CS5351 Software Engineering

2021/22 Semester A

Quiz1: Summary & Reference

Summary

Max	86
MEDIAN	62.5
MEAN	61.58
MIN	34

RANGE	**GRADE**
70 - 100	A
55 - 69	B
40 - 54	C
30 - 39	D
0 - 29	F

Question 1: Fuzzing (a)

- It is a grey-box technique: the fuzzer begins with an input seed 13 and measures the branch coverage in g(·).
- It is a feedback-oriented technique: the fuzzer determines a new test seed (e.g., 11) if it covers new branch compared with the historical coverage. (seed 11 covers B1 and B3)
- It is a mutation-based technique: the input to the function under test is generated from the seed. (e.g., seed 12 is generated via subtracting 13 by 1)

Question 1: Fuzzing (b)

- Yes.
- When the initial seed is 12/11/10/9/8/7/6/5/4/3, the fuzzer can detect the vulnerability.
- Coverage-guided fuzzer is effective because region of new coverage is with higher probability to reveal crash problem.
- Mutation-based fuzzer is effective due to the choice of the initial seed and the design of mutation operators, which is possible to explore the entire input space.

Question 1: Fuzzing (c)

- Explanation
  - Yes. In this example, the buggy test cases lie in the middle part of the input space such that applying O2(dividing by 2) is faster to approach to the buggy cases.
- Illustration
  - When applying O2 ahead of O1, the executed test cases are <13, 6, 12, 3>, which is shorter than the one from O1-O2 order.

Question 1: Fuzzing (d)

- An example
  - New O1: $y = \lfloor {}^{3}/_{2}\, x \rfloor$
  - New O2: $y = \lceil {}^{1}/_{2}\, x \rceil$
  - Executed sequence of test cases: 19, 7

Input executed	Coverage [new?]	Seed queue after mutated test case execution
	NIL	<13>
O1(13) = 19	INFEASIBLE	<**13**>
O2(13) = 7	B5, B7, B8, B9 [yes]	<13, 7>

Question 1: Fuzzing (d)

- Another example
  - New O1: y = 12 – (x * 0.8)
  - New O2: y = x + (-1)^x
  - Executed sequence of test cases: 2, 12, 11, 3

Input executed	Coverage [new?]	Seed queue after mutated test case execution
	NIL	<13>
O1(13) = 2	B2, B4, B5 [yes]	<**13**, 2>
O2(13) = 12	B2, B4, B5, B6 [yes]	<**13**, 2, 12>
O1(2) = 11	B1, B3, B5 [yes]	<13, **2**, 12, 11>
O2(2) = 3	B5, B7, B8, B9 [yes]	<13, **2**, 12, 11>

Question 2: Fundamental Principles for Design

(a) Benefits of abstraction, information hiding and encapsulation:

- Abstraction: help programmers avoid writing low-level code, code duplication and increase maintainability, reusability.
- Information Hiding: help the program to increase security because only important details are visible and accessible to the user.
- Encapsulation: programmers can change the internal implementation of class independently and flexibly without affecting the user.

Question 2: Fundamental Principles for Design

(b) OCP Example

```
class MyShape{
    private ShapeType shapeType;
    double area() {
        shapeType.calculateArea();
        ...
    }
}

class ShapeType{
    double calculateArea(){ … }
}
```

```
class Circle extend ShapeType{
    private double radius;
    ...
    double calculateArea(){
      return 3.14 * radius * radius;
    }
}

class Square extend ShapeType{
    private double sideLength;
    ...
    double calculateArea(){
      return sideLength * sideLength;
    }
}
```

Question 2: Fundamental Principles for Design

(b) OCP Example

```
class MyShape{
   private ShapeType shapeType;
   double area() {
      if (shapeType instanceof Circle){          // violate OCP
            shapeType. calculateArea();
      } else if (shapeType instanceof Square){  // violate OCP
            shapeType.calculateArea ();
      }
      ...
   }
}

class ShapeType{
   double calculateArea(){ … }
}
```

```
class Circle extend ShapeType{
   private double radius;
   ...
   double calculateArea(){
      return 3.14 * radius * radius;
   }
}

class Square extend ShapeType{
   private double sideLength;
   ...
   double calculateArea(){
      return sideLength * sideLength;
   }
}
```

Question 2: Fundamental Principles for Design

(b) OCP Example

- **OCP** is open for extension, and close for modification. A basic rule is to allow th**e** behavior of a class to be extended without modifying the existing code.
- My **OCP** example is implemented on MyShape class. The class is closed for modification but open for extension through the calculateArea function.
- After modification, the example violates OCP. Because a new shape is added each time, the class MyShape will be modified.

Question 2: Fundamental Principles for Design

(c) Reasons

- Figure 1: For all the refactoring types except for "Rename", most of time half of programmers want to do refactoring manually.
- Figure 2: The process of code refactoring is not an easy one since it may result in high number of regression bugs and build breaks.
- Figure 3: Code smells are not the driving factors for refactoring. Figure 3 shows improved readability of code is the highest benefit of refactoring.
- Figure 4: Readability is the highest reason given by developers for initiating refactoring. This does not support the notion of code smells being the driving factors for refactoring.

I will conduct code refactoring to remove the violation of OCP because it will improve readability of code and will not involve significant risks.

Question 3: Modern Code Review

- Process
  - Assign roles in the MCR process: moderator M (you), author A (a developer), reviewers R (the rest developers)
  - Email and GitHub are selected as the tools used in the MCR process, because it is the common available tools to the team's situation.
  - Author subsequently makes a patch P on a code block C to address some problems and sends the patch P via Email or GitHub. If it is the latter, the author should notify the moderator and reviewers. The writing should be in English.
  - Each reviewer R evaluates P to get output O, provides the ideas about code readability and maintenance, and makes a decision: either deem P good or reject P. The communication could be in English or Putonghua.
  - After getting the consensus acceptance on the Patch P, the author A requests a merge to the moderator M (with the patch P if via Email). And the moderator M is responsible to merge the code to the code repository.

Question 3: Modern Code Review

- Explanation
  - The MCR process is informal. They could use any common language and available tools for communication.
  - The logistic is tool-based. They should use some platform to record the code change and leave the comments.
  - The MCR focus on the review of patch, not the entire code base.
  - The code review should have at least one reviewer in order to satisfy one reviewer per code review constraint.

Question 4: Software Process

(a) Customer involvement:

- The manager’s requirement of customer involvement is improper.
- The manager does not want to involve customers too much in the development process, which violates the rule of Scrum. We should know that the requirements cannot be fully understood or defined at one time. If we don’t communicate with customers continuously in the development process, the product may not be accepted by the customer finally.
- Hence, in each sprint, the developer group should be together with customers to discuss the goal of the current Sprint, prioritize functions to be completed and divided into detailed tasks when there is any uncertainty for developers.

Question 4: Software Process

(b) One sprint for one release:

- The manager’s requirement of one sprint for one release is improper.
- The deadline for Release 1 is two months later, therefore the first sprint will be last for 2 months. For one sprint, the development group should conduct periodic short planning and review meetings for tasks completed and not completed in time and revise the set of backlog tasks accordingly. A too long sprint violates the rule of Scrum and the development group cannot summarize and adjust the current work in time. And also cannot draw the burndown chart to track the project.
- Two months is too long for only one sprint. It is better to divide 2 months into 4 sprints and each sprint is 2 weeks.

Question 4: Software Process

(c) Use case prioritization:

- The manager’s requirement of use case prioritization is improper.
- The manager is responsible for prioritizing the backlog during Scrum development. In different release, the manager plans for different targets, from the basic use cases to high value, until all use cases. Prioritizing use cases is a good way to guide the development effort and conduct the incremental development. Because customers can know about the development result directly and give feedbacks to the developers to guide the next release. However, the prioritization process should be talked with customers and agreed by the customer at first, but not only decide by the manager.
- The manager should conduct a meeting with customers and prioritize the requirements with customers and the prioritization should be decided by customers.

Question 4: Software Process

(d) Project backlog versus sprint backlog:

- The manager’s requirement of backlog is improper.
- The project backlog is a prioritized and structured list of deliverables that are a part of the scope of a project, which is often a complete list that breaks down work that needs to be completed. The manager himself cannot decide how to complete use cases in which release and should first talk with customers.
- The sprint backlog lists tasks to complete during a sprint, which is updated once a day. The manager can decide scenarios or stories to be completed in which sprint.
- Both project backlog and sprint backlog should be used in the development process. When the project is delayed, the manager should talk with customers and decide which use cases to be removed from the corresponding release.