SOFT3202/COMP9202 Assignment 4

Software Construction and Design 2

1 Assignment Description

Please complete the trial examination by filling in your answers in the provided template markdown file, `answers.md`. Submit the file.

Academic Declaration

By submitting this assignment, you declare the following:
I declare that I have read and understood the University of Sydney Student Plagiarism: Coursework Policy and Procedure, and except where specifically acknowledged, the work contained in this assignment/project is my own work, and has not been copied from other sources or been previously submitted for award or assessment.
I understand that failure to comply with the Student Plagiarism: Coursework Policy and Procedure can lead to severe penalties as outlined under Chapter 8 of the University of Sydney By-Law 1999 (as amended). These penalties may be imposed in cases where any significant portion of my submitted work has been copied without proper acknowledgment from other sources, including published works, the Internet, existing programs, the work of other students, or work previously submitted for other awards or assessments.
I realise that I may be asked to identify those portions of the work contributed by me and required to demonstrate my knowledge of the relevant material by answering oral questions or by undertaking supplementary work, either written or in the laboratory, in order to arrive at the final assessment mark.
I acknowledge that the School of Computer Science, in assessing this assignment, may reproduce it entirely, may provide a copy to another member of faculty, and/or communicate a copy of this assignment to a plagiarism checking service or in-house computer program, and that a copy of the assignment may be maintained by the service or the School of Computer Science for the purpose of future plagiarism checking.

Multiple-choice questions (+20)

1. What is the Dependency Inversion Principle?

   (a) A class should have only one responsibility.

   (b) Subclasses should be substitutable for their base class.

   (c) High-level modules should depend on low-level modules to have direct control over their data and control-flow.

   (d) Depend on abstractions, not concrete classes.

2. Which of the following is a limitation of mutation analysis?

   (a) It incurs a high computational expense.

   (b) It is a weaker form of test adequacy compared to branch coverage

   (c) It guarantees the detection of all real faults in the program.

   (d) It removes the need for a human developer in writing test cases.

3. What are the advantages of using the Decorator pattern?

   (a) It enforces static behavior and prevents unwanted changes

   (b) It makes it easy to modify code of the original class to add new behaviors.

   (c) It allows for the adding of responsibilities to objects dynamically at runtime.

   (d) It eases the process of combining instances of the same class from different codebases.

4. Which of the following is a creational pattern?

   (a) Visitor

   (b) Factory Method

   (c) Decorator

   (d) Strategy

5. Which of the following is a structural pattern?

   (a) Facade

   (b) Strategy

   (c) Observer

   (d) Factory Method

6. What is entailed by the competent programmer hypothesis?

   (a) Most faults in code are minor deviations from the correct implementation

   (b) Testers should focus only on integration testing due to programmer error.

   (c) Most programmers write incorrect code that needs full rewrites to be fixed.

   (d) Software testing is important because bugs are hard to catch

7. Which logical component under the Model-View-Controller architectural pattern should handle the storage of data?

   (a) View
   (b) Controller
   (c) Storage
   (d) Model

8. What are the advantages of using the Proxy pattern?

   (a) Eliminates the need for the actual object.
   (b) Allows developers to breaks encapsulation for performance gains.
   (c) Controls and manages access to the real object.
   (d) Allows the process of object creation to bypass controls

9. Unix Signals are an instance of

   (a) Composite Pattern
   (b) Singleton Pattern
   (c) State Pattern
   (d) Observer Pattern

10. REST is an

    (a) Architectural Pattern
    (b) Architectural Style
    (c) Design Pattern
    (d) Design Style

11. Which of the following best describes the Composite design pattern?

    (a) It ensures only one instance of a class is created.
    (b) It allows us to represent hierarchical structures with a uniform interface.
    (c) It notifies multiple objects when the state of one object changes.
    (d) It provides a way to extend an object's behavior without modifying it.

12. Given a user requirement "The system shall allow administrators to export borrowing statistics in a structured format (e.g., JSON or CSV)," which of the following is part of the SRS acceptance criteria for covering it?

    (a) When an administrator selects the Export Borrowing Data option, the system must generate a JSON file containing borrowing statistics and allow the user to download it.
    (b) When a member logs in, they must only be able to view their own borrowing history and not any other member's records.
    (c) Any data stored must be encrypted using industry-standard encryption techniques.

(d) The system displays an error message indicating a timeout if the system takes too much time to generate the report.

13. Which of the following is a good set of circumstances for using the chain of responsibility design pattern?

    (a) When you need to dynamically configure request handling logic

    (b) When you need to ensure only one class is responsible for object creation.

    (c) When you need to encapsulate interchangeable behaviors and select one at runtime.

    (d) When you want to reduce memory usage by sharing instances of similar objects.

14. What is the difference between blackbox and whitebox testing?

    (a) Only whitebox testing verifies the output of a program

    (b) There is no real difference; both black-box and white-box testing aim to test code coverage and branch conditions.

    (c) Only black-box testing involves measuring the coverage of test cases.

    (d) Black-box testing is done without knowledge of the program's internal code, while white-box testing uses that knowledge to guide testing.

15. Which of the following is a principle of GRASP?

    (a) Information Expert

    (b) Singleton

    (c) Chain of Responsibilities

    (d) Liskov Substitution Principle

16. Given a function that accepts two parameters, a and b. If the valid values of a ranges from 1 to 5, and the valid values of b range from 1 to 10, which of the following would NOT be a test case derived from Boundary Value Testing under the Weak Robust assumption?

    (a) f(-1, 11)

    (b) f(0, 6)

    (c) f(4, 0)

    (d) f(4, 11)

17. Given a function that accepts two parameters, a and b. If the valid values of a ranges from -1 to 5, and the valid values of b range from 1 to 10, which of the following would NOT be a test case derived from Boundary Value Testing under the Strong Normal assumption?

    (a) f(-1, 1)

    (b) f(0, 1)

    (c) f(5, 10)

    (d) f(-1, 11)

18. Which of the following is a correct description of generational fuzzing or mutational fuzzing?

(a) You can use a sanitizer with a mutational fuzzer, but not with a generational fuzzer

(b) A generational fuzzer can only detect issues associated with file formatting

(c) A generational fuzzer rewrites the code reached in the program

(d) A mutational fuzzer starts with a set of seed inputs, and then mutates them

19. What is a fault?

(a) An incorrect program behaviour

(b) A code artefact causing a failure

(c) A test case that failed

(d) A program that fails a test case

20. Which of the following best describes Branch Coverage in software testing?

(a) It verifies that every individual statement in the code is executed at least once

(b) It verifies that each possible branch (decision outcome) is executed at least once

(c) It focuses on testing loops and iterations

(d) It verifies that every branch through is executed at least once

END OF MULTIPLE CHOICE QUESTIONS

Question 21. Design patterns (10 marks)

The following program shows an `Expression` abstract class that represents nodes in an arithmetic expression language for a calculator. It defines a language grammar as a class hierarchy. Each subclass corresponds to a rule in the grammar of the language and implements the evaluate() method to recursively interpret the expression it represents. For Questions 21a and 21b, refer to this program.

```
from abc import ABC, abstractmethod

class Expression(ABC):
    @abstractmethod
    def evaluate(self, context: dict) -> float:
        ...

class Number(Expression):
    def __init__(self, value: float):
        self.value = value

    def evaluate(self, context: dict) -> float:
        return self.value

class Variable(Expression):
    def __init__(self, name: str):
        self.name = name

    def evaluate(self, context: dict) -> float:
        try:
            return context[self.name]
        except KeyError:
            raise NameError(f"Variable '{self.name}' isn't in context")

class Multiply(Expression):
    def __init__(self, left: Expression, right: Expression):
        self.left = left
        self.right = right

    def evaluate(self, context: dict) -> float:
        return self.left.evaluate(context) * self.right.evaluate(context)

context = {'x': 2}
# represents x * 2 * 3
expr = Multiply(Variable('x'),
           Multiply(Number(2), Number(3)))

print(expr.evaluate(context))  # prints 12
```

a. Identify the design pattern used above. State one advantage and one disadvantage of using the pattern in general (5 marks)

b. You are tasked to add a new operation to support addition. Which class or methods would you write or modify? Give a brief description of the implemented code. (5 marks)

Question 22. Decision tables (5 marks)

a. Fill in the decision table based on the following description of a program. Parts of the answer have been pre-filled. Assume that all necessary information has been given and that all inputs to the function are valid. If you need to make any further assumptions about the function or test cases, please state them.

```
def classify_package(length, width, height):
    """
    Classify a package based on its volume and dimensions.

    The function calculates the volume of a package
    using its length, width, and height,
    and returns a classification string based on the following rules:

    - If the largest single dimension (length, width, or height)
      is greater than 500cm,
      the package is classified as "Oversized".
      This check takes precedence over volume.
      Note: A dimension of exactly 500cm is not considered oversized.
    - Otherwise:
        - Volume < 1000 cm^3: "Small Package"
        - 1000cm^3 <= Volume < 5000cm^3: "Medium Package"
        - Volume >= 5000cm^3: "Large Package"

    Parameters:
        length (int): The length of the package in cm.
        width (int): The width of the package in cm.
        height (int): The height of the package in cm.

    Returns:
        str: The classification of the package.
    """
```

Conditions	**R1**	**R2**	**R3**	**R4**	**R5**
length * width * height < 1000					
1000 <= length * width * height < 5000					
Output					
Small Package					

Question 23. Whitebox testing techniques. (20 marks)

The following code snippet shows a function that analyzes a collection of scores and determines whether the majority of them correspond to passing grades (A or B). Refer to this program for Questions 23a, 23b, 23c, and 23d.

```
1     def analyze_scores(scores):
2         total = 0
3         passed = 0
4         for score in scores:
5             if score >= 90:
6                 grade = 'A'
7             elif score >= 80:
8                 grade = 'B'
9             else:
10                grade = 'F'
11
12            if grade == 'A' or grade == 'B':
13                passed += 1
14            total += 1
15
16        if total > 0 or passed / total >= 0.6:  # BUG:  "or" should be "and"
17            return "Majority passed."
18        else:
19            return "Majority failed."
```

The following code snippet is a test suite designed to test the program shown above.

```
assert analyze_scores([90]) == "Majority passed."
```

a. Evaluate the above test suite in terms of statement coverage by identifying the line numbers of the covered statements. The first statement has been pre-filled for you. (4 marks)

Lines 1,

b. Evaluate the above test suite in terms of branch coverage by identifying the line numbers and conditions of the covered branches. The first branch has been pre-filled for you. (5 marks)

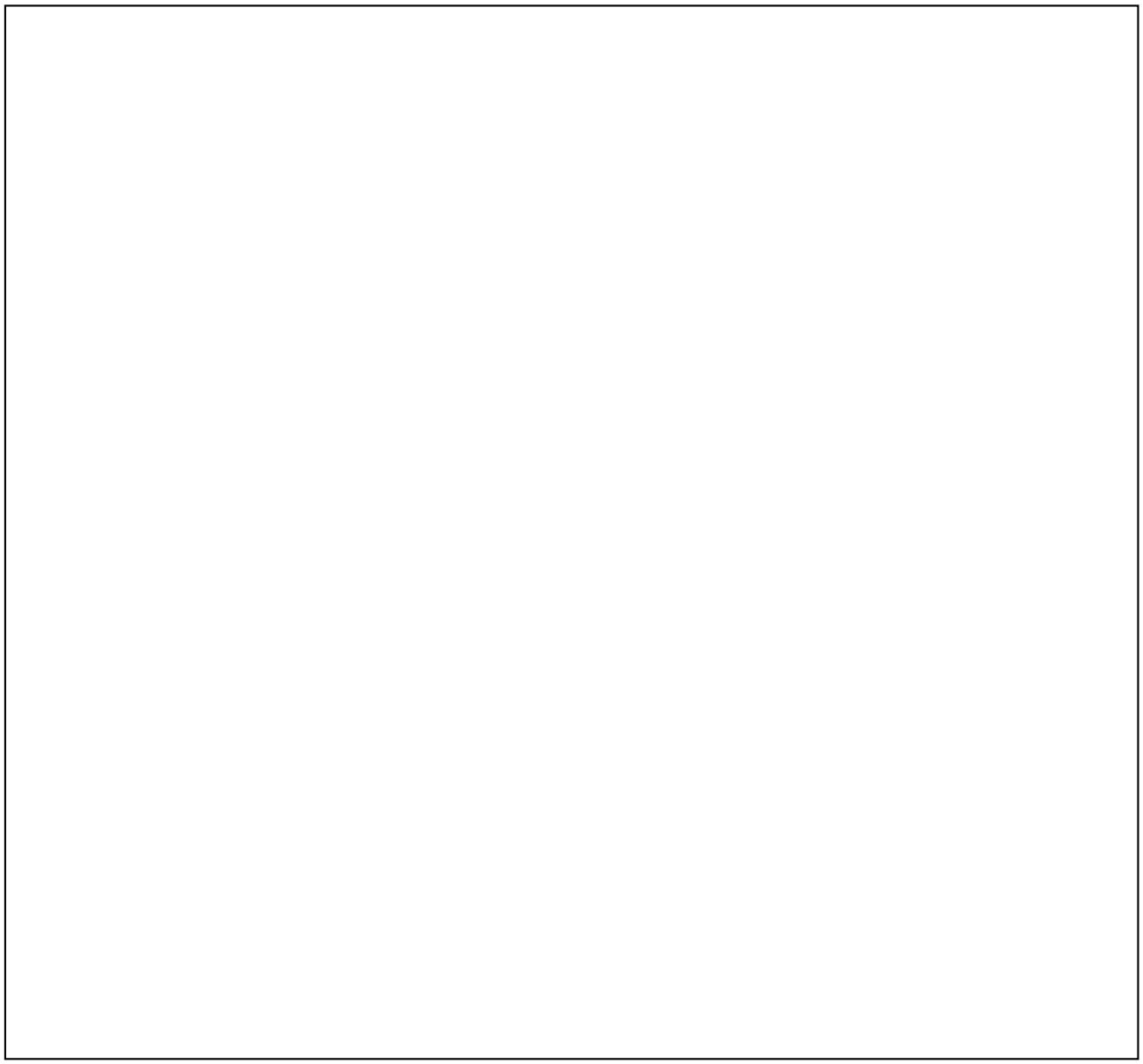
Line	Condition	True Taken? (Yes/No)	False Taken? (Yes/No)
4	`for score in scores`	Yes	Yes

c. Design a new test suite to maximize statement coverage of `analyze_scores`. For each test case, provide the input and specify which statements it covers. One test input has been pre-filled for you. (5 marks)

Test input	**Statements covered**
[0]	1, 2, 3, 4, 5, 7, 10, 11, 13, 14, 17

d. Notice how the program above contains a bug, with the lines contributing to the bug marked with the "BUG" comment.

Achieving full branch coverage does not guarantee that the buggy behavior will be detected during testing. Explain why. Then, recommend a more suitable coverage criterion, and explain why the recommended criterion would be more effective. (6 marks)



Question 24. Fault-based testing (18 marks)

The following is a program (the "**original program**") that checks if each element in an input sequence is greater than its preceding element.

```
def is_increasing(xs):
    for i in range(1, len(xs)):
        if xs[i] <= xs[i - 1]:
            return False
    return True
```

Consider the 5 mutation operators.

1. Arithmetic Operator Replacement, e.g., changing a "+" to a "-"
2. Relational Operator Replacement, e.g., changing a "!=" to "=="
3. Constant Replacement, e.g., changing "1" to "10"
4. Break Continue Replacement, e.g., changing "break" to "continue"
5. Slice Index Removal, e.g., changing "xs[:5]" to "xs"

Mutant A:

```
def is_increasing(xs):
    for i in range(1, len(xs)):
        if xs[i] >= xs[i - 1]: # BUG: a mutation operator is applied here
            return False
    return True
```

a. Which of the given mutation operators, when applied to the original program, can generate Mutant A? (1 mark)

Mutant B:

```
def is_increasing(xs):
    for i in range(1, len(xs)):
        if xs[i] <= xs[i * 1]: # BUG: a mutation operator is applied here
            return False
    return True
```

b. Which of the given mutation operators, when applied to the original program, can generate Mutant B? (1 mark)

Mutant C:

```
def is_increasing(xs):
    for i in range(2, len(xs)): # BUG: a mutation operator is applied here
        if xs[i] <= xs[i - 1]:
            return False
    return True
```

c. Which of the given mutation operators, when applied to the original program, can generate Mutant C? (1 mark)

d. A mutant is killed by a test case if the output of the mutant is different from the output of the original program when executing the test case. For each mutant, describe its killing conditions and state whether or not would be killed by the following test case? The answer for Mutant A has been pre-filled for you. (6 marks)

```
assert is_increasing([1, 1]) is False # the single test case
```

Mutant	**Killing condition/ input description**	**Killed by the test case (Yes/No)**
A	A list with at least two elements that is in either a strictly increasing or decreasing order	No

e. Write a test case that would kill Mutant A. (2 marks)

f. Given the following three mutants, Mutants D, E, F, describe the killing conditions of each mutant, and then explain which mutant subsumes another. Part of the answer has been pre-filled for you. (7 marks)

Mutant D:

```
def is_increasing(xs):
    for i in range(1, len(xs)):
        if xs[i] == xs[i - 1]: # BUG: checks for equality
            return False
    return True
```

Mutant E:

```
def is_increasing(xs):
    for i in range(1, len(xs)):
        if xs[i] <= xs[i - 1]:
            return False
    return False # BUG: always returns false
```

Mutant F:

```
def is_increasing(xs):
    for i in range(1, len(xs)):
        if xs[i] <= xs[i - 1]:
            return True # always returns true
    return True
```

Mutant	Killing condition/input description
D	A list that has at least one element less than its predecessor and has no two consecutive elements equal.

Mutant x	Mutant y	Does x subsume y? (Yes/No)
D	E	No
D	F	
E	D	
E	F	
F	D	
F	E	

Question 25. Contracts (7 marks)
The following is a program that removes duplicates from an input collection.

```
def remove_duplicates(xs: Iterable[Any]) -> List[Any]:
    seen = set()
    result = []
    for x in xs:
        if x not in seen:
            seen.add(x)
            result.append(x)
    return result
```

a. Given the implementation of `remove_duplicates` above, one postcondition is that the length of the return value must be less than or equal to the length of the input collection. Write this postcondition as an `@ensure` contract. Assume that all necessary dependencies have been imported. (2 marks)

```
@ensure(lambda result:
```

@ensure (lambda result , xs: len(result) <= len(xs))

b. Considering the following pre-condition of `remove_duplicates`:

```
@require(lambda xs: len(set(map(type, xs))) <= 1,
    "All elements must be of the same type.")
```

For each of the following test case, state whether the precondition would be violated. The first test case has been answered for you. (5 marks)

Test case	Violates Precondition? (Yes/No)
`remove_duplicates([2, 3, 4])`	No
`remove_duplicates([2, "3", 4])`	
`remove_duplicates(["2", "3", "4"])`	
`remove_duplicates((2, 3, 4))`	
`remove_duplicates([2.0, 3, 4])`	
`remove_duplicates([None])`	

Question 26. Software development lifecycle (5 marks)

Note: This question is ONLY for COMP9202 students.

You have recently joined a new organization as a software developer. Upon reviewing the test suites, you observe that the team follows a mockist testing strategy, making extensive use of mocks to isolate units of code. However, one developer strongly argues for a classical approach, where tests focus on the actual behavior of components and their real collaborators.

Critically evaluate the developer's claim by describing the classical and mockist testing strategies, and discussing their advantages and disadvantages.