Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano

Prof. Elisabetta Di Nitto and Matteo Rossi

20133 Milano (Italia)
Piazza Leonardo da Vinci, 32
Tel. (39) 02-2399.3400
Fax (39) 02-2399.3411

Software Engineering II

June 17th, 2020

Last Name

First Name

Id number (Matricola)

Notes

This exam is handled online. Rules

1. Only use a computer, NOT a tablet, NOR a smartphone
2. Activate the feed of your webcam
3. Share the screen of your computer
4. Keep the microphone on
5. No dual screens
6. No virtual machines
7. When you upload a file through the form, make sure to include your "person id" (the 8-digit number that starts with "10") in the name of the file, AT THE BEGINNING OF THE NAME (so the name of the file should be, say, "10143828_etc.", if your person id is "10143828").
8. The exam is open book, so you can check the course materials (notes, slides, books, past exams, etc.), which can be in paper form, or in electronic form. If the materials are in electronic form, you MUST use the same computer on which you are taking the exam to display them.
9. You cannot interact with other people during the exam.
10. The exam is composed of three exercises. Read carefully all points in the text!
11. **Total available time: 1h and 30 mins**

Scores of each question:

Question 1 (MAX 7) _____

Question 2 (MAX 6) _____

Question 3 (MAX 3) _____

Question 1 Alloy (7 points)
We want to develop an integrated system that supports the execution of online exams. Each online exam is handled by one or more instructors, it can be taken by some students, it is associated with a virtual room and with the exam text, plus the answers by the students. The following rules hold:

1) Only students who registered for the exam can enter the virtual room.
2) While the exam is running, students in the room must have their camera, mic and screen sharing on, otherwise the system assumes they have exited the room.
3) If the exam is running or it has ended, students that result to have exited the room cannot enter again.

Consider the following Alloy signatures that provide a partial model for the problem at hand:

```
sig Person {}
sig Instructor extends Person {}
sig Exam {
   instructors: some Instructor,
   students: set Student,
   vr: VirtualRoom,
   test: ExamTest,
   examStatus: EStatus
}

sig Description {}

sig ExamTest {
   description: Description,
   answers: set Student
}

sig EStatus {}
one sig Planned extends EStatus{}
one sig Running extends EStatus{}
one sig Ended extends EStatus{}
```

Point A (2 points). Define the concept of *Student*, who must own some devices (camera, mic and screen sharing). Add all signatures/constraints/facts needed to make the *Student* concept self-contained and coherent with the description above.

Point B (1 point). Define the concept of *VirtualRoom* that separately captures the students who entered the room at some point, those that are currently in the room, and those that exited the room, in addition to the instructors supervising the room. Add all signatures/constraints/facts you need.

Point C (1 point). Define a fact modelling constraint 1) "Only students who registered for the exam can enter the virtual room".

Point D (1 point). Define a fact modelling constraint 2) "While the exam is running, students in the room must have their camera, mic and screen sharing on, otherwise the system assumes they have exited the room".

Point E (2 points). Define a predicate modelling operation *letIn* that is invoked to let students into the room, and which enforces constraint 3) "If the exam is running or it has ended, students that result to have exited the room cannot enter again. If the exam is planned, they can get in even if they have been out".

Solution

Point A

```
sig Student extends Person {
   devices: some Device
}{#(devices & Camera) = 1 and #(devices & Mic) = 1 and #(devices & Screen) = 1 }

sig Status {}
one sig On extends Status {}
one sig Off extends Status {}

sig Device {
   status: Status
}

sig Camera extends Device {}
sig Mic extends Device {}
sig Screen extends Device {}
```

Point B

```
sig VirtualRoom {
   studentsEnteredRoom: set Student,
   studentsInRoom: set Student,
   instructorsInRoom: some Instructor,
   studentsExitedRoom: set Student
}{
studentsInRoom & studentsExitedRoom = none
studentsEnteredRoom = studentsInRoom + studentsExitedRoom
}
```

Point C

```
fact onlyRegisteredStudentsInVR {
   all s: Student | all e: Exam | s in e.vr.studentsEnteredRoom implies
                                  s in e.students
}
```

Point D

```
fact studentsMustHaveDevicesOn {
 all e: Exam, s: Student |
        e.examStatus in Running and s in e.vr.studentsInRoom
        implies
        ( all d: Device | d in s.devices implies d.status in On )
}
```

Point E

```
pred letIn(e, e': Exam, s: Student) {
   //precondition
   s in e.students and
   ( not (e.examStatus = Planned) implies (not s in e.vr.studentsEnteredRoom) )


   //postcondition
```

```
    e'.instructors = e.instructors
    e'.students = e.students
    e'.vr.studentsInRoom =  e.vr.studentsInRoom + s
    e'.vr.instructorsInRoom =  e.vr.instructorsInRoom
    e'.vr.studentsExitRoom = e.vr.studentsExitRoom - s
    e'.test = e.test
    e'.examStatus = e.examStatus
}
```

Question 2 Testing (red) (6 points)

Consider the following fragment of C code.

```
0   mycode(int S[], int Slength)
1     int h, x, p, res = 0;
2     if (S == null)
3         return 23;
4     x = Slength;
5     if (x == 0)
6         return -5;
7     h = 0;
8     while (h < Slength) {
9         p = S[h] - S[x-(h+1)];
10        if (p != 0)
11            return 14;
12        h  = h+1;
13    }
14    return -5
```

a. Identify the def-use pairs for program mycode and say whether they highlight anything unusual.

b. Consider the following paths for program mycode.

1. 0, 1, 2, 4, 5, 7, 8, 9, 10, 12, 13, 8, 14

2. 0, 1, 2, 4, 5, 7, 8, 9, 10, 12, 13, 8, 9, 10, 12, 13, 8, 14

For each path: symbolically execute the program covering it, highlight the path condition associated with it, and define a test case that realises the path.

Solution

a.
S: (0, 2), (0, 9)
Slength: (0, 4), (0, 8)
h: (7, 8), (7, 9), (7, 12), (12, 8), (12, 9), (12, 12)
x: (4, 5), (4, 9)
p: (9, 10)

res: (1, ?)

Def-use analysis shows that variable res is defined, but never used.

b.1
0. S = SV_S, Slength = SL
2. SV_S ≠ null
4. x = SL
5. SL ≠ 0
7. h = 0
8. 0 < SL
9. p = SV_S[0] – SV_S[SL-1]
10. SV_S[0] – SV_S[SL-1] == 0
12. h = 1
8. 1 ≥ SL
Path condition: SV_S ≠ null and SL ≠ 0 and 0 < SL and SV_S[0] – SV_S[SL-1] == 0 and 1 ≥ SL
Focusing on SL, this means that SL == 1
This also implies that it is true that SV_S[0] – SV_S[SL-1] == 0 for any value in S[0]

Test case: S = [4], Slength = 1

b.2
0. S = SV_S, Slength = SL
2. SV_S ≠ null
4. x = SL
5. SL ≠ 0
7. h = 0
8. 0 < SL
9. p = SV_S[0] – SV_S[SL-1]
10. SV_S[0] – SV_S[SL-1] == 0
12. h = 1
8. 1 < SL
9. p = SV_S[1] – SV_S[SL-2]
10. SV_S[1] – SV_S[SL-2] == 0
12. h = 2
8. 2 ≥ SL

Path condition: SV_S ≠ null and SL ≠ 0 and 0 < SL and SV_S[0] – SV_S[SL-1] == 0 and 1 < SL and SV_S[1] – SV_S[SL-2] == 0 and 2 ≥ SL
Focusing on SL: SL ≠ 0 and 1 < SL and 2 ≥ SL, this means that SL == 2
Focusing on SV_S: SV_S ≠ null and S[0] == SV_S[1]

Test case: S[-2, -2], Slength = 2

Question 3 Project Management
Note: This exercise is from a different exam where the case study was already presented in a previous exercise.

Consider a system that counts the number of passengers on public transports and at their respective stops by scanning what Bluetooth devices are in the vicinity of the vehicles and of the stops.
The system considers 2 types of public transportation means: buses and trams. Vehicles of different types are differently equipped, in the sense that buses are equipped with 2 Bluetooth devices (one at the front, one at the back), whereas trams, which are smaller, are equipped with only one Bluetooth device in the

center of the vehicle. Bus/tram stops are also equipped with one Bluetooth device each. On the passenger side, each passenger has a mobile device, which also has a Bluetooth device. Each Bluetooth device has a unique identifier, and it can be used to scan what other Bluetooth devices are in its vicinity (the result of the scan is the list of ids of detected Bluetooth devices).
Each bus/tram and each stop has its own id. Vehicles and stops keep track of the Bluetooth ids that they detect through Bluetooth scans. More precisely, each vehicle/stop has an internal function that periodically retrieves and updates the list of Bluetooth ids that are detected in its vicinity through scans. Each vehicle/stop offers a function that allows clients to retrieve the current list of stored Bluetooth ids.

The system offers to operators the possibility to look for specific buses/trams/stops and retrieve the number of passengers currently in the system. In addition, it allows operators to set, for each bus/tram/stop, the threshold of number of passengers over which the bus/tram/stop should be considered overcrowded. Finally, it periodically provides a report with the highest number of passengers reached in each bus/tram/stop, highlighting the overcrowded situations.

Calculate the function points for the part of the system targeted to operators. Motivate your choices.

Refer to the following table to associate weights to the function types:

	Weights		
Function types	Simple	Medium	Complex
External Inputs	3	4	6
External Outputs	4	5	7
External Inquiry	3	4	6
Internal Logic Files	7	10	15
External Interface Files	5	7	10

Solution

ILF
Vehicles (vehicle ID, line number, capacity, Vehicle ID) → simple (weight 7)
Stops (stop ID, address) → simple (weight 7)
CollectedDeviceIDs (vehicle ID or stop ID, time of collection, deviceID) → medium as typically this is a table with a large number of elements (weight 10)

ILF tot = 17

EIF none

E Inputs
Set overcrowded threshold for nus/tram/stop → simple (weight 3)

E Outputs
Produce a periodic report with all deviceIDs per vehicle/stop → medium, as the number of elements to be considered and the visualization of the situation might not be trivial (weight 5)

E Inquiry
Get the number of passengers currently in a certain vehicle or stop → medium because it requires the processing of CollectedDeviceIDs joined with Vehicles or Stops (weight 4)

Total number of FP = 17 + 3 + 5 + 4 = 29

Question 3: Architecture (3 points)

We want to build a pipeline of the type

DataCollection → DataProcessing → DataStorage

using components that are at our disposal.
More precisely, *DataCollection* components are cheap (200 euros each), but not very available (availability of 80%). *DataProcessing* components, instead, are quite expensive (5000 euros each), but have good availability (availability of 99.9%). Finally, *DataStorage* components cost 2000 euros each, and have availability of 95%.
We have a budget of 20000 euros.
Define an architecture that stays within the budget while achieving an availability of 99.99%.

Motivate adequately your choices.

Solution:

None of the components at our disposal has the required availability of 99.99%, which means that each of them will have to be replicated and put in parallel with others.

For what concerns the *DataCollection* components, putting 6 of them in parallel achieves an availability of $1-0.2^6 = 0.999936$, or 99.9936% availability.
Putting 2 *DataProcessing* components in parallel achieves an availability of $1-0.001^2 = 0.999999$, or 99.9999%.
Putting 4 *DataStorage* components in parallel achieves an availability of $1-0.05^4 = 0.99999375$, or 99.999375%.
In total, we obtain an availability of 0.999936*0.999999*0.99999375, which is 0.99992875 and greater than the desired one.
The total cost is 200*6+5000*2+2000*4 = 19200, which is within the budget.