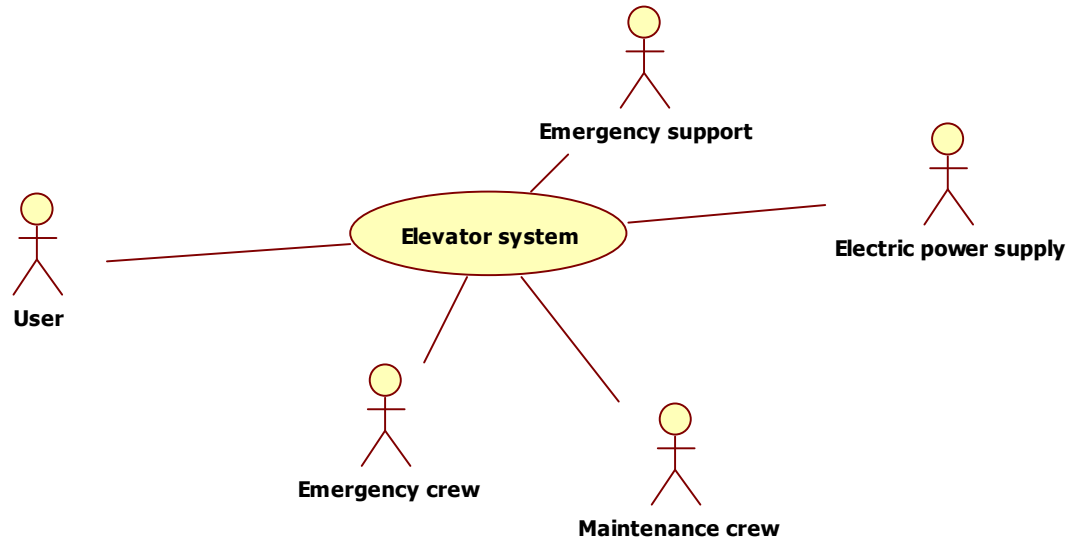
Software Engineering

Elevator system
Consider an elevator system, such as the one in Politong building in Siping campus. It is composed of E elevators, side by side, in one building, serving F floors. At each floor there are buttons to call elevators, displays to show the position of elevators, and doors. Inside each elevator there are buttons and displays too. Each elevator is moved by one engine. All these elements are connected to a computer that controls everything.

In the following you should analyze and model the elevator system.

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



Actor	Logical interface	Physical interface
User		Buttons, displays on floors and on elevators
Maintenance crew	GUI menus and windows to analyze and change state of system	Screen and keyboard
Electric Power supply		220V , 20A, 3 poles
Emergency crew	Procedures to switch on / off / operate the system in case of emergency	Specific buttons and switches to power off / on
Emergency support	Procedure to ask intervention in case of emergency	BJ11 port (public telephone network)

Remark that an elevator system typically comprises the elevators (the cabins going up and down), all displays and buttons, the engines to move the elevator, the (computer based) controller. Those components are shown in the system design diagram. It is possible to define a different context diagram (for instance with only the controller inside, and all the rest outside), but then the relative system design is different (contains only the computer and *hardware software interfaces* to engines, displays etc).

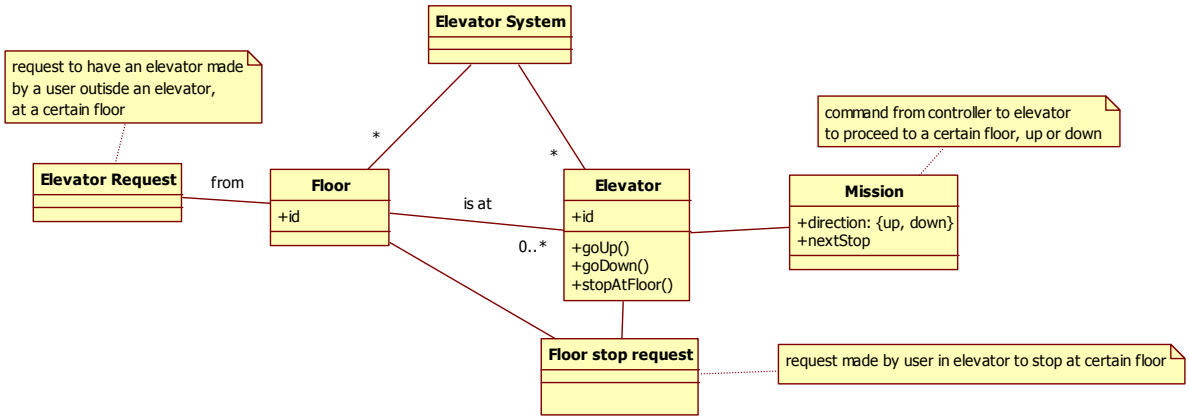
List the requirements in tabular form

ID	Type (Functional Non Functional)	Description
1	F	Call an elevator at floor F (request on button from user at floor F)
2	F	Compute mission M (include choice of what elevator to send)
3	F	Request elevator E to floor F (request from user inside Elevator)
4	F	Show position of Elevator E inside Elevator (manage display in elevator)
5	F	Show position of Elevator E at floor F (manage display at floor F)
6	F	Open doors for Elevator E
7	F	Close doors for Elevator E
8	F	Open doors at floor F, elevator E
9	F	Close doors at floor F, elevator E
10	F	Send mission M to elevator E
11	F	Send request for emergency intervention
12	F	Get / set status of elevator for maintenance tasks
	NF	Performance – update displays in <1 sec

Business rules

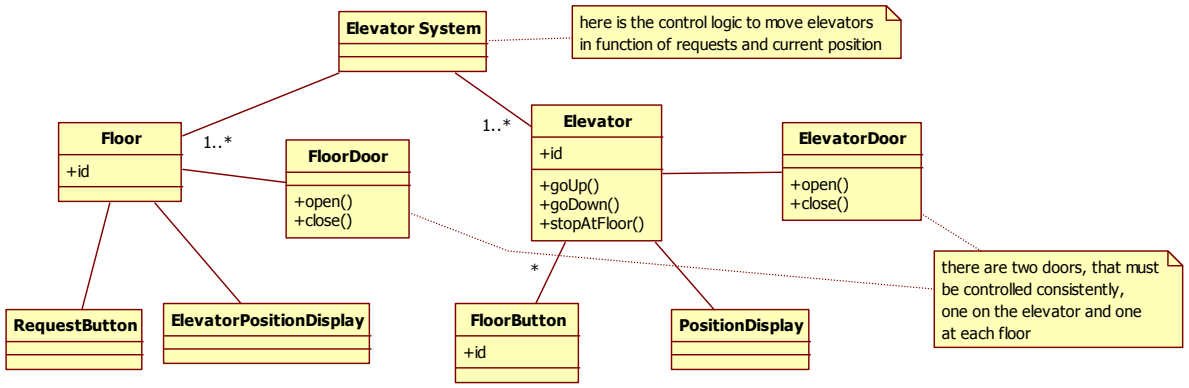
1 floor doors must be closed when elevator is not at floor
2 elevator doors should be closed when elevator travels
3 missions should be defined to minimize waiting time for users

Define the glossary (UML class diagram) for the system



There are many elevators (see multiplicity). Other key concepts are the floor, the position of an elevator at a certain floor, the request of an elevator from a floor (request from outside), the request to bring an elevator to a floor (command from inside elevator). The mission abstracts the command given to an elevator by the controller. Deciding the missions is the key function of the system. It requires to know the current status (positions or current missions of all elevators, all requests from inside and outside) and in function of it to decide the next missions. A flexible elevator system can change missions in real time.

Define the system design model (UML class diagram)



As explained, the system design shows the components of the system (the shopping list of parts). While the glossary shows the key concepts.
It is also possible to merge the two class diagrams in one, loosing some clarity (classes Elevator system, Elevator, Floor are common, plus Elevator request, Mission, Floor stop request from Glossary, Floor Door, Elevator Door, buttons and display from System design)
There is no Computer class. It could be added in the System design, as part of Elevator System.

Define one scenario describing an elevator travel from floor x to y
Precondition: elevator 1 is free and standing at floor x. user requests elevator from floor y. elevator 2 is at floor z
Postcondition: elevator 1 is at floor y

Step	Description	Req ID
1	Call elevator at floor Y	1
2	Compute mission M (elevator 1 from x to y) (elevator 2 is not chosen either because in use or because farther than 1)	2
3	Close doors on elevator 1	7
4	Close doors at floor x, elevator 1	9
5	Send mission M to elevator 1	10
6	When elevator1 is at floor y Open doors on elevator 1	6
7	Open doors on floor y, elevator 1	8
8		

2 (8 points) -Define black box tests for the following function

The function implements an alarm on a clock

int alarm(int hour, int minute, int alarmHour, int alarmMinute)

where hour, minute represent the current time, alarmHour, alarmMinute represent the time of the alarm; the function returns 1 if current time == alarm time, 0 otherwise

ex

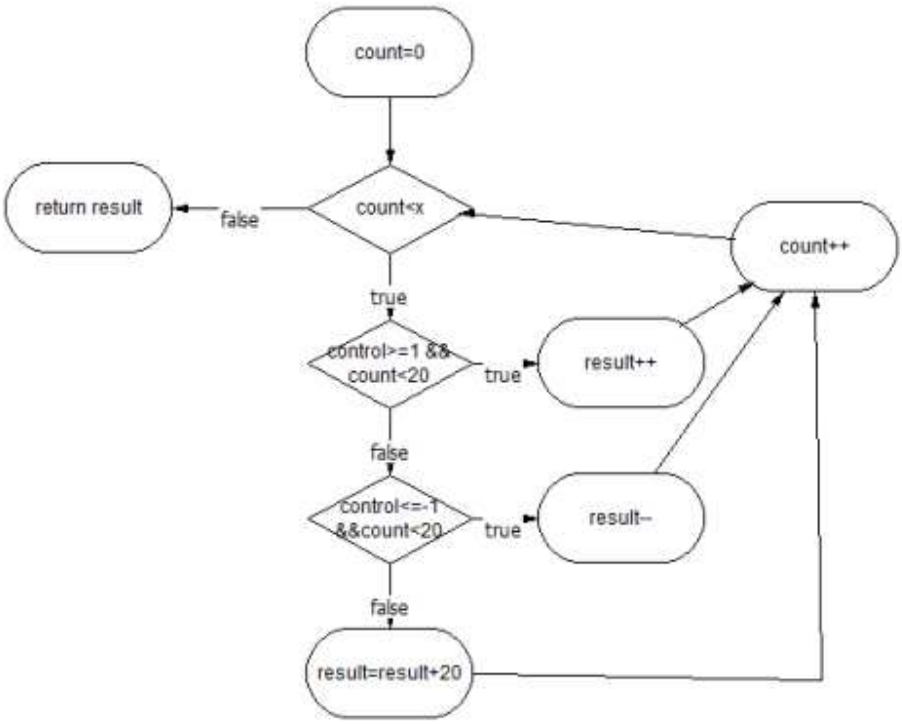
alarm(10, 20, 10, 30) → 0
alarm(10, 20, 10, 20) → 1

hour	min	alarmHour	alarmMin	alarm	valid	Test case	Boundary
[minint,0[[0, 23] [24,maxint]	[minint,0[[0, 59] [60,maxint]	[minint,0[[0, 23] [24,maxint]	[minint,0[[0, 59] [60,maxint]	Y, N	V, I		
[minint,0[	-	-	-	-	I	T(-5,1,1,1,), err	T(minint,1,1,1) T(0,1,1,1)
[0, 23]	[minint,0[	-	-	-	I	T(10,-5,1,1) err	
	[0, 59]	[minint,0[	-	-	I		
		[0, 23]	[minint,0[	-	I		
			[0, 59]	Y	V	T(1,1,1,1), 1	
				N	V	T(1,1,2,1), 0	
			[60,maxint]	-	I		
		[24,maxint]	-	-	I		
	[60,maxint]	-	-	-	I		
[24,maxint]	-	-	-	-	I		
Parameter not an int				-	I		
Wrong number of parameters				-	I		

Five conditions are defined (range of hours and minutes, alarm should ring or not)
Number of classes: 3x3x3x3x2 = 162 - It is important to prune some combinations
Only test cases for some classes are reported
It is important to distinguish error conditions (invalid classes) from the others (in other words a test case with output zero is different from a test case with output error)
More classes could be defined for the ‘normal’ case, using other than the Boolean condition ‘alarm should ring or not’. For instance, alarmMinute <, = ,> minute, and the same for alarmHour

3 (6 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 int function (int x, int control, int result){
2   int count;
3   for (count = 0; count < x; count++ ){
4      if (control >= 1 && count < 20) {
5          result++;
6      }
7      else if (control <= -1 && count < 20){
8          result--;
9        } else { result= result +20;}
10   }
11   return result;
12 }
```



Coverage type	Feasibility (Y/N)	Coverage obtained (%)	Test cases
Node	Y, 2 or 3 test cases	100%	T1(1,1,10) T2(1, -1, 10) T3(1, 0, 10)
Edge	Y, 2 or 3 test cases	100%	Same as node coverage
Multiple condition	Y, 4 test cases	100%	T1, T2

(line 7)			T3(21,1,10) T4(21, -1, 10)
Loop	Y, 3 test cases	100%	T1, T3 T5(0, ..)
Path	It depends on variable x, so in general is Npaths <= 3^x In fact not all paths are feasible because of the *control* variable Npaths = 3x So the growth is not exponential but 100% coverage is not feasible anyway		

4 (1 points) – Give a definition of configuration management

5 (1 points) – Why configuration management is needed?

To keep a software system (made of many different parts, or CI) consistent over time and changes

6 (1 points) – Give an example of non functional requirement

7 (1 points) – What is the difference between fault and failure?