Aston University

School of Engineering and Applied Science

CS2020 – Software Engineering

Second Year Examination

CLOSED BOOK

Date: 16th January 2018
Time: 14:00 – 17:00
Duration: 3 hours

Instructions to Candidates

1. **Answer ALL questions from Section A (40 marks)**
2. **Section A contains TEN questions**
3. **Answer THREE questions from Section B (60 marks)**
4. **Section B contains FOUR questions, each question is worth 20 marks.**
5. **Use of calculators IS NOT allowed.**
6. **In all questions that involve the writing of Java program code, a few MINOR syntactic errors will not be penalised. Program code that contains more substantial errors may still gain a substantial proportion of the available marks provided the intended meaning of the code is clear. Thus, candidates are advised to include appropriate comments that briefly explain the intended meaning of their code.**

Materials provided

Answer booklets

Please note that the exam questions are printed on both sides of the exam paper.

This exam paper must not be removed from the exam room.

THIS PAGE HAS BEEN LEFT INTENTIONALLY BLANK

Section A — Answer ALL questions

1. STATE ONE **emergent property** of a car engine system, and ONE **non-emergent property** of this system. JUSTIFY your answers.

(4 marks)

2. a) STATE the main difference between the goals of the **elaboration** and **construction phases** in the Unified Process.

(2 marks)

b) For EACH of the following software engineering activities, STATE a strong argument why it should be performed in either elaboration or construction phase, whichever is more appropriate:

- Developing a prototype game in each of several game development frameworks.
- Running a questionnaire to evaluate a game design expressed in wireframes and storyboards.
- Adding a new variation of an enemy character to an action game.

(3 marks)

3. STATE THREE reasons why it is beneficial to create models of software systems.

(3 marks)

4. STATE in what context one prefers vague requirements over precise requirements.

(2 marks)

5. a) EXPLAIN how a **conflict** can arise in a **repository** of a version control system.

(2 marks)

b) Assume that a team of programmers use a version control system to manage their shared source code. DESCRIBE a process that the team should adopt to minimise the occurrences and impact of conflicts.

(2 marks)

6. A **sequence diagram** may contain **synchronous** and **asynchronous** messages.

   a) EXPLAIN the difference between a synchronous and asynchronous message. Also, DRAW ONE specific example of a synchronous message and ONE example of an asynchronous message in the UML sequence diagram notation.

   (3 marks)

   b) WHICH type of message can an actor send to an object inside a software system? EXPLAIN your answer.

   (2 marks)

7. Making use of a diagram, briefly EXPLAIN the main difference between **client-server** and **peer-to-peer** communication styles between different subsystems.

   (4 marks)

8. a) Making reference to its purpose, briefly explain what is meant by a **persistence framework**.

   (3 marks)

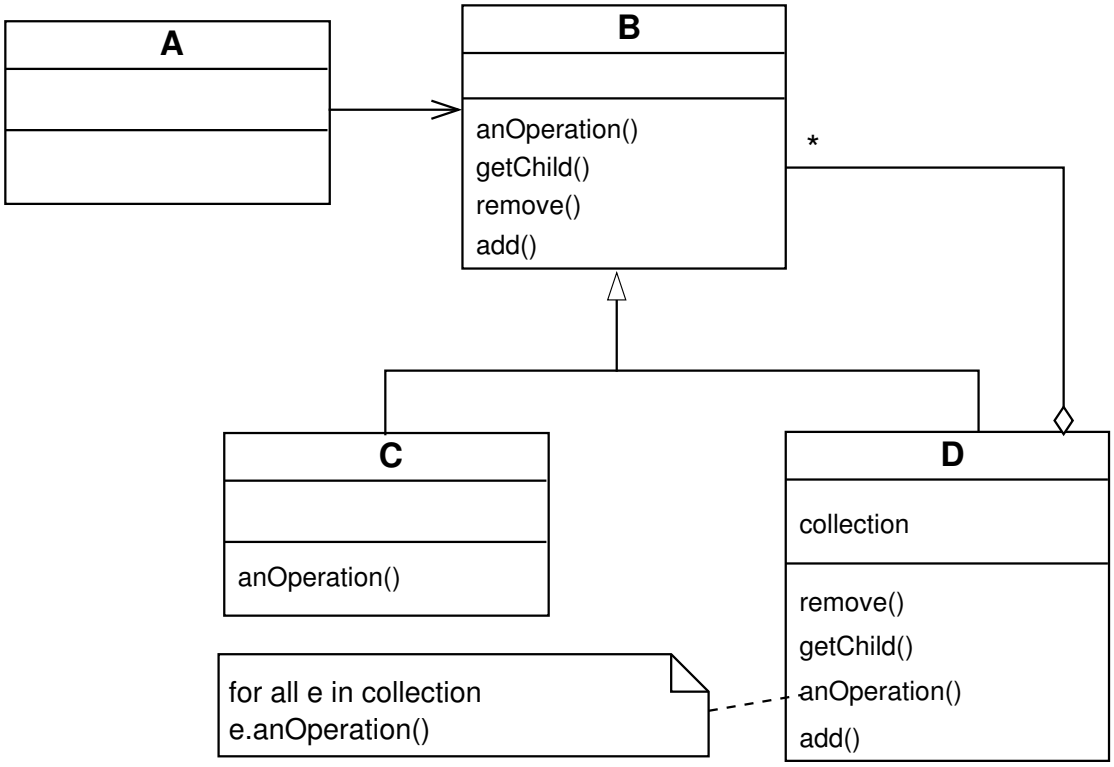
   b) NAME the object-oriented design pattern that is often used to support **lazy materialization** in a **persistence framework**.

   (1 mark)

9. Usability is measurable. Briefly describe TWO measures that can be used to assess usability.

   (4 marks)

10. Consider the following detailed UML **class diagram** which shows the general form of an object-oriented design pattern as defined by the "Gang of Four". This design pattern is designed to model recursive tree structure that is required to capture a complex part-whole relation in a class design.



a) NAME the design pattern described by the given UML **class diagram**.

(1 mark)

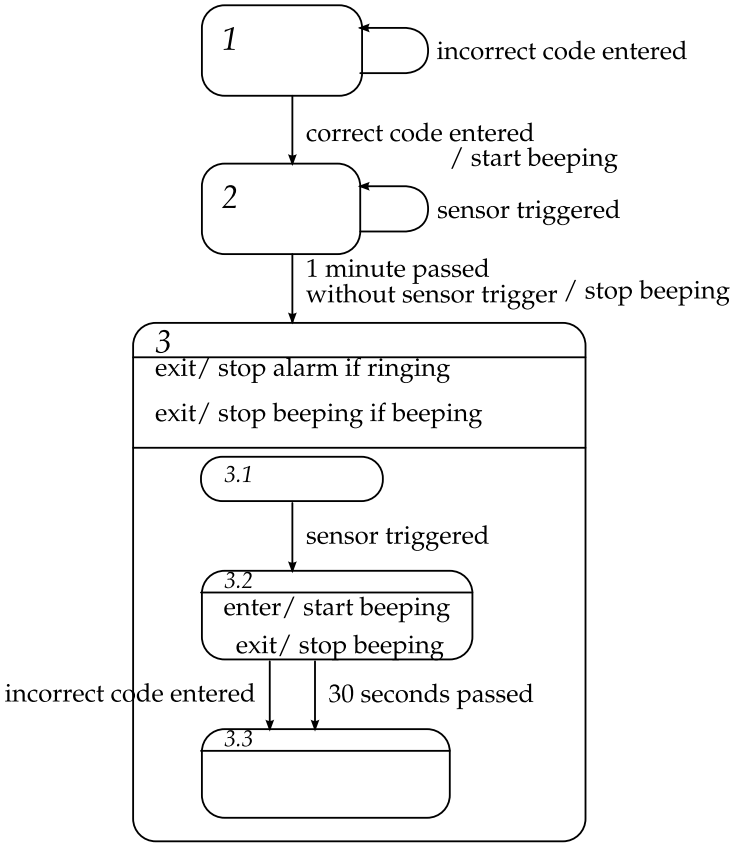
b) GIVE a suitable name to EACH of the classes `A`, `B`, `C` and `D` shown in the given UML **class diagram**.

(4 marks)

END OF SECTION A

Section B — Answer THREE questions

11.a) Consider the operation of an intruder alarm system in a house. The system is pre-activated by entering its code. The system becomes active one minute after it stops sensing movement in the house. If movement is detected when the system is active, it beeps for 30 seconds or until the code is entered. If the code is not entered within 30 seconds, it starts ringing loudly until the code is entered. The following **state machine diagram** attempts to describe the behaviour of this system, but it is not complete:



i) DESCRIBE how to complete the diagram above so that it represents the specified behaviour. In particular, describe how to add: initial state indicator(s), ONE important transition, and ONE important action.

(6 marks)

ii) EXPLAIN why the states in a **state machine diagram** do not have to be named.

(2 marks)

(question continues on next page. . .)

(Question 11 continued. . .)

b) Consider the following objects:

- pen
- ink cartridge
- company logo printed on a pen
- paper

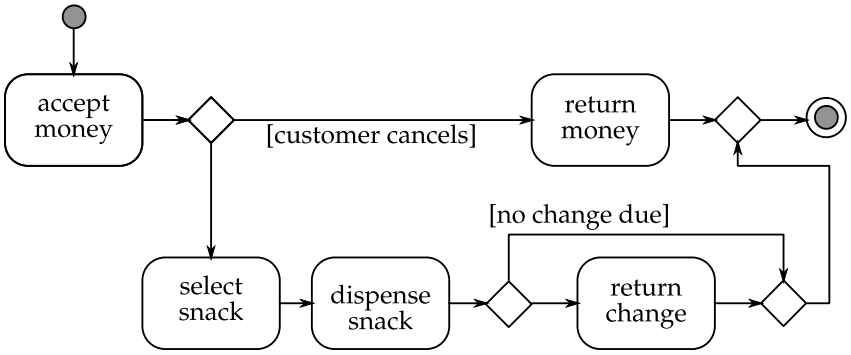
i) Identify among these objects:

- a **composition** (2 marks)
- an **aggregation** that is not a **composition** (2 marks)
- one **association** that is not an **aggregation** (2 marks)

ii) DRAW a **class diagram** showing the THREE associations identified in (i), including their **multiplicities** in both directions. (3 marks)

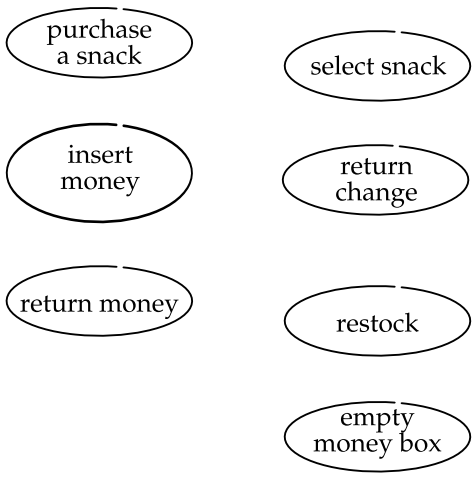
c) EXPLAIN why **business analysis** is relevant in software engineering, using examples from the context of developing either a software simulator of a car or a software system for assisting professional cooking. (3 marks)

12. Consider a certain type of vending machine that facilitates, among others, the process described by the following **activity diagram**:



Assume the manufacturer of the vending machine commissioned you to develop a computer system to control the machine.

The following is an incomplete **use case diagram** of the machine:



a) DRAW the above **use case diagram** and ADD actors into it, appropriately connecting them with use cases.

(2 marks)

b) DRAW any appropriate relationships between use cases in your copy of the above use case diagram.

(5 marks)

(question continues on next page. . .)

(Question 12 continued. . .)

c) Consider the following partially filled use case description form:

Use case number: 1	**Name**: Purchase a snack
Goal: Let customer purchase a snack from the vending machine.	
Brief description: Involves accepting money, selecting a snack and dispensing the snack+change.	
Actors: *(deliberately omitted)*	
Frequency of execution:	
Scalability:	
Criticality:	
Preconditions:	
Postconditions:	
Primary path:	
Alternatives:	
Exceptions:	

In the following tasks you will continue completing this form.

In your answers you can make any further reasonable assumptions on the details of the vending machine parameters and users' preferences.

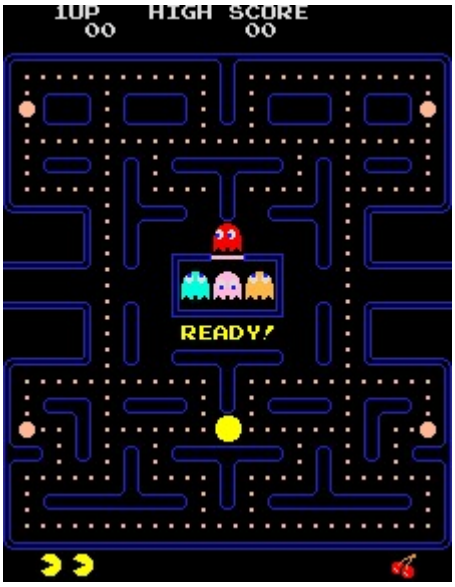
i) Some of the fields in the form refer to **non-functional** requirements. STATE these **non-functional** requirements. (3 marks)

ii) STATE ONE **precondition** and ONE **postcondition** of the use case. (4 marks)

iii) DESCRIBE the **primary path** of this use case. (2 marks)

iv) DESCRIBE ONE **alternative** and ONE **exception** of this use case. (4 marks)

13. a) Consider the following partial description of the classic computer game Pac-Man:



> The player controls Pac-Man through a maze, eating pac-dots. When all pac-dots are eaten, Pac-Man is taken to the next stage.
>
> At each stage, four ghosts (Blinky, Pinky, Inky and Clyde) roam the maze, trying to catch Pac-Man. Each ghost has a distinct personality. For example, Inky is “fickle” and sometimes heads towards Pac-Man and other times away, and Pinky always chases Pac-Man.
>
> If Pac-Man is touched by a ghost, he loses a life. When Pac-Man loses all of his lives, the game ends.
>
> Near the corners of the maze are four larger, flashing dots known as power pellets. As soon as Pac-Man has eaten a power pellet, he enters a temporary state of power-boost. Pac-Man is then able to eat the ghosts. The ghosts, on the other hand, turn deep blue (i.e. in a state of distress), reverse direction of travel and usually move more slowly. Once the time for the power-boost is up, Pac-Man returns to his normal state.

(question continues on next page. . .)

(Question 13 continued. . .)

i) Assume that `Ghost` is a class in a Java implementation of the computer game Pac-Man. As each ghost (i.e. Blinky, Pinky, Inky and Clyde) has a different personality, it has been decided that classes `Blinky`, `Pinky`, `Inky` and `Clyde` are also introduced to the design of the computer game.

DRAW a detailed UML class diagram that shows how the **state design pattern** can be applied to classes `Ghost`, `Blinky`, `Pinky`, `Inky` and `Clyde`.

*Your class diagram should include the use of the UML comment notation to outline how the operation(s) that are typical to the **state design pattern** would be implemented.*
*Your class diagram may omit details that are irrelevant to the application of the **state pattern**.*

(13 marks)

ii) GIVE a brief definition of **specialization cohesion**. (2 marks)

iii) EVALUATE your class design for question part a(i) in terms of **specialization cohesion**. (2 marks)

b) Explain what is meant by **regression testing**. (2 marks)

c) NAME a **framework** that is suitable for performing **regression testing** of Java code. (1 mark)

14. Consider a Java application which contains the following outline Java code with seven classes and one interface.

Listing 1: Class `Location`

```
1 package gameEnvironment;
2
3 public class Location {
4   private int x;
5   private int y;
6   private Occupant occupant;
7
8   // other details omitted
9 }
```

Listing 2: Class `Maze`

```
1 package gameEnvironment;
2
3 public class Maze {
4   private Location[] locations;
5
6   // other details omitted
7 }
```

Listing 3: Interface `Occupant`

```
1 package gameEnvironment;
2
3 public interface Occupant {
4   void makeVisible();
5   void makeInvisible();
6 }
```

Listing 4: Class `Stone`

```
1 package gameEnvironment;
2
3 public class Stone implements Occupant {
4   // details omitted
5 }
```

Listing 5: Class `Diamond`

```
1 package gameEnvironment;
2
3 public class Diamond extends Stone {
4   // details omitted
5 }
```

(question continues on next page. . .)

(Question 14 continued. . .)

Listing 6: Class `Explorer`

```
1  package gameCharacters;
2
3  public class Explorer implements Occupant {
4    private static final int MAX_TREASURE = 15;
5
6    private Stone[] bag;
7
8    // other field definitions omitted
9
10   public Explorer(String name) {
11     this.name = name;
12     bag = new Stone[MAX_TREASURE];
13   }
14
15   public void move(int direction) {
16      // method detail omitted
17   }
18
19   public boolean pickUp(Stone treasure) {
20      // method detail omitted
21   }
22
23   // other details omitted
24 }
```

Listing 7: Class `Species`

```
1  package gameCharacters;
2
3  public class Species {
4    private String name;
5    private int strength;
6
7    public Species(String name, int strength) {
8       // details omitted
9    }
10
11   public int strength() {
12      // details omitted
13   }
14 }
```

(question continues on next page. . .)

(Question 14 continued. . .)

Listing 8: Class `Snake`

```
1  package gameCharacters;
2
3  public class Snake implements Occupant {
4    private String colour;
5    private Species species;
6
7    public void attack(Explorer explorer) {
8      // details omitted
9    }
10
11   // other details omitted
12 }
```

a) DRAW a detailed UML **class diagram** containing ALL of the classes and the interface, showing ALL attributes, operations and relationships with appropriate multiplicity, defined in the given Java code in listings 1–8.

(9 marks)

b) Briefly DESCRIBE what is meant by **refactoring**. (2 marks)

c) Class `Snake` in Listing 8 is the result of a refactoring. Initially, class `Snake` was defined as shown in Listing 9:

Listing 9: Class `Snake` before refactoring

```
1  public class Snake implements Occupant {
2    private String colour;
3    private String species;
4
5    public void attack(Explorer explorer) {
6       // details omitted
7    }
8
9    // other details omitted
10 }
```

NAME the type of **refactoring** that has been applied to the version of class `Snake` in Listing 9.

(1 mark)

d) Making reference to the typical process in **agile** software development, EXPLAIN why it is beneficial to perform the kind of **refactoring** in part (c) on class `Snake`.

(3 marks)

(question continues on next page. . .)

(Question 14 continued. . .)

e) GIVE a BRIEF definition of **Defensive Programming**. (1 mark)

f) **Enforce Intention** is one technique for Defensive Programming. IDENTIFY TWO examples of enforcing intention that can be observed in the given Java code. Briefly EXPLAIN why this technique is used. (4 marks)

END OF EXAMINATION PAPER