A smart energy management and monitoring system

EBU5304 – Software Engineering Group Coursework Report

Group Number: 37

Group Leaders Name: Wei Sun

Group Members:

Name	QM Student No.	BUPT Student No.
Wei Sun	151012930	2015213287
Xiaorui Zhang	151011771	2015213172
Shuhui Li	151012284	2015213222
Siqi Miu	151013225	2015213316
Sihong Luo	151012217	2015213215
Yiwen Zhang	140921724	2014213173

CATALOGUE

1.Project Management

Project management is to ensure the software is delivered on time, within budget, with quality.

1.1 Project planning

In this period, our team make a decision about what we should have done in each period, delivery data and staff available. We write a quality plan to describe the quality procedures and standards that will be used in this project, so that our program could meet the requirements better.

1.2 Agile project management

To obtain full software engineering skills, we choose Agile project management as our requirement, everyone in our group work on all aspects of the project.

1.3 Project scheduling

1.3.1 Work Breakdown & Scheduling

Preparing (2018/3/24--2018/3/30)	**T1: Read the handout**	**-**
Requirement (2018/4/1--2018/4/8)	T2: Collect story	T1
	T3: Summary story	T2
	T4: Complete the product backlog	T3
Analysis (2018/4/8--2018/4/16)	T6: Type of classes	T4
	T7: Relationship between classes	T5
	T8: Draw sequence diagram	T6,T7
	T9: Summary	T8

Design **(2018/4/17--2018/4/27)**	T10: Design variables and methods of classes	T9
	T11: Drew class diagram	T10
	T12: Design prototype of GUI	T10
Implementation **(2018/5/1--2018/5/6)**	T13: Job distribution	T11
	T14: Individual working	T12
	T15: Integrate each part	T13
Testing **(2018/5/7--2018/5/8)**	T16: Class Test	T14
	T17: Integration Test	T15
	T18: Performance Test	T16
Reporting **(2018/5/20--2018/5/28)**	T19: Write report	T4,T9,T11,T14,T17
	T20: Check all works	T18

1.4 Monitoring and Reporting

1.4.1 Stand-up Meeting

We organize daily stand-up meeting to discuss our ideas and difficulties, if someone's tasks have unpredictable absence, we can get this in time and find ways to solve this.

1.4.2 Weekly Meeting

We also organize weekly meeting to share our current mission progress, and make our own recommendations then can make appropriate changes.

1.4.3 Working Around Table

If we are free at the same time, we often get together to face-to-face programming so that we can find the problems in time.

1.5 Risk management

Risks can be divided into: Project risks, Product risks, Business risks

Risk Type	Potential Risks	Probability	Effects
project risks	Team member can't complete the task on time	Moderate	Tolerable
	Because of one un-finished, can't go on this project	High	Serious
Product risk	Team members' skills and knowledge are at different levels	Moderate	Tolerable
	The design may have error at beginning	High	Serious
	There may be other errors, such as logical error, performance, security	High	Catastrophic
Business risk	Development program software has bugs	Moderate	Serious

2. Requirement

The customer must be able to read and understand the results of requirements capture. We must use the language of the customer to describe these results.

With the purpose of satisfying the customer, we need to determine the requirements that our system must do. There are 4 requirements capture stages: 1. List candidate requirements 2. Understand the system context 3. Capture functional requirements 4. Capture non-functional requirements.

2.1 Fact-finding techniques

2.1.1Background Reading

Our main responsibility is to develop the software of a smart energy management and monitoring system. After meeting discussion and requirement forms, we divided the user into two groups: consumers and energy provider. Consumers can view their own consumption and cost, set budget and get alert, view history, check tariff, send meter readings to energy provider. Energy provider can manage each household, set tariff, receive meter readings and generate bills automatically.

- Company reports & Policy manuals:

A smart energy company's target is to provide more convenient system for consumers and their own to control energy consumption. As an agile software development team, our job is to find and capture their requirements.

- Job description:

To complete these functions, the main tasks are reading from files and writing into files. Almost all of the functions are based on those operations, so we must determine how to build a file system, such as how many files we need, what should be recorded in the files. Then we need to write some methods to finish other jobs. At last, there is a requirement about time, so we need to determine time system.

2.1.2 Interviewing

In order to understand requirements thoroughly, we need interview our two types of users: consumer and energy provider.

Consumer:

To control and view his usage of energy, the consumer should be provided following information: real-time and historical energy consumption, energy tariff, and the functions about budget, there also should be an e-mails alert.

Energy provider:

To manage households and energy, the system should have functions for provider to add/view/remove consumer and receive monthly readings, generate bills and send to consumer's e-mail automatically.

2.1.3 Observation

Observation provides the developer with a better understanding of the system. We can get

shortages of our system by observing the user using the current system, so that we can improve our system according users' feedback.

- Provider finds that's difficult for him to remove consumer's information, because there are too many users, so he needs a searching bar to enter consumer's ID to manage these things more convenient. To solve this problem, we add a search system.

2.2 Functional Requirements

The functionalities our system provided are the functional requirements, which are based on fact-finding technique.

View current energy consumption and cost: the current data of electricity meter and gas meter is recorded in .txt files; consumption can be updated frequently.

Set budget: the budgets of electricity and gas are set separately; the budget can be viewed and modified at any time.

Give alert: the alert can be given in a proper way: if the energy usage exceeds the budget, a red signal and an alert must be displayed; otherwise a green signal must be displayed.

View historic energy consumption: the previous consumption of energy is recorded daily; consumers can choose which period to view: daily, weekly or monthly.

Send meter reading: monitoring software will automatically send the 30-days meter reading to provider.

Check tariff: household monitoring software can get access to and display the current tariff; consumer cannot change the tariff.

Manage consumer information: when a consumer is removed, his historic readings will be removed at the same time; consumer information must be stored in the system and can be viewed and modified by provider.

View consumers' energy records: users' readings and bills information must be recorded, and can be accessed by provider.

Set tariff: once provider changes the tariff, it must be applied to every energy monitoring software.

Send monthly bill to consumers: a bill is generated with date, userID, energy type, consumption, and total cost.

Consumer login: only when the password and userID are matched, can the user log into the account.

Consumer modify the password: the password can be modified only when the old one is matched with userID; the old password will be replaced and the new one must be recorded.

2.3 Non-functional Requirements

2.3.1 Product Requirements

The user interface should be implemented as GUI. The GUI should be flexible and extensible. The records in files are generous, so they must be correct. And each function should be tested before using because it's important for system to operate.

2.3.2 Organization Requirements

It includes delivery, implementation, standards requirements.

The system development process must be based on product backlog. It needs standalone Java application program, user do not need to install any other java libraries, just use command line to run.

2.3.3 External Requirements

The system should not disclose any personal information of users. Users will not receive any spam and advertisements.

2.4 Software Prototypes

Provider: Appendix figure 1

Consumer: Appendix figure 2-3

2.5 Backlog Evolution

During the process of program engineering, we find some requirements should to be changed, some stories should be broken up, and some details should be added. We also change the priority and iteration.

2.5.1 Stories Break-up and Iterations Changes

First submission			Final Version		
Story	Priority	Iteration	Story	Priority	Iteration
View current energy consumption and cost	1	1	View current energy consumption and cost	1	1
Manage consumer information	1	1	View consumer information	1	1
			Search for consumer	1	1
			Add consumer	2	2
			Remove consumer	2	2
Consumer login	2	1	Consumer login	2	2
Check tariff	1	5	Check tariff	3	3
Set tariff	1	5	Set tariff	3	3
Consumer modify the password	4	6	Consumer modify the password	3	3
View historic energy record	1	4	View historic energy record	3	4
Set budget	1	2	Set budget	4	4
Give alert	1	2	Give alert	4	4
Send meter reading	1	3	Send meter reading to provider	5	5
Send monthly bill to consumers	1	3	Receive monthly bill to consumers	5	5
View consumers' energy records	1	4	View consumers' history bills	5	5
Build an e-mail system	5	6	(deleted)		

We break up the "manage consumer information" story into four, to make it convenient to complement and improve the reusability of code. We also delete the story about e-mail, since it is kind of over-design and the similar function can be achieved in other stories. Besides, we re-estimated priority based on requirements and relationship between stories, and then adjust the iteration to match it.

2.5.2 Details Added

Details about file information and necessary fields are added. And some limited conditions are clarified. For instance, following statements are added into story "Add a new user".

1.Verify that user information can be recorded: in user_info.txt

2.Verify that recorded user information contains required fields: user_id, user_name, e-mail address, and initial password

3.Verify that user_id can be randomly generated by the system, and must be unique

4.Verify that user_name and e-mail address can be gained from system input: neither of them can be null

2.6 Iteration and Estimation

Our group also do iteration and estimation. To increase work efficiency, we divide functions into units of works so that every iteration they could run singly.

Iteration	Story ID	Story name	Priority	Interpretation
1	C01	View current energy consumption and cost	1	Those stories are the basic functions for the system. From the user requirements, these functions are important.
	P02	View consumer information	1	
	P03	Search for consumer	1	
2	C04	Consumer login	2	Those stories mainly about provider management operation, which is the essential part in the system. The program should allow provider to add and remove consumers.
	P05	Add consumer	2	
	P06	Remove consumer	2	
3	C07	Check tariff	3	Those stories mainly about consumers' basic operation.
	P08	Set tariff	3	
	C09	Consumer modify the password	3	
4	C10	View historic energy consumption	3	Those stories are important for consumers but they are based on previous iterations.
	C11	Set budget	4	
	C12	Give alert	4	
5	C13	Send meter reading to provider	5	Those stories are mainly about meter readings and bill operations.

	P14	Send monthly bill to consumers	5	
	P15	View consumers' history bill	5	

3. Analysis and Design

3.1 Class Analysis

3.1.1 Class Design

After the analysis, we designed the stereotype, the attributes and operation of classes. Specify all details before implementing the design. Details of classes diagram are presented as below.

Class	Stereotype	Attribute	Operation	Association
User	Entity	String: id, user_name, password, email, tariff Int: g_usage, e_usage, g_cost, e_cost	Present the user information.	Class gas, electric, energy, manage, GUI, modify
Gas	Entity	Double: amount, budget, price	Store the gas information of users	Class GUI
Electric	Entity	Double: amount, budget, price	Store the electric information of users	Class GUI
Energy	Entity	String: id Date: date	The whole id and data related to user	Class GUI
Manage	Control	-	Control the method of some file operation.	Class GUI, user

Time	Control	String: time Int: t	Provide the timing function, the thread control the system when it is open.	Class GUI
Login	Control	String: password, id	To check for the different and login in the system.	Class GUI
Modifier	Control	<list>String: user_info	Provide the method to change the user's password	Class GUI
GUI	Boundary	-	Provide the interaction between	

3.1.2 Class Diagram

Appendix figure 4

3.1.3 The Issue of Re-usability of System Components

The re-usability refers that the component of the system can be used in the other system and previous developed object. According to the Intelligent system we could design the reusability in these aspect.

1. Because in this system there could have many part that related to the file operation. So the read and write operation of the systems could be write in a class and by extending the file operation and use the method in it.
2. The system contains many part related to the random number. And this method is written in the 'creator' method so in many situations it could be used with slightly change.

3.2 Design of Software

3.2.1 Architecture Design

Object-Oriented Architecture

Java is a typical Object Oriented program. So we design the system fit the requirement of

object oriental program.

1.Inheritance

The inheritance of thread realized the function of changing the energy with relative time, so in this way could cloud seen other function that change with time aggregation.

2. Polymorphism

We write the record_control which are implement by electric and gas, to have different way to realize the function

Layered Architecture

The system will be designed into four layers, each of which offers different type of service. Layer 1 will be the basic method that support for the function. Second layer is the fundamental entity class, for example in this class the entity of the system will be defined. The third layer functions as the control the whole system. The last layer provide user GUI. With the architecture, it will realize the modules of the code, and insure low dependency between different layers which means when some code changed in one layer the other layer won't be affected.

3.3 GUI interface design

Appendix figure 5

GUI Interface: the basic use interface, which show the electric and gas change with time, and provide the button provide the option for change the budget, change the default password and check bill etc.

The manager interface provide the option for manage user, send bill and etc. and the other detailed designs are shown in the picture.

3.4 Design principle applied

3.4.1 Open-Closed Principle (OCP)

The open-closed principle means that software modules is open for extension and closed for modification. This requires the module behave in new way when the requirement are changing. For example, we add change budget by method rather than change the whole structure of the code. The basic structure is settled and the requirement is adding accordingly.

3.4.2 Dependency-Inversion Principle (DIP)

It can be stated as: high level modules should not depend on low-level modules; instead, both

should depend on abstractions. In order to apply this principle, we defined the abstractions we will need and use, the codes are through variables of an interface type is abstraction with what interface type defines.

3.4.3 Interface-Segregation Principle (ISP)

This principle means that the user should not be forced to use the methods that they don't need. We use the ActionListener as interface who listen to button that only when users click with requirement.

3.4.4 Single-Responsibility Principle (SRP)

Every object in a system should have a single responsibility, and all the object's services should focus on carrying out that single responsibility.

In our program, we design our classes to obey this principle. For example, we have two classes named ERecord and GRecord, they all inherent the Record class, and implement the RecordControl interface.

3.4.5 The Liskov Substitution Principle (LSP)

The LSP is "methods should not be overridden in a way that changes assumptions about their behaviour".

Though we may not use that frequently in our program, we meet the principle by the appropriate rewriting existing codes in some parts that need to be re-implemented.

4.Implementation and Testing

4.1 Integrate build plan

For the whole system should be built up step by step, a construction should involve several functions that contribute to integration of a subsystem.

In the first iteration, the basic functions such as verify current energy consumption and cost, check consumer information includes name，ID password and email, and let provider search consumer.

In the second iteration, we expect to implement functions such as consumers login their own system, provider add or remove consumer while add or remove the consumer's information.

In the third iteration, we plan to focus on tariff that provider can set tariff and consumers can check tariff but can't modify, then we add a function that consumer can modify their own

passwords.

In the fourth iteration, we want to add functions such as consumers can view historic energy consumption, and costs regularly and set budgets for both electricity and gas, and get alert.

In the final iteration, we refine the previous subsystem and add some functions such as provider can view the historic readings and bills of every consumer, system sends meter reading to provider and provider sends monthly bill to consumers.

Finally refine all the system.

4.2 Testing strategy

After implementation phase, we carried out several unit testing to compare the test results with expectant outputs. We assumed that almost half of the tests would be automatically carried out, such as using Junit. The rest would be manual to test whether the class is correct, which covers untrue inputs. We supposed that around 90 percent of tests passed and there was no high priority defects meant the testing is successful.

4.3 Test techniques

4.3.1 Regression Testing

Regression testing is used during each build stage to test the functions of the build. Since new functions are added to the build, the number of regression testing is also increasing. For example, we created testing cases for the first build, and then for the integrated build, we also created several testing cases. In this way, testing cases are added and we continue to run all tests for each build without being affected by build updates.

4.3.2 White-box Testing

White-box testing is carried out to test the internal program logic using Basis Path Testing and Loop Testing. Internal operations are performed according to specifications, and white-box testing tests all components.

Appendix figure 6-7

4.3.3 Black-box Testing

Black-box testing is carried out to test software requirements using Partition testing.

Partition: (separate each function as a partition)

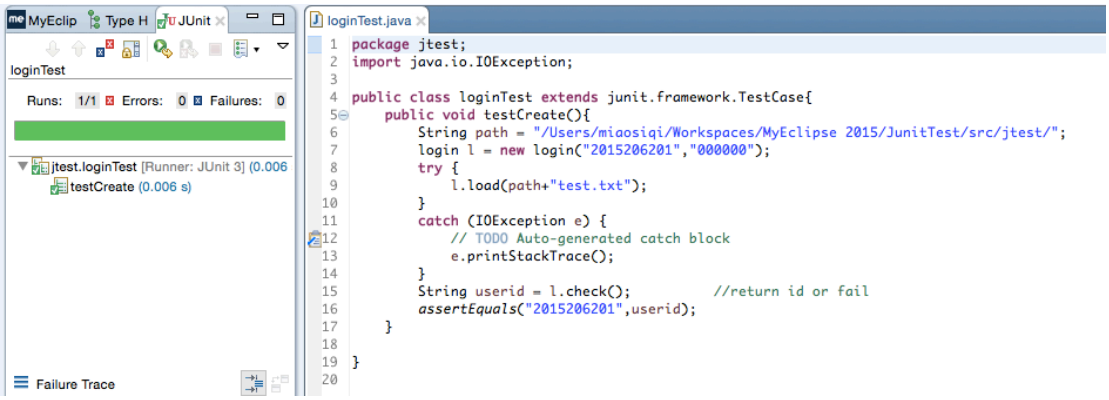
Function	Input	Output
Log in	The user enters id and password, and clicks “Login”	All information about electric and gas is shown in page.
Change budget	The user enters new budget and clicks “OK”	The budget is changed and recorded.
Check history record	The user chooses the energy and period of time, clicks “OK”	The history record is displayed in table.
Check bill	The user clicks “Bill”	The bills of every month are displayed in table.
Check Tariff	The user clicks “Check Tariff”	The current prices of electric and gas are shown.
Check Current Usage	The user enters home page	The current usage and cost (in pounds) are updated.
Modify password	The user clicks button and enters new password	The new password is recorded.
View user information	The manager clicks “Manage user”	All the user information is displayed in table.
Search user	The manager enters user id and clicks “Search”	The information of this user is shown.
Add user	The manager enters new user’s information and clicks “Add”	The new user is added and his information is recorded.

Remove user	The manager enters the user id and clicks “Remove”	The user is removed from this system.
Set tariff	The manager enters new tariff and clicks “Set tariff”	The current tariff is changed and recorded.
Check bill	The manager clicks “Check bill”	All the bills are displayed in table.

4.4 TDD

Before writing the production code, TDD is required to test function unit in system, and we use Junit as a simple unit-testing framework. We designed unit test for each class and write code to meet the specification. The following Junit code is an example test of login class.

(User id and password are transferred into login( ) method, and this method will return user id or fail. So we use assertEquals( ) to test whether the output is right.)



Success is detected and user id is matched.

Appendix

Figure 1

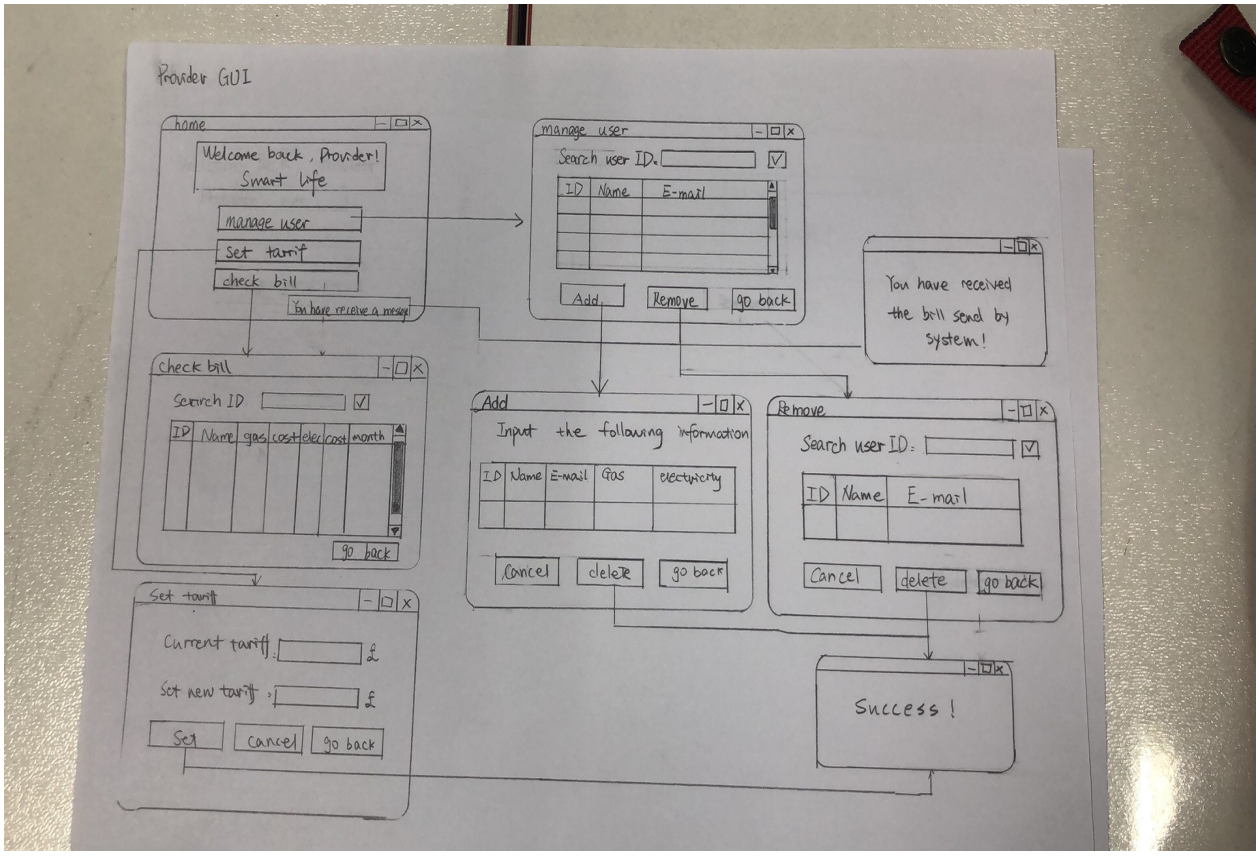
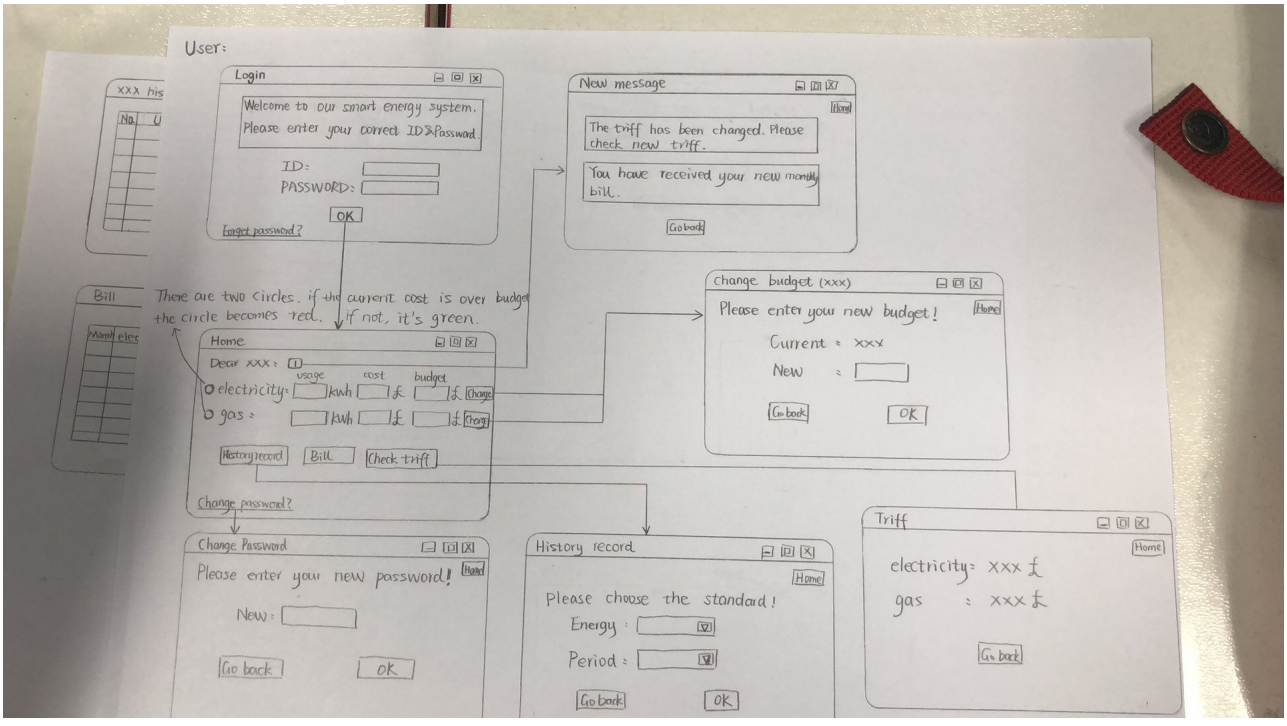


Figure 2-3



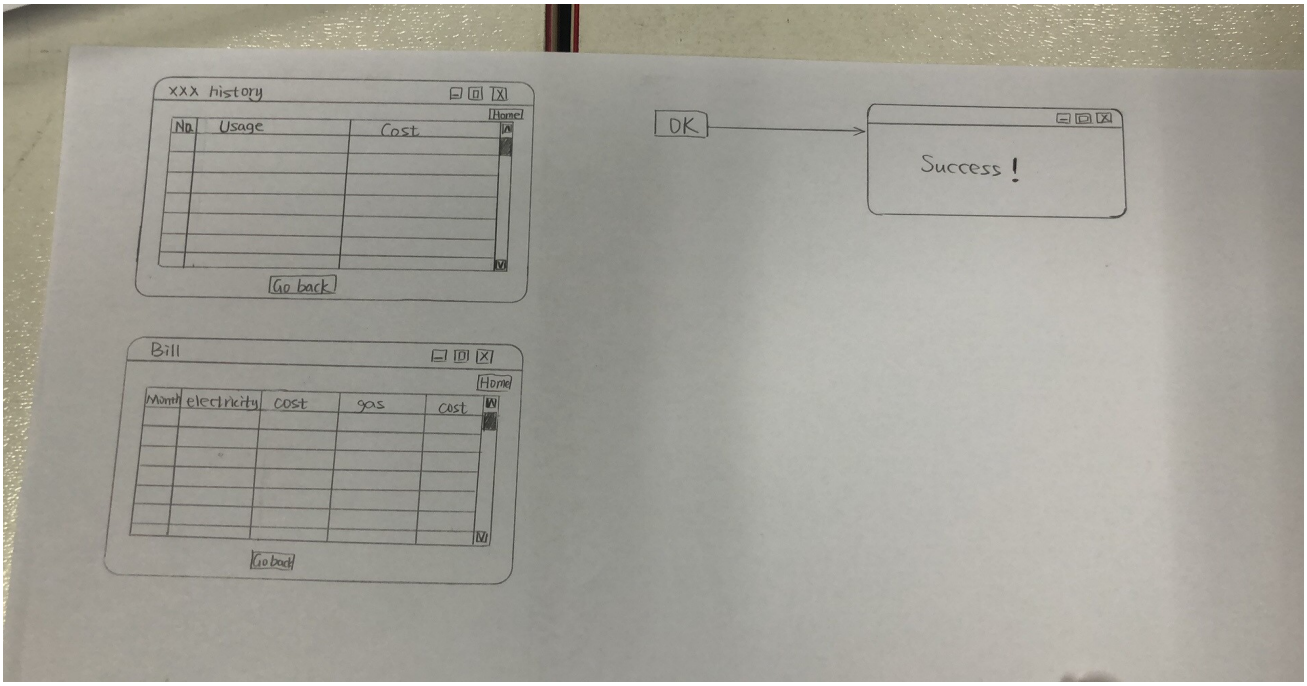

xxx history
No.
Usage
Cost
Home
Go back
OK
Success !
Bill
Month
electricity
cost
gas
cost
Home
Go back

Figure 4

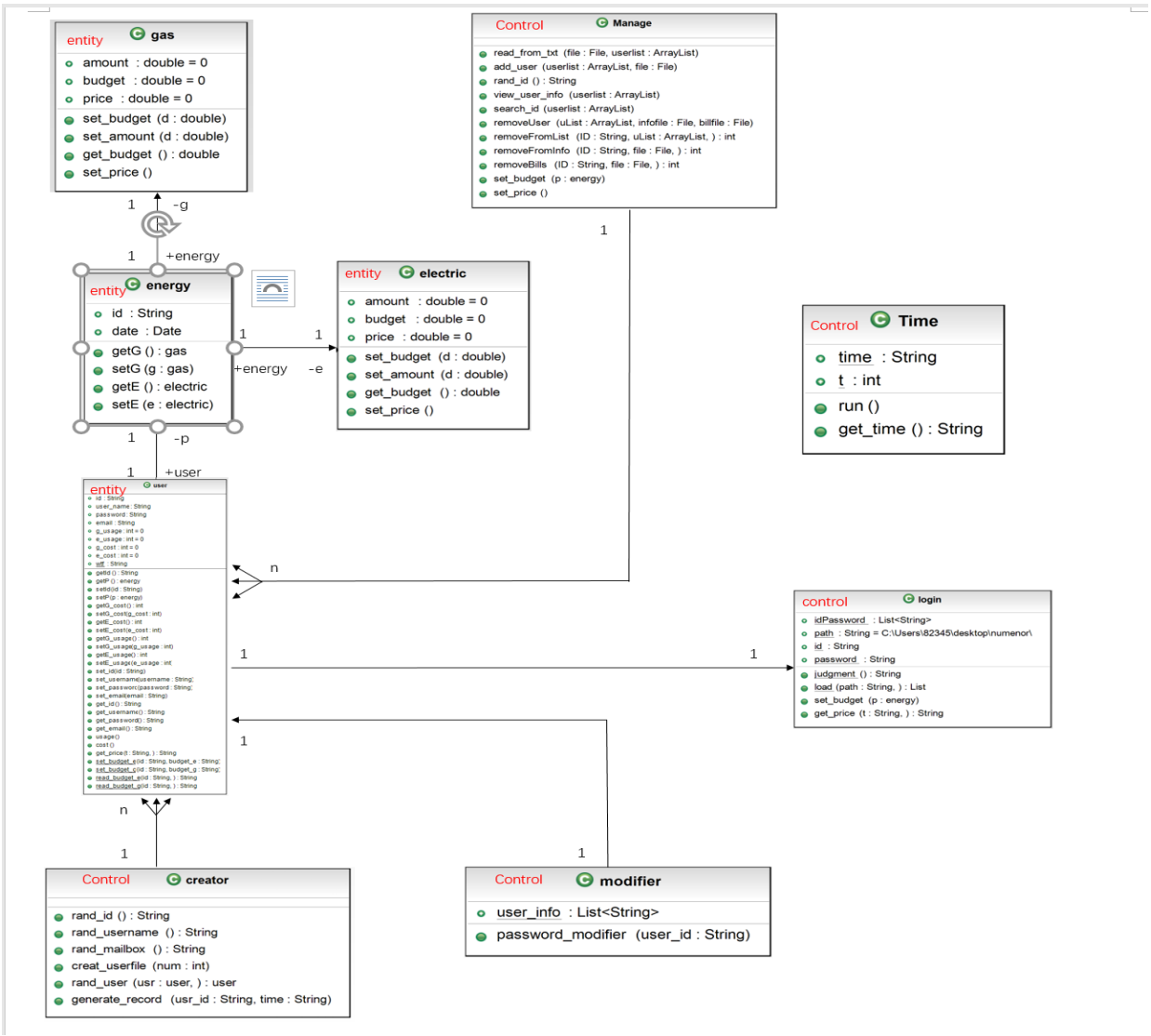

entity gas
amount : double = 0
budget : double = 0
price : double = 0
set_budget (d : double)
set_amount (d : double)
get_budget () : double
set_price ()
Control Manage
read_from_txt (file : File, userlist : ArrayList)
add_user (userlist : ArrayList, file : File)
rand_id () : String
view_user_info (userlist : ArrayList)
search_id (userlist : ArrayList)
removeUser (uList : ArrayList, infofile : File, billfile : File)
removeFromList (ID : String, uList : ArrayList,) : int
removeFromInfo (ID : String, file : File,) : int
removeBills (ID : String, file : File,) : int
set_budget (p : energy)
set_price ()
-g
+energy
entity energy
id : String
date : Date
getG () : gas
setG (g : gas)
getE () : electric
setE (e : electric)
+energy
-e
entity electric
amount : double = 0
budget : double = 0
price : double = 0
set_budget (d : double)
set_amount (d : double)
get_budget () : double
set_price ()
Control Time
time : String
t : int
run ()
get_time () : String
-p
+user
entity user
control login
idPassword : List<String>
path : String = C:\Users\82345\desktop\numenor\
id : String
password : String
judgment () : String
load (path : String,) : List
set_budget (p : energy)
get_price (t : String,) : String
Control creator
rand_id () : String
rand_username () : String
rand_mailbox () : String
creat_userfile (num : int)
rand_user (usr : user,) : user
generate_record (usr_id : String, time : String)
Control modifier
user_info : List<String>
password_modifier (user_id : String)

Figure 5

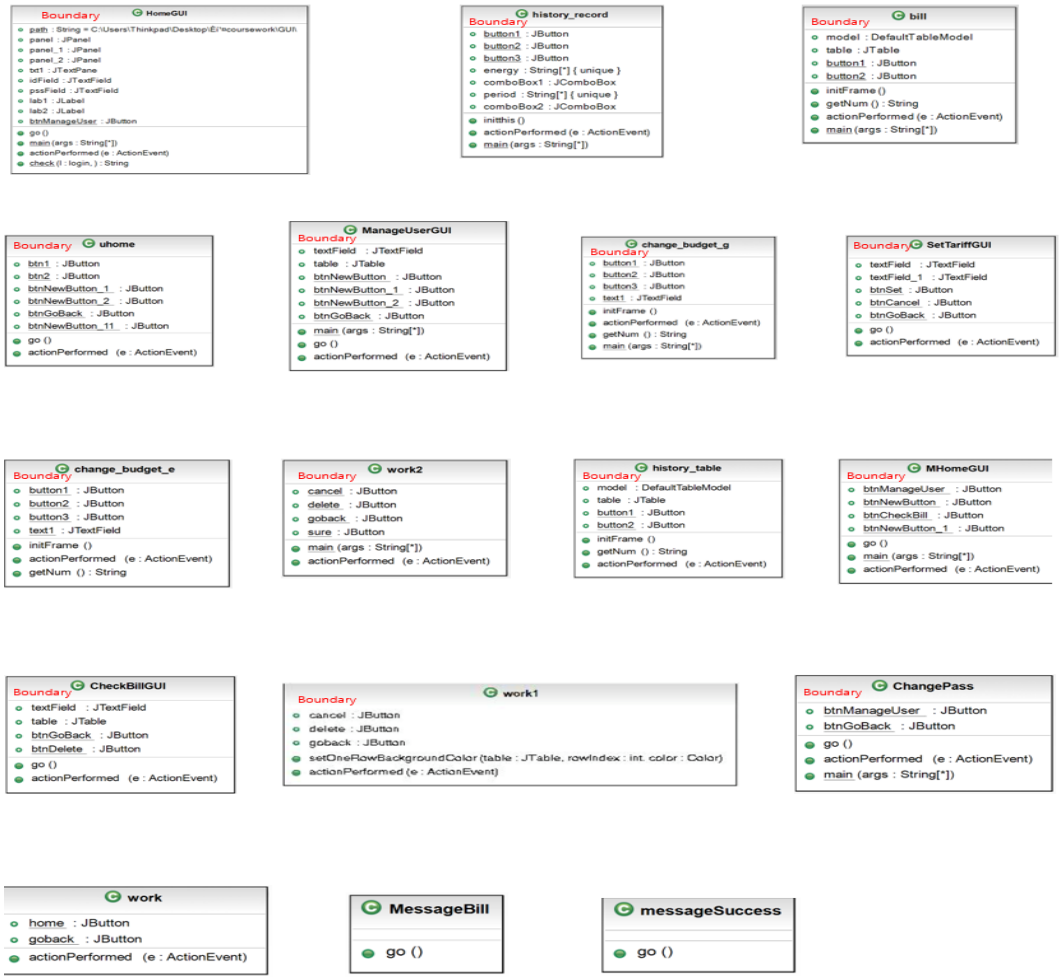
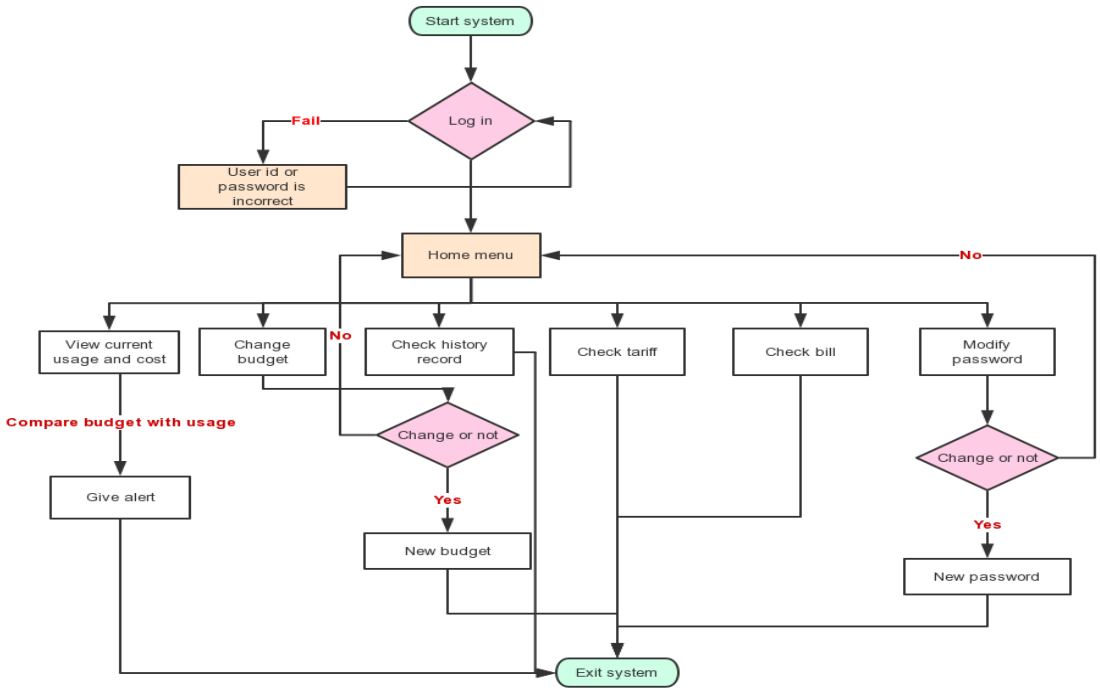
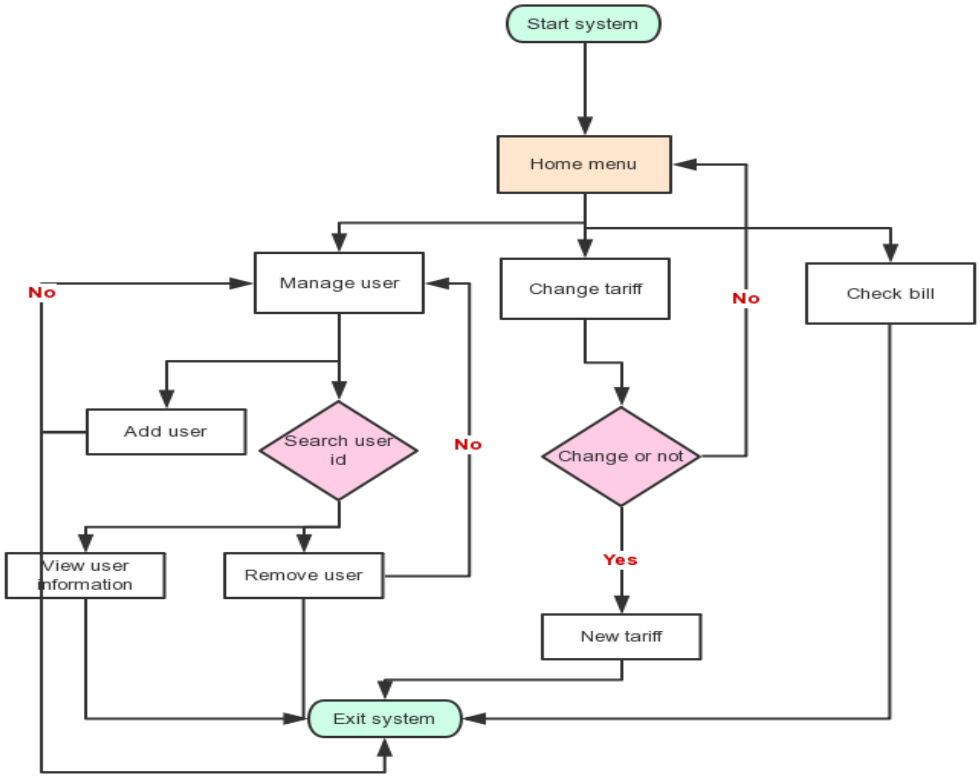

Boundary
HomeGUI
path : String = C:\Users\Thinkpad\Desktop\Ei^#coursework\GUI\
panel : JPanel
panel_1 : JPanel
panel_2 : JPanel
txt1 : JTextPane
idField : JTextField
pssField : JTextField
lab1 : JLabel
lab2 : JLabel
btnManageUser : JButton
go ()
main (args : String[*])
actionPerformed (e : ActionEvent)
check (l : login,) : String
history_record
Boundary
button1 : JButton
button2 : JButton
button3 : JButton
energy : String[*] { unique }
comboBox1 : JComboBox
period : String[*] { unique }
comboBox2 : JComboBox
initthis ()
actionPerformed (e : ActionEvent)
main (args : String[*])
bill
Boundary
model : DefaultTableModel
table : JTable
button1 : JButton
button2 : JButton
initFrame ()
getNum () : String
actionPerformed (e : ActionEvent)
main (args : String[*])
uhome
Boundary
btn1 : JButton
btn2 : JButton
btnNewButton_1 : JButton
btnNewButton_2 : JButton
btnGoBack : JButton
btnNewButton_11 : JButton
go ()
actionPerformed (e : ActionEvent)
ManageUserGUI
Boundary
textField : JTextField
table : JTable
btnNewButton : JButton
btnNewButton_1 : JButton
btnNewButton_2 : JButton
btnGoBack : JButton
main (args : String[*])
go ()
actionPerformed (e : ActionEvent)
change_budget_g
Boundary
button1 : JButton
button2 : JButton
button3 : JButton
text1 : JTextField
initFrame ()
actionPerformed (e : ActionEvent)
getNum () : String
main (args : String[*])
SetTariffGUI
Boundary
textField : JTextField
textField_1 : JTextField
btnSet : JButton
btnCancel : JButton
btnGoBack : JButton
go ()
actionPerformed (e : ActionEvent)
change_budget_e
Boundary
button1 : JButton
button2 : JButton
button3 : JButton
text1 : JTextField
initFrame ()
actionPerformed (e : ActionEvent)
getNum () : String
work2
Boundary
cancel : JButton
delete : JButton
goback : JButton
sure : JButton
main (args : String[*])
actionPerformed (e : ActionEvent)
history_table
Boundary
model : DefaultTableModel
table : JTable
button1 : JButton
button2 : JButton
initFrame ()
getNum () : String
actionPerformed (e : ActionEvent)
MHomeGUI
Boundary
btnManageUser : JButton
btnNewButton : JButton
btnCheckBill : JButton
btnNewButton_1 : JButton
go ()
main (args : String[*])
actionPerformed (e : ActionEvent)
CheckBillGUI
Boundary
textField : JTextField
table : JTable
btnGoBack : JButton
btnDelete : JButton
go ()
actionPerformed (e : ActionEvent)
work1
Boundary
cancel : JButton
delete : JButton
goback : JButton
setOneRowBackgroundColor (table : JTable, rowIndex : int, color : Color)
actionPerformed (e : ActionEvent)
ChangePass
Boundary
btnManageUser : JButton
btnGoBack : JButton
go ()
actionPerformed (e : ActionEvent)
main (args : String[*])
work
home : JButton
goback : JButton
actionPerformed (e : ActionEvent)
MessageBill
go ()
messageSuccess
go ()

Figure 6-7


Start system
Log in
Fail
User id or password is incorrect
Home menu
No
View current usage and cost
Change budget
No
Check history record
Check tariff
Check bill
Modify password
Compare budget with usage
Change or not
Change or not
Give alert
Yes
Yes
New budget
New password
Exit system


Start system
Home menu
Manage user
Change tariff
Check bill
No
No
Add user
Search user id
No
Change or not
View user information
Remove user
Yes
New tariff
Exit system