Technical University of Moldova

Chair Computer Science

Course Thesis

On Designing Informational Systems

Smart Parking Tax System

Done by: st. gr. FAF-091 Bostan Constantin

Checked by: Zarea Ivan

Chişinău – 2013

Task: Design a system that will amend people whose cars are parked illegal, this sytem will use a mobile device to collect the data.

Plan of the course thesis:

Name of the stage	Description	Deliverables	Criterions of acceptance	Nr. of hours Min/max	Start date-end date
System requirements and architecture	In this stage we are creating functional and non-functional requirements of the system. Also there is architecture description.	Functional and non-functional features, system architecture.	Detecting and describing each system features. Design the system according to architectural principles	12/24	24/09/2012 10/10/2012
Test Driven Development	Creating the system features according to some unit tests	Three features with their unit test	Passing the unit test	24/48	11/10/2012 12/11/2012
Code refactoring and architectural patterns	Using refactoring techniques to improve the code of system features. Applying some architectural pattern to the code	Refactored code	Code refactoring according to "Improving the Design of Existing Code" and "Patterns of Enterprise Application Architecture" by Martin Fowler. The code should be more understandable and easy to implement	24/48	13/11/2012 14/12/2012

1. System requirements and architecture

1.1 Creating the system vision

1) Business Context

The program Smart Parking Tax will be used by the policemen to amend people whose cars are parked illegal. It will be installed on Android smart phones. Therefore this program will be portable and it will solve the main problem in Chisinau: agglomeration of parked cars on the downtown streets. Of course this solution will create some difficulties for the policeman to study the new method, but in the end their speed to amend drivers will grow.

2) Business Opportunities

In Chisinau there are a lot of cars that are parked illegal and drivers are not amended regularly. Maybe because there aren't enough people in the road police department or old method is very slow, till you will complete the form, driver could escape. Using new method with a Smartphone you could take a picture of the car parked illegal and the driver will receive his penalty tax via email, or a simple letter. Thus the number of amendment tax will increase very fast, causing drivers to park their cars legally. The amount of money collected from taxation will also grow.

3) Problem Description

Problem

-The main problem is that policemen cannot handle the increase number of illegal parked cars; therefore drivers can easily park their cars wherever they want.

Concerns

- The people who encounter this problem are the pedestrians and other drivers.

Consequences

- Accidents and traffic jams on the roads.

Successful solution

- Less illegal parked cars on the roads, increasing number of amendments, transparency on taxation, in long terms less accidents on the roads.

Task of the system

- Capture on the digital cameras these parked cars and amend drivers.

1.2 Identifying the stakeholders

Stakeholder	Role	Description
Road Policeman	Controls the car traffic on the roads	Giving amends to drivers.
System Administrator	Maintaining the data base	Introduces the new matriculation numbers of a car.
Person who receives tax at the police department	-	Deletes the record of the amendment form database of a driver if he receives money.
Drivers	-	They usually park they car wherever they want.
Pedestrians	-	People who usually don't have cars and uses public transport.
Bostan Constantin	Project manager/ software programmer	-Approving the milestones; -Approving documents and functionalities; - Responsible for project execution; -Approving the application requirements; - Creating the system.

1.3 Documenting the functional requirements of the system

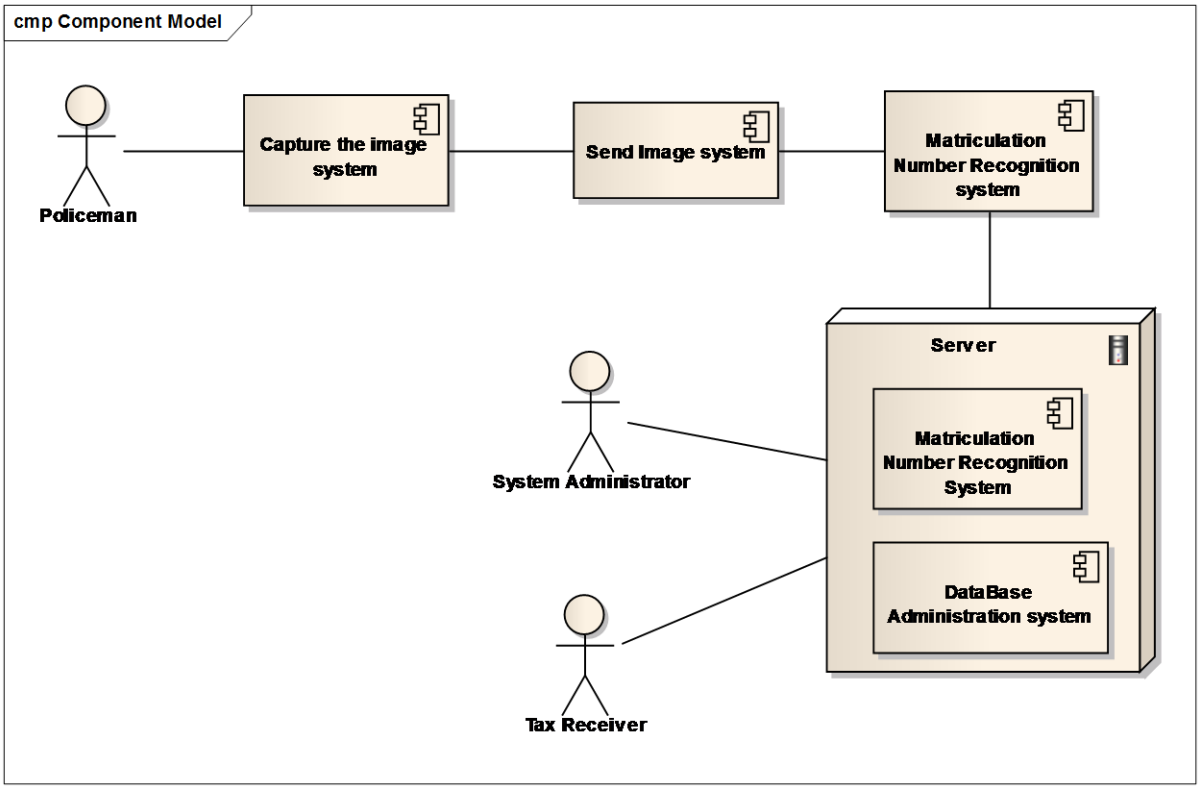
System functionalities:

- Program for Smartphone: the program will be installed on an Android based Smartphone.
- Smartphone will connect to the server, that has the database of all the drivers, via internet connection.
- Image recognition: on the digital photo it will recognize the matriculation number of a car.
- Sending the tax: the amendment will be send to the driver via email, if he has one, and a simple letter. The message on the letter will show the picture of the car, when and where the car was found.

1.4 Non-functional requirements:

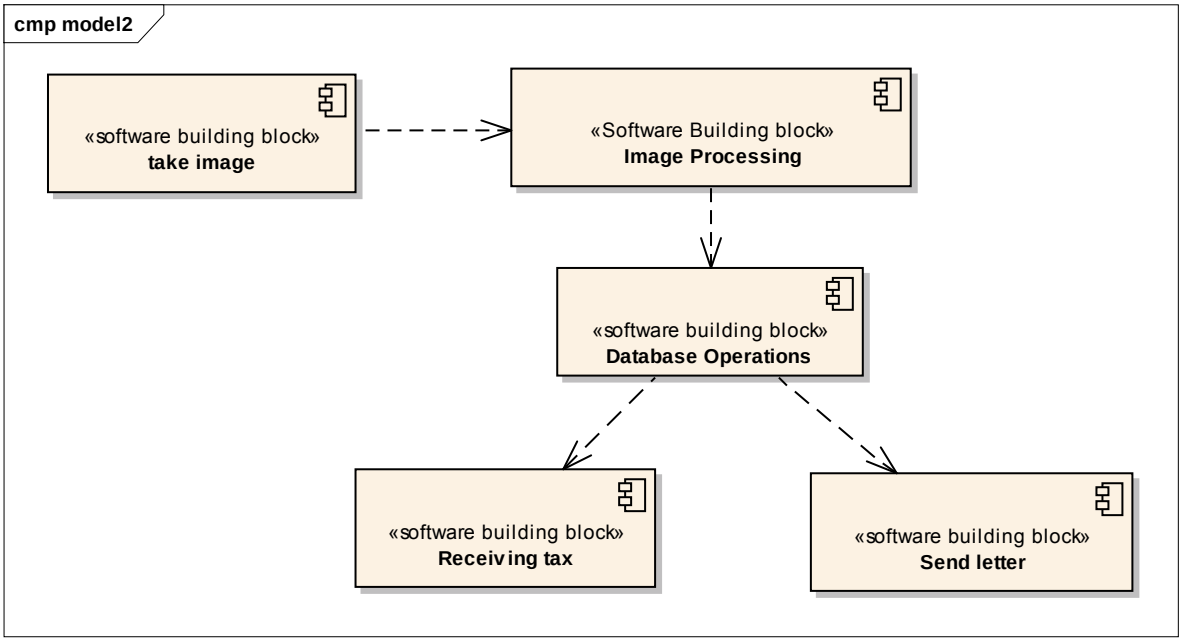
- Usability- program will have help contents of how to use it;
- Must run on Java Virtual Machine;
- Security- Only authenticated users must be able to access functionalities and data for which they have authorization. From the Smartphone program it will be impossible to delete some data, only to send the matriculation number.
- User friendly: program should not have too much functionalities, it will be simple to use and maintain.

1.5 Architectural sketch:



Capture the image system- is responsible to capture the image form an android Smartphone
Sending the image- Smartphone connects to the server via internet and sends the image for processing.
Matriculation Number Recognition System- is an image recognition system that detects the car matriculation number and converts it to a string.
DataBase Administration system- this part is responsible to insert, deletes and update data in database.

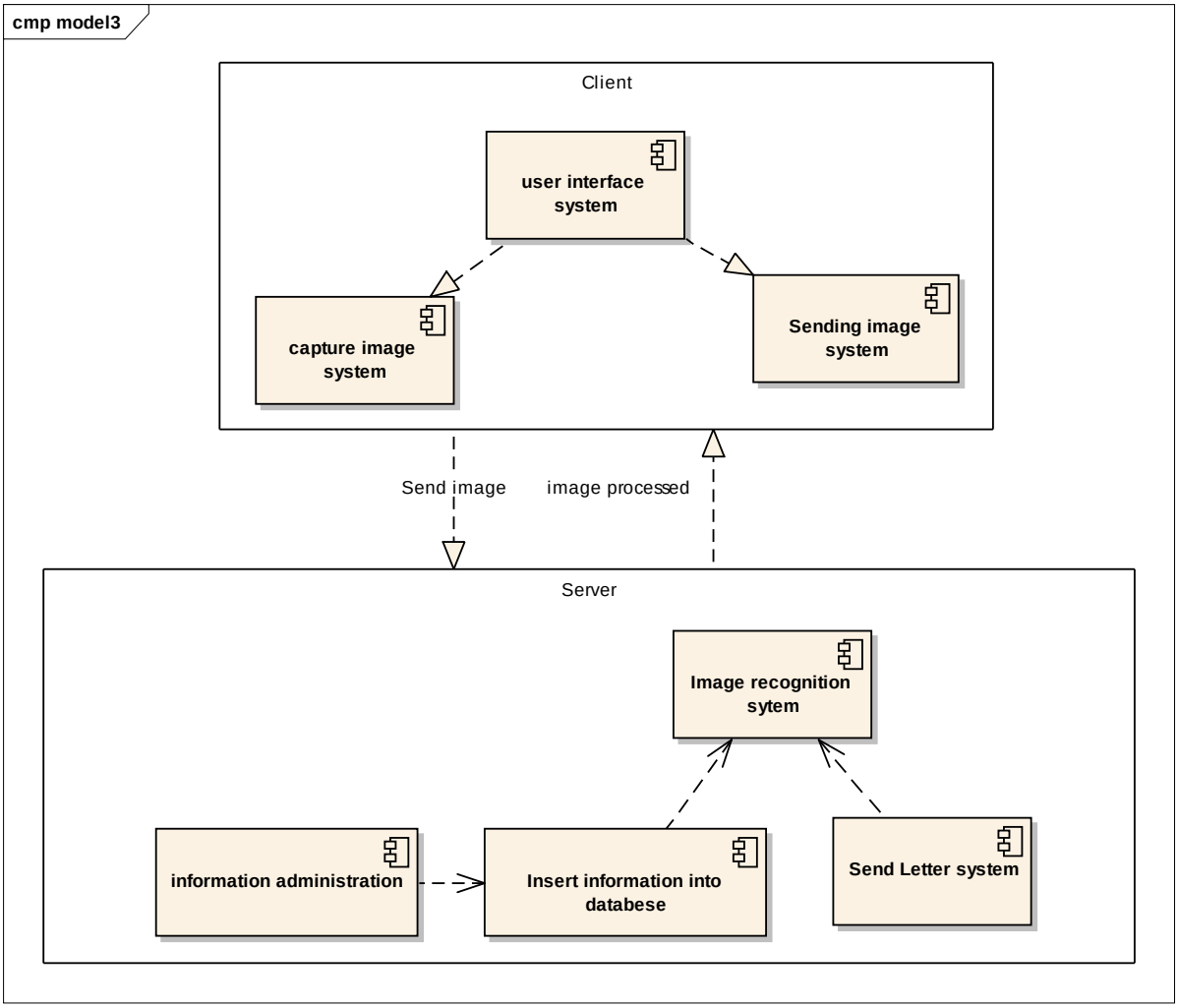
1.6 Functional Blocks



First building block is: Take image in this block is responsible for capturing image.
Image processing- we have image recognition system that detects car number and transform it to string.
Database operations: this block is responsible for operations on database: insert, update and delete.
Receiving tax: finds the driver and deletes the records of the amendment if the driver has paid this amend.
Send letter block: is responsible for sending an email to the driver and a simple letter.

1.7 selecting architecture

The chosen architecture is Client server:



Client-will capture and send image to the server.

Server will process the image and will send a letter to the driver.

1.8 Architectural principles

Loose Couplings: I have divided my project into 2 parts client installed as an Android application and server therefore each part is working independently.

Principle of high cohesion: Each building block for example user interface, capture image, sending image are grouped in one place named client, other functionalities are implemented on the server.

Design for change: Client building block could be easily changed if for example a new Android version will come out or a new more developed Smart phone will appear, server part will remain intact.

Separation of concerns: The easiest part is taking and sending the picture will be implemented on the Android part. The hardest and CPU consuming part: image recognition process will be called on the server. Therefore the problem is divided and processed separately.

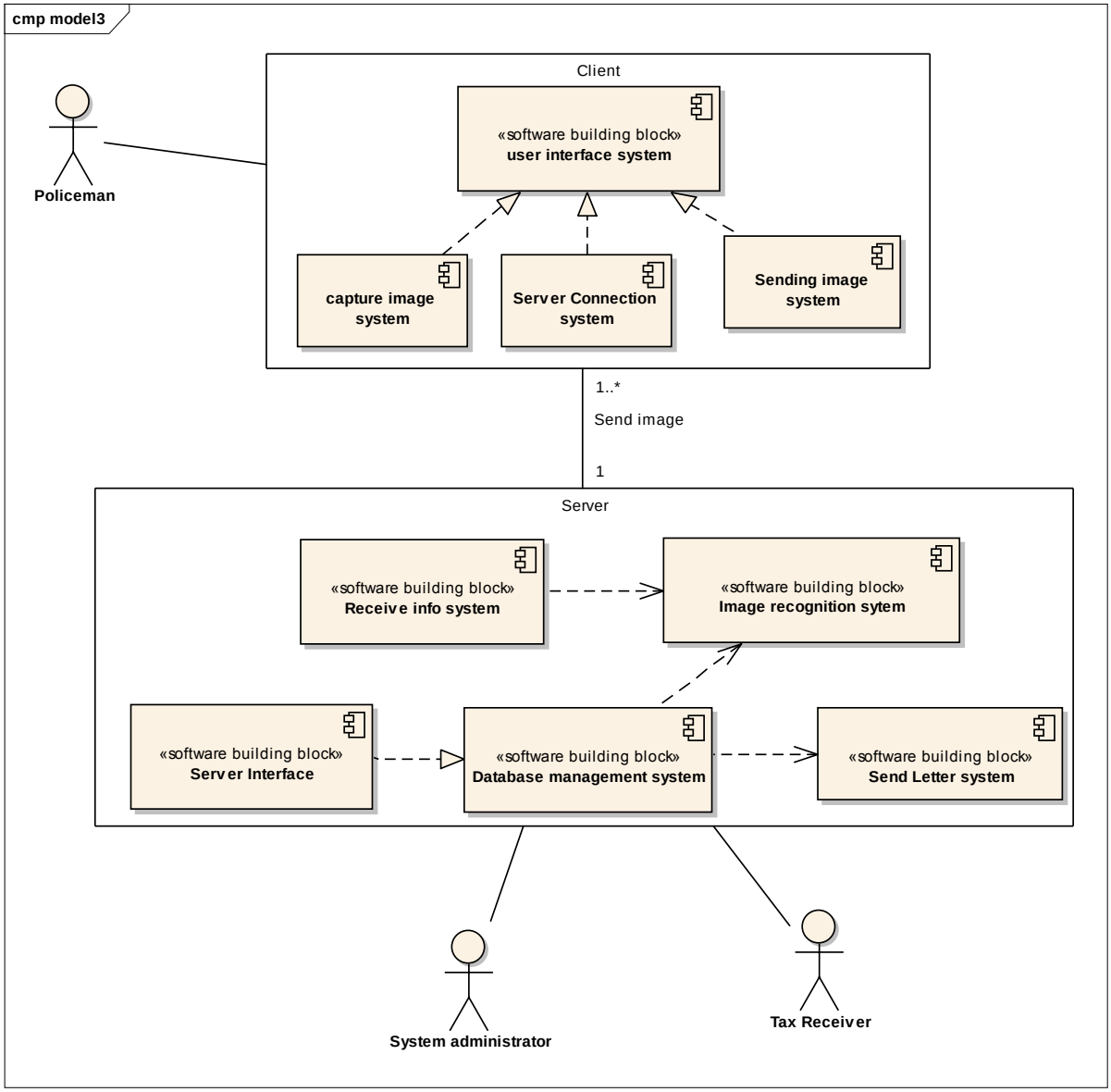
Informational hiding: The most important part is server side therefore client only can send photos not insert and delete data. In my architecture this is implemented.

Liskov substitution: User interface uses this principle to specify methods: capturing and taking picture.

Interface segregation: Client has its own interface and server too.

Modularity: each of the system building blocks is self-contained, modifiable, extensible, and reusable.
Convention over configuration: Android program could be easily installed and configured and server also therefore if some functionality could be added step by step.

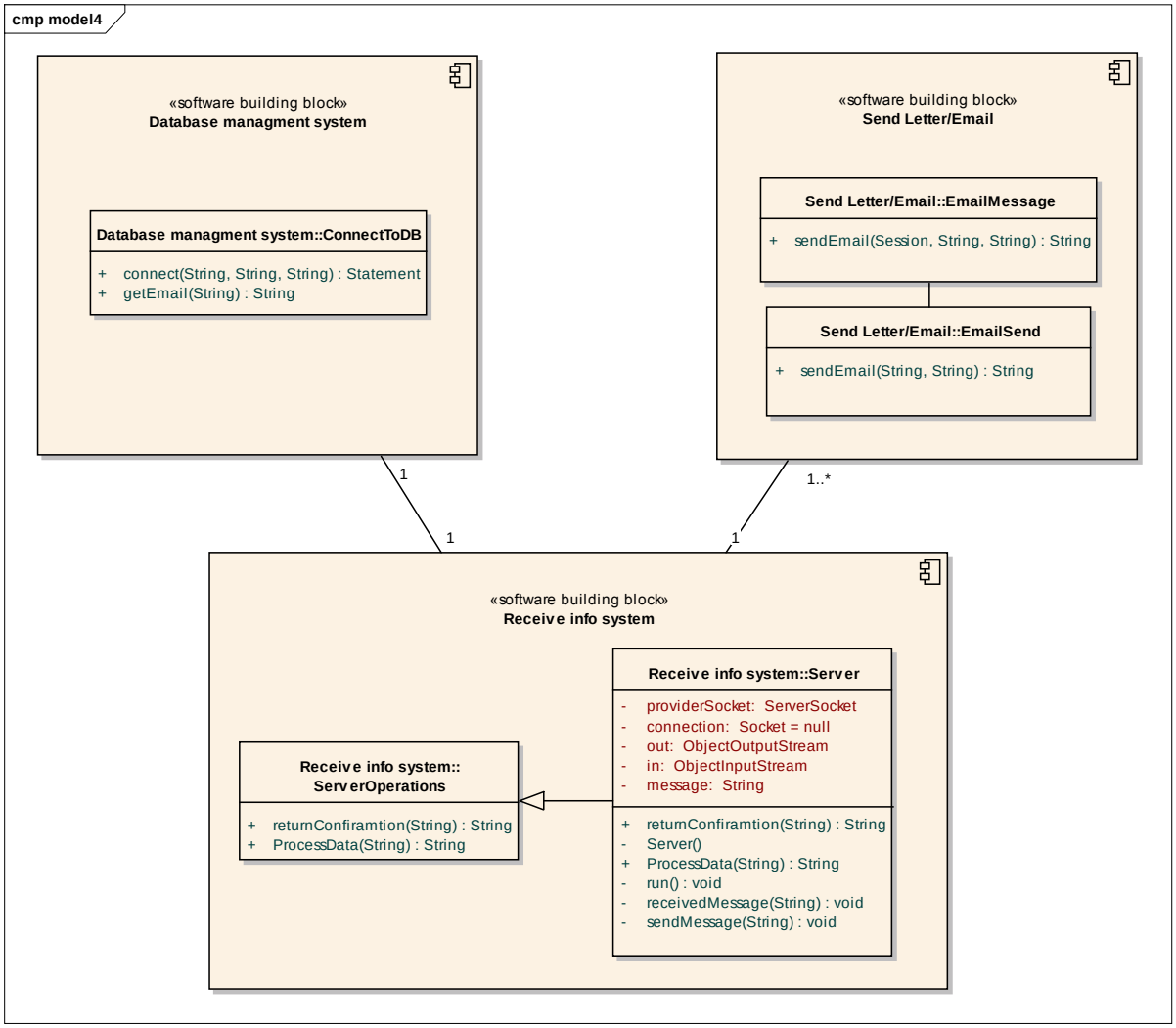
1. 9. An architectural sketch (The final version)


cmp model3
Client
«software building block»
user interface system
Policeman
capture image
system
Server Connection
system
Sending image
system
1..*
Send image
1
Server
«software building block»
Receive info system
«software building block»
Image recognition sytem
«software building block»
Server Interface
«software building block»
Database management system
«software building block»
Send Letter system
System administrator
Tax Receiver

2. Test Driven Test of system features

2.1 Test descriptions (User story):

As a client I want to connect to the server using sockets so that I could send to this server some data.
As a server I want to connect to the database so that I could find the email of a driver.
As a server I want to send an email to people who have parked illegal so that every pedestrian could be happy.
Description of the classes that implements those three functionalities:
connetToDB - class responsible to connect to database and select some information
EmailSend - sends the emails using an exterior mail.proprieties file to store the email configurations
Client - connects to the server and sends some information.
Server - receives the information form class client and processes it.



Functional blocks with implemented classes

```
public class Test1 {
        @Test
        public void testDB() {
                connectToDB tester = new connectToDB();
            assertEquals("Result", "costea.bostan@gmail.com",
tester.getEmail("CRNM778"));

            assertEquals("Result", "No record found",
tester.getEmail("asdf"));
}
}


public class Test2 {

        @Test
        public void testSendEmail() throws Exception, Throwable {
                EmailSend tester2 = new EmailSend();
            assertEquals("Result", "Message send",
tester2.SendEmail("costea.bostan@gmail.com", "432"));
        }
}


public class Test3 {

        @Test
        public void testRun() {
                Client tester3 = new Client();
            assertEquals("Result", "Connection successful", tester3.run("A
simple message to server"));
        }
}
```

3. Code refactoring and architectural patterns

1. Code inspection
The chosen toolkit to check code is: checkstyle pug-in for Eclipse because it is easy to use and configure.
Sonar is also a good tool but it is more restrictive, hard to run, doesn't like my project's name and is not easy to configure.
2. Refactoring
1) Extract Method
Server Side:
//lines of codes to create socket and wait for the connections
//receiving the message from the client
```
try{
```

```
                        message = (String)in.readObject();
                        System.out.println("client>" + message);
                        if (message.equals("bye"))
                        sendMessage("bye");
}

The code was changed to this:

try{
                        message = (String)in.readObject();
                        receivedMessage(message);
                        if (message.equals("bye"))
                        sendMessage("bye");
                                }

void receivedMessage(String msg){
                System.out.println("client>" + msg);
        }
```

2) Split Temporary Variable

Server Side:

```
//lines of codes to create socket and wait for the connections
//receiving the message from the client
// message is a temporal variable that is used more than ones.
do{
                                try{
                                        message = (String) in.readObject();
                                        msg = message;
                                        receivedMessage(msg);
                                        sendMessage(serverMsg);
                                        message = "bye";
                                        sendMessage(message);
                                        }

                                        catch (ClassNotFoundException classNot) {
                        System.err.println("data received in unknown format");
                                        }
                        } while (!message.equals("bye"));
The code was changed to:
do{
                                try{
                                        message = (String) in.readObject();
                                        receivedMessage(message);
                                        sendMessage(serverMsg);
                                        serverMessage = "bye";
                                        sendMessage(message);
                                        }
```

```
                                catch (ClassNotFoundException classNot) {
                                        System.err.println("data received in
unknown format");
                                }
                        } while (!serverMessage.equals("bye"));
```

Benefits: The temporary variable message was used often and was getting in the way of other refactoring such as: implementing Local Extension for Server Class. Also it was confusing the code for other developers.

Therefore after implementing this refactoring the code became more flexible and understandable.

3) Introduce Local Extension

Before:

```
                do{
                                try{
                                message = (String)in.readObject();
                                receivedMessage(message);
                                                                //a lot of code to
process message
                                if (message.equals("bye"))
                                sendMessage("bye");
                        }
                        catch(ClassNotFoundException classnot){
                        System.err.println("Data received in unknown format");
                                }
                        }
                        while(!message.equals("bye"));
                }
```

After:

```
public class Server extends ServerOperations{
//Creating sockets and waiting for connections lines of code
do
{
try
{
message = (String)in.readObject();
receivedMessage(message);
ProcessData(message);
if (message.equals("bye"))
sendMessage("bye");
}
```

```
catch(ClassNotFoundException classnot){
System.err.println("Data received in unknown format");
}
}
while(!message.equals("bye"));
}

public class ServerOperations {
 public String returnConfiramtion(String number)
 {
         ConnectToDB email = new ConnectToDB();
         String driverEmail = email.getEmail(number);
        return driverEmail;
 }

public String ProcessData(String line) {
        ServerOperations obj = new ServerOperations();
        String[] parts = new String[2];
        parts=line.split(" ");
        String line1 = parts[0];
        String line2="";
        String respond= "bad request";
        if (parts.length==2)
        {line2 = parts[1];}
        else
        {line2="";}
    if (line1.equals("email"))
        {
        respond = obj.returnConfiramtion(line2);
        }
        return respond;
}
}
```

Benefits: The Server side class is more flexible to changes and in case if we cannot modify the server class we can easily change the class responsible for operation: ServerOperations.

4) Inline Method

// a code example from function getEmail() where we have two return statements that //basically means the same thing that there is no such record.

```
                try {
                        rs.next();}
                catch (SQLException e) {
                        return "no record next";
                }
                try {
                                email = rs.getString("email");
                        } catch (SQLException e) {
                                return "No record found";
                        }
```

```
                  return email;
```

The code was changed to this:

```
try   {
if (rs.next()){
email = rs.getString("email");
}
}
catch (SQLException e)
{
return "no record found";
}
```

Benefits: The functions which have seemed badly factored are grouped together to short the code.

5) Replace Method with Method Object

```
Before:
//code responsible for setting the email from an external file and send it to
someone
try {
                        Message message = new MimeMessage(session);
                        message.setFrom(new InternetAddress("administrator"));
                        message.setRecipients(Message.RecipientType.TO,
                        InternetAddress.parse(email));
                        message.setSubject("You've been amendet ");
                        message.setText("The sum is: "+Sum);
                        Transport.send(message);

                } catch (MessagingException e) {
                        throw new RuntimeException(e);


                }
                return "Message send";
        }

After:
ClassEmailMessage message = new ClassEmailMessage();
                        String responce = message.sendEmail(session, email,
sum);
                return responce;


public class ClassEmailMessage {
public String sendEmail (Session session,String email, String sum)
```

```
{
        Message message = new MimeMessage(session);
        try {
                message.setFrom(new InternetAddress("administrator"));
        } catch (MessagingException e) {
        return "Cannot Login with current user and password";
        }
        try {
                message.setRecipients(Message.RecipientType.TO,
                InternetAddress.parse(email));
        } catch (MessagingException e) {
                return "Cannot connect to mailbox";
        }
        try {
                message.setSubject("You've been amendet ");
        } catch (MessagingException e) {
                return "Cannot set the subject";
        }
        try {
                message.setText("The sum is:"+sum);
        } catch (MessagingException e1) {
                return "Cannot set the text";
        }
        try {
                Transport.send(message);
        } catch (MessagingException e) {
                return "Cannot send the message";
        }

   return "the message was sent";
}
}
```

Benefits: By extracting pieces out of a large method, the things became much more comprehensible.

Table Data Gateway

It is implemented in my system in the class ConnectToDatabase that it is a gateway that holds all the SQL for accessing a single table or view: selects, inserts, updates, and deletes. The other code: Server class calls these database methods. As a benefit we are avoiding introducing SQL queries as a parameter for methods.

Service Layer

In my system I have used three layers:

- Database layer- connecting to database and retrieve information;
- Server layer - image recognition, email sending;
- Client layer- Sending the image to server.

The benefits for this architecture are that my system becomes more flexible and easy to understand. Also client and Server have their own interface.

Pessimistic Offline Lock

Prevents conflicts between concurrent business transactions by allowing only one business transaction at a time to access data. I'm using PostgresSQL by default at the end of each query a commit is applied and I only have a select statement therefore I think there is no need to use this architectural pattern. Pessimistic Offline Lock is used to work with long transactions.

Conclusion

In this course thesis I have studied how to create design and create software systems using architectural patterns and architectural principles. The most useful parts I think are Test Driven Development where first of all I have created user stories that tells what user does in the system or what he need to do in order to obtain some result. Using these stories I have created three features: connecting to database, send email and client-server connection. These three features are created using JUnit tests. A *unit test* is a piece of code written by a developer that executes a specific functionality in the code which is tested. *JUnit* in version 4.x is a test framework which uses annotations to identify methods that are test methods.

At refactoring I have used extract method to make code more understandable easy to modify especially then you need to quickly change the output of what the client says. Slit Temporary Variable- temporary variable message was often used in the code and was creating confusions, therefore by introducing more variables it becomes clearer to understand. Another term used was “Local Extension”- If I had wrote the code in the server’s while loop that process client messages it would become a nightmare for someone to understand that cod.

The program Smart Parking Tax will be used by the policemen and simple citizens to amend people whose cars are parked illegal. It will be installed on Android smart phones also it will need a server that could process all the requests to the database and email send.