Q1 Testers
3 Points

Explain the difference between a developer testing their own code versus an independent tester testing the original developer's code. Include an example.

When a developer tests his/her own code, he/she understands the system and the code base, so he/she will gently test the code, and the test is driven by deadlines.
When an independent tester is testing the original developer's code, he/she must learn the system, and will attempt to break the system, and the test is driven by "quality".

Q2 Requirements Specification Document
3 Points

Which of the following does a Requirements Specification Document **not** contain?

- [ ] Overview of what the system will do
- [x] Info about the algorithm or logic
- [x] Technical specifications
- [x] UI discussion
- [ ] A list of the functional requirements
- [ ] Description of all the requirements

Q3 Structured Basis Testing
4 Points

According to the structured basis testing, what is the minimum number of tests that you would need for the following piece of code?

8

```cpp
#include <iostream>

using namespace std;

int main() {

    int num;
    bool b = true;

    while (b) {

        cout << "Enter an integer: ";
        cin >> num;

        if (num % 2 == 0 && num > 0) {
            cout << "Your integer is even!" << endl;
        }

        bool prime = true;
```

```
        for (int i = 2; i < num; i++) {
            if (num % i == 0) {
                prime = false;
            }
        }

        if (prime) {
            cout << "Your integer is prime!" << endl;
        }

        cout << "Continue? (y/n) ";
        string c;
        cin >> c;
        if (c.compare("n") == 0) {
            b = false;
        }
    }
    return 0;
}
```

Q4 Mutant Analysis

2 Points

A developer has their base code as well as 30 mutants. They have a set of tests that they run on their base code. All the tests pass! They then run their tests on their mutants. What would be the best-case scenario for the developer?

- ○ The tests all fail for each mutant
- ○ Some tests pass and some tests fail for each mutant
- ○ The tests all pass for each mutant
- ◉ Most tests pass and one test fails for each mutant

Q5 Testing

3 Points

What purposes do Unit, Regression, and Integration testing serve? Give 1-2 sentences for each. From there, give an example of usage for each of the test types.

Unit testing tests a single class. It only test each individual code.
Integration testing tests multiple classes. It puts several units together and tests when unites of functionality are integrating, no errors are introduced.
Regression testing tests that future changes won't break the system. It is performed whenever anything changes in the system, to check that no new bug is introduced.
For example, unit testing only tests if it can save images. Integration testing also test the functionality of checking color correctness. Regression testing tests when we add a new feature to that Image class, say, a change color feature, if any new bugs are introduced.

Q6 Cohesion and Coupling

3 Points

Explain ‘high cohesion and low coupling’ and why it’s desirable.

High cohesion is to keep parts of the code base that are related to each other. Low coupling means separating unrelated parts of the code base as much as possible.
"High cohesion, low coupling" is desirable because high cohesion makes code simpler,

extensible, and reusable; low coupling enables us to make changes without too much worrying about break other modules; It increases readability and maintainability.

Q7 Requirements

2 Points

What is the difference between functional and non-functional requirements?

Functional and non-functional requirements describe different aspects of the requirements. Functional requirement describes what the software does; Non-functional requirement defines limitations that the software has.

Q8 SOLID

5 Points

List the 5 solid principles discussed in the class. Explain and Discuss 2 of them. Your explanation per principle should be at least a full paragraph containing conceptual example(s).

1. Single Use Responsibility
Single Use Responsibility means that a class should only have a single responsibility. Foe example, when we create a journal class for adding journals, it would be better to use another separate class for saving the journal, like PersistenceMgt().save(journal, "1.txt");

2. Open to extension, closed to modification

3. Liskov Substitution Principle
Liskov Substitution Principle suggests that objects should be replaceable with instances of their subtypes without altering program correctness. For example, Square as the subtype of Rectangle would violate the principle and influence the correctness of the program.

4. Interface Segregation Principle

5. Dependency Inversion.

Quiz 02

● GRADED

STUDENT
Ruijun Ni

TOTAL POINTS
23 / 25 pts

QUESTION 1

Testers

1 / 3 pts

✔ + 1 pt	Correct difference between the two testers
+ 2 pts	Good example

+ 0 pts Incorrect

QUESTION 2

Requirements Specification Document **3** / 3 pts

QUESTION 3

Structured Basis Testing **4** / 4 pts

QUESTION 4

Mutant Analysis **2** / 2 pts

QUESTION 5

Testing **3** / 3 pts

QUESTION 6

Cohesion and Coupling **3** / 3 pts

QUESTION 7

Requirements **2** / 2 pts

QUESTION 8

SOLID **5** / 5 pts