Software Engineering

Books or notes are **not** allowed.
Write only on these sheets. **Concise** and **readable** answers please.

Surname, name, matricola ____

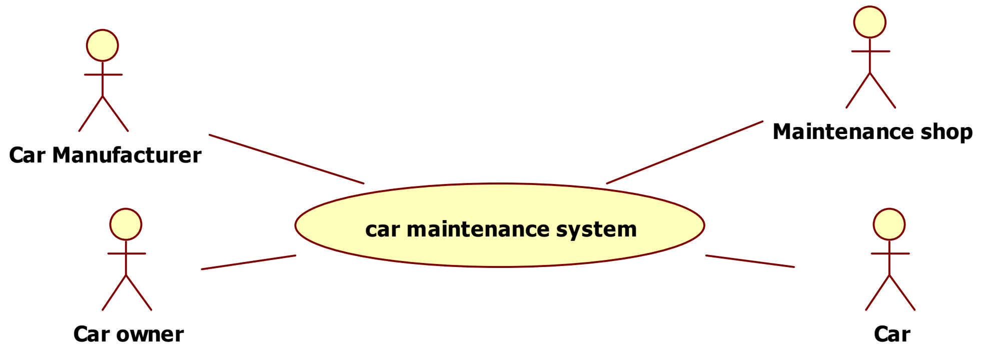
Car maintenance management
During the lifetime of a car many maintenance and repair interventions are needed, some scheduled, some not. It is important for the owner of the car to keep track of these interventions. On the car producer side, it is important to keep track of these interventions both for safety issues (recalls due to defects) and for customer care. Many car producers or car dealers have developed applications to handle car maintenance.

In the following you should analyze and model the application that supports web based car maintenance management.
Key roles to be considered are the car owner, the car manufacturer (defines maintenance schedules and jobs), the maintenance shop (records maintenance interventions).

Key high level functions to be considered are:

- Define the regular maintenance jobs for a car (typically performed by car manufacturer). Remark that jobs (both type and schedule) depend on the car model.
- Record a set of interventions on a car, due to normal maintenance or not (typically performed by a maintenance shop). Record also effort spent and cost for each job, to be used both for owner records and for payment.
- Remind owner about a scheduled maintenance
- Browse and analyze jobs for a car (typically performed by car owner or maintenance shop)

1 (14 points) – a. Define the context diagram (including relevant interfaces)



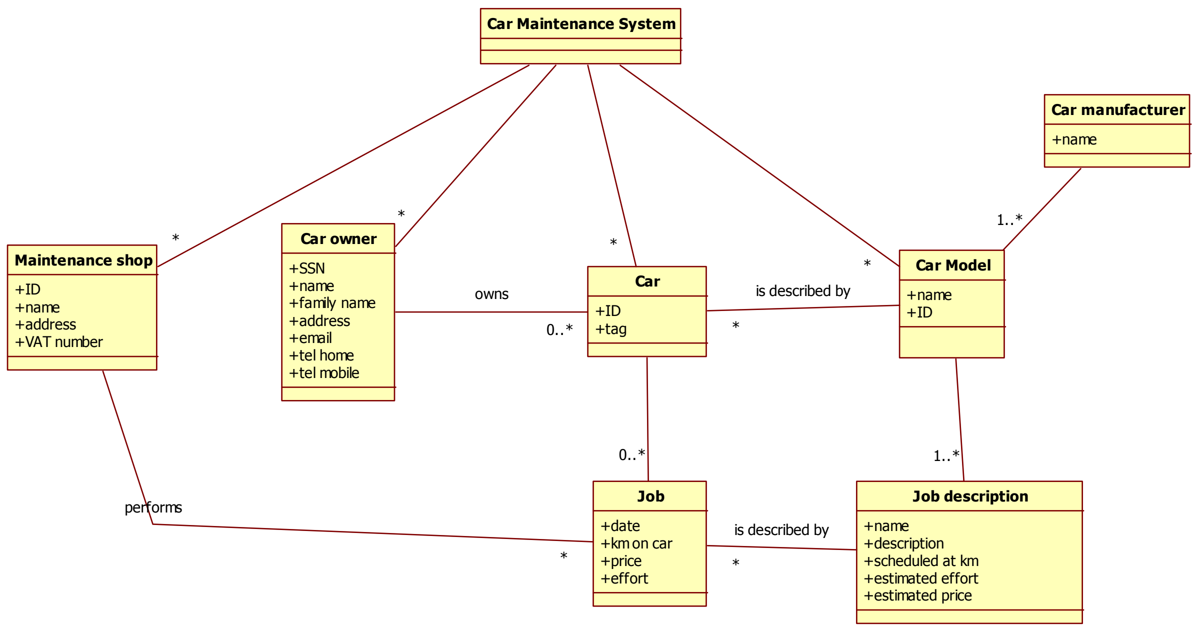
Interfaces. For all actors, physical = internet connection, logical = GUI within browser with specific windows and menus.

Also the car, for more recent models, is an actor in the system. In this case it could have an interface (typically RS232 or CAN) between its main ECU (Electronic Control Unit) and the maintenance system. This kind of access is typically restricted to authorized maintenance shops. The logical interface is made of functions to read data, and data formats for the data exchanged.

List the requirements in tabular form

ID	Type (Functional Non Functional)	Description (see class diagram for Concepts mentioned)
1	F	Car management
1.1 1.2 1.3	F	Add a car, modify a car, attach car to owner
2	F	Car Model management
2.1 2.2	F	Add a model, modify a model
3	F	Job description management
3.1 3.2	F	Add job description, attach job description to car model
4	F	Car Owner management
4.1 4.2	F	Add owner, modify owner
5	F	Car management
5.1 5.2 5.3	F	Add a car, attach car to car model, attach car to car owner
6	F	Job management
6.1 6.2 6.3 6.4	F	Add job, modify job, attach job to job description, attach job to car
7	F	Scheduled maintenance management
7.1 7.2	F	Compute scheduled maintenance job for a car, send reminder to owner
	NF	Performance. All functions should have response time < 0.5sec
	NF	Security. All functions and data about Car Model accessible only by Car manufacturer (Similarly for other functions)

Define the key concepts and entities and their relationships (UML class diagram) for the application



'Car' describes an actual car (has an ID like the chassis number and a tag - the chassis number never changes) with a specific owner, 'Car Model' describes characteristics common to cars of the same model. Similarly 'Job description' describes generic jobs, while 'Job' models an actual Job made on an actual Car. Therefore 'Job description' are attached to 'Car Model', 'Job' to 'Car'. Job descriptions allow defining the schedule of Jobs for a specific car (attribute 'scheduled at km').

Define one scenario describing a maintenance recall (invite owner of car for a scheduled maintenance job, for a number of times or until job done)
Precondition: car C has job description JB to be performed in < 14 days from today
Postcondition: job J corresponding to job description JB performed on car C

Step	Description	Req ID
1	Send reminder to CO (car owner of car C) about job JB	7.2
	One week passes	
2	Send reminder to CO (car owner of car C) about job JB	7.2
	Three more days pass	
3	CO brings car C to maintenance shop and requires to perform job	
4	Shop records job J corresponding to JP on car C	6.1 6.3 6.4

2 (7 points) -Define black box tests for the following function, using equivalence classes and boundary conditions.

int computeElapsedMinutes (int hourStart, int minutesStart, int hourEnd, int minutesEnd)

The function computes the elapsed time, in minutes, between two hours (e.g. the elapsed minutes between 15:22 and 16:25 is 63, because one hour and 3 minutes elapsed).
The function returns -1 if there is an error in the input parameters (start time posterior to end time, impossible time, etc)

hourStart	*minuteStart*	*hourEnd*	*minuteEnd*	*Start time vs end time*	*Valid Invalid*	*Test case*
[minint, 0[[0, 23] [24, maxint]	[minint, 0[[0, 59] [60, maxint]	[minint, 0[[0, 23] [24, maxint]	[minint, 0[[0, 59] [60, maxint]	Starttime <= endtime Starttime > endtime		
[minint, 0[	-	-	-	-	I	T(-100, 1,1,1)→-1
[0,23]	[minint, 0[	-	-	-	I	T(12, -100,1,1)→-1
	[0, 59]	[minint, 0[	-	-	I	T(12,11,-12,1)→-1
		[0,23]	[minint, 0[	-	I	T(12,11,12,-12)→-1
			[0, 59]	Starttime <= endtime	V	T(12,11,12,13)→ 2 T_b(12,11,12,11)→ 0 T_b(0,11,12,11)→ 720 T_b(-1,11,12,11)→ -1 T_b(23,11,23,12)→ 1 T_b(24,11,12,11)→ -1 T_b(12,59,13,59)→ 60 T_b(12,60,12,11)→ -1
				Starttime > endtime	I	T(12,11,12, 1)→ -1
			[60, maxint]	-	I	T(12,11,12, 70)→ -1
		[24, maxint]	-	-	I	T(12,11, 25,10)→ -1
	[60, maxint]	-	-	-	I	T(12,65, 20,10)→ -1
[24, maxint]	-	-	-	-	I	T(26,11, 20,10)→ -1

Each row in the table represents an equivalence class.

Total number of equivalence classes = 3x3x3x3x2 = 162
For this reason some combinations are pruned (character '-' in the table) at the first invalid range encountered.
For the same reason boundary test cases are defined around one equivalence class only (see T_b test cases).
The criterion start time vs end time (in the table above with two values) could have been defined in further detail (ex hourEnd == hourStart , hourEnd < hourStart, hourEnd > hourStart and then consider minutes).

3 (7 points) – For the following function define the control flow graph, and define test cases to obtain the highest possible node coverage, edge coverage, multiple condition coverage, loop coverage.
For the test cases, **write only the input value**.

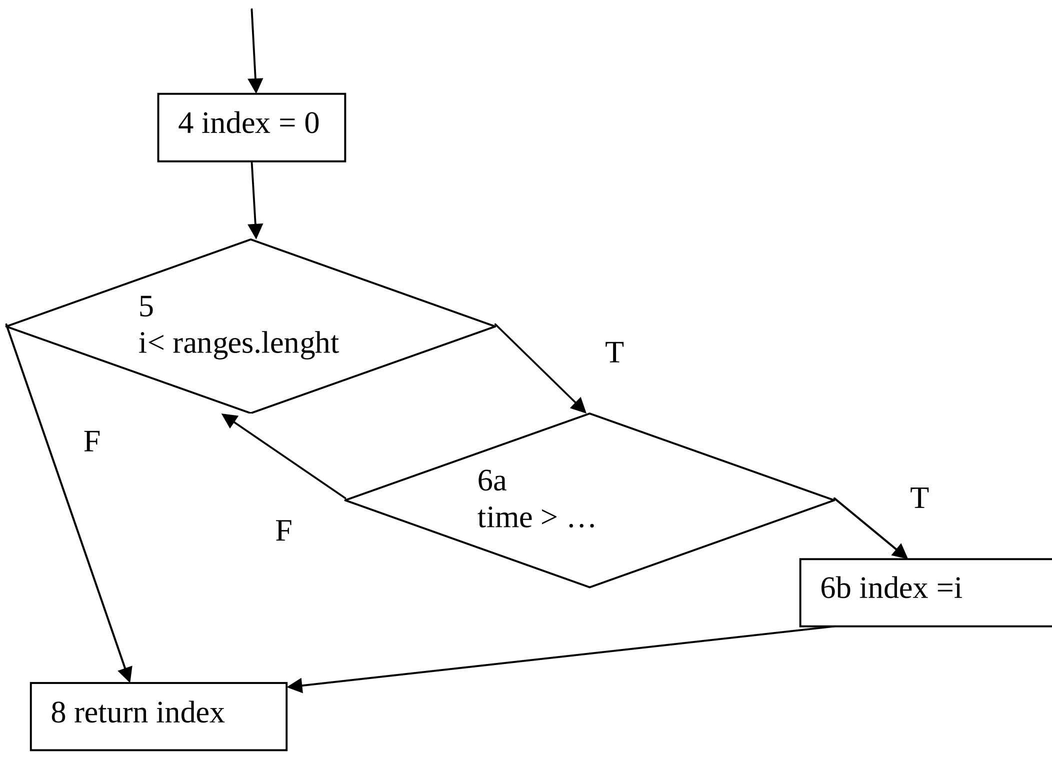
```
1     public int searchRange(int time){
2           int[] ranges = {0, 8, 24};
3
4           int index = 0;
5           for (int i=1; i< ranges.length; i++){
6           if (time > ranges[i-1] && time < ranges[i]) {index = i; break;}
7             }
8           return index;
9     }
10    }
```

Coverage type	Feasibility (Y/N)	Coverage obtained (%)	Test cases
Node	Y	100%	T(12)
Edge	Y	100%	T(12)

Multiple condition (line 6)	Y	75%
		TT
		TF
		FT
		FF
			
			T(1) or T(12)
			T(12)
			T(-6)
			Not feasible

Loop	Y	The loop is executed at maximum 2 times (== ranges.length-1). There is no way not to enter the loop, nor to loop only once (it is possible to loop once, but using the break, and not the condition within the for) 1/3 max coverage possible (2/3 considering the loop once with break)	T(12)

Path	Y	Only 3 paths are possible, so 100% path coverage is feasible:
		4 5 6a 6b 8
		4 5 6a 5 6a 6b 8
		4 5 6a 5 6a 5 8
			
			T(1)
			T(12)
			T(30), or T(-6)



4 (1 points) – Give the definition of effort on the context of project management, and its unit of measure

Time spent by staff to perform an activity. Unit: Person * hours

5 (1 points) – Describe the Copy-ModifyMerge strategy for controlling changes

See slides

6 (1 points) – Describe what are mixed revisions in Subversion

Mixed revisions are revisions of a working copy whose files have different revision numbers. Mixed revisions are possible only in working copies and not in the central repository.

Example: Suppose you have a working copy entirely at revision 10. You edit the file foo.html and then perform an svn commit, which creates revision 15 in the repository.
Therefore the Subversion marks the file foo.html as being at revision 15. The rest of the working copy remains at revision 10. This is a mixed revision.

7 (1 points) – Describe shortly the perspective based inspection technique.
Inspection where readers use different points of view (ex end user, designer, tester) to read the requirement document.