Arab Open University- Lebanon

Faculty of Computer Studies
Information Technology and Computing Department

Elevator Simulation

Ibrahim Deryan (7551); Mostafa Akdanez (7288)
T471: Final Year Project, May 2010

Supervisor:
Mr. Fadi Fayyad
Abstract

Starting from a hand powered elevator till reaching to what we are dealing with today of systematically complex one’s, clarify to us that technological development don’t stop, and whenever time is passing, new developments will be available to us helping to solve every problem we face.

Simulation is a way of developing application or design simulating real world, the way which help in clarifying the real idea. The reasons upon using simulations may be for literature, cost, time, or safety reasons.

The elevator challenge was born out of a long time curiosity in the control of elevators. Elevator simulation is becoming increasingly more flexible and powerful. Elevator simulation models of varying sophistication have been written and applied for many years. The continuing improvements in computer technology and software development tools make increasing complex and comprehensive simulation models feasible.
Acknowledgment

First and Foremost, Special thanks to our University “Arab Open University” That helps and stands beside us in our four years trip... And then our thanks and appreciation goes to our tutor Mr. Fadi Fayyad for his guidance, advice and encouragement toward the successful completion of this project. Additional thanks go to the “ITC department” in AOU, faculty and students, and everyone involved in this project, for their help in making information technology an interesting and knowledgeable learning experience throughout our studies.
Contents

CHAPTER 1: Introduction .............................................. 6
  1.1. History......................................................... 6
  1.2. General approach ........................................... 7

CHAPTER 2: Specification ............................................ 9
  2.1. Problem specification and features.......................... 9
  2.2. Limitations.................................................... 9

CHAPTER 3: Analysis ................................................... 11
  3.1. Problem Domain............................................... 11
      3.1.1. Structure.................................................. 11
      3.1.2. System Events............................................ 11
      3.1.3. Classes.................................................... 12
  3.2. Application Domain.......................................... 12
      3.2.1. Actors..................................................... 12
      3.2.2. Functions............................................... 13

CHAPTER 4: Design ..................................................... 15
  4.1. UML: A brief introduction.................................. 15
  4.2. UML Overview............................................... 16
  4.3. Modeling the static aspects of the system................. 16
      4.3.1. Use Case Diagram........................................ 16
      4.3.2. Class Diagram............................................ 18
  4.4. Modeling the dynamic aspects of the system.............. 20
      4.4.1. Sequence Diagram........................................ 20
      4.4.2. State Chart Diagram.................................... 22

CHAPTER 5: Implementation ........................................ 25
  5.1. Technical Platform.......................................... 25
      5.1.1. The NetBeans Platform and IDE....................... 25
      5.1.2. Design Language......................................... 26
  5.2. Implementation.............................................. 27
5.3. System Interface.........................................................28

CHAPTER 6: Testing and Reviewing.................................32
6.1. Test Cases.................................................................32
6.2. Performing Test.........................................................33

CHAPTER 7: Conclusion..................................................35
7.1. Summary .................................................................35
7.2. Future work perspectives............................................35

CHAPTER 8: References..................................................36
8.1. References.................................................................36
8.2. Appendices...............................................................36
CHAPTER 1: Introduction

1.1. History

The first reference to an elevator is in the works of the Roman architect Vitruvius, who reported that Archimedes built his first elevator probably in 236 B.C. In some literary sources of later historical periods, elevators were mentioned as cabs on a hemp rope and powered by hand or by animals. It is supposed that elevators of this type were installed in the Sinai monastery of Egypt.

In 1000, the Book of Secrets by the Arab Ibn Khalaf al-Muradi in Islamic Spain described the use of an elevator-like lifting device, in order to raise a large battering ram to destroy a fortress. In the 17th century the prototypes of elevators were located in the palace buildings of England and France.

In 1793 Ivan Kulibin created an elevator with the screw lifting mechanism for the Winter Palace of Saint Petersburg. In 1816 an elevator was established in the main building of sub Moscow village called Arkhangelskoye. In 1823, an "ascending room" made its debut in London.

In 1852, Elisha Otis introduced the safety elevator, which prevented the fall of the cab if the cable broke. The design of the Otis safety elevator is somewhat similar to one type still used
today. A governor device engages knurled roller(s); locking the elevator to its guides should the elevator descend at excessive speed. He demonstrated it at the New York exposition in the Crystal Palace in 1854.

On March 23, 1857 the first Otis passenger elevator was installed at 488 Broadway in New York City. The first elevator shaft preceded the first elevator by four years. Construction for Peter Cooper's Cooper Union building in New York began in 1853. An elevator shaft was included in the design for Cooper Union, because Cooper was confident that a safe passenger elevator would soon be invented.\[6] The shaft was cylindrical because Cooper felt it was the most efficient design.

The first electric elevator was built by Werner von Siemens in 1880.\[5] The safety and speed of electric elevators were significantly enhanced by Frank Sprague.

The development of elevators was led by the need for movement of raw materials including coal and lumber from hillsides. The technology developed by these industries and the introduction of steel beam construction worked together to provide the passenger and freight elevators in use today.

In 1874, J.W. Meaker patented a method which permitted elevator doors to open and close safely.

In 1882, when hydraulic power was a well established technology, a company later named the London Hydraulic Power Company was formed. It constructed a network of high pressure mains on both sides of the Thames which, ultimately, extended to 184 miles and powered some 8,000 machines, predominantly lifts (elevators) and cranes.\[6]

In 1929, Clarence Conrad Crispen, with Inclinator Company of America, created the first residential elevator.

1.2. General approach

The company problem appears when it is intended to equip its building with an elevator and want to know according to its needs the way to settle the elevator to best satisfy this needs. Where this company my need from 1 to 6 elevators and the building will be of 10 floors (not including ground floor). The way where an Object-Oriented Software Simulator is to be installed, concerning the logic required to move the elevator between floors, giving the company a clear idea and leading it to choose the best design which satisfy this needs.

An elevator (or lift) is a vertical transport vehicle that efficiently moves people or goods between floors of a building. They are generally powered by electric motors that either drive traction cables and counterweight systems, or pump hydraulic fluid to raise a cylindrical piston.
GUI Interface starts by choosing the number of elevators the user want to see their simulation starting from 1 to 6 elevators; This GUI will be divided in to 2 modules:

- Building; showing the whole Building with its floors and the interaction between elevator(s) them selves and their interaction with the floors when requests are applied.

- Elevator button panel; Showing the buttons inside the elevator cabin when request is applied and its illumination when pressed, in addition to the close door button which take the request to the visited floor.

Where the company can test all what it want including the elevator cabin and floors. Reaching to have a clear idea about what it needs to settle in its building.
CHAPTER 2: Specification

2.1. Problem specification and features

The problem concerns the logic required to move elevators between floors according to the following constraints:

- Having a unique control system, which control the elevators and pick up the requests.
- Each elevator has a set of m buttons, one for each floor. These illuminate when pressed and cause the elevator to visit the corresponding floor. The illumination is canceled when the elevator visits the corresponding floor.
- Each floor, except the first floor and top floor has two buttons, one to request an up-elevator and one to request a down-elevator. These buttons illuminate when pressed. The illumination is canceled when an elevator visits the floor and then moves in the desired direction.
- Continue traveling in the same direction while there are remaining requests in that same direction.
- When an elevator has no requests, it remains at its current floor with its doors closed.

The elevator algorithm is implemented in computer operating systems as an algorithm for scheduling hard disk requests.

2.2. Limitations

Because we are dealing with a simulation meaning that not all features of the real world can be implemented, these limitations can be summarized as follows:

- The door system includes several safety devices. Sensors detect passengers or objects in the door opening, preventing the continued closing of the doors. Older systems use mechanical "safety edges" which cause the doors to stop or retract when they make contact with a person or object. More modern systems use a large number of invisible light rays to detect people or objects in the doorway and reverse or stop the doors without having to make physical contact.
- Door operators contain devices which limit the amount of closing force. Newer systems are better able to keep the closing force consistent even under unusual conditions such as the "stack effect" which can cause heavy air movement in elevator shafts.
- To discourage the very dangerous practice of passengers trying to open the door of a stalled elevator, door restraints can allow normal operation of the door when the car is near the
floor level, but will restrict forcible movement of the door when the car is away from the floor.

- In most elevators, an emergency telephone or intercom can serve as a link to assistance in any emergency case.

- A special fire emergency system has been installed. It may be manually activated, or may respond to smoke sensors in the building. Exact operation varies by local codes, but generally such systems return the elevator to the main floor, open the doors to allow passengers to exit, and make the elevators available to emergency personnel.
CHAPTER 3: Analysis

3.1. Problem Domain

3.1.1. Structure

The problem concerns the logic required to move elevators between floors according to the following structure:

- Having a unique control system, which control the elevators and pick up the requests by using FIFO (First In First Out) vector or stack. Where requests enter the vector and the controller will start picking these requests.
- Each elevator has a set of m buttons, one for each floor. These illuminate when pressed and cause the elevator to visit the corresponding floor. The illumination is canceled when the elevator visits the corresponding floor.
- Each floor, except the first floor and top floor has two buttons, one to request an up-elevator and one to request a down-elevator. These buttons illuminate when pressed. The illumination is canceled when an elevator visits the floor and then moves in the desired direction.
- Continue traveling in the same direction while there are remaining requests in that same direction.
- When an elevator has no requests, it remains at its current floor with its doors closed.

3.1.2. System Events

The system events can be summarized as follows:

- **Process Elevator Calls**: These scenarios includes that the elevator receives calls from the passengers, turns on or turns off the light of elevator call buttons, updates the record of elevator calls stored in system controlling parts, etc.
- **Process Floor Calls**: Similar to Elevator Call processing, this use case includes that the elevator receives Floor calls from the passengers, turns on or turns off the light of Floor calls buttons, updates the record of floor calls in system controlling parts, etc.
- **Move/Stop elevator**: The main function of an elevator, how to make the decision of stop, and driving directions of the elevator.
- **Indicating Moving Direction**: The elevator should have this mechanism to let the passengers know the current moving direction of the elevator such that the passenger might decide whether to enter the elevator or not.
- **Indicating Elevator Position:** Similarly, the elevator should let the passengers know whether his/her destination floor is reached so that the passenger may decide to leave the elevator.

- **Open/Close the Doors:** The elevator should be able to open and close the doors for the passengers to get in and out of the elevator. The functional areas of this use case should also enable the passengers to make door reversals when the doors are closing and the passenger wants to get in the elevator.

- **On/Off illumination:** The elevator and floor buttons should be able inform the passenger that his call has been scheduled and inform him that his call has been requested, in which button illumination will be needed.

### 3.1.3. Classes

The main classes necessary for this problem will be as follows:

- **Controller:** The central controlling object in the elevator system. Controller communicates and controls all other objects in the system.

- **Door:** The Command the doors to open and close, according to the situation stated in the use case.

- **Elevator:** The Elevator is being controlled to move up and down, to make stops at floors when necessary.

- **State:** The indicators are controlled to show the information about the current position and moving direction of the elevator.

- **Button:** The Controller class also controls the button class, which further generalizes two subclasses Floor Button and Elevator Button. The control object communicates with the Button objects, get the information whether a button is pressed and in turn controls the illumination of Button lights.

### 3.2. Application Domain

#### 3.2.1. Actors

The only actor in elevator system is the passenger, which is the role that humans play when interacting with the system. The passenger interacts with the Elevator system by:

- Observing the state of elevator (position and direction)
- Making elevator calls
- Making floor calls
- Makes decision whether to enter/leave the elevator or not by observing the indication of moving direction and elevator position
- Makes decision to close door
3.2.2. Functions

The elevator algorithm will be based on the following functions:

- **Move elevator:**
  - void start()
  - Animates the elevator to the next stop based on commands issued from ElevatorButtonPanel:
    - void animateElevator()
- **Stop elevator:**
  - void stop()
  - Checks if there are anymore stop requests:
    - boolean moreStops()
- **Set the next stop location going UP:**
  - int getNextStopGoingUp()
- **Set the next stop location going Down:**
  - int getNextStopGoingDown()
- **Get the precise location:**
  - double getCurrentLevel()
- **Return if any more stop requests:**
  - boolean[] getStopRequests()
- **List of stops that have been requested by ElevatorButtonPanel:**
  - void setRequestedStops(int level)
- **Checks for floor requests if one was made (can be used anytime, even while animation):**
  - boolean pickUpRequests()
- **Sets for a pick up request:**
  - void setPickUpRequests(int level)
- **For the controller to set the moving direction of this elevator:**
  - void setMovingDirection(char a)
- **For the controller to get the moving direction of this elevator:**
  - char getMovingDirection()
- **For the ElevatorButtonPanel to give ok to include this in Controller's decision:**
  - void setAvailable()
- **For the Controller to see if this elevator is available to include in decision:**
  - boolean getAvailable()
- **Remove all queued requests and calls:**
  - void reset()
- **Change the state of an existing elevator:**
  - void updateStates()
- **Return a queued request:**
  - void retryQueuedRequests()
- **Decide which elevator will apply the queued request:**
  - int decide(int destination, char floorDir)
• Find the elevator with minimum cost to apply the queued request:
  ```c
  int findMinCostIndex(int destination, char floorDir)
  ```
• To set the new elevator position:
  ```c
  void setElevator(int elevatorNum, int pos, char[] jobs, char direction)
  ```
CHAPTER 4: Design

4.1. UML: A brief introduction

The Unified Modeling Language (UML) is the industry-standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as other non-software systems. UML simplifies the complex process of software design, making a "blueprint" for construction, and is now the standard notation for software architecture. UML provides both the structural views and behavioral views of the system. A set of diagrams with different graphical elements is the core part as well as the most expressive presentation in UML. The UML includes nine kinds of diagrams, for the sake of grasp the most representative aspects of the design of elevator system, in this paper only following UML diagrams are used and analyzed:

**Use Case diagram** shows a set of use cases and actors (a special kind of class) and their relationships. Use case diagrams address the static use case view of a system, these diagrams are important in organizing and modeling the behaviors of a system.

**Class diagram** shows a set of classes, interfaces, and collaborations and their relationships. Class diagrams are the most common diagrams used in modeling object-oriented systems. Class diagrams address the static design view of a system.

**Sequence diagram** is an interaction diagram. Interaction diagrams address the dynamic view of a system, besides sequence diagram, the other interaction diagram in UML is the **Collaboration diagram**. Sequence diagram emphasizes the time ordering of messages between objects in the system, while collaboration diagram emphasizes the structural organization of the objects that send and receive messages. Sequence diagrams and collaboration diagrams are isomorphic, and can be transformed from one into the other. Since either of them contributes to the same extend of understanding of our system, while sequence diagrams give more ideas of time, which is essential for real time systems, only the sequence diagrams are given in this report.

**State chart diagram** shows a state machine, consisting of states, transitions, events and activities. State chart diagrams address the dynamic view of a system. State chart diagrams are especially important in modeling the behavior of an interface, class, or collaboration and emphasize the event-ordered behavior of an object, which is especially useful in modeling reactive systems. The rest four kinds of UML diagrams are: **Object diagram**, showing a set of objects and their relationships; **Activity diagram**, a special kind of State chart diagram showing the flow from activity to activity within a system; **Component diagram**, showing the organizations and dependencies among a set of components; and **Deployment diagram** showing the configuration of run-time processing nodes and the components that live on them.
4.2. UML Overview

In real time systems, performance requirements are as important as functional requirements, so not only do we have to perform the correct functions, but there are clear bounds within which these must be completed. An embedded computer system is a system that uses a computer as a component, but whose prime function is not that of a computer. As one of the object-oriented techniques, UML is basically suitable for real time system development. There are techniques within UML definition that are a natural fit for specifying and designing real-time systems. Use cases allow the designers to describe the way in which humans and external devices, interact with the system. Object sequence diagrams, describe for a given use case, the events which cause the interaction and the detailed system response, including timing.

Class diagrams helps to separate system components and define interfaces between them. These techniques are good enough to capture usage scenarios and identify likely time problems.

4.3. Modeling the static aspects of the system

4.3.1. Use Case Diagram

All systems interact with human or automated actors that use the system for some purpose, and both human and actors expect the system to behave in predictable ways. In UML, a use case models the behaviors of a system or a part of a system, and is a description of a set of sequences of actions, including variants, that a system performs to yield an observable result of value to an actor.

A use case diagram models the dynamic design view of systems. Use case diagrams are central to modeling the behavior of a system, a subsystem, or a class. Use case diagram shows a set of use cases and actors and their relationships. The main contents of a use case diagram are:

- Use Cases
- Actors
- Dependency, generalization, and association relationships

According to the requirements document in our class, the use case diagram of elevator systems is showed in Figure 2:
There are seven use cases based on the requirement documentation of the elevator system in our class, as shown in Figure 2:

- **Process Elevator Calls**: These scenarios include that the elevator receives calls from the passengers, turns on or turns off the light of elevator call buttons, updates the record of elevator calls stored in system controlling parts, etc.

- **Process Floor Calls**: Similar to Elevator Call processing, this use case includes that the elevator receives Floor calls from the passengers, turns on or turns off the light of Floor calls buttons, updates the record of floor calls in system controlling parts, etc.

- **Move/Stop elevator**: The main function of an elevator, how to make the decision of stop, and driving directions of the elevator.

- **Indicating Moving Direction**: The elevator should have this mechanism to let the passengers know the current moving direction of the elevator such that the passenger might decide whether to enter the elevator or not.

- **Indicating Elevator Position**: Similarly, the elevator should let the passengers know whether his/her destination floor is reached so that the passenger may decide to leave the elevator.

- **Open/Close the Doors**: The elevator should be able to open and close the doors for the passengers to get in and out of the elevator. The functional areas of this use case should also enable the passengers to make door reversals when the doors are closing and the passenger wants to get in the elevator.
• **On/Off illumination:** The elevator and floor buttons should be able inform the passenger that his call has been scheduled and inform him that his call has been requested, in which button illumination will be needed.

The only actor in elevator system is the passenger, which is the role that humans play when interacting with the system. The passenger interacts with the Elevator system by making elevator calls and floor calls. The passenger also makes decision whether to enter/leave the elevator or not by observing the indication of moving direction and elevator position. Therefore the use case diagram shows that the actor has relationship with four use cases of the system: Process Elevator Calls, Process Floor calls, Indicate Moving Direction, and Indicate Elevator Position.

### 4.3.2. Class Diagram

Class diagram, one of the most commonly used diagrams in object-oriented system, models the static design view for a system. The static view mainly supports the functional requirements of a system, the services of the system should provide to the end users. We will see from our practical experience that lots of fun comes out when modeling out system with class diagrams. The discussion on different views of class diagrams for the system will be put into emphasis later in this paper.

A class diagram shows a set of classes, interfaces, and collaborations and their relationships. Class diagrams involve global system description, such as the system architecture, and detail aspects such as the attributes and operations within a class as well. The most common contents of a class diagram are:

- Classes
- Interfaces
- Collaborations
- Dependency, generalization, and association relationships
- Notes and constraints
Figure.3: Class Diagram

**Controller**: The central controlling object in the elevator system. Controller communicates and controls all other objects in the system.

**Door**: The Command the doors to open and close, according to the situation stated in the use case.

**Elevator**: The Elevator is being controlled to move up and down, to make stops at floors when necessary.

**Button**: The Controller class also controls the button class, which further generalizes two subclasses Floor Button and Elevator Button. The control object communicates with the Button objects, get the information whether a button is pressed and in turn controls the illumination of Button lights.

**State**: The indicators are controlled to show the information about the current position and moving direction of the elevator.

The classes captured in this diagram can cover all the functional aspects of the system: for moving or stopping the elevator, we have the class Elevator, and the control class Controller;
for opening or closing the doors, we have the class Door; for the passenger to know the position and direction of the elevator, we have State class, for the passenger to make elevator calls or floor calls, we have the Button classes. All the classes have interfaces with the central controller, whose job is in charge of the actions of other objects. From the point of view of object division and system functioning, this class diagram helps understanding the basic design idea of the system.

4.4. Modeling the dynamic aspects of the system

4.4.1. Sequence Diagram

Sequence diagram is one kind of interaction diagrams, which shows an interaction among a set of objects and their relationships (another kind of interaction diagram is collaboration diagram). The purpose of the Sequence diagram is to document the sequence of messages among objects in a time based view. The scope of a typical sequence diagram includes all the message interactions for (part-of) a single use case. There may be multiple sequence diagrams per use case, one per use case scenario.

The state diagrams commonly contain:

- Objects
- Links
- Messages
- Respond Time (especially useful in real-time systems)

The vertical “lifelines” represents objects of interest. Messages are shown flowing between object lifelines. UML supports the notation of respond time in the sequence diagrams, which makes it feasible to specify the performance requirements for a real time system. Time flows from top to bottom.

In following sections, the objects in sequence diagrams are based on the class diagram. The reason for doing that is we want to neither stay in the object construction view, in which the functions of objects are obscure and inadequate, nor go too further in the system architecture view, where many technical details obstruct a quick understanding of interaction among objects.

In some sequence diagrams the passenger appears to be an object of the system, since some of the messages are coming out from the passengers.
Figure 4: Sequence Diagram for Serving Elevator Button
4.4.2. State Chart Diagram

A State chart diagram shows a state machine. Usually the state machine in a state chart models the behaviour of a reactive object, whose behaviour is best characterized by its response to events dispatched from outside its context. The object has a clear lifetime whose current behaviour is affected by its past. State chart diagrams are important for constructing executable systems through forward and reverse engineering.

It is admitted that there exists a gap in the process of designing a system from requirements to state charts, not enough direction methods can be followed when drawing the state chart diagram from the requirements. In this section, some practical methods used during our designing the state charts for the elevator system are introduced. These methods may not be as serious as rules or instructions of how to draw state chart diagrams from the requirement document, but they are helpful in practice.
A detailed UML documentation for a simulated elevator control system is given. The UML diagrams used in this documentation are Use Case Diagram, Class Diagram, Sequence Diagram, and State Chart Diagram.
Every diagram in UML is just a graphical presentation of some of the aspects of a system. No single diagram could capture everything about a system’s design view. The UML diagrams have be combined to express a complete description of a real-time system. The three different views of class diagrams of the system can help to understand better the structural aspect of the system.
CHAPTER 5: Implementation

5.1. Technical Platform

5.1.1. The NetBeans Platform and IDE

The NetBeans Integrated Development Environment (IDE) platform is an open source IDE. NetBeans is written in the Java programming language and provides the services common to desktop applications.

NetBeans also provides support for CVS/Version control access, FTP functionality, databases, scripting, and servlet and JavaServer Pages support through Tomcat.

With NetBeans you can create desktop applications, as well as web applications. NetBeans contains extensions to the Swing APIs that make it easier to write in a syntax-highlighting code editor. The editor is capable of mixed-mode operation in which correct syntax highlighting, code-completion, formatting, and macros are provided for documents that contain content in more than one language, such as a JSP page containing both HTML and Java code.

If you are writing applications to run strictly on the desktop, NetBeans provides everything you need. In addition, it makes creating menuing systems easy. Rather than write code to manage menus, simply write the logic that's important - what should happen when a user clicks on a menu item. You can also create web applications with NetBeans.

NetBeans is also good for programmers and developers new to the Java programming language because it's simpler to parse through the output of the compiler when dealing with an application than when dealing with a call stack from an application server.

Provided features:

- Write and edit source code
- See errors as you type
- See highlighted code syntax
- Automate repetitive tasks
- Compile code
- Browse class structures
- View JavaDocs
- Use drag-and-drop utilities for easy building of features, such as graphic objects or creating database connections
- Provide templates for quick creation of JSP pages, servlets and other web components
5.1.2. Design Language

Java is a programming language originally developed by James Gosling at Sun Microsystems and released in 1995 as a core component of Sun Microsystems. The language derives much of its syntax from C and C++ but has a simpler object model and fewer low-level facilities. Java applications are typically compiled to bytecode (class file) that can run on any Java Virtual Machine (JVM) regardless of computer architecture. Java is general-purpose, concurrent, class-
based, and object-oriented, and is specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere".

There were five primary goals in the creation of the Java language:

1. It should be "simple, object oriented, and familiar".
2. It should be "robust and secure".
3. It should be "architecture neutral and portable".
4. It should execute with "high performance".
5. It should be "interpreted, threaded, and dynamic".

5.2. Implementation

Iterative and incremental processes represent best practice that has been identified from the limitations of the waterfall process. The term ‘iterative’ indicates the repetition of one or more activities; the term ‘incremental’ indicates that development proceeds from an initial subset of the requirements to more and more complete subsets, until the whole system is addressed. Incremental development involves an initial partition of the intended functionality; some or all of the subsequent development activities can be carried out independently and in parallel. The final product results from the total integration of the partitions.

In iterative and incremental processes there is still a need for analysis, design, implementation and testing activities, but these activities are carried out in a more flexible way than in the waterfall process, since not all requirements are received at once, the requirements from customer goes on getting added to the list even after the end of "Requirement Gathering and Analysis" phase, this affects the system development process. Although the problems with one
phase are never solved completely during that phase and in fact many problems regarding a particular phase arise after the phase is signed off, that’s why iterative process considered better than Waterfall process. An iterative and incremental process consists of several cycles of analysis, design, implementation and testing. Each cycle is short and provides feedback for the next cycle, in which a more refined and enhanced development is achieved. With an incremental model, development starts from small subsets of the requirements, reducing the complexity and scope of each analysis, design and coding cycle. Each increment is carried through the development activities to produce a working subset of the system, and is developed through several iterations. The integration of the increments results in the final system. However, this integration can be progressively achieved by successive releases of the software, each release achieving more functionality.

5.3. System Interface

The application starts by the Main Form shown in Figure 10:

![Figure 10: Main Form](image)

The user can choose the number of elevator needed, from one to six, for simulation using the combo Box in the top of the Form shown in Figure 11:

![Figure 11: Selecting Number of Elevators](image)
After choosing the number of elevators needed Simulation starts after pressing the “Start Simulation” Button, a new Simulation form appears showing the ten floors and the elevators chosen in addition to UP and DOWN Buttons, shown in Figure 12 & 13:
The simulation starts after calling any existed elevator by pressing UP or DOWN Button, the elevator moves to the called floor and stop and open the Elevator Button panel (showing the buttons in the elevator inside) where the user can choose the requested floor and then press the Close Door Button, where the elevator then after that can move again to the requested floor. As shown in Figure 14:
CHAPTER 6: Testing and Reviewing

6.1. Test Cases

Software Testing is the process of executing a program or system with the intent of finding errors. Or, it involves any activity aimed at evaluating an attribute or capability of a program or system and determining that it meets its required results. Software is not unlike other physical processes where inputs are received and outputs are produced. Where software differs is in the manner in which it fails. Typically, more than 50% percent of the development time is spent in testing. Testing is usually performed for the following purposes:

- **To improve quality**

  Quality means the conformance to the specified design requirement. Being correct, the minimum requirement of quality, means performing as required under specified circumstances.

- **For Verification & Validation (V&V)**

  Another important purpose of testing is verification and validation (V&V).

<table>
<thead>
<tr>
<th>Functionality (exterior quality)</th>
<th>Engineering (interior quality)</th>
<th>Adaptability (future quality)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correctness</td>
<td>Efficiency</td>
<td>Flexibility</td>
</tr>
<tr>
<td>Reliability</td>
<td>Testability</td>
<td>Reusability</td>
</tr>
<tr>
<td>Usability</td>
<td>Documentation</td>
<td>Maintainability</td>
</tr>
<tr>
<td>Integrity</td>
<td>Structure</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Typical Software Quality Factors

- **For reliability estimation**

  Software reliability has important relations with many aspects of software, including the structure, and the amount of testing it has been subjected to.
6.2. Performing Test

The **White-Box Testing** depends on rechecking the chunks of code written. White-Box testing took place from the start of the implementation of the software. Since the declaration of the first table and the creation of the first query, revision of the code started, with one purpose in mind, which is to make sure that no code conflicts with another leading to a semantic error that is a logical error where the written code doesn't perform the specific required task. During White-Box testing the software system proved to be fine with no conflicts occurring and very few errors were detected all related to specific cases where non relative values are entered in the fields.

The **Black-Box Testing** depends on viewing the software from an external point of view, no revision of the code takes place the code is obscured from the person who is performing the test to the software system. Here the person testing the software system is a normal user, exploring the software and testing its functionality by entering samples of data and the software is unfamiliar to him which means that the person testing the software application is working on the software for the first time. Samples of data have to be values taken at random, the values taken should be at the boundaries of the acceptable range and at the middle of the range. When to stop testing? Testing is potentially endless. We can not test till all the defects are unearthed and removed -- it is simply impossible. At some point, we have to stop testing and ship the software.

**Consistency:**

<table>
<thead>
<tr>
<th>Test Conditions</th>
<th>Expected Result</th>
<th>Actual Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Menu positioning</td>
<td>The menu position must be consistent throughout</td>
<td>Yes, the main menu is always situated at the top of the application window</td>
</tr>
<tr>
<td>Color</td>
<td>A small collection of different colors must be used</td>
<td>Yes, The use of color in background, controls etc. is consistent</td>
</tr>
<tr>
<td>System responses</td>
<td>The way the system responds to a particular user action must be consistent</td>
<td>Yes, All system responses are displayed</td>
</tr>
<tr>
<td>User feedback</td>
<td>The illumination of buttons and the visual interaction of user with the application</td>
<td>Yes, All system interaction are providing consistent feedback</td>
</tr>
</tbody>
</table>

**Functionality:**
<table>
<thead>
<tr>
<th>Test Conditions</th>
<th>Expected Result</th>
<th>Actual Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Receiving call</td>
<td>The system take into consideration the call received from the user</td>
<td>Yes, by requesting the call after receiving it</td>
</tr>
<tr>
<td>Requesting call</td>
<td>The system is requesting the received call</td>
<td>Yes, by showing the result</td>
</tr>
<tr>
<td>Indication of elevator movement</td>
<td>The system indicate the position and direction of the elevator</td>
<td>Yes, All system indications are displayed</td>
</tr>
<tr>
<td>Multithreading of the system</td>
<td>The system must be capable of receiving requests while simultaneously replying to previously requested quotes.</td>
<td>Yes, The server can handle requests and replies at once.</td>
</tr>
</tbody>
</table>

Table.3: Functionality table testing
CHAPTER 7: Conclusion

7.1. Summary

Simulation is a way of developing application or design simulating real world, the way which help in clarifying the real idea. The reasons upon using simulations may be for literature, cost, time, or safety reasons. Company problem appears when it is intended to equip its building with an elevator and want to know according to its needs the way to settle the elevator to best satisfy this needs. The way where we settle an Object-Oriented Software Simulator, concerning the logic and algorithm required to move the elevator between floors, giving the company a clear idea and leading it to choose the best design which satisfies this needs.

7.2. Future work perspectives

Some of future work which could be implemented:
- Speed control may be available to the user of the simulator to see different simulations with different speeds.
- The elevator needs the ability to change directions if the user decides to go down instead of up (and vice versa) when there are no other requests for that elevator.
- Elevator will be supplied by inventor, in the case of electricity failure elevator will go down to the nearest floor and open its doors.
- An emergency alarm switch will sound an alarm when activated by a passenger.
- Elevator will be supplied by Priority system, which is used by specific users like managers, where they can press certain code making the elevator moving straight forward to the wanted floor without stopping in any floor.
CHAPTER 8: References

8.1. References

- http://www.us.schindler.com/sec_index/sec_kg/sec_kg_safety/sec_kg_profile_safety_elevatorsafetyfeatures.htm
- http://www.elevatorchallenge.com
- http://www.ece.cmu.edu/~koopman/des_s99/sw_testing/
- http://www.cnet.com/Content/Features/Dlife/Bugs/?dd
- Alan Moore and Niall Cooling; Developing Real-Time Systems using Object Technology; A white paper from Artisan Software Tools.

8.2. Appendices