Student Name: Khaled Banjaki
Student #: 101058264
Course: COMP3004

FINAL EXAM

PART 1 – The Therac-25: 30 Years Later

a. Can we say that software by itself is safe or not?

As stated in the article; “Software by itself isn’t safe or unsafe --safety depends on context.” All in all, software entirely depends on the way it is used. It is said that software could be potentially safe only if all of it’s requirements are met. But this is obviously not true, since most software errors mostly happen due to poor implementations rather than the software requirements. Meaning it could be safe depending on how we handle the requirements and implementations.

b. At what phase of software development does safety first come into play?

Security comes into play in the software design phase, that is, the creation of the software model itself, as well as the model between the components contained within the software itself. In most software, before, the main "protection" was based on an assessment of the likelihood of risk. Often, this approach has led to severe malfunctions and accidents in the operation of the existence for which the software was responsible (an example of such an accident in The Therac-25 software. An example of an accident is the incident at the Chernobyl nuclear power plant, present-day Ukraine). The risk likelihood assessment convinced users, ie. operators, that the error would not occur and did not involve the operation of the system in the most extreme (most difficult) conditions. Instead of this approach, security and security today are based on specific data collected in the previous software operating period, which is a good basis for starting to configure security and security in today's software.

c. Is it safer to reuse software or build from scratch?

If a software error occurs that results in unwanted system behavior, it is safer, but also more expensive, to build new software. If we only implement a patch on a particular problematic part of the software, we will solve the problem immediately. However, this does not guarantee us that in the coming period the malfunction will not be repeated or even spread to other pieces of software. It is also possible that an error that has occurred may cause errors to occur in another piece of software that previously worked. Overall, when it comes to software that delivers a critical system, the recommendation is to rebuild the software from scratch, but this time to model and program to meet all the requirements of good software such as integrity, security, sustainability, efficiency, and more.

d. Does using object-oriented technology lead to safer software?

Object-oriented design is very good for data-oriented systems but the opposite for control-oriented systems. Using this technology it would make software more complex for testing when evaluating safety measures and make sure we meet the correct changes when dealing with critical safety requirements.

e. Is it better, from the point of view of safety, to first implement normal and second error-handling behavior, or first error-handling and then normal behavior?

It would save more time and money if we take care of error handling first. Dealing with error handling at the beginning would save more time and money when working on a business project. This is key when designing ways to prevent or even detect software errors before the normal behaviour. Type-checking in programming languages isn't very popular but provides lots of protection in systems.

PART 2 – Elevator installation process modeling in ArchiMate

ELEVATOR INSTALLATION PROCESS

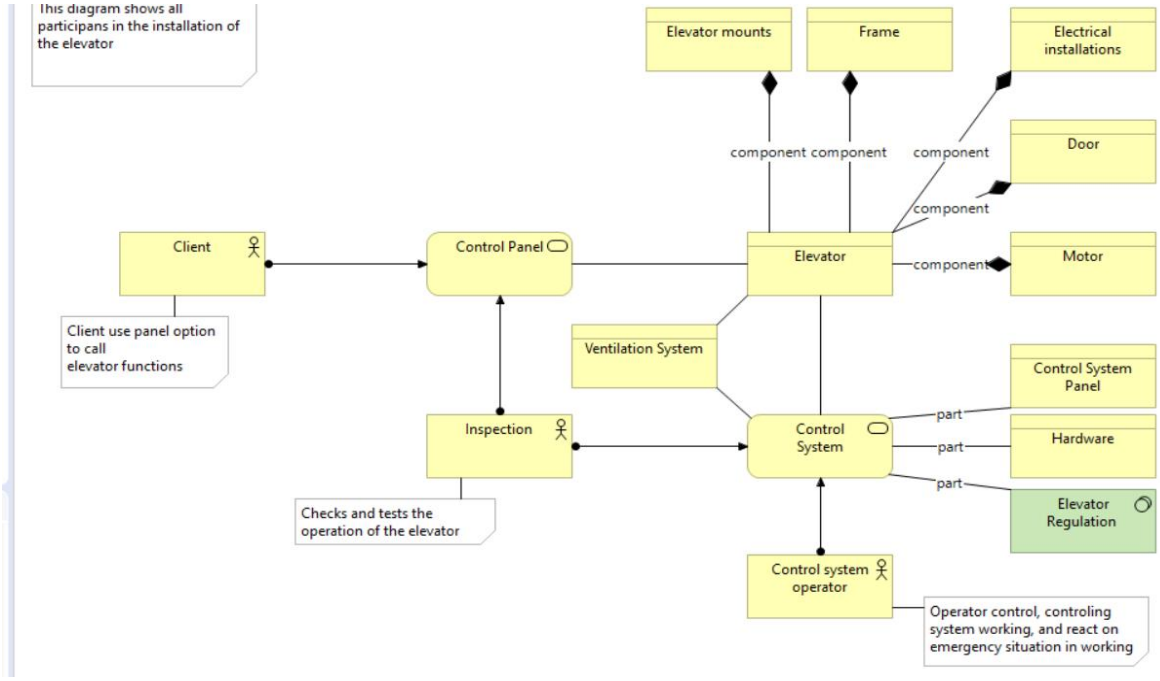
The elevator system itself consists of multiple subsystems that must be in perfect collision and in harmony with each other. Each of the subsystems has its parts and functionalities whose measure of functioning depends on the measure of functioning of the other subsystems. Throughout the history of construction, various elevators have been constructed that have had different modes of operation, as viewed by the use of propulsion systems, alarm systems and other functionalities. The elevator under consideration is an elevator in use today and is called "**Island Elevator**".

The installation takes place in several stages, which are listed below:
1) First of all, it is necessary to build a structure within the construction that will be able to withstand the elevator system in its durability and durability performance. By this we mean a stable construction of an object (building) in which the said system is to be made.
2) The installation of the elevator itself begins with the installation of wall mounting brackets. It is important that the handles are of solid material (titanium or iron) to support the weight of the elevator cabin itself. The installation itself should be accurate in the limiter, so as not to cause frequent malfunction of the elevator itself or in the worst case of elevator failure and injury to the elevator user.
3) A hydraulic motor is then installed, the task of which is to produce energy that will drive the elevator itself. The engine operates on the principle of oil movement under the primitive oil, which is an outflow of hydraulic engines.
4) The construction of the elevator, i.e its frame, is then installed. The material used to build the frame itself should be solid, durable, long lasting and resistant to all the forces that occur in the work. It should also provide full security for the user. The platform also includes the platform on which users stand.
5) The next phase is to install the hardware. This implies equipment that will serve as an elevator control subsystem. The hardware itself consists of many wires, cables and accessories to communicate with the elevator. The idea is that clicking on one of the options in the elevator will send instructions to the control system, while he will respond by executing the instruction

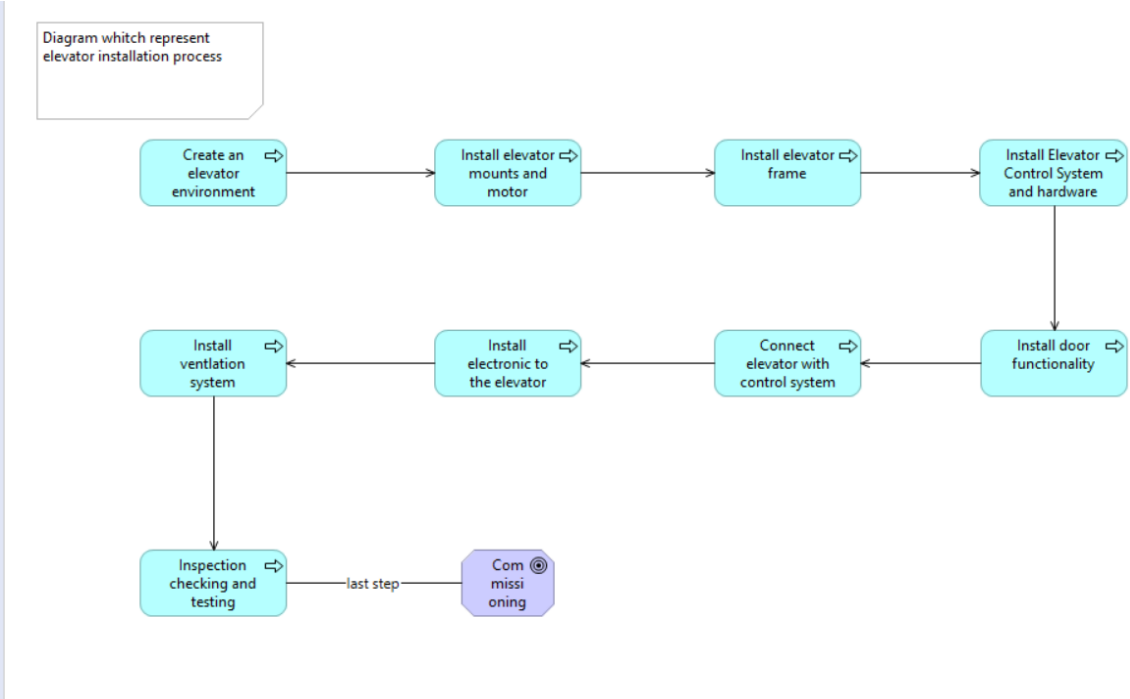
given by the user. One of the most important goals of a management system is the ability to respond promptly and promptly to instructions. This increases the safety and security of the user.
6) Sliding doors are then installed, which physically connect to the elevator frame on the one hand, and software to the control system.
7) The next phase is the addition of electronics to the elevator system itself. This includes the addition of a panel with options for using the elevator, as well as the addition of a special panel used in emergency shots. At this stage, we have fully connected the machine part of the elevator system with the control system, training all the options and possibilities that the elevator has to offer, as well as the introduction of sensors (sensor door, alarm sensor, fire sensor, wear list of system parts, ...).
8) The last stage of construction is the addition of a ventilation system, which increases the safety and security of use.
9) At the end comes a special inspection to check and test the operation of the elevator and if the elevator meets the standard prescribed, it is put into operation. If the elevator has not been tested, the system must be repaired and completed in accordance with the inspection recommendations and procedures defined in the standard. A special service is responsible for maintaining the elevator.

Each of these stages is equally important in the installation of the system. Failure to do one of the phases as prescribed or with some omissions may result in serious material and human loss. Therefore, special emphasis in the installation of this system is on the safety, security, reliability and proper operation of the system. The conclusion is that it is much better and more profitable to invest in the safety and security of the system during the construction and installation itself, than the work, when a failure occurs.

Diagrams describing the installation participants/model/contributors for Part 2:

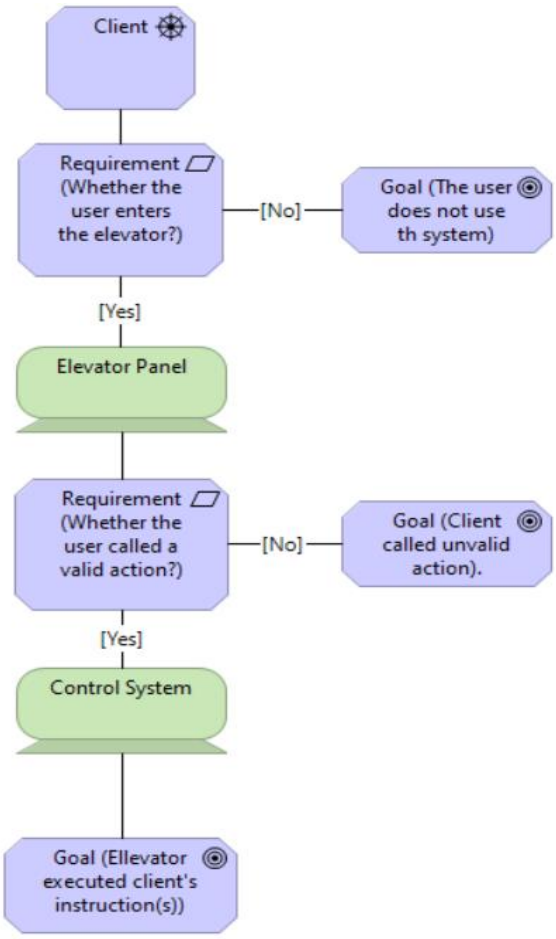


The above slice diagram shows the process of installation of all participants in building the elevator system. How clients interact with the control panel and how the inspector checks and tests the operation of the elevator. There is also a Control System operator which checks if everything is in working order.



This installation process (**Activity** diagram) diagram represents the elevator installation process. How the elevator is built from scratch, the steps needed from creating the environment for the elevator to being used by clients. It models the logic of complex logic within the elevator system.

This diagram shows all action elevator working



The work process diagram depicts one way to model the logic of the elevator system. How the elevator system works when a client interacts with it and how the control system reacts.

Part 3: Elevator Control System

1. **Use Cases**

Use Case 1: Door Open

Primary Actor: Client
Scope: Island Elevator
Level: User Goal
Prerequisite: Navigate the buttons panel

1. User interacts with the operator buttons
2. Button illuminates until the elevator has arrived
3. Elevator door and floor doors open for 10 seconds signaling the User to enter the elevator

Use Case 2: Door close

Primary Actor: Client
Scope: Island Elevator
Level: User Goal
Prerequisite: Navigate the buttons panel

1. User waits in the elevator
2. User is displayed with current floor number visually
3. User could manually close the doors or wait 10 seconds for them to close
4. Bell rings signaling User the elevator doors will now close
5. User proceeds to another floor after selecting desired floor number

Use Case 3: Level Up

Primary Actor: Client
Scope: Island Elevator
Level: User Goal
Prerequisite: Navigate the buttons panel

1. User selects a floor number above their current floor
2. User is signaled by operator that they have reached their desired floor
3. User is displayed current floor number visually and audio messages
4. User exits elevator

Use Case 4: Level Down

Primary Actor: Client
Scope: Island Elevator
Level: User Goal
Prerequisite: Navigate the buttons panel

1. User selects a floor number below their current floor
2. User is signaled by operator that they have reached their desired floor
3. User is displayed current floor number visually and audio messages
4. User exits elevator

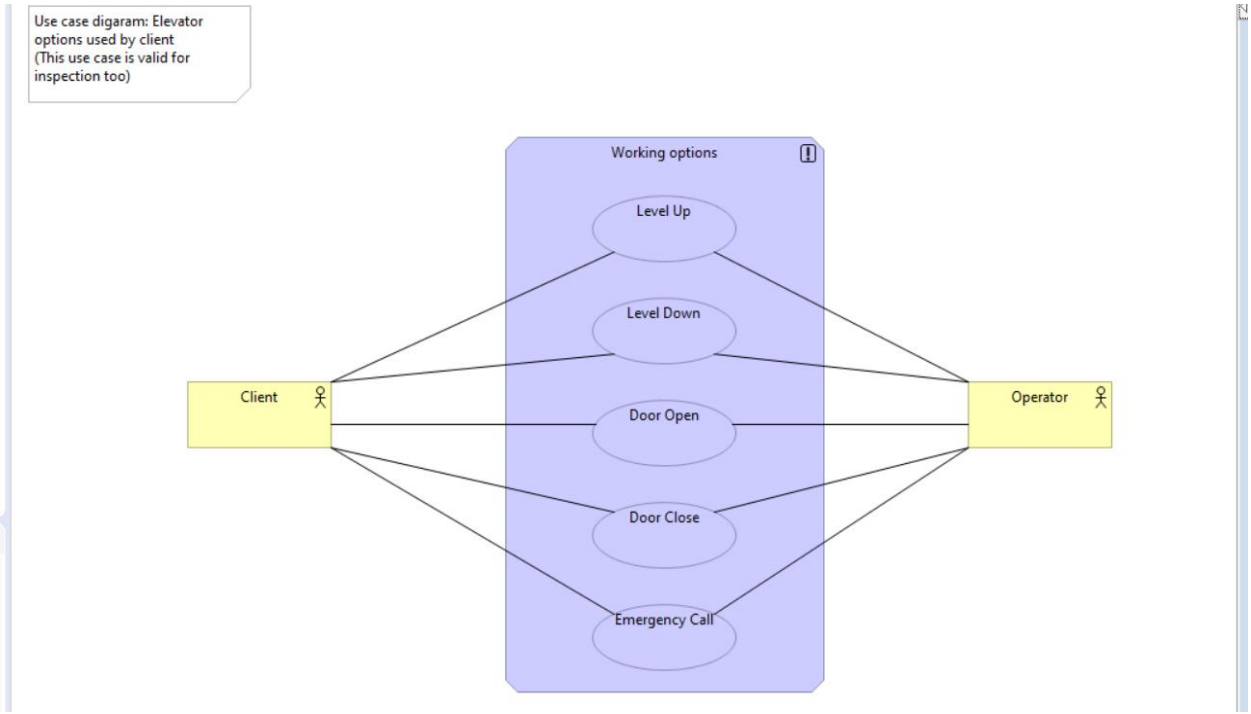
Use Case 5: Emergency Call

Primary Actor: Client
Scope: Island Elevator
Level: User Goal
Prerequisite: Navigate the buttons panel

1. User is currently in the elevator
2. User navigates button panel and selects one of 5 emergency buttons
3. User receives Audio and visual messages based on the selected safety feature
4. User exits elevator once help has arrived

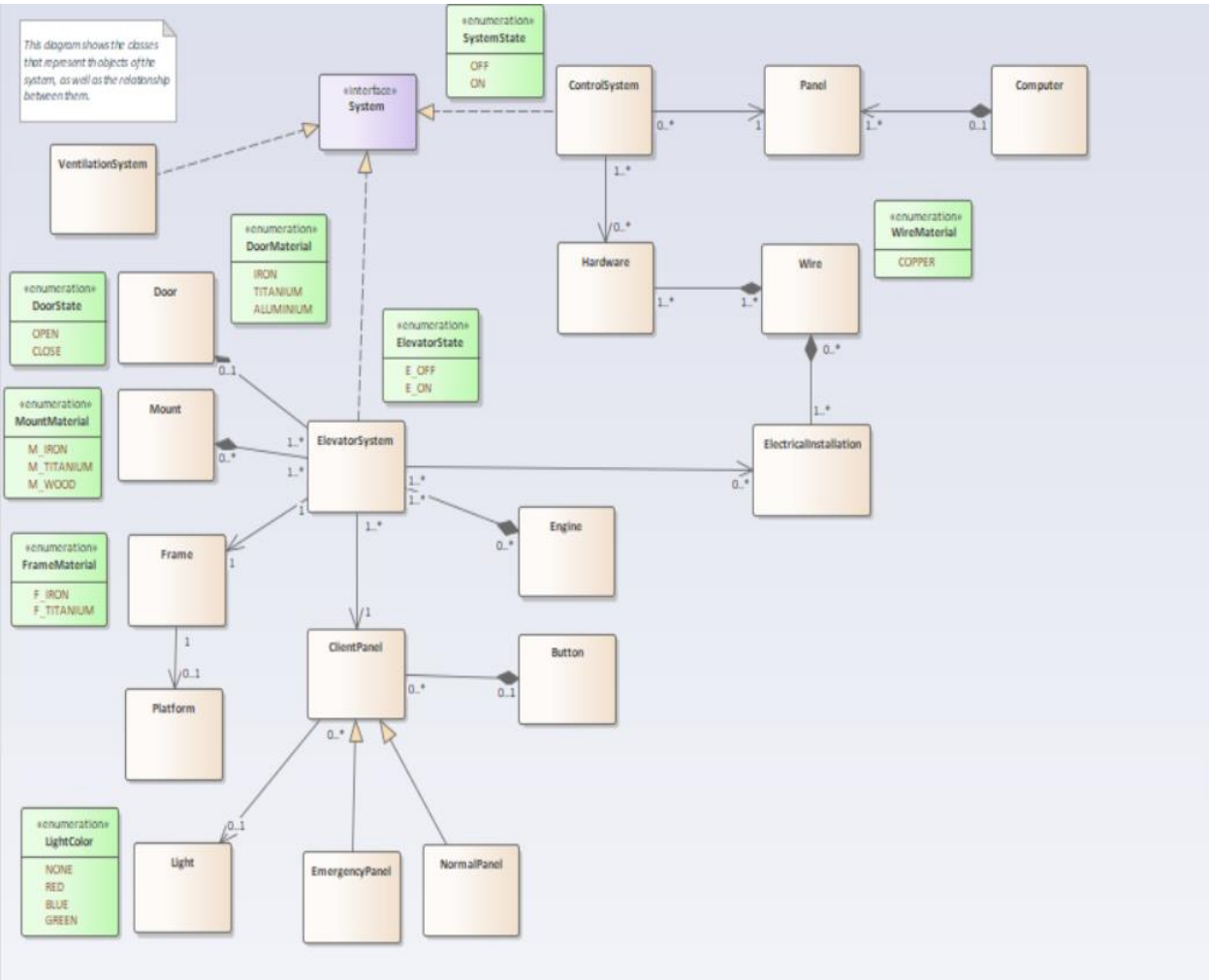
Use case Diagram:

2. Actor/Use cases

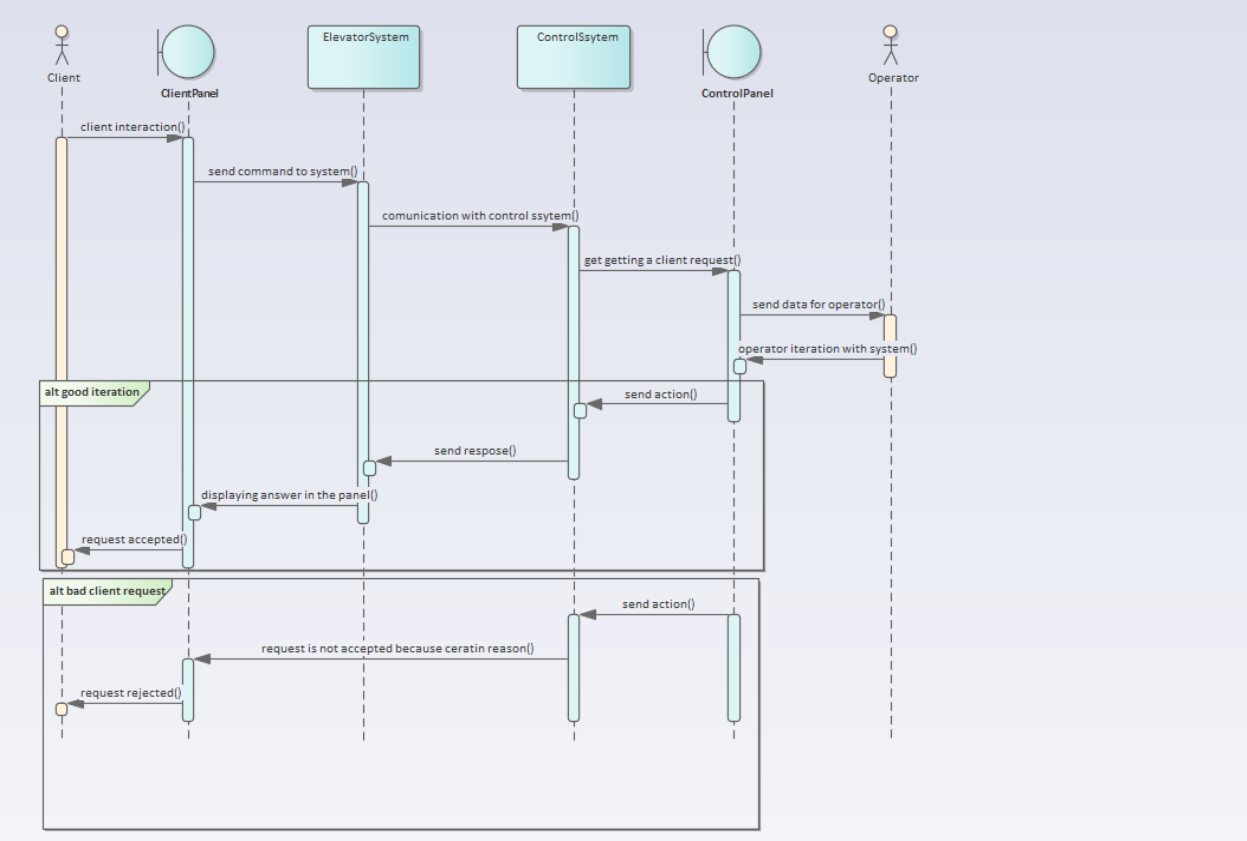


This use case diagram depicts the relationship between the actors, use cases and their interrelationships. How the client interacts with a function of the elevator and how the operator controls it. Both the user and operator are actors in this scenario, they both interact with the same functions, this is how and why the elevator functions appropriately.

3. Control System diagrams



The above Class Design Diagram shows a collection of skeleton classes that describe the relationship between these objects for the elevator system. **This diagram was designed with empty classes since it is a simple skeleton for an elevator system. It serves as a basis to further develop a more complex elevator system.



This sequence diagram models the sequential logic of the elevator system. It depicts the state of a certain functionality and the ordering of messages between classifiers. How an elevator operates from beginning to finish.

Files included for a potential elevator system:

Button.h
ClientPanel.h
Computer.h
ControlSystem.h
Door.h
ElectricalInstallation.h
ElevatorSystem.h
EmergencyPanel.h
Engine.h
Frame.h
Hardware.h
ISystem.h
Light.h
main.cpp
Mount.h
NormalPanel.h
Panel.h
Platform.h
VentilationSystem.h
Wire.h

Skeleton Code Below

```
#ifndef DOOR_H_INCLUDED
#define DOOR_H_INCLUDED
using namespace std;

enum DoorMaterial {IRON, TITANIUM, ALUMINIUM};
enum DoorState {OPEN, CLOSE};

//Door Class
class Door
{
   private:
      double height;
      double width;
      DoorMaterial material;
      bool haveSensor;
      DoorState currentState;

   public:
      Door();
      Door(double h, double w, DoorMaterial mat, bool hs, DoorState ds);
      void setHeight(double h);
      double getHeight();
      void setWidth(double w);
      double getWidth();
                  //sensors
      void setHaveSensor(double hs);
      bool getHaveSensor();
      void setMaterial(DoorMaterial m);
      DoorMaterial getMaterial();
                  //state of door
      void setCurrentState(DoorState cs);
      DoorState getCurrentState();
      void openDoor();
      void closeDoor();
};
#endif // DOOR_H_INCLUDED
```

```
#ifndef ELECTRICALINSTALLATION_H_INCLUDED
#define ELECTRICALINSTALLATION_H_INCLUDED
#include "Wire.h"
#include <list>

//ElectricalInstallation Class
class ElectricalInstallation
{
   private:
      list<Wire> wires;
```

```
   public:
      ElectricalInstallation();
      ElectricalInstallation(list <Wire> w);
      void setWires(list<Wire> w);
      list<Wire> getWires();
};
#endif // ELECTRICALINSTALLATION_H_INCLUDED
```

```
#ifndef ELEVATORSYSTEM_H_INCLUDED
#define ELEVATORSYSTEM_H_INCLUDED
#include "Door.h"
#include "Engine.h"
#include "Mount.h"
#include "Frame.h"
#include "ClientPanel.h"
#include <list>
#include "ElectricalInstallation.h"

enum ElevatorState {E_OFF, E_ON};

//ElevatorSystem Class
class ElevatorSystem: public ISystem
{
   private:
      ElevatorState state;
      list<Door> doors;
      list<Engine> engines;
      list<Mount> mounts;
      Frame frame;
      ClientPanel panel;
      ElectricalInstallation installation;

   public:
      ElevatorSystem();
      ElevatorSystem(ElevatorState s, list<Door> d, list<Engine> e, list<Mount> m, Frame f, ClientPanel p,
ElectricalInstallation ei);
      void setState(ElevatorState s);
      void setDoors(list<Door> d);
      void setEngines(list<Engine> e);
      void setInstallation(ElectricalInstallation i);
      void setMounts(list<Mount> m);
      void setFrame(Frame f);
      void setClientPanel(ClientPanel cp);
      ElevatorState getState();
      list<Door> getDoors();
      list<Engine> getEngines();
```

```
    ElectricalInstallation getInstallation();
    list<Mount> getMounts();
    Frame getFrame();
    ClientPanel getClientPanel();
    virtual void onSystem()
    virtual void offSystem();
};
#endif // ELEVATORSYSTEM_H_INCLUDED
```

```
#ifndef EMERGENCYPANEL_H_INCLUDED
#define EMERGENCYPANEL_H_INCLUDED
#include "ClientPanel.h"
#include <list>

//EmergencyPanel Class
class EmergencyPanel: public ClientPanel
{
  public:
    EmergencyPanel();
    EmergencyPanel(list<Button> b, Light l) : ClientPanel(b, l);
};
#endif // EMERGENCYPANEL_H_INCLUDED
```

```
#ifndef ENGINE_H_INCLUDED
#define ENGINE_H_INCLUDED

//Engine Class
class Engine
{
  private:
    string name;
    double literOfOil;
    double enginePower;
                double capacity;

  public:
    Engine();
    Engine(string n, double l, double e);
    void setName(string n);
    string getName();
    void setLiterOfOil(double liter);
    double getLiterOfOil();
    void setEnginePower(double e);
    double getEnginePower();
    void consuptionOil();
    void stopWorking();
    void addOil();
```

```
};
#endif // ENGINE_H_INCLUDED
```

```
#ifndef FRAME_H_INCLUDED
#define FRAME_H_INCLUDED
#include "Platform.h"

enum FrameMaterial {F_IRON, F_TITANIUM};

//Frame Class
class Frame
{
   private:
      double width;
      double height;
      double length;
      FrameMaterial material;
      Platform platform;
   public:
      Frame();
      Frame(double w, double h, double l, FrameMaterial m, Platform p);
      void setWidth(double w);
      void setHeight(double h);
      void setLength(double l);
      void setMaterial(FrameMaterial m);
      void setPlatform(Platform p);
      double getWidth();
      double getHeight();
      double getLength();
      FrameMaterial getMaterial();
      Platform getPlatform();
};
#endif // FRAME_H_INCLUDED
```

```
#ifndef CONTROLSYSTEM_H_INCLUDED
#define CONTROLSYSTEM_H_INCLUDED
#include "Hardware.h"
#include "Panel.h"

enum SystemState {OFF, ON};

//ControlSystem Class
class ControlSystem : public ISystem
{
   private:
      Hardware hardware;
      Panel panel;
      SystemState state;
```

```
   public:
      ControlSystem();
      ControlSystem(Hardware h, Panel p, SystemState s);
      void setHardware(Hardware h);
      void setPanel(Panel p);
      void setState(SystemState s);
      Hardware getHardware();
      Panel getPanel();
      SystemState getState();
      virtual void onSystem();
      virtual void offSystem();
};
#endif // CONTROLSYSTEM_H_INCLUDED
```

```
#ifndef COMPUTER_H_INCLUDED
#define COMPUTER_H_INCLUDED

//Computer class
class Computer
{
   private:
      string cpu;
      double memoryGB;
      string monitorMark;

   public:
      Computer();
      Computer(string c, double m, string mm);
      void setCpu(string c);
      void setMemoryGB(double m);
      void setMonitormark(string mm);
      string getCpu();
      double getMemoryGB();
      string getMonitorMark();
};
#endif // COMPUTER_H_INCLUDED
```

```
#ifndef CLIENTPANEL_H_INCLUDED
#define CLIENTPANEL_H_INCLUDED
#include "Button.h"
#include "Light.h"
#include <list>

//ClientPanel Class
class ClientPanel
{
   private:
```

```
    list<Button> buttons;
    Light light;

  public:
    ClientPanel();
    ClientPanel(list<Button> b, Light l);
    void setButtons(list<Button> b);
    void setLight(Light l);
    list<Button> getButtons();
    Light getLight();
};
#endif // CLIENTPANEL_H_INCLUDED
```

```
#ifndef BUTTON_H_INCLUDED
#define BUTTON_H_INCLUDED

//Button Class
class Button
{
  private:
    int buttonNumber;
    bool isPress;

  public:
    Button(){}
    Button(int bn);
    void setButtonNumber(int bn);
    void setIsPress(bool isP);
    int getButtonNumber();
    bool getIsPress();
    void pressButton();
    void unPressButton();
};
#endif // BUTTON_H_INCLUDED
```

```
#ifndef WIRE_H_INCLUDED
#define WIRE_H_INCLUDED

enum WireMaterial{COPPER};

//Wire class
class Wire
{
  private:
    string id;
    WireMaterial material;
  public:
    Wire();
```

```
        Wire(string i, WireMaterial m);
        void setId(string i);
        string getId();
        void setMaterial(WireMaterial wm);
        WireMaterial getMaterial();
};
#endif // WIRE_H_INCLUDED
```

```
#ifndef VENTILATIONSYSTEM_H_INCLUDED
#define VENTILATIONSYSTEM_H_INCLUDED
#include "ISystem.h"

//VentilationSystem Class
class VentilationSystem : public ISystem
{
   private:
      double permabilityAir;
      double permabilityWater;
      bool stateAir;
      bool stateWater;

   public:
      VentilationSystem();
      VentilationSystem(double a, double w, bool sa, bool sw);
      void setAir(double pa);
      void setWater(double pw);
      void setStateAir(bool sa);
      void setStateWater(bool sw);
      double getAir();
      double getWater();
      bool getStateAir();
      bool getStateWater();
      virtual void onSystem();
      virtual void offSystem();
};
#endif // VENTILATIONSYSTEM_H_INCLUDED
```

```
#ifndef PLATFORM_H_INCLUDED
#define PLATFORM_H_INCLUDED

//Platform class
class Platform
{
   private:
      double height;
      double width;
```

```
   public:
      Platform(); //defautl constructor
      Platform(double h, double w);
      void setHeight(double h);
      double getHeight();
      void setWidth(double w);
      double getWidth();
};
#endif // PLATFORM_H_INCLUDED
```

```
#ifndef PANEL_H_INCLUDED
#define PANEL_H_INCLUDED

#include "Computer.h"
#include <list>

//Panel Class
class Panel
{
   private:
      list<Computer> computers; //panel will include computers

   public:
      Panel();
      Panel(list<Computer> c);

      void setPanel(list<Computer> c);
      list<Computer> getPanel(); //return the list of cps
};
#endif // PANEL_H_INCLUDED
```

```
#ifndef NORMALPANEL_H_INCLUDED
#define NORMALPANEL_H_INCLUDED
#include "ClientPanel.h"
#include <list>

//NormalPanel class
class NormalPanel: public ClientPanel
{
   public:
      NormalPanel(); //default constructor
      NormalPanel(list<Button> b, Light l) : ClientPanel(b, l);
};
#endif // NORMALPANEL_H_INCLUDED
```

```
#ifndef MOUNT_H_INCLUDED
#define MOUNT_H_INCLUDED
```

```
enum MountMaterial {M_IRON, M_TITANIUM, M_WOOD};

//Mount Class
class Mount
{
   private:
      MountMaterial material;
   public:
      Mount();
      Mount(MountMaterial m);
      void setMaterial(MountMaterial m);
      MountMaterial getMaterial();
};
#endif // MOUNT_H_INCLUDED
```

```
#ifndef LIGHT_H_INCLUDED
#define LIGHT_H_INCLUDED
enum LightColor {NONE, RED, BLUE, GREEN}; //red for alarm, green for up, blue for down

//Light Class
class Light
{
   private:
      bool isOn;
      LightColor light;

   public:
      Light();
      Light(bool i, LightColor lc);
      void setIsOn(bool i);
      void setLight(LightColor cl);
      bool getIsOn();
      LightColor getLight();
      void on();
      void off();
};
#endif // LIGHT_H_INCLUDED
```

```
#ifndef ISYSTEM_H_INCLUDED
#define ISYSTEM_H_INCLUDED

//ISystem Class
class ISystem
{
   public:
      virtual void onSystem();
      virtual void offSystem();
```

```
};
#endif // ISYSTEM_H_INCLUDED
```

```
#ifndef HARDWARE_H_INCLUDED
#define HARDWARE_H_INCLUDED
#include <list>
//Hardware class
class Hardware
{
   private:
      list<Wire> wires;
      int numberOfElements;
   public:
      Hardware();
      Hardware(list<Wire> w, int n);
      void setWires(list<Wire> w);
      void SetNumberOfElements(int n);
      list<Wire> getWires();
      int getNumberOfElements();
};
#endif // HARDWARE_H_INCLUDED
```

```
#include <iostream>
#include "Door.h"
#include "Engine.h"
#include "Wire.h"
#include "ElectricalInstallation.h"
#include "Platform.h"
#include "Mount.h"
#include "Frame.h"
#include "Hardware.h"
#include "Computer.h"
#include "Panel.h"
#include "Button.h"
#include "Light.h"
#include "ClientPanel.h"
#include "NormalPanel.h"
#include "EmergencyPanel.h"
#include "VentilationSystem.h"
#include "ControlSystem.h"
#include "ElevatorSystem.h"
using namespace std;
//Program Entry Point
int main()
{
 //empty
}
// EOFs
```