**II.** **Assessment Criteria**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Class/grade  Assessment criteria | First | | | Upper second | | | Lower Second | | | Third | | | Fail | | | Zero |
| **\*Exceptional 1st** | | | High 2.1 | Mid 2.1 | Low 2.1 | High 2.2 | Mid 2.2 | Low 2.2 | High 3rd | Mid 3rd | Low 3rd | Marginal fail | \*Mid Fail | \*Low Fail | zero |
| High 1st | Mid 1st | Low 1st |
| **Functionality (70%):** | As previous plus extra features as agreed beforehand with tutor which use highly sophisticated and considered language constructs, and achieves considerable functionality or difficult goals, that will have required the use of considerable design skills. Clear reflective thought on design, and pertinent testing. Can discuss code and rationale highly professionally, with consideration of alternatives, design and implementation issues etc.  \*Exceptional programming skills and coding quality. Program satisfies specification completely and correctly. | | | Working program that achieves major requirements and substantial content. Program shows use of extra features appropriately deployed and performs a task of some complexity, requiring good design. | | | Working program that has basic interface handling, but can only achieve partial success in key requirements. A reasonable amount of code, though size may be gained from repetition rather than novel sections. Includes some features beyond those in live test. Program performs a reasonable task of moderate intrinsic complexity. | | | Working program that has interface handling but cannot perform much functionality, or task relatively simplistic, and requiring very little design. Limited understanding of code or code so simple that needs little understanding. Cannot explain how additional features might be added. | | | A small part of program works but main functionality of the program is not working correctly.  \*Program that does not satisfy any key requirements. Program shows lower level of skills than those used within the live test. Understanding of code minimal - explanation of even simple aspects unsatisfactory | | | Work of no merit.  Absent from test |
| **Object-Oriented Programming (15%)** | Excellent object-oriented design and well-thought out implementation, using sophisticated constructs where appropriate. Shows effort/learning well beyond taught material. Program succeeds completely at achieving its aims, or approaches very challenging aims very well  \*Exceptional object-oriented design which uses of most object-oriented programming concept that binds together the data and functions that manipulate the data and uses polymorphism. | | | Very good Understanding of how to identify, illustrate and describe all the classes in the program. All classes are accurately and completely discovered and implemented. | | | Good understanding of how to identify, illustrate and describe all the classes in the program. | | | Sufficient understanding of object-oriented modelling but the implementation is not complete. Many classes and their characteristics are missed or, not implemented correctly including some major ones. | | | Marginal understanding of object-oriented implementation.  \*Insufficient understanding of object-oriented implementation. | | |  |
| **Modularity (5%)** | Program is decomposed into coherent and reusable units, and unnecessary repetition has been eliminated.  \*Exceptional modularity which uses header files, use of encapsulation. Program is fully decomposed into coherent and reusable units. Ability to decompose a problem into coherent and reusable functions, files, classes, or objects (as appropriate for the programming language and platform). | | | Program is decomposed into coherent units, but may still contain some unnecessary repetition | | | Program is decomposed into units of appropriate size, but they lack coherence or reusability. Program contains unnecessary repetition | | | Program is one big function or is decomposed in ways that make little sense. | | | Marginal understanding of modularity.  \*Insufficient understanding of modularity. | | |  |
| **Comments and Documentation (5%)** | Very good documentation and comments. Well-formatted, understandable code; appropriate use of language capabilities  \*Exceptional ability to format and document code for human consumption. Program contains appropriate documentation for all major functions, variables, or nontrivial algorithms. Formatting, indentation, and other white space aids readability. | | | Very Good documentation and comments. Understandable code. Report shows clear process from requirements, through design decisions to quality testing. Discussion of code is good and competent, with understanding of its limitations. | | | Good documentation and code comments, but the code is hard to follow in one reading; poor use of language capabilities. Discussion of code has some understanding, but may be missing in depth or consideration of limitations of particular choices or structure. | | | Sufficient documentation and a reasonable use of comments and code | | | Incomprehensive code, appropriate language capabilities not used. The code has no comments.  \*No code comments No program documentation | | |  |
| **Testing (5%** | Excellent testing strategy and excellent use of unit testing  \*Exceptional testing strategy and use of testing tools | | | Very good unit testing | | | Good use of unit testing | | | Sufficient use of unit testing. | | | Marginal understanding of unit testing in python.  \*No unit testing. | | |  |

|  |
| --- |
| **Comments**  Collection management program with windows GUI. Adding and deleting an item implemented, and good use of custom dialog boxes. Event handlers made use of to respond to table interactions. Collection details are stored to a csv file specified by the user through a file dialog box. The SaveTableToFile function is written to a high quality, good attention to detail in dealing with cases where the file extension is not present. WXgrid data structure used to store collection data in memory while the program is running.  Each user interface item is it’s own class, (window, toolbar, table, menu, main, help, dialog). All classes have been identified and successfully implemented. Report and demo show good understanding of OOP.  Header files used extensively. Each class has it’s own header file, no repeated code. To make it even better, it might be possible to “decouple” your program from the GUI framework used (reduce use of WxFrame to make interface-independent)  Code fairly self-explanatory, even so many of the comments are quite brief. Class diagram excellent, although some of the report is a bit incoherent in places and your behaviour diagram isn’t legible.  Description of the test should describe how to carry out the test, and the expected result should document what you expect to happen. For this reason many of your tests aren’t very well defined.  Overall: LOW1 |