**FIT2107-S2-2017-Exam-Attempt** [55.25/66 = 84%]

**Question 1** [11.25/12]

a)

Efficiency [1.5]

- The software should fulfill the requirements within the device's hardware capabilities

- MYDAM taking over a minute to save marks means that it is violating the time behaviour aspect by taking too much time to work

Usability [1.5]

- The software should be easy to use for the user

- MYDAM has a user interface that is hard to figure out without help, meaning that it does not have a high level of usability for the user

Functionality [1.5]

- The software should have functions that fulfill the software's original requirements

- MYDAM violates this by rounding up rather than to the nearest integer, which violates the original requirements set by the university policy

Reliability [1.5]

- The software should have a low error rate with high uptime, including proper maintenance of data stored

- MYDAM experiences crashes with marks lost this morning, meaning that the servers for the system is very unreliable and information is not easily retained

b)

[Design/code review for any of them]

Efficiency

- System testing should have been applied including tests for submitting the marks, this will include timing of the action and will prevent the code from hitting production if the action took too little time. [0.75]

- System design review

Usability: (?)

- Prior usability testing could have been done with a sample set of users (lecturers/markers) using the software in a mock situation. Acquiring their feedback will allow some feedback to improve the user interface before going to production. [0.75]

- If mock-ups were created before the software was developed, it could be verified with internal UI designers through a formal inspection to ensure proper usability standards are met [0.75]

Functionality

- Unit testing for the module that computes the final subject mark could have been written and executed before reaching production. The unit test would have set inputs and expected outputs, allowing it to pick up the rounding error. [0.75]

- Code inspections could possibly pick up this defect through the reviewers noticing the error [0.75]

Reliability

- Chaos engineering testing to test the sudden shutdown of a server and ensuring that data is backed up according [0.75]

- System testing to simulate multiple users using the system and ensuring 99.9% uptime [0.75]

**Question 2 (??)** [19/20]

a) Paper [6]

b) [5]

Tests

- mylist = [ 5, "hi", Mock.MockObject()]

- mylist = []

2 test inputs are required to achieve 100% branch coverage.

The first test input contains

- a Number (to satisfy the True condition at branch 10),

- a string (to satisfy the False conditon at branch 10 and the true condition at branch 13)

- and an Object (to satisfy the False condition at branch 10 and the False condition at branch 13)

This test input then satisfies the True condition at branch 17

The second test input contains no items. Satisfying the False condition at branch 17.

Altogether all decisions are covered, hence 100% branch coverage

c) [5]

It is not a good set of tests for conducting unit tests as it would be hard to debug each individual scenario if the unit test failed. Instead, the inputs should split into multiple unit tests to cover a singular, specific scenario for easier debugging.

e.g. One test for a list of Numbers, one with a list of strings, one with a list of objects, an empty list

d) (?) [3]

[MC/DC involves branch coverage + condition coverage + all entry/exit points]

[Only one entry point in methods and all exit points covered already]

MC/DC testing cover both condition coverage and decision coverages. Because each decision here in this function is composed of only one condition, covering every decision will already cover every condition here. Hence, no additional tests are required.

**Question 3** [19/22]

1. [8]

Categories can be selected from the input fields of the page, with the various choices including valid and invalid possible inputs

Category: Assignment Name

Choices:

- A string (assuming no max length is specified)

- Empty string (assuming a min length of 1 is specified)

Category: Available from

Choices:

- Date no earlier than current date and no later than end of semester's date

- Date earlier than current date

- Date later than end of teaching period's date

- Empty date

Category: Due date

Choices:

- Date later than "Available from"

- Date earlier than "Available from"

- Date later than end of teaching period's date

- Empty date

Category: Maximum size (LOC)

- Integer more than or equal to 0 (assuming students can submit empty assignments)

- Integer less than 0

- String

- Float

- Empty field

Category: Execution timeout (seconds)

- Float more than 0.0s

- Float less than or equal to 0.0s

- Empty field

- String

1. [2]

There can be 2x4x4x5x4 possible test cases. Calculated from multiplying the number of choices of each category.

There are invalid test frames present as there are test frames where one or more invalid inputs are selected and would cause an error (e.g. if two fields are empty).

c) [5]

Test frame #1

- Assignment name: A string

- Available from: Date no earlier than current date and no later than end of semester's date

- Due date: Date later than "Available from"

- Maximum size (LOC): - Integer more than or equal to 0

- Float more than 0.0s

Test case #1

- "Hi there"

- 20/12/2020

- 27/12/2020

- 1000

- 10

Test frame #2

- Assignment name: Empty string

- Available from: Date no earlier than current date and no later than end of semester's date

- Due date: Date later than "Available from"

- Maximum size (LOC): - Integer more than or equal to 0

- Float more than 0.0s

Test case #2

- ""

- 20/12/2020

- 27/12/2020

- 1000

- 10

d) [4]

Pairwise testing can be used to reduce the number of test cases. It can reduce the number of test cases by drawing up test frames that ensure that every pair of choices are selected at least once. This is in comparison to category-partitioning where every possible combination of choices are selected.

**Question 4** [0/0]

-----------

**Question 5** [6/12]

a) [6]

Yes, the evidence in the graph supports the idea. This is because of the positive correllation where, as the maintainability coefficient increased (the software is deemed maintainable), the average time to close issues decreased (meaning that defects were solved quickly). This may be due to using object-oriented metrics like RFC, CBO, etc. that pose hints at a softwares maintainability.

However in actuality, the maintainability coefficient may not provide a full picture in whether a software is actually maintainable or not. Maintability of a software is defined by the ease of fixing defects, as well as the ease of modifying and extending the current codebase (which the coefficient does not cover). More metrics would need to be thought of and recorded to ensure a better picture of software maintainability.

b) [0]

I forgot.