SEACSTA
IBDP CS Moderation Check-List

This document has been compiled by teachers in the SEACSTA WhatsApp group.
Any recommendations and adjustments should be added as comments and will be changed by Carl Turland pending discussions with other teachers in the group if necessary.
Hopefully this document can act as a guide to ensure greater transparency for how the IAs are moderated.

Crit A

Taken from the IA moderation guidance statements

Evidence	**0**	**1-2**	**3-4**	**5-6**
Appropriate Scenario		Stated	Stated with evidence of consultation	Described with evidence of consultation
Product Rationale		Rationale identified	Partially explained	Justified
Success Criteria		Generally inappropriate	Some appropriate	Range of appropriate
Client quotes		Yes/No		
Ref Appendix		Yes/No		
No. of Interviews		How many?		

Essentials

- Scenario
  - Quotes from a referenced consultation with client
  - 2 interviews in appendix
    - One identifying the problem
    - One agreeing on success criteria
  - How is the problem currently solved?
  - Consequences of not solving this problem?
- Rationale
  - Product is justified
  - Compare language/tech to be used against at least one other
- Success Criteria
  - Numbered/bullet pointed testable success criteria statements
  - Around 9-15 criteria

Better If

Justification is on the basis of some research with supplied references
Research evidence included in appendix
Further interviews present in appendix showing consultation with client throughout all stages of the development
Success criteria should include the client's desired deployment platform. This can be used to justify the language selection in the rationale.

Crit B

Taken from the IA moderation guidance statements

Evidence	**0**	**1-2**	**3-4**	**5-6**
ROT		Limited	Partially complete	Detailed and complete
Test plan		Limited	Partially complete	Detailed and complete
Design Overview 1		Limited	Partially complete	Detailed and complete
Design Overview 2		Difficult to see development	Basic understanding of development	Clear how it was developed

Essentials

Design documentation (where appropriate)

Structure/hierarchy chart
System flowcharts
Algorithms (Flowcharts / Pseudocode)
Data flow diagram (DFD)
HCI (Human-Computer Interaction)/Screen designs
Data structures
OOP: UML diagrams and relationships
Database: design
Database: queries
Database: Entity-Relationship (E-R) diagrams
File structures
Hardware selection
Diagram annotations
No extended writing
No designs should be reverse engineered (or look like they are)
ROT uses template

ROT covering all areas of development (planning, design, development, testing, and implementation)
ROT time estimate - entered in hours
A test plan which can be used to evaluate success criteria statements (show which SC the test is being used for in the table)

Crit C

Taken from the IA moderation guidance statements

Evidence	**0**	**1-4**	**5-8**	**9-12**
Level of complexity		Low level	Moderate level	High level
Level of ingenuity		Low level	Moderate level	High level
Existing tools		Limited use	Some appropriate	Appropriate
Explanation of techniques and how adequate		None	Some	Full
Sources		Not identified	Identified	

		Complexity (at SL)		
Ingenuity		High	Moderate	Low
	High	9-12	7-10	5-8
	Moderate	7-10	5-8	3-6
	Low	5-8	3-6	1-4

Essentials

Content page of skills covered
Cover 7 of the most complex skills used. Explain each in around 150 words
Use screenshots and annotations to help explain
Describe the use of the technique in the product rather than explaining the technique itself
All references to sources used should be listed in Crit C - not in an appendix

Better If

Have used skills beyond those taught on the course

Crit D

Taken from the IA moderation guidance statements

Evidence	**0**	**1-2**	**3-4**
Video shows product		Partially functions	Functions well
Expansion and modifications		Possible, but difficult	Straightforward

Essentials

Video goes through each success criteria showing that the product functions as intended
Video should not show any code
Video must not be longer than 7 minutes
Product files should be organised into relevant folders
Code should use good naming conventions for variables, functions, objects etc
Comments should be present throughout to aid understanding
Finished code should be included as an appendix

Better If

By following a good test plan, the video should go through each test proving every success criteria has been met

Crit E

Taken from the IA moderation guidance statements

Evidence	**0**	**1-2**	**3-4**	**5-6**
Evaluate against success criteria		Limited attempt	Partial	Fully
Feedback from client / advisor		Limited evidence	Included	
Recommendations for improvement		Trivial or unrealistic	Largely realistic	Realistic

Essentials

Client should have been given time to use the product before final interview
Evaluation should be written against the agreed success criteria
Evaluation should include feedback from a client interview (quoted and referenced)

Improvements suggested should include some suggested by the client based off the interview - at least 2
Developer can also suggest improvements (can be more technical - related to coding methods for example)
Final client interview included as an appendix

Better If

Skills List

This is taken from the old IB course and does not directly relate to the current one. However, it can be useful for students to get some ideas on which skills to talk about in Criteria C

Standard Level

The rules are slightly different for SL. The rule of thumb used to be that a project had to **have at least 10** of the **following 15 aspects** to be counted as 'sufficiently complicated'. They are

1. Arrays
2. User-defined objects
3. Objects as data records
4. Simple selection (if–else)
5. Complex selection (nested if, if with multiple conditions or switch)
6. Loops
7. Nested loops
8. User-defined methods
9. User-defined methods with parameters (the parameters have to be useful and used within the method body)
10. User-defined methods with appropriate return values (primitives or objects)
11. Sorting
12. Searching
13. File i/o
14. Use of additional libraries (such as utilities and graphical libraries not included in appendix 2 Java Examination Tool Subsets)
15. Use of sentinels or flags

HL/SL IA: How to get marks for 'complexity'

In 2014 the Computer Science was dramatically updated, but we are still coding Java and bits that were complex before, appear to still count as 'complex'. The rule of thumb used to be that a **HL project** had to have at **least 10** of the **following 18 aspects** to be counted as 'sufficiently complicated'. They are:

1. **Adding data to an instance of the RandomAccessFile** class by direct manipulation of the file pointer using the seek method

2. **Deleting data from an instance of the RandomAccessFile** class by direct manipulation of the file pointer using the seek method. (Data primitives or objects may be shuffled or marked as deleted by use of a flag field. Therefore files may be ordered or unordered)

3. **Searching** for specified data in a file

4. **Recursion**

5. **Merging two or more sorted data structures**

6. **Polymorphism**

7. **Inheritance**

8. **Encapsulation**

9. **Parsing a text file** or other data stream

10. **Implementing a hierarchical composite data structure**. A composite data structure in this definition is a class implementing a record style data structure. A hierarchical composite data structure is one that contains more than one element and at least one of the elements is a composite data structure. Examples are, an array or linked list of records, a record that has one field that is another record, or an array

11.Use of **additional libraries** (such as utilities and graphical libraries not included in appendix 2 Java Examination Tool Subsets)

12. **Inserting data into an ordered sequential file** without reading the entire file into RAM

13. **Deleting data from a sequential file** without reading the entire file into RAM

14. **Arrays of two or more dimensions**.

15.to 18. Up to four aspects can be awarded for the **implementation of abstract data types** (ADTs)

ADT Name	1 aspect	2 aspects	3 aspects	4 aspects
General criteria	An incomplete ADT is implemented.	An ADT is implemented with all key methods implemented.	An ADT is implemented that includes some error checking.	An ADT is implemented completely and robustly.
Lists, implemented using references (such as, a dynamically linked list).	A node style class with appropriate constructors and methods to set and get data elements.	Methods are implemented to add at / remove from the tail and the head of the list.	Proper checks are made for errors such as attempting to get an element from an empty list or inserting the same element twice.	All error conditions are checked for, and all appropriate methods are implemented. For a doubly linked list these could be: `size, isEmpty, first, last, before, after, insertHead, insertTail, insertBefore insertAfter`
Tree (simple, ordered binary tree is sufficient using arrays or dynamically linked object instances)	A class or interface with appropriate constructors and methods to set and get data elements.	Methods are implemented to add at / remove from the correct point in the tree.	Proper checks are made for errors such as attempting to get an element from an empty tree or not inserting the same element twice.	All error conditions are checked for, all appropriate methods are implemented. For a simple ordered, binary tree these could be: `size, isEmpty, root, parent leftChild, rightChild`
Stack implemented dynamically or statically.	A class or interface with appropriate constructors and methods to push and pop items.	Methods to test for full and empty stack are added.	Proper checks are made for errors such as attempting to get an element from an empty stack.	Probable methods `push, pop, top, isEmpty isFull, size`
Queue implemented dynamically or statically	A class or interface with appropriate constructors and methods to enqueue and dequeue items.	Methods to test for full and empty queue are added.	Proper checks are made for errors such as attempting to get an element from an empty queue.	Probable methods `enqueue, dequeue front, rear, isEmpty, isFull, size`
Hash table implemented in an array.	A class or interface with appropriate constructors and methods to insert and remove items.	Methods to test for full table and duplicate keys are added.	Proper checks are made for errors such as attempting to get a non-existent key, clashes are dealt with properly	Probable methods `hashFunction, insertKey removeKey, isDuplicate isEmpty, isFull, size`

The "**Non-trivial" principle** means that the programmer must demonstrate that the program benefits from the use of the aspect.
Where one aspect includes others, all are credited (always provided that the use is non-trivial, well-documented and appropriate).