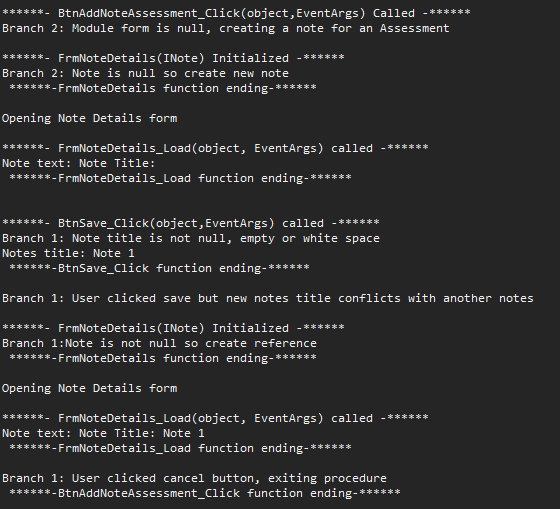
##### Adding a note to an assessment

I started by choosing an assessment and clicking ‘add’ on the notes form, this then opened the note details form where I proceeded to give the note a title, and then test adding an Image and a Link and also cancelling mid way through both to cover all branches. After that I clicked the ‘save’ button and the note was added to the assessment.

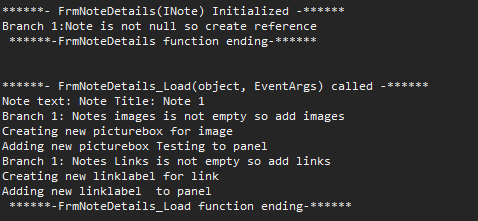
##### Cancelling adding a note to an assessment

 I then decided to cancel adding a note to achieve a large branch coverage.

##### Adding a note to an assessment with a conflicting name

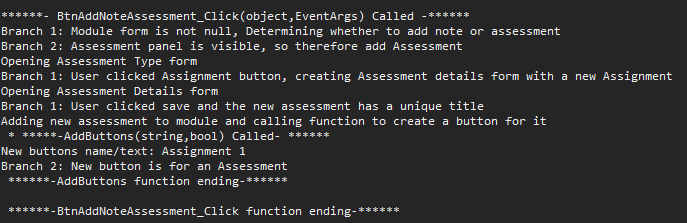
Since each note must have a name unique to that notes list I needed to test whether a statement would stop the user adding a note with the same name as another notes

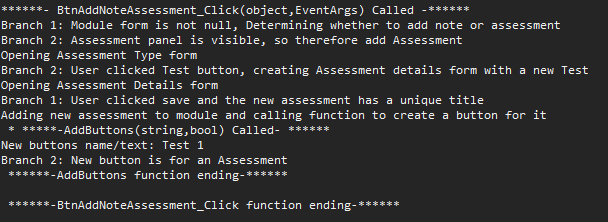
##### Opening an assessments note

I also needed to cover branches of statements to add images and links if a note was passed to the form with images and links

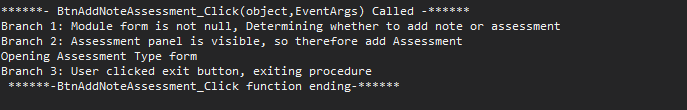
##### Adding an assessment to a module

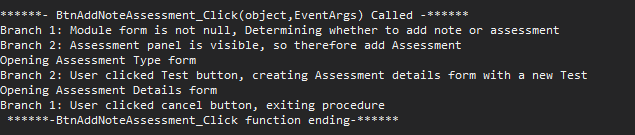
Although I covered all the methods in the previous tests, I needed to test adding an assessment to a module to cover more statements and branches.

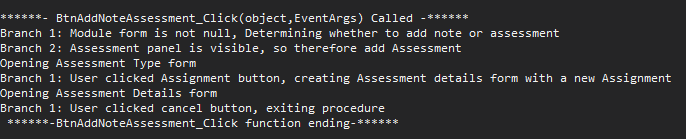
I first chose to add an assessment, and covered adding both a test and an assignment. Then I added a title to each assessment and clicked save.



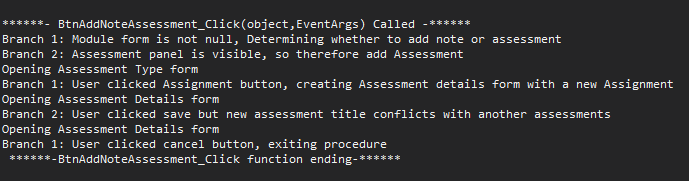
##### Cancelling adding an assessment

I then tested cancelling adding each type of assessment, first by closing the assessment type form, and then by clicking the ‘cancel’ button on each assessment details form.

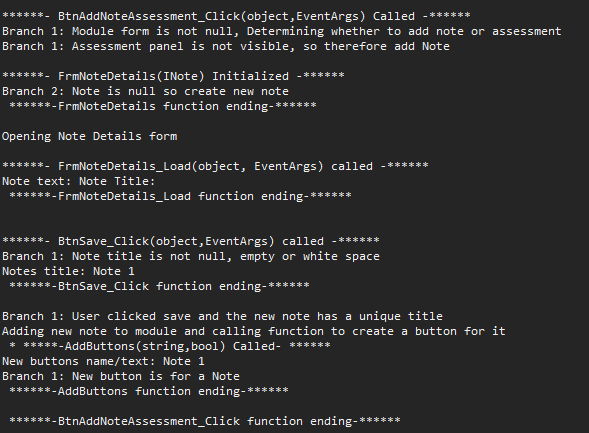




##### Adding an assessment to a module with a conflicting name

Just like notes, assessments must have titles unique to their assessment list, so I needed to test whether my logic in determining whether the title was unique or not worked.

##### Adding a note to a Module

In order to cover every branch of each statement, I needed to add a note to a module.

##### Cancelling adding a note to a module

##### Adding a note to a module with a conflicting name

##### Conclusion

To conclude the testing, the method coverage, statement coverage and branch coverage of these actions are shown below.

|  |  |
| --- | --- |
| Measurments | Results |
| method coverage | 70% |
| statement coverage | 71% |
| branch coverage | 68% |

## Video URL

https://www.youtube.com/watch?v=0VUsGcgliNA

References

Mccutchen, R. (2010, May 17). *C# Interfaces, what are they and why use them?* Retrieved from Dzone: https://dzone.com/articles/c-interfaces-what-are-they-and