Office Use Only

MONASH University

Semester Two 2017
Examination Period

Faculty of Information Technology

EXAM CODES: **FIT2107**

TITLE OF PAPER: **Software Quality and testing - SAMPLE PAPER!**

EXAM DURATION: 2 hours writing time

READING TIME: 10 minutes

THIS PAPER IS FOR STUDENTS STUDYING AT: (tick where applicable)

☐ Berwick	☑ Clayton	☑ Malaysia	☐ Off Campus Learning	☐ Open Learning
☐ Caulfield	☐ Gippsland	☐ Peninsula	☐ Monash Extension	☐ Sth Africa
☐ Parkville	☐ Other (specify)			

During an exam, you must not have in your possession any item/material that has not been authorised for your exam. This includes books, notes, paper, electronic device/s, mobile phone, smart watch/device, calculator, pencil case, or writing on any part of your body. Any authorised items are listed below. Items/materials on your desk, chair, in your clothing or otherwise on your person will be deemed to be in your possession.

No examination materials are to be removed from the room. This includes retaining, copying, memorising or noting down content of exam material for personal use or to share with any other person by any means following your exam.
Failure to comply with the above instructions, or attempting to cheat or cheating in an exam is a discipline offence under Part 7 of the Monash University (Council) Regulations.

AUTHORISED MATERIALS

OPEN BOOK	**☑ YES**	**☐ NO**
CALCULATORS	**☐ YES**	**☑ NO**
SPECIFICALLY PERMITTED ITEMS **if yes, items permitted are:**	**☐ YES**	**☑NO**

Candidates must complete this section if required to write answers within this paper

STUDENT ID: __ __ __ __ __ __ __ __ __ DESK NUMBER: __ __ __ __ __

INSTRUCTIONS TO STUDENTS

This sample exam is of a similar format and the questions are indicative of some of the types of questions you might see on the real exam. Some of them are directly from past exams in FIT4004; some have been modified, and some are completely new. I think, if anything, they are *slightly* harder than the real exam will be.

There are a total of 75 marks available in this exam. The number of marks allocated to a question is a *rough* guide to how long you should spend on it.

Question 1 – Quality (12 marks)

Programmable Logic Controllers (PLCs) are a specialized type of embedded computer system used by engineers to control industrial processes. They can be used to control motors, heating elements, valves (automated taps to control liquid or gas flow), and many other devices that are present in automated factories. They respond to a variety of sensors, including "on/off" sensors as well as analogue sensors that measure things like temperature, pressure, and speed.

Rather than conventional programming languages, PLCs are programmed using a specialized graphical language – the "programs" look like electrical circuit diagrams. They can often specify exact timing constraints – for instance, that a motor is to stop 50 milliseconds after a temperature sensor reads a certain value. Programs for a PLC are written on PCs and uploaded to PLCs via a USB or network connection, and interpreted by the PLC firmware.

The factories, and the hardware that these PLCs are used to control, often lasts for decades.

You are responsible for developing the firmware for a new PLC model. The firmware acts as the operating system for the PLC – it provides the environment for the programs written by the engineers for the specific factory installation to run. It includes the interpreter for the PLC programming language, and facilities to upload new or modified programs. It does not include the development environment that runs on the PCs, nor the programs written for a specific factory installation.

System quality attributes are defined in ISO/IEC9126 as:

- **Functionality**
- **Reliability**
- **Usability**
- **Efficiency**
- **Maintainability**
- **Portability**

For each quality attribute, give *two* examples of a specific, assessable requirement/property that *might plausibly* be applicable for the PLC firmware. If an attribute is of limited or no relevance in this application and therefore you can't give two such examples, explain why. If the attribute is relevant to the application, but it is difficult to describe assessable requirements relating to it, explain why.

Your understanding of the system quality attributes is much more important than your knowledge of the specifics of PLCs, so your answer does not have to go into great detail about the operational aspects of factories or other systems that PLCs are used to control.

(12 marks)

1 mark per reasonable requirement/property.
2 marks if they provide a reasonable explanation of why property is of limited/no relevance. Partial marks granted if explanation is not completely insane but somewhat unclear or slightly off-topic.

Some example answers (note – you do NOT have to give these exact answers to receive the marks, these are purely to give an indication of acceptable answers to this question might be):

- **Functionality:**
  - **Administrator must be able to upload a new program to the PLC via Ethernet connection.**

    - System must be able to switch a gas load valve on and off according to an uploaded program.
- Reliability
    - PLC must be able to execute an uploaded program continuously for 1 week without interpretation errors.
    - PLC must restart execution of uploaded program after a power failure.
- Usability:
    - As this device has such a limited user interface, usability is of limited relevance here.
- Efficiency:
    - Programmed actions should take place within 1ms of the specified response timeframe.
    - The system must be able to execute programs containing 1000 diagram elements
- Maintainability
    - Very hard to give quantifiable maintainability requirements in general, so would accept that it's not easy to give testable maintainability requirements for this system.
    - Would accept, for instance: firmware must be updateable via USB stick or network.
    - Also would accept something related to level of unit test coverage, eg automated unit tests for the system must cover 90% of statements.
- Portability
    - Would accept that it's of limited relevance given firmware is designed for specific hardware.

Question 2 – inspections (2 + 12 + 8 = 22 marks)

a) For a Fagan inspection of a source code artefact, should the programmer of that artefact attend the inspection? Explain why or why not in a couple of sentences. (2 marks)

Yes. (Answers with no are worth max 0.5 marks – literature is very clear on this).

Fagan's paper explains that the job of the programmer/designer is to explain the artefact to the review panel. They will also be responsible for fixing identified issues; it will be easier if they are present in the meeting where they are identified. It can also reduce false positives because the programmer can explain how particular elements work.

2 marks for a Yes answer that clearly explains at least two major reasons.
1.5 mark for a Yes answer that clearly explains one major reason, or mentions two reasons but the explanations aren't great.
1 mark for a Yes answer but with a poor explanation.
1 mar
0.5 marks for Yes without a justification, or a no mark with a semi-plausible rationale.
0 marks for a no answer without justification.

b) You are developing a new Fagan inspection checklist for source code written in Python.

Consider the following candidate items for the checklist and indicate whether they are suitable as-is for such a checklist, need to be modified (and propose a modified version), or are not suitable for a Fagan inspection checklist. In each case, state a rationale.

i) All variables must be initialized before use.
ii) Class and attribute names must be consistent with reviewed UML class diagram.
iii) All method names must reflect the purpose of the method.
iv) All methods must use the best algorithm available.
v) All methods that are not part of a class or module's public API should have names that begin with an underscore ('_') character.
vi) Attributes should be made part of a class or module's public API only when necessary.

(12 marks)

2 marks per item. Award 2 marks if a reasonable item is stated reasonable – or 2 marks if they give a *strong* argument as to why it's not really reasonable and a good rewrite to deal with the issue. 1 mark for a semi-plausible reason, 0 for a poor or no reason.
2 marks for a not-reasonable item if they say it should be dropped altogether and give a good rationale, 1 mark if dropping is reasonable but rationale is unclear. 2 marks if they say it should be modified and the modification is reasonable and clear, 1 mark if they say modify but modification sucks.

i) This is IMO reasonable.
ii) This is also reasonable.
iii) This is also reasonable, but would accept "too vague" and an argument for droping/.
iv) This is not reasonable, as Fagan inspections are supposed to detect clear defects, not try and optimize. Would accept "should be dropped altogether". Modifications need to be good.
v) This is reasonable.
vi) "Necessary" is vague, so would accept a rewrite or dropping here.

c) Some software engineering experts have proposed, based on the fact that more defects are found by inspections than unit testing, to eliminate unit testing entirely and rely exclusively on inspections and system testing.

In two or three paragraphs, analyse the strengths and weaknesses of such a proposal. Do you think it is likely to be an improvement on current practice? Justify your answer.

(8 marks)

Key points to consider (note – I'm not suggesting you answer this type of question using a succession of dot points! Write in normal prose) :
Unit testing requires additional work.
Good system testing should cause the units to execute those parts relevant to the system.
Therefore, by this argument unit testing is redundant.
However…unit testing finds *different* bugs to inspections.
Inspections are more likely to find maintainability defects, where unit tests find more functional correctness bugs.
Unit tests can find bugs more quickly than system tests, because you need a system to do unit testing, and unit testing is easier to automate in many cases.
FWIW, my conclusion is that unit testing is worth doing in most systems.
8 marks: A coherent argument making points backed up by empirical evidence, clearly written and structured.

6 marks: A satisfactorily structured answer making at least some relevant points, sufficiently well written to be understandable.

4 marks: an answer that makes some relevant points but has substantial gaps in evidence or logic, or is hard to understand.

2 marks: something useful said in the ballpark of addressing the question.

Question 3 – Mocking (6 marks)

In your assignment 3, you were asked to "mock" calls to the Tweepy library when unit testing your program. The Tweepy library provides the ability to access information from the Twitter social network from within a Python program.

Describe and clearly explain two advantages of using mocking to test this particular program. Take a few sentences to explain each advantage – including providing specific scenarios where mocking is helpful
(6 marks)

Advantages include:

- Avoid nondeterministic behavior (that is, mock calls always give same result, unlike Twitter searches), so we can easily test correctness.
- Tests run faster (because no network accesses)
- Account credentials don't need to be stored in repository (minor, but would accept).

  1.5 marks for each reasonable advantage identified, 1.5 marks for the corresponding explanation.

Question 4 – unit testing (2 + 4 + 3 + 6 + 4 = 18 marks)

The questions relate to the following Python source code:

```
class Student:
    def __init__(self, firstName, familyName, grade):
        self.givenName = firstName
        self.familyName = familyName
        self.grade = grade

    def letter_grade(self):
        '''Return the letter grade from the numerical mark
        for this student
        according to the standard Monash scale'''
        if self.grade >= 80:
            return "HD"
        elif self.grade >=70:
            return "D"
        elif self.grade >=60:
            return "C"
        elif self.grade >=50:
            return "P"
        else:
            return "N"

    def pass_subject(self):
        '''Return true if the student has received a passing grade for
        this subject'''
        return self.letter_grade() is not "N"

    def result_summary(self):
        return self.firstName + " " + self.familyName + " " +
str(self.grade) + " " + self.letter_grade()

class Unit:
    def __init__(self, students):
        self.students = students

    def fail_report(self):
        for student in students:
            if not student.pass_subject():
                print(student.result_summary())
```

(question on next page)

a) Give the smallest possible set of *test inputs* (that is, values for `self.grade`) that would achieve 100% *statement coverage* for the method `letter_grade()`. (2 marks)

For example ([80, 70, 60, 50, 49]) would do.

1 mark if statement coverage achieved, 1 mark for smallest.

b) Give the smallest possible set of *test inputs* for `Unit.fail_report()` (where a test input involves calling the constructor to create a `Unit` object and calling `fail_report()` on it) to achieve 100% *branch* coverage for `fail_report()` (not including methods invoked by it). (4 marks)

I don't care how you express this, but I'd say something like,

Myunit = Unit([Student("Fred", "Bloggs", 48), Student("Jane", "Nguyen", 70)]).

1 test input, a list with two students in it!

2.5 marks if you have test inputs that achieve branch coverage, 1.5 marks if it's the smallest set (1 input)

c) Do you think it would be a good idea to test `fail_report()` just using your answer to part b)? Explain why or why not and if not, provide an alternative approach in about a paragraph. (3 marks)

2 marks Testing for all the possible conditions is kinda bundled up in a single test, which makes it harder to debug than necessary.

1 mark Suggest you split the cases up into separate tests (eg an empty list).

d) Suggest an appropriate *black-box* testing strategy for `letter_grade()`. Explain why it is appropriate for this method, and use it to come up with a set of *test cases* for `letter_grade()`. (6 marks)

2 marks for a clear strategy:

1 mark for explanation Generally, domain testing is highly suitable for this as the input is one number with well-defined partition boundaries.

2 marks for tests following that strategy, 1 mark if there are clear statements of expected outputs.

e) Write Python code using the `unittest` module that implements the *first* of your tests in part d). (4 marks)

4 marks: correct Python code that would execute the test (modulo minor syntax errors).

2 marks: something that involves defining a test and checking an assertion.

Question 5- metrics (3 + 3 + 4 + 4 = 14 marks)

One of the Chidamber and Kemerer metrics is Coupling Between Objects.

It is defined as:
CBO for a class is a count of the number of other class to which it is coupled…ie methods of one use methods or instance variables of another. (Chidamber and Kemerer (1994))

a) In general, do you think a system with many classes with a high CBO metric would be easier or harder to *unit test* well than an equivalent system with lower CBO values? Explain your answer in about a paragraph. (3 marks)

It's going to be harder, in general, because the more classes that interact with each other with, either the more bugs are going to be the result of interaction rather than within the class, the harder the results are going to be to interpret, or the more mocking you'll have to do.

1 mark for just stating "harder", 2 marks for quality of explanation (2 marks for a clearly argued paragraph that makes key points along the lines of that above, 1 for a partial or poorly structured justification).

b) Is it likely to be easy to reuse classes with very high CBO score? Justify your answer. (3 marks)

In general, no, because high CBO scores mean that it depends on a whole bunch of other classes, all of which have to be available and relevant to the reuse. Makes it hard to repurpose
1 mark for just stating "harder", 2 marks for quality of explanation (2 marks for a clearly argued paragraph that makes key points as above, 1 for a partial or poorly structured justification).

c) Are there aspects of good design related to coupling that CBO *does not* measure adequately? Explain your answer – if there are such aspects, include examples to illustrate. (4 marks)

Clearly, yes, as CBO does not take into account the nature of the coupling between the classes. There are different types of ways a class can be coupled, and not all of them are as tight. For instance, access to shared data, such as public attributes, is a much harder-to-change and bug prone way for classes to be coupled than access through a well-defined API.

1 mark for yes, 2 marks for quality of explanation, 1 mark for clearly stated example.

d) Chidamber and Kemerer's Response for a Class metric (RFC) is defined as the size of the set of the following methods:
   - The methods defined in the class.
   - Any method called directly by a method defined in the class.

If RFC is high, does that tell us anything about the likely value of the CBO score? What about the other way around – if CBO is high, is RFC likely to be high? Explain your answers. (4 marks)

If RFC is high, CBO will often be high, but not always. Could call many methods in one or

a few classes. A single-class program would have a very high RFC but a CBO of 0.

If CBO is high, RFC will generally be high. Possible excceptions include if all coupling is via public attributes, but this would be very bad design and uncommon in real systems.

2 marks for each answer. For full marks, must clearly consider trends and possible exceptions, and explain clearly. 0.5 marks for a reasonable trend answer with no further explanation.