**FIT2107-S2-2017-SampleExam-Attempt** [53/75] = 71%

**Question 1 – Quality** [7/12]

Functionality [2]

- The firmware should be able to successfully interpret, compile and run the programs written for the specific factory installation in the PLC programming language, barring any errors on the programmer's part

- The firmware should be able to successfully upload new or modified PLC programs

Reliability (?) [1]

- The firmware is expected to adhere to the correct timing of the program (e.g. stopping the motor 50ms exactly after a temperature sensor reads a certain value)

- The program is expected to correctly store and maintain the uploaded PLC programs, potentially with multiple backups across different storage devices (e.g. multiple SD cards). It is important to store the programs reliably as they control various industrial processes that may endanger human lives if not executed correctly.

Usability [1]

- Assuming that the firmware has a command line interface or graphical interface to interact with...

- The firmware should have clear instructions for reading in new/modified PLC programs

- The firmware should have clear instructions for running the programs

Efficiency [0]

- Because the PLC firmware is written specifically for the PLC hardware, with no other tasks to allocate system resources to, the firmware would be expected to run on the hardware with full dedicated resources, this attribute is of limited relevance to the application

Maintainability [1]

- The hardware is stated to last for decades, hence there is limited relevance in ensuring ease of maintenance for the firmware's source code.

Portability [2]

- Because the PLC firmware is written specifically for the PLC hardware, there is little need for portability as the firmware will not be installed across multiple software/hardware configurations.

**Question 2 – Inspections** [17/22]

[2] a) The programmer should attend the inspection. One reason is that they would be able to answer any queries about the source code asked by the reviewers, such as why a function was implemented in a certain way, or the meaning of any variables. They would also be able to listen to any alternate perspectives from the reviewers about potential changes to the code, providing internal knowledge transfer throughout the team.

b) This Fagan inspection checklist needs modifications, especially in the context that it is written for Python source code. A modified version would be...

[2] i) Omitted (Python does not need to initialise variables before assigning)

[2] ii) Class and attribute names must be consistent with reviewed UML class diagram

[2] iii) All method names must reflect the purpose of the method

[1] iv) All methods must use a sufficiently performant algorithm fitting the purpose of the function depending on the program's requirement for optimising time or space (best is very subjective and requires more clarification, algorithms have tradeoffs between space and time and this needs to be clarified correctly)

[2] v) All methods that are not part of a class or module’s public API should have names that begin with an underscore (‘\_’) character

[2] vi) Attributes should be made part of a class or module’s public API at all times(if keeping up with proper OOP practices, attributes should always be part of the class/module's API for encapsulation purposes)

[4] c)

Strengths

Assuming more reliance on inspections and system testing would imply more time spent on such processes, there would be an overall greater level of team communication and understanding through more inspections. This will increase team cohesion on the software as there will be more frequent points where the code author's understanding is relayed to the rest of the team.

Weaknesses

- Having no unit tests will mean that individual classes/components will now lack quality assurance in that individual unit. If a system error occurs, every unit will have to be investigated rather than pinpointing it to an individual class easily with unit tests.

- Though more defects are found by inspections and system testing overall, there will be different defects found by unit tests that neither inspections or system testing will find. Due to the human aspect of inspections, the reviewers may not be able to find the specific edge case in which the code does not work.

Overall, it is not likely to be an improvement to the current practice due to the heavy disadvantages. More testing throughout different levels of the software (unit, integration, system) is required to "filter" out bugs and removing unit tests overall lead to more defects passing through.

**Question 3 – Mocking** [6/6]

[3] - Using mocking in this application allows testing of the application without actually affecting the Twitter account on which the app is based on. For example, the program can test for the sending of tweets by ensuring that the program passes the correct arguments to the mock object without calling the actual send function. Doing so without mocking will mean that the twitter account will constantly be sending out "testing" tweets, which the user would want to avoid.

[3] - Using mocking in this application also ensures that the test suite will run at a faster speed. For example, calling a mock object to retrieve tweets will take milliseconds to return as the return value is already set to the mock object. This is in contrast to using the actual API which may take seconds to connect to, retrieve and return back. Having a fast test suite ensures easier adherence to continuous integration efforts and quicker feedback loops for any defects.

**Question 4 - Unit tests** [18/18]

a) [2]

1) self.grade = 80

2) self.grade = 70

3) self.grade = 60

4) self.grade = 50

5) self.grade = 49

b) [4]

1)

students = [Student("A", "B", 49), Student("C", "D", 50)]

unit = Unit(students)

unit.fail\_report()

c) (?) [2]

No it would not be a good idea. Multiple unit tests should be created for a single function that each serve one clear purpose. As an alternative, we can have one unit test covering a set of students that are all failing (satisfying the condition of "not student.pass\_subject()) in a function named test\_fail\_report\_all\_fail(). We can then have another set of students all passing (not satisfying the condition) in a function named test\_fail\_report\_all\_pass().

d) (?) [6]

Equivalence testing would be a good option as there is only a single input which can be divided into equivalence classes, where every input inside the class' domain all give the same unique behaviour.

Given the input of self.grade (let's call this G), we can partition this into the following classes

- G >= 80 -> Returns HD

- 70 <= G < 80 -> Returns D

- 60 <= G < 70 -> Returns C

- 50 <= G < 60 -> Returns P

- G < 50 -> Returns N

We can then select a random value in each domain to draw up our test cases.

Case #1: G = 85 (expect HD)

Case #2: G = 72 (expect D)

Case #2: G = 69 (expect C)

Case #2: G = 51 (expect P)

Case #2: G = 4 (expect N)

e) (?) [4]

def test\_letter\_grade\_hd(self):

stu = Student("Nick", "Chong", 85)

self.assertEqual(stu.letter\_grade(),"HD")

**Question 5 – Metrics** [5/14]

[3] a) It would be harder to unit test This is in contrast to a class with a high CBO value. Because the class heavily depends on the behaviours and values of other classes more, there would need to be more mocking done to artificially emulate the various behaviours of the said classes. This is in contrast to a class with a lower CBO which can test its behaviour with less mock objects having to be created.

[0] b)

[0] c)

[2] d) There should be little correlation if RFC is high. Classes with a high RFO could simply be calling alot of methods from one other class, this would lead to a low CBO value as it is just referencing 1 unique class.

On the other hand, if the CBO is high, it is likely that RFC will be high. This may be because classes that reference a high number of different classes are likely to call their methods, which directly contributes to the calculation of the CBO.

==================================================================

END (Duration = 83m35s, Remaining = 46m25s)