SOFTWARE ENGINEERING

A Campus Scooter Sharing System

GROUP 84

Summary of Responsibilities and Achievements

Group Number: <84>

Group Members:

Name	QM Student Number	BUPT Student Number
Yinze Li	161189147	2016213609
Fan Ding	161188243	2016213516
Shiwen Zou	161188966	2016213591
Zijing Tian	161187800	2016213469
Pengran Wang	161189169	2016213611
Yu Sun	161189228	2016213617

I. Individual member: Yinze Li (Leader)

Individual member contribution:

- **Code:** Control classes and GUIs of user enrollment and management
- **Report:** Chapter 3 Analysis and Design

II. Individual member: Fan Ding

Individual member contribution:

- **Code:** Entity classes of docks and slots, main GUI and function selection GUIs
- **Report:** Integration and Layout

III. Individual member: Shiwen Zou

Individual member contribution:

- **Code:** GUIs of showing docks and slots state, querying usage information and sending email
- **Report:** Chapter 4 Implementation and Testing

IV. Individual member: Zijing Tian

Individual member contribution:

- **Code:** GUIs of picking up and returning scooters
- **Report:** Chapter 3 Analysis and Design

V. Individual member: Pengran Wang

Individual member contribution:

- **Code:** Entity classes of users and transactions
- **Report:** Chapter 0 Introduction

  Chapter 1 Projectment

  Chapter 2 Requirements

VI. Individual member: Yu Sun

Individual member contribution:

- **Code:** GUIs of user logging and fine payment
- **Report:** Chapter 4 Implementation and Testing

Table of Contents

Chapter 0: Introduction

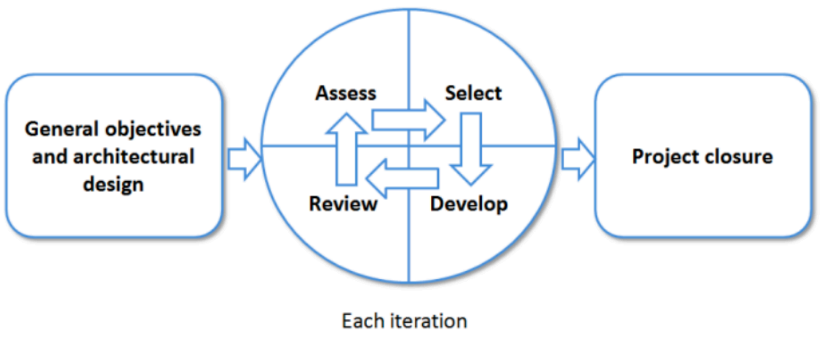
Our agile software team developed a scooters sharing system for QM students and staff. This system is used in scooter docks and manager computers. After QM users register, they can borrow and return scooters at docks through the system. And system manager can use system to register new users and monitoring all the current state of docks and usage. They also can get users' information and send email to users to inform them with usage information.

This report will introduce the whole process of agile software development. The system is developed by agile method with four iterations. All steps of development is run by cooperation of six group members. The planning, analysis and design are defined on teamwork meetings on which we share our ideas and communicate with each other.

Chapter 1: Project Management

1.1 Project plan

1.1.1 Scrum approach



(Figure 1: Scrum process)

The Scrum approach is a general agile method but its focus is on managing iterative development rather than specific agile practices.[1] There are three phases in Scrum---Outline planning, Sprint cycle and Project closure. And our Agile Project Management uses Scrum approach.

1.1.2 Outline planning and architecture design

The objective of the scooter sharing system we design is to build a system for QM users to use free scooters in QM and managers to monitor and manage system conveniently.

The users are QM registered users and managers. Each QM registered users can borrow or return scooters at dock A, B or C in QM. And if they use scooters overtime, their ID will be banned automatically. Only they pay the fine by system can they use scooters again. Each manager can use system to monitor the current state of each dock and usage of scooters. All usage information will be recorded in system. Managers can get information of each registered users. Then a manager has authority to register, ban, release and delete a user ID directly. And they can send email to users to inform them with their details.

About the architecture design, we divide it into three phases. First one is boundary class which is used to interact with users. Second one is entity class is used to build the relation. And final one is control class which is used to control internal interaction of system.

1.1.3 Scheduling

Activity	Contents	Duration (days)	Dependencies
T1	Overview the whole project	2	
T2	Risk analysis and management	1	T1
T3	Requirement analysis	1	T1
T4	Paper prototype design	1	T3
T5	Identify entity, boundary and control class.	1	T3, T4
T6	Identify class relationship and build class diagram	2	T5
T7	Analyze attributes and constraints	2	T5, T6
T8	Designing architecture	1	T5, T6
T9	Build class diagram	1	T5, T6
T10	Plan and design testing	10	T5-T9
T11	Test implementation	10	T10
T12	Pair programming from design to code	20	T5-T9
T13	Integration system and evaluate testing	10	T12
T14	Writing report	5	T1-T13

(Chart 1: Gantt task chart)

1.2 Sprint Cycle

Sprints are fixed length, normally 2 to 4 weeks. The starting point for planning is the product backlog. And at the end of the sprint, the work done is reviewed and presented to stakeholders. The next sprint cycle then begins.[1]

1.2.1 Iterations

The project should be finished on May 31, 2019. Our agile software development team has six staffs. And we are divided into three sub-groups to pair programming. Through analysis, we decide to divide project into four iterations.

1.2.2 Selecting and making decisions

In selection phase involves the team leader who also can be called "Scrum master" [1] to read the handout to select the features and functionality to be developed during the sprint. After each time of selection, the whole time will attend a short stand-up meeting to analysis about the selection. We talk about the reasonability of features and functionality and decide whether to build. Finally the tasks will be assigned to sub-groups to develop.

1.2.3 Adapting to change

Any changes occurring, "Scrum master" will communicate with other staffs. On the meeting, agile team decide to how to deal with the changes.

1.3 Project Closure

This is the last phase of Scrum approach, we finish the project and complete the project report.

1.4 Agile Teamwork

The six staffs of group 84 are divided into three sub-groups, and finish software development by pair programming. Then we attend the group meeting frequently.

(Figure 2: Teamwork)

1.5 Risk Management

Risk management is the process of identifying possible failures, establishing where alternative plans should be developed. And risks can be divided into project risks, product risks and business risks.[1]

- **Project risks**

Possible risks	Probability	Solutions
Skilled staff leave	High	Pair programming
Time is not enough for some changes	High	Finish the iteration ahead of time in plan
Code and record lose	Low	Make copies

(Chart 2-1: Risk Management)

- **Product risks**

Possible risks	Probability	Solutions
User interface proves unattractive to customers	Low	Changing GUI according to feedback from demo

Serious design error discovered during implementation	Low	Allocate some for changing at a large scale

(Chart 2-2: Risk Management)

- **Risks which are both Project and Product Risks**

Possible risks	Probability	Solutions
Larger number of changes to requirements	Low	Allocate some for changing at a large scale
Implementation reveals the team are lacking in skills	High	Study the skill immediately and individually

(Chart 2-3: Risk Management)

- **Business Risks**

No business risks in our project.

1.6 Quality Management

Quality management is to ensure the high quality software[1], and in agile approach to software process, it emphasis on developing software by continuous release of new versions with extra features.

For design and programming, we ensure:

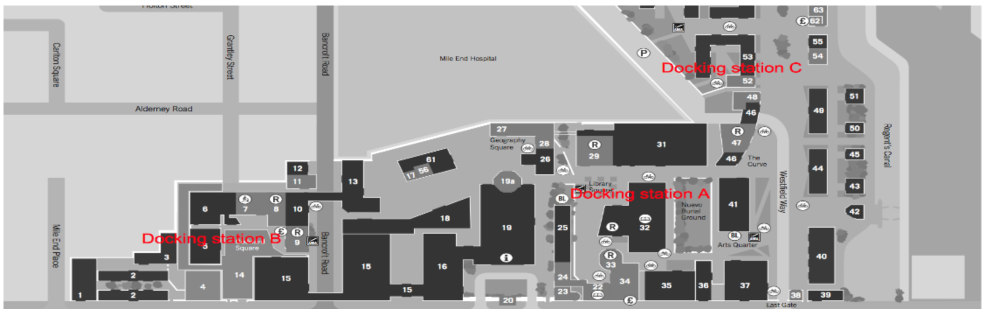
- Assessing the quality of system in users' view.
- It should be efficient and easy to use.
- It should be easy to modify to meet changing requirement.
- Paying attention to internal quality code in terms of codes to get a good structure.
- It should be "fitness for purpose" rather than specification conformance.

Chapter 2: Requirements

2.1 Fact-finding Techniques

2.1.1 Background reading

Our agile software development team develops the scooters sharing system for QM. And we got the handout about this project on QMplus. We also found the details of Mile End Campus on QMplus too.[2] Then we could know functional and non-functional requirements about the system.



(Figure 3: map of QM)

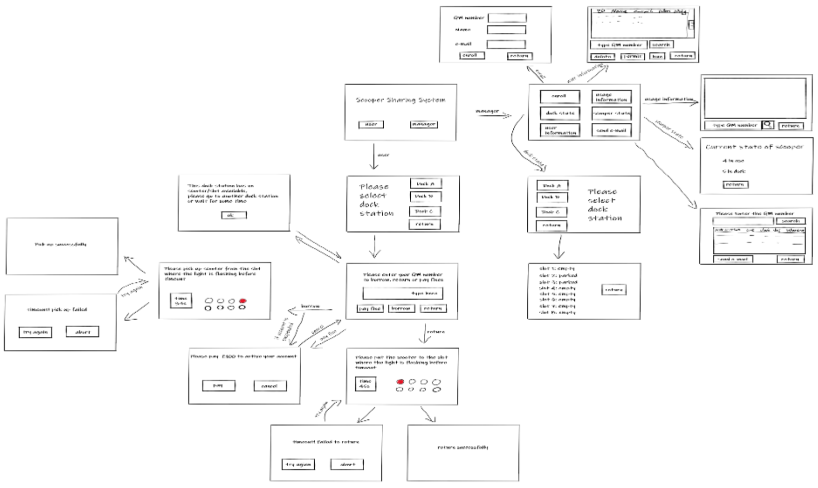
2.1.2 Interviewing

We interviewed the students in BUPT. We asked them that if there would be scooters sharing system in BUPT, what do they want? Then we got some basic all round needs from interviewer. We knew how to design GUI will be attractive to users, how to pay the fine will be convenient and so on.

2.1.3 Observation

We want to get a deep knowledge of the system. We observed the current cycle sharing system in China---mobile sharing cycling. Although it was a little different from the system we designed. The part of registering and borrowing or returning cycles gave us some ideas.

2.2 Paper Prototype



(Figure 4: Paper Prototype)

2.3 Product Backlog Changes

After first submission of backlog, we find that though our backlog is complete in functional requirements, it is not refining enough and lacks some details of non-functional requirements. Because we think that these details are obvious and do not add them in backlog at first submission. Then we add them to backlog now.

2.3.1 Functional requirements

- We add light function requirements when a scooter is to be picked up or returned. The flashing light will be closed after user picks or returns.
- We point out that only the students and staffs in QM who have registered in system can use the scooters sharing system.
- We refine the function about pick up or return scooters and pay fine.
- We add the user ID check when they want to register. If they want to use a QM number to register more than one user ID, which will be not allowed.
- We add some demand on timers. We need two timers to count the usage time for single time and whole day.
- We add some function for managers. They can delete, ban and release the user ID directly.

2.3.2 Non-functional requirements

- We add that the system should be developed by Java and it should be a stand-alone application.
- We add that the system should use files to store data, in simple text file format. And we can use txt, CSV or XML.
- We add that the system cannot use database.

2.3.3 Acceptance criteria

We add more acceptance criteria on legality verification function with more detail format demand.

2.4 Iteration and Estimation of the Stories

According to agile requirements, we should develop some functions in each iteration and it should be run successfully. At early iteration, we should develop the essential function with high prioritization. Then according to prioritization of stories, the project is divided into four iterations.

2.4.1 Iteration 1

In first iteration, we mainly design the system framework and implement some essential functions --- borrow or return scooters, light flashing function, enrollment function, legality verification, error checking when input information and development environment requirement. At the same time, we design the GUI.

2.4.2 Iteration 2

In second iteration, we implement one minute countdown function, screen function with friendly hint, timing function to record usage time, payment function to pay fine and the function of monitoring docks.

2.4.3 Iteration 3

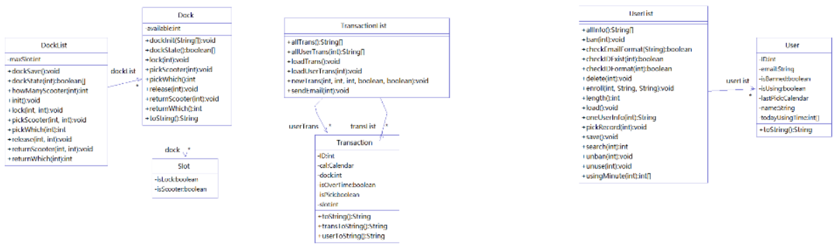
In third iteration, we start to implement extra functions --- some notification about user need to register, user need to pay the fine, dock has no scooters and dock has no slot available, prohibiting function, manager operation function and inquiring users' information for managers.

2.4.4 Iteration 4

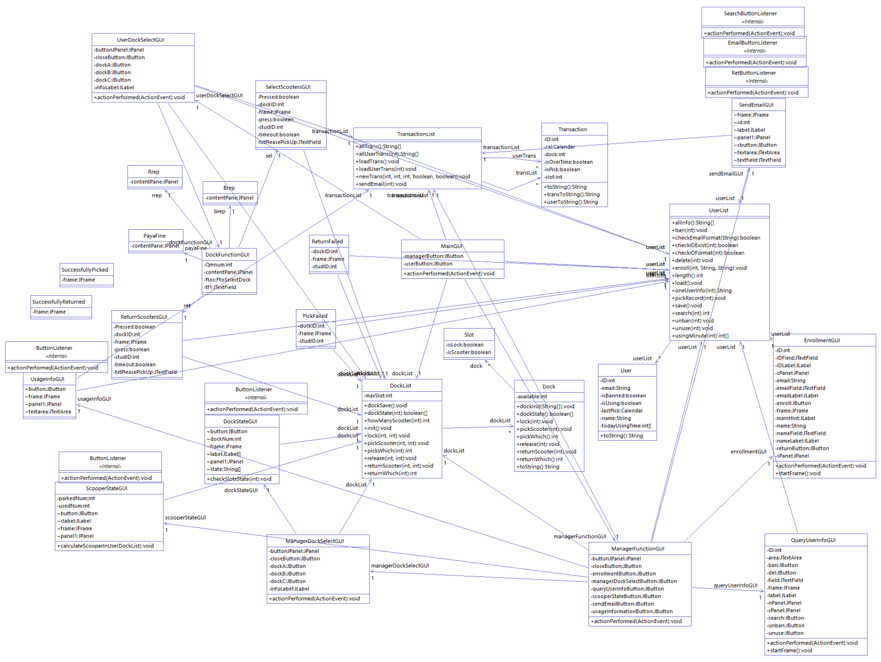
In the fourth iteration, we design the less important functions --- email function, friendly interaction function and future developing demand.

Chapter 3: Analysis and Design

3.1 Analysis



(Figure 5: Class diagram of entity and control classes)



(Figure 6: Class diagram of all classes)

3.1.1 Class relationship

We divided all the classes into three sorts: entity, control and boundary.

- Entity: used to model information that is long-lived and persistent, such as Class Slot, Dock and User.
- Control: used to model the interaction, concretely speaking, in this system, control classed receive user’s interaction, do some calculation and control the data in file. There are three control classes, UserList, DockList and TransactionList.
- Boundary: used to encapsulate control and coordination of the main actions and control flows. In this system, boundary classes are GUIs. We create a class for every GUI.

The first picture shows the class diagram only contain the control and entity classes. One control class is related to many entity classes, because control classes contain an array or array list to store and control entities. For example, Class UserList have an array list to reserve all the users, and it have some methods to manage users, such as to enroll a user and to ban a user. Then boundary classes

(GUIs) interact with users and send command to control classes. The second picture shows the class diagram of all classes.

3.1.2 Reusability of software components

There are three main models in the system, there are user model, slot & dock model and transaction model. User model can be revised and adapted to other system such as bank system. Transaction model can used for library system for recording borrowing & returning. Correspondingly, GUIs such as EnrollmentGUI and UserManagementGUI also can be used in other system with Class User and UserList.

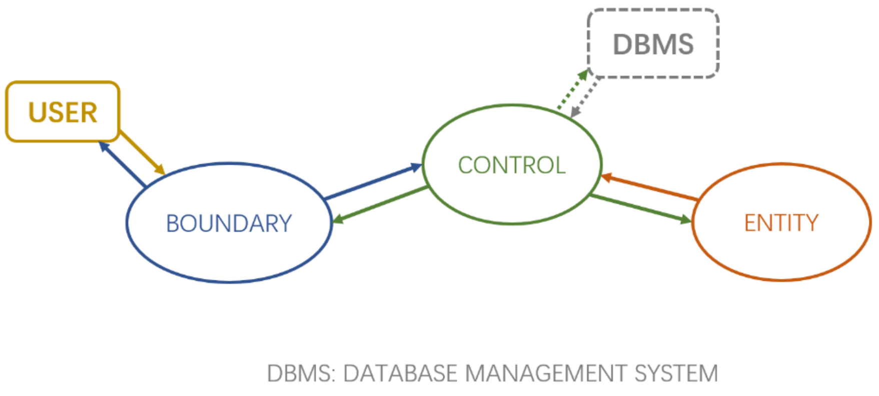
3.1.3 Adaptable to change

The system has some degrees of ability to adapt to prospective changes. For example, all the information of slots and docks is stored in a file, if the client want to establish a new dock station or add some new slots in a current dock station, you need only to revise the file not the code. Then when you boot the system, the program can recognize the new slots or docks automatically.

3.2 Design

3.2.1 Design patterns

We focus on creating a software system based on highly interacted three types of classes functioning in strong cooperation. We classified the classes into Entity Class, Boundary Class and Control Class. The relationship between three types of classes, data management systems and users are shown below.



(Figure 7: class relationship)

Boundary classes are specified for interactions with mankind, including customers, manager and other not-targeted users. Boundary classes includes GUI classes and UI classes.

Entity classes are used to model long-lived and persistent information, including data structures. Entity classes are responsible for the getters and setters of those significant information.

Control classes are used to encapsulate control and coordination of the main actions and control flows. Control classes have the right to communicate directly with entity classes and play a role of intermediary when boundary classes try to reach or modify the information stored by the Entity classes. Control classes also communicates with database management systems however, **DBMS are not allowed to use in this project**, so we replace the storage system with manually writing into CSV documents.

3.2.2 Class design

As we discovered our backlogs and reach the agreement of achievement, we draw the diagram of classes, and figure out the class type and operations of each class (Figure 6: Class diagram of all classes) .

TYPE	CLASS	OPERATION
Control	DockList	To manage and implement operations towards docks, including saving, loading, etc.
	TransactionList	To manage and implement operations towards transactions, including creating, saving, loading, etc.
	UserList	To manage and implement operations towards users, including creating, saving, loading, etc.
Entity	Dock	To store the information of each dock.
	Slot	To store the information of each slot.
	Transaction	To store the information of each transaction.
	User	To store the information of each user.
Boundary	Brep	To remind users they have repeatedly borrowed.
	DockFunctionGUI	To enter the QM number and then choose next operations.
	DockStateGUI	For manager to check the state of each slot that if they are occupied or empty in the dock.
	EnrollmentGUI	For user to input information for enrollment.
	FtoFelectDock	To remind users that the dock has no scooter or empty slot available.
	MainGUI	To start the whole system.
	ManagerDockSelectGUI	For manager to select a target dock.
	ManagerFunctionGUI	For manager to select a management function.
	PayaFine	For users to pay fines.
	PickFailed	To inform users that a fail occurs when picking up a scooter.
	QueryUserInfoGUI	To query the information of users.
	ReturnFailed	To inform users that a fail occurs when returning a scooter.
	Rrep	To remind users they have repeatedly returned.
	ScooterStateGUI	For manager to check the number of scooters that are used or available.
	SelectScootersGUI	For user to pick or return a scooter.
	SendEmailGUI	For manager to search for a user's transaction and send an email to the user.
	SuccessfullyPicked	To inform users that a scooter has been picked up successfully.
	SuccessfullyReturned	To inform users that a scooter has been returned successfully.
	UsageInfoGUI	To display all scooters' usage information to the manager.
	UsageInformationGUI	To display the usage information of a user.
	UserDockSelectGUI	For user to select a dock to do further operations.

(Chart 3: design principle)

3.3 DESIGN PRINCIPLES

SRP

SRP suggests that each object has a single responsibility in a system, and all the object's services should only carry out that single responsibility. We follow SRP through all the coding, for example, our GUI objects would only create a window of itself, and every new window with different function would be created by other objects.

OCP

OCP leads to extensible but unmodifiable system modules, to make a module behave in new without changing any code. Our system modules are well-designed to meet extension while any modification is forbidden. We can modify the number of slots in a dock, the number of total docks by one single management operation.

DRY

DRY stands for cutting out those similar codes in system modules. We understand the negative effect that simply copy some part of the written codes in reaching of same function. Therefore, at the very beginning of coding, we set up standardizations of how classes communicate. We define all those methods beforehand, and we call out those methods when necessary.

DIP

Abstractions and details should all depend on abstractions. This is one of the main thoughts of DIP. In fact, we insist in using interfaces through the whole system, and ensuring that major classes are operating under the wide use of interfaces.

ISP

ISP requests the use of a small interface type which has fewer methods in it. Most of our interfaces contains only one method.

LSP

LSP suggests that Inheritance should only weaken preconditions and strengthen postconditions. Our implementation in software coding always ensures that our preconditions would not be extended,

Chapter 4: Implementation and Testing

4.1 Implementation

4.1.1 Integration build plan

Planning builds: To ensure the successful implementation of the whole system, we implement our complete use cases in several builds. Each build adds functionality to the previous one and contains a few new or refined components so that to be manageable and has not too many integrations or test problems. We have built during the development, which are shown in the chart below. These builds are set according to their priority. This incremental approach helps to demonstrate system features. This table actually helps to make the testing process easier.

Build	Functionality	Implementation
01	Read student ID when pick/return the scooter	DockFunctionGUI, Dock, DockList

	Light at the slot when pick/return the scooter	SelectScootersGUI, ReturnScootersGUI
	Enrollment and record the information	EnrollmentGUI, User, UserList, Transaction, TransactionList
02	Pick/return after scanning the ID card	SelectScootersGUI, ReturnScootersGUI, UserList, TransactionList
	Screen display the message and countdown timer	SelectScootersGUI, ReturnScootersGUI
	One minute countdown when pick/return a scooter	SelectScootersGUI, ReturnScootersGUI
	Notification of no available scooter/slot	DockFunctionGUI, FtoSelectDock, DockList, Slot
	Legality verification	EnrollmentGUI, UserList, User
	Timing the usage of users	SelectScootersGUI, ReturnScootersGUI, UserList, TransactionList
	Pay for the fine Prohibited the user who is banned	PayaFine, UserList DockFunctionGUI, UserList
	Monitor the record of the dock and scooters usage	ManagerFunctionGUI, ScooterStateGUI, DockStateGUI, ManagerDockSelectGUI, DockList, TransactionList
	User ID check	DockFunctionGUI, UserList
03	Lock the slot after returning/picking the scooter	SelectScootersGUI, ReturnScootersGUI, Slot
	Query and check information about the users	UsageInfoGUI, QueryUserInfoGUI
	Manager’s operation on user ID	ManagerFunctionGUI
04	Sending email to users	SendEmailGUI, UserList, TransactionList
	Friendly GUI for guiding	Brep, Rrep, PickFailed, ReturnFailed, SuccessfullyPicked, SuccessfullyReturned
05	Integration	Integration

(Chart 4: build plan)

4.1.2 Pair programming

Our teammates were divided into three teams to finish the code. Both team 1, 2 and 3 took part in the four iterates. The code of GUIs, functions, error checking and test were distributed in average. Working in pairs and sitting together to develop code helps to develop common ownership of code and spreads knowledge across our team. We put our heads together and sharpen our teamwork skills.

4.2 Testing

The aim of testing our software is to find errors from implementation stage before delivered to end users. We wrote tests prior to codes during each build using TDD method and testing occupies a long time during our whole process (approximately 45%).

4.2.1 Test Strategy

We tested each software build including internal builds, intermediate builds and final customer builds. Besides, all build tests are implemented from two aspects: validation testing (shows intended outputs) and defect testing (expose defects on purpose). Our policy of choosing to test is selecting all individual functions, combinations of functions accessed through menus and functions required user's inputs to test. In order to run, we obeyed testing model: firstly designed test cases and prepared test data, then run program with test data and compare results. If 90% of test case passed, the testing effort is successful. Furthermore, our testing process also conforms to unit testing, system testing and acceptance testing.

4.2.2 Test Techniques

We test the software from two levels: firstly at component level and then integration level. There are four techniques we applied into our software: regression testing and black-box testing are used for integration test, and white-box testing and object-oriented testing are used for unit test.

1. Regression testing

Since we have four builds, at each build stage, test cases will be created to test the functionality of the build. Since the development is incremental, we run all test for each build to ensure earlier functionalities are not broken by updated build.

2. White-Box testing

It is to test internal program logic, all internal statements and conditions have been executed at least once.

1) basis path testing

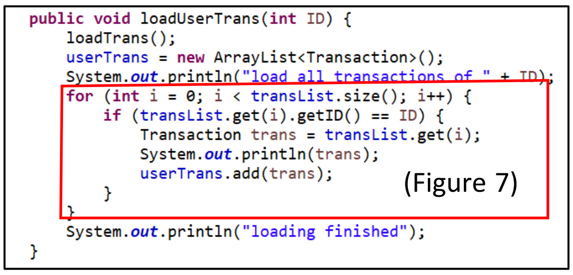
First we calculate the cyclomatic complexity to decide the number of indipendent path. Then we drive test cases to exercise all pathes to ensure all statements are examined.

2) loop testing

Different combinations of loops have different methods to examine.

a) simple loops

For example, in this module, there is simple loop. We skip the loop entirely, allow one passes,…, allow m (<transList.size()) pass,…,allow (transList.size()-1) pass, allow transList.size() pass and allow (transList.size()+1) pass to test. (Figure 7)



```
public void loadUserTrans(int ID) {
    loadTrans();
    userTrans = new ArrayList<Transaction>();
    System.out.println("load all transactions of " + ID);
    for (int i = 0; i < transList.size(); i++) {
        if (transList.get(i).getID() == ID) {
            Transaction trans = transList.get(i);
            System.out.println(trans);
            userTrans.add(trans);
        }
    }
    System.out.println("loading finished");
}
```

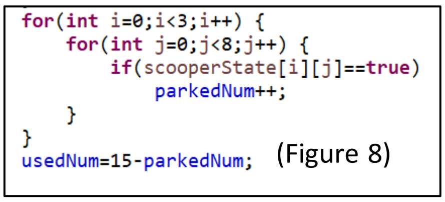
(Figure 7)

b) nested loops

Start from innermost loop and set outer loop to minimum iteration parameter value 0.

Test 1, typical value, 6,7 at inner loop and keep outer loop values 0.

Hold inner loop at typical value and outer at 1, typical, 1,2. (Figure 8)



```
for(int i=0;i<3;i++) {
    for(int j=0;j<8;j++) {
        if(scooperState[i][j]==true)
            parkedNum++;
    }
}
usedNum=15-parkedNum;
```

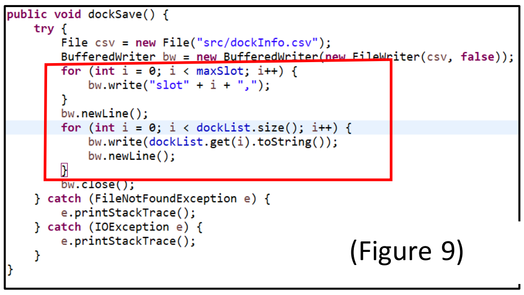
(Figure 8)

c) concatenated loops

Since the two concatenated loops are independent, we treat them as two simple loops. (Figure 9)

d) unstructured loops

We redesigned to structured loops.



```
public void dockSave() {
    try {
        File csv = new File("src/dockInfo.csv");
        BufferedWriter bw = new BufferedWriter(new FileWriter(csv, false));
        for (int i = 0; i < maxSlot; i++) {
            bw.write("slot" + i + ",");
        }
        bw.newLine();
        for (int i = 0; i < dockList.size(); i++) {
            bw.write(dockList.get(i).toString());
            bw.newLine();
        }
        bw.close();
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

(Figure 9)

3. Black-Box testing

It focuses on functional requirements and we attemp to find errors, like missing or incorrect functions, interface errors and behavior errors.

1) Partition testing：

We made equivalent partition of inputs according to their attributes, and then choose test case in each partition.

Example: *we need 9 digits QM inputs when registering.*

a) Test Case Design

- **Partition1:** <100000000

Test Case for QM number register with incorrect input

Input:

Upper edge：99999999 (<9 digit).

Result:

The system confirms the QM number doesn't exist and number of digit is smaller than 9. It will not be registered for further transaction.

Conditions:

No conditions exist.

- **Partition2:** 100000000-999999999

Test Case for QM number register with correct input

Input:

Lower edge: 100000000(9 digit); Middle value: 555555555(9 digit); Upper edge: 999999999(9 digit)

Result:

The system confirm the QM number doesn't exist and number of digit is exactly 9. It will be registered for further transaction.

Conditions:

No conditions exist.

- **Partition3:** >999999999

Test Case for QM number register with incorrect input

Input:

Lower edge: 1000000000 (>9 digit)

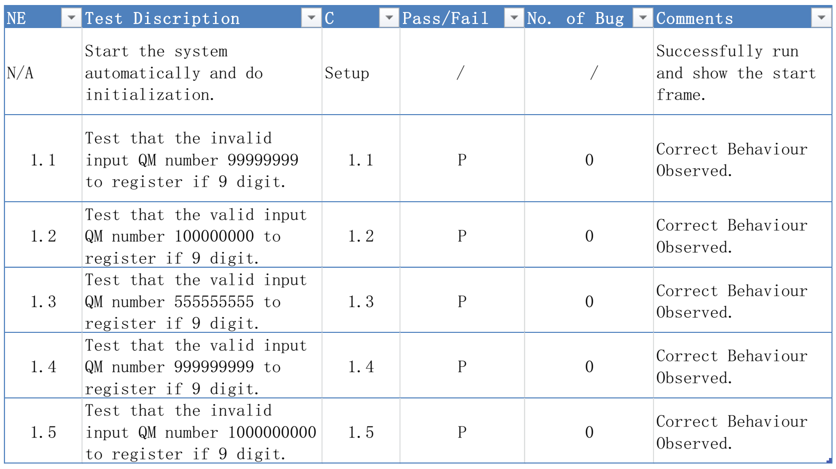
Result:

The system confirm the QM number doesn't exist and number of digit is larger than 9. It will be not registered for further transaction.

Conditions:

No conditions exist.

b) Test Matrix



NE	Test Discription	C	Pass/Fail	No. of Bug	Comments
N/A	Start the system automatically and do initialization.	Setup	/	/	Successfully run and show the start frame.
1.1	Test that the invalid input QM number 99999999 to register if 9 digit.	1.1	P	0	Correct Behaviour Observed.
1.2	Test that the valid input QM number 100000000 to register if 9 digit.	1.2	P	0	Correct Behaviour Observed.
1.3	Test that the valid input QM number 555555555 to register if 9 digit.	1.3	P	0	Correct Behaviour Observed.
1.4	Test that the valid input QM number 999999999 to register if 9 digit.	1.4	P	0	Correct Behaviour Observed.
1.5	Test that the invalid input QM number 1000000000 to register if 9 digit.	1.5	P	0	Correct Behaviour Observed.

(Chart 5: test matrix)

2) Scenario-based testing

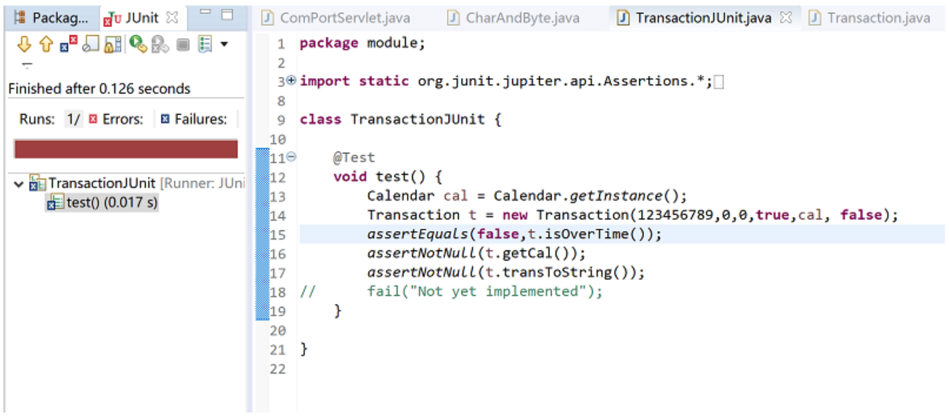
We check each requirement or user story in backlog is realized by validation testing technique. It is a system test.

4. Object-Oriented Testing

We tested individual classes and classes integration. Each test case is uniquely identified.

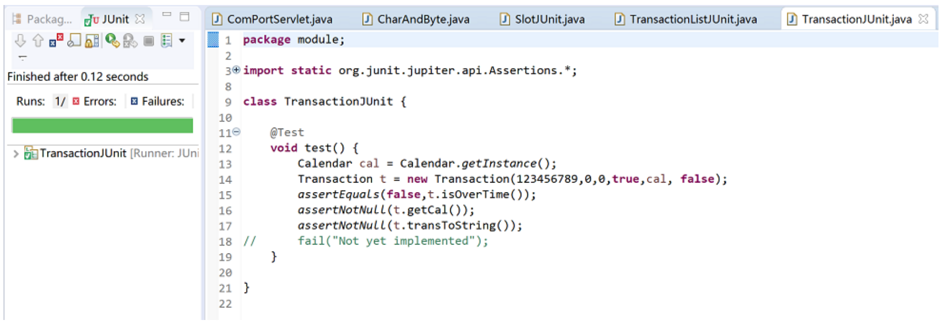
4.3 Test Driven Development

In the project development process, we applied TDD technique. In TDD cycle, we write tests prior to write the production code. And we use Junit as an unit-testing framework. During each iteration, we first decide the specification to be implemented and test the unimplemented functionality, which will demonstrate test failure. Then write code to meet the specification, so we demonstrate test success. Besides, we also unit test previous classes and methods to verify the functionalities are not broken.



```java
package module;

import static org.junit.jupiter.api.Assertions.*;

class TransactionJUnit {

    @Test
    void test() {
        Calendar cal = Calendar.getInstance();
        Transaction t = new Transaction(123456789,0,0,true,cal, false);
        assertEquals(false,t.isOverTime());
        assertNotNull(t.getCal());
        assertNotNull(t.transToString());
//      fail("Not yet implemented");
    }

}
```

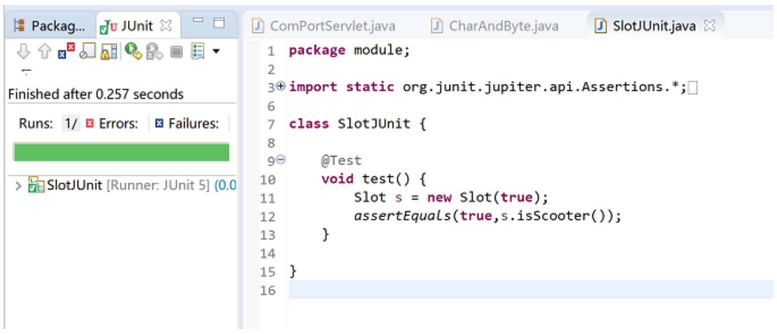
(Figure 10: Undefined Object 'Transaction' —— Failure)



```java
package module;

import static org.junit.jupiter.api.Assertions.*;

class TransactionJUnit {

    @Test
    void test() {
        Calendar cal = Calendar.getInstance();
        Transaction t = new Transaction(123456789,0,0,true,cal, false);
        assertEquals(false,t.isOverTime());
        assertNotNull(t.getCal());
        assertNotNull(t.transToString());
//      fail("Not yet implemented");
    }

}
```

(Figure 11: Defined Object, Methods And Expected Output —— Success)

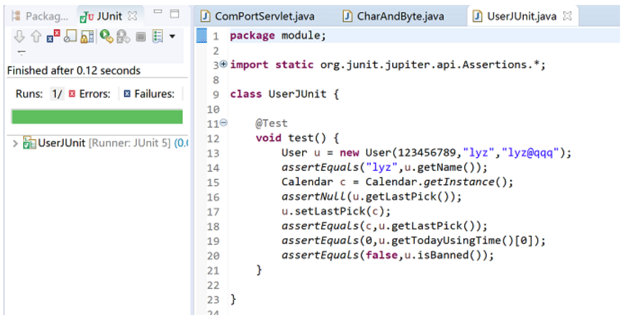
Thus, we carried out unit tests for each classes, which has functionality in requirement.



```
Finished after 0.257 seconds
Runs: 1/ Errors: Failures:
SlotJUnit [Runner: JUnit 5] (0.0
```

```java
package module;

import static org.junit.jupiter.api.Assertions.*;

class SlotJUnit {

    @Test
    void test() {
        Slot s = new Slot(true);
        assertEquals(true,s.isScooter());
    }

}
```

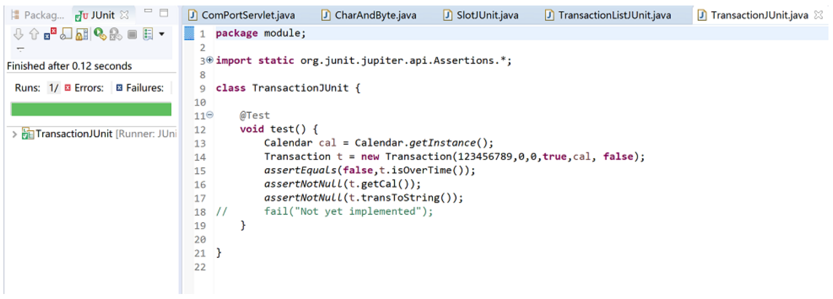
(Figure 12: Successful Unit Test for Slot Class)



```
Finished after 0.12 seconds
Runs: 1/ Errors: Failures:
UserJUnit [Runner: JUnit 5] (0.
```

```java
package module;

import static org.junit.jupiter.api.Assertions.*;

class UserJUnit {

    @Test
    void test() {
        User u = new User(123456789,"lyz","lyz@qqq");
        assertEquals("lyz",u.getName());
        Calendar c = Calendar.getInstance();
        assertNull(u.getLastPick());
        u.setLastPick(c);
        assertEquals(c,u.getLastPick());
        assertEquals(0,u.getTodayUsingTime()[0]);
        assertEquals(false,u.isBanned());
    }

}
```

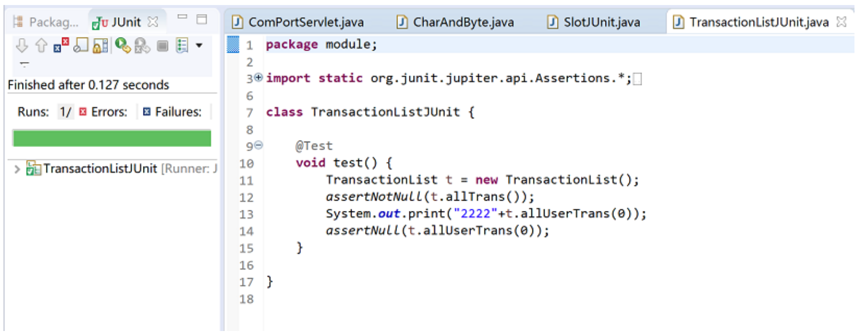
(Figure 13: Successful Unit Test for User Class)



```
Finished after 0.12 seconds
Runs: 1/ Errors: Failures:
TransactionJUnit [Runner: JUni
```

```java
package module;

import static org.junit.jupiter.api.Assertions.*;

class TransactionJUnit {

    @Test
    void test() {
        Calendar cal = Calendar.getInstance();
        Transaction t = new Transaction(123456789,0,0,true,cal, false);
        assertEquals(false,t.isOverTime());
        assertNotNull(t.getCal());
        assertNotNull(t.transToString());
//      fail("Not yet implemented");
    }

}
```

(Figure 14:Successful Unit Test for Transaction Class)



```
Finished after 0.127 seconds
Runs: 1/ Errors: Failures:
TransactionListJUnit [Runner: J
```

```java
package module;

import static org.junit.jupiter.api.Assertions.*;

class TransactionListJUnit {

    @Test
    void test() {
        TransactionList t = new TransactionList();
        assertNotNull(t.allTrans());
        System.out.print("2222"+t.allUserTrans(0));
        assertNull(t.allUserTrans(0));
    }

}
```

(Figure 15:Successful Unit Test for TransactionList Class)

Appendix

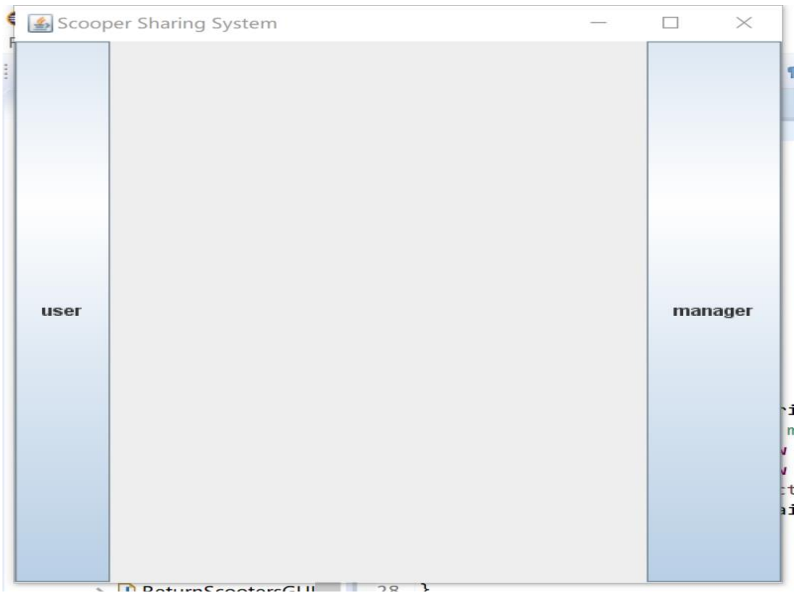
References

[1]. Software Engineering (9th Edition) by Ian Sommerville

[2]. Handouts for class sessions

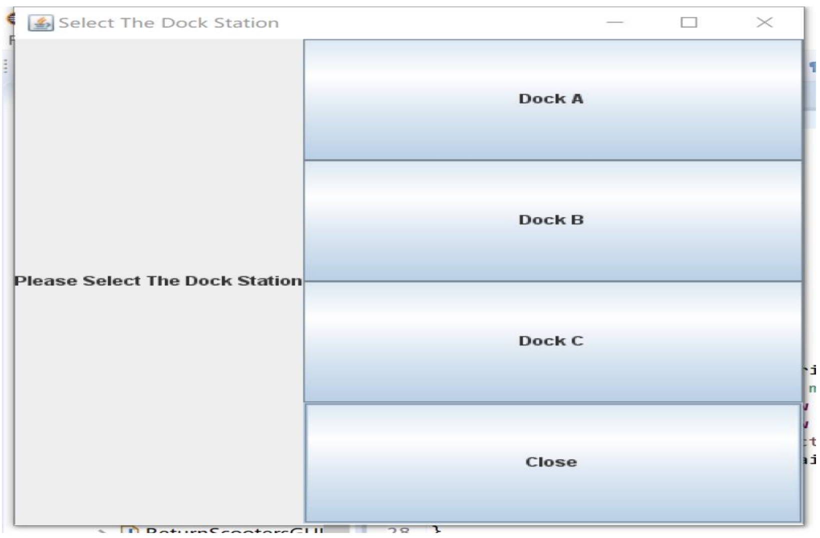
Screenshots of the system

1. In figure 1, this is the main frame of system. Choosing "user", you will go into user section, and choosing "manager", you will go into manager section.



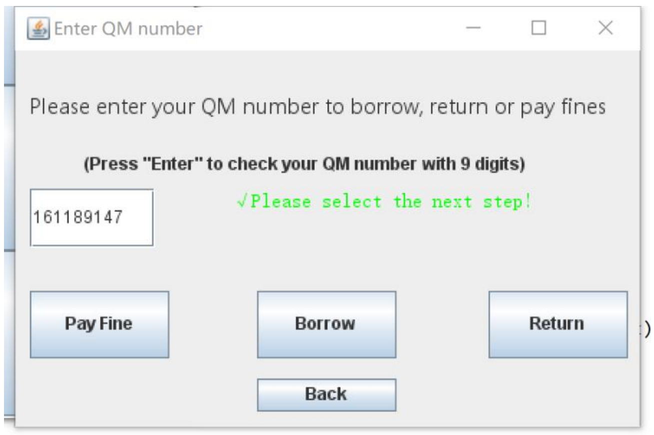
(Figure 1)

2. If you choose the user section, in figure 2, you can choose the dock you want to pick up scooters or return them.



(Figure 2)

3. After choosing the dock, in figure 3, you should input your QM number and choose pay fine , borrow or return.


Enter QM number

Please enter your QM number to borrow, return or pay fines

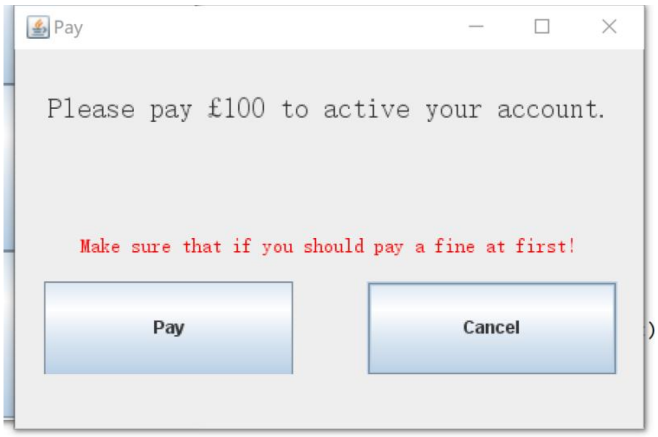
(Press "Enter" to check your QM number with 9 digits)

161189147

√Please select the next step!

(Figure 3)

4. If choosing “pay fine”, you will enter figure 4. And press button “pay”, you will pay the fine and can use scooters after banning.

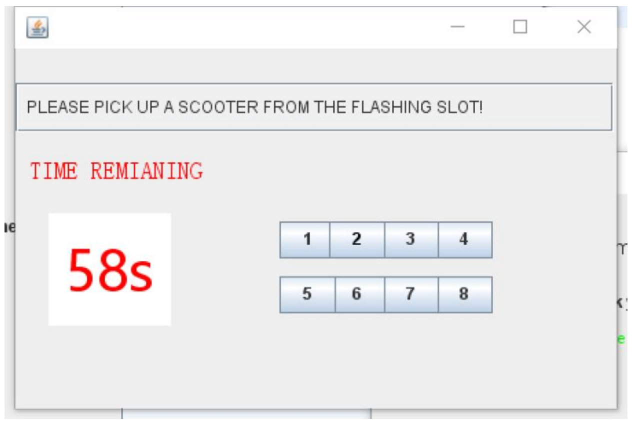

Pay

Please pay £100 to active your account.

Make sure that if you should pay a fine at first!

(Figure 4)

5. If choosing “borrow”, and if your user ID is being banned, you will enter figure 4 again to pay the fine. If not, you will go into figure 5. There are 8 LEDs representing slots. And there will be a LED that is flashing. If you press the flashing LED, the slot will be unlocked and you can pick up scooter. But you should pick it up before the timeout and there is a screen that displays time from 60s to 0s.

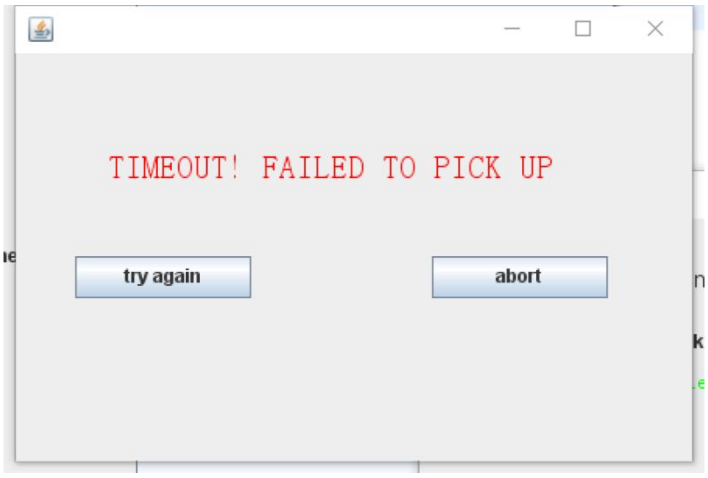

PLEASE PICK UP A SCOOTER FROM THE FLASHING SLOT!

TIME REMIANING

58s

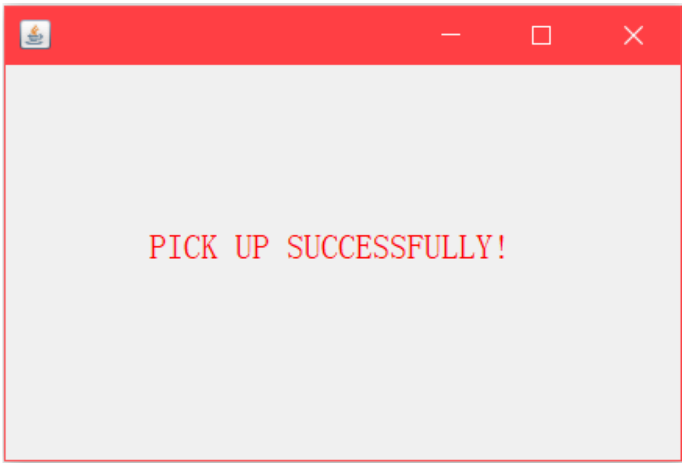
(Figure 5)

6. If you do not pick up scooter before timeout, you will enter figure 6. It shows “timeout! pick up failed”. And you can press “try again”, you will borrow scooters at next time.


TIMEOUT! FAILED TO PICK UP

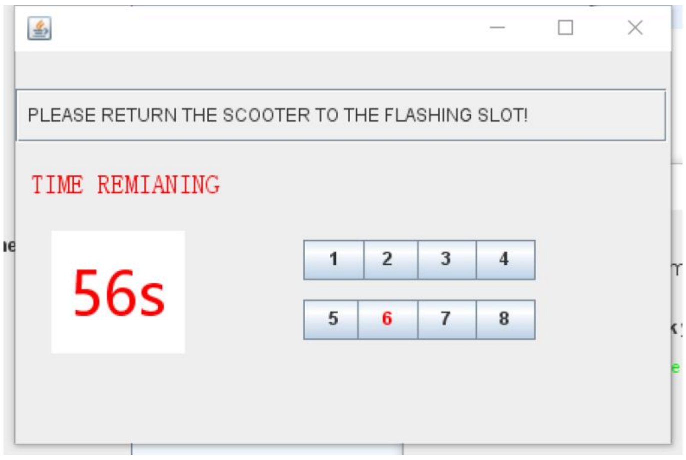
(Figure 6)

7. If you pick it up successfully, and in figure 7, it will show "Pick up successfully".


PICK UP SUCCESSFULLY!

(Figure 7)

8. Back to figure 3, you can choose "return" and enter figure 8. In this, there are 8 LEDs too. And there will be a LED flashing randomly, which represents this slot is available and you should return scooter to this slot. Pressing flashing LED means you have finished returning. And there is a time counter too.

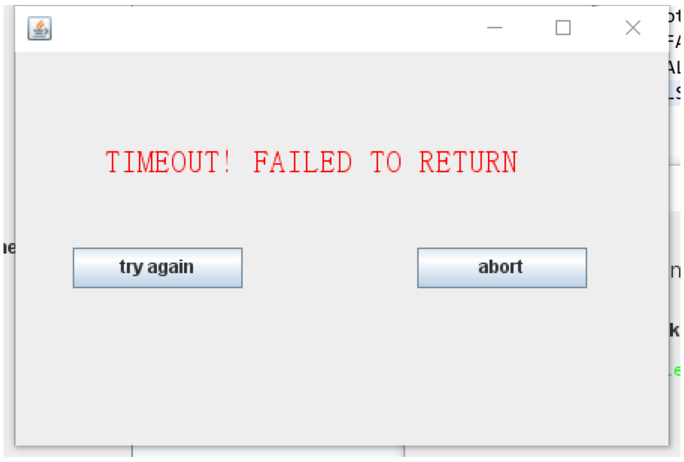

PLEASE RETURN THE SCOOTER TO THE FLASHING SLOT!

TIME REMIANING

56s

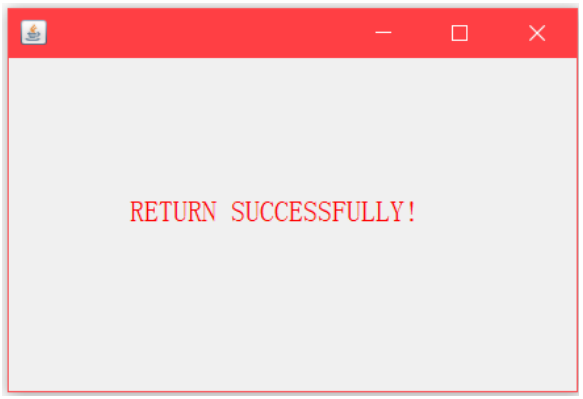
(Figure 8)

9. If you do not return before timeout, entering figure 9.


TIMEOUT! FAILED TO RETURN

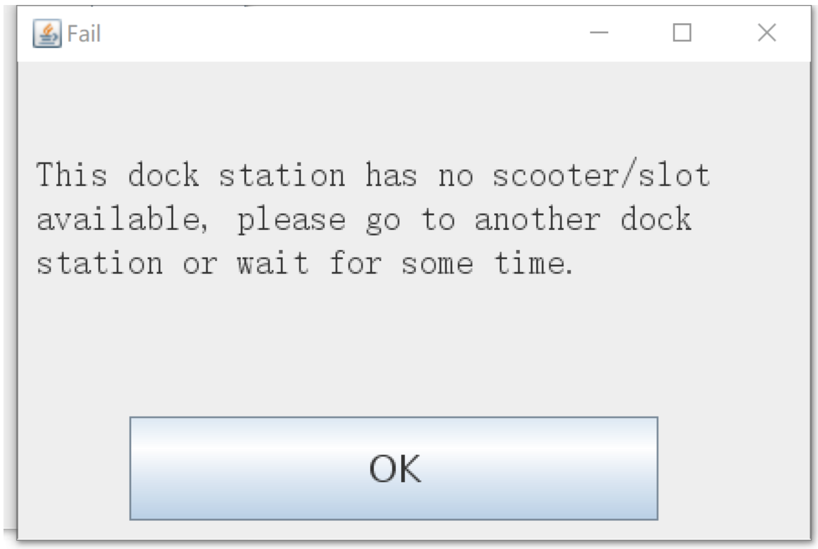
(Figure 9)

10. If you finish returning, entering figure 10.


RETURN SUCCESSFULLY!

(Figure 10)

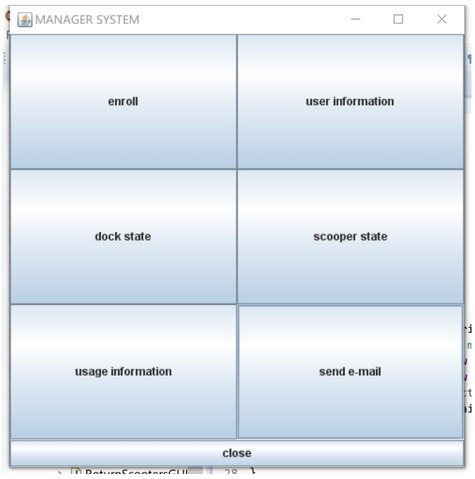
11. Back to figure 3 again, if you decide to borrow or return scooter and there is no slot available in the dock you choosing, you will enter figure 11. The system will tell you this dock does not have available slot.



This dock station has no scooter/slot available, please go to another dock station or wait for some time.

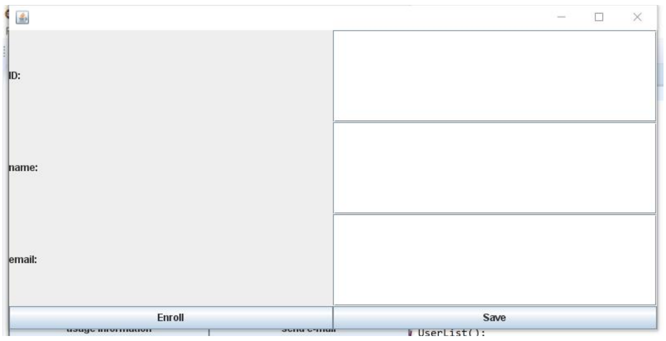
(Figure 11)

12. If you choose “manager” in main frame, you will enter figure 12. In this frame, you can choose “enroll”, “usage information”, “dock state”, “scooter state”, “user information” and “send email” to operate.



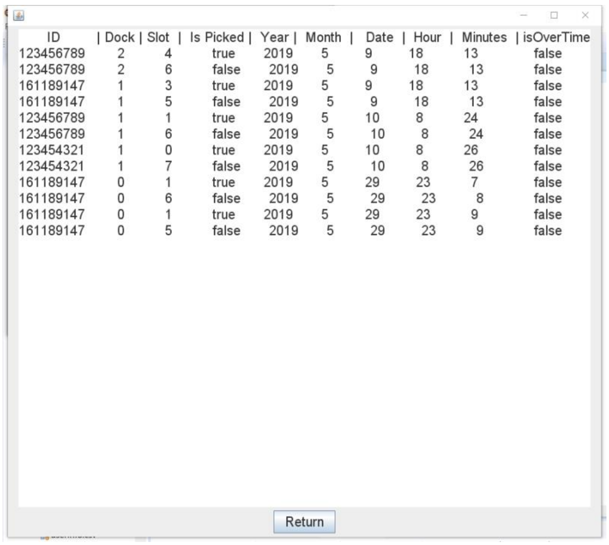
(Figure 12)

13. If you choose "enroll", you register users' information to system in figure 13. You can input QM number, name and email. Then pressing the "enroll" button, and users' information will be registered into system. And this user will have right to use scooter sharing system.



(Figure 13)

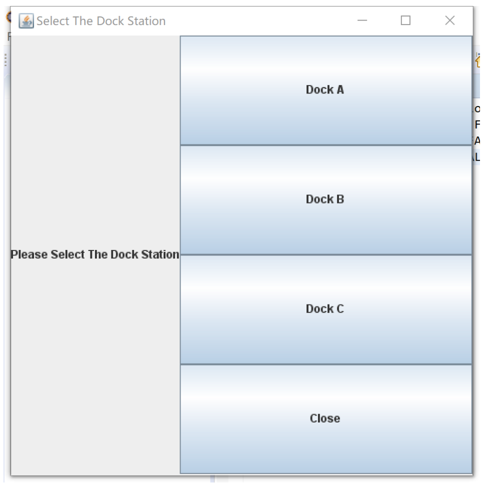
14. Pressing "usage information" button and enter figure 14. To input the users' QM number and press "search" button, you will get his information.



```
     ID        | Dock | Slot  |  Is Picked |  Year |  Month  |   Date  |  Hour  |  Minutes  | isOverTime
123456789        2       4         true     2019     5        9      18        13          false
123456789        2       6         false     2019     5        9      18        13          false
161189147        1       3         true     2019     5        9      18        13          false
161189147        1       5         false     2019     5        9      18        13          false
123456789        1       1         true     2019     5       10       8        24          false
123456789        1       6         false     2019     5       10       8        24          false
123454321        1       0         true     2019     5       10       8        26          false
123454321        1       7         false     2019     5       10       8        26          false
161189147        0       1         true     2019     5       29      23         7          false
161189147        0       6         false     2019     5       29      23         8          false
161189147        0       1         true     2019     5       29      23         9          false
161189147        0       5         false     2019     5       29      23         9          false
```

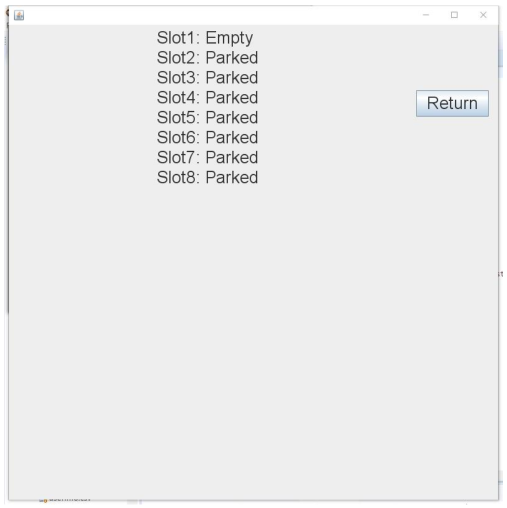
(Figure 14)

15. Pressing "dock state" button and enter figure 15. You can choose the dock A, B or C to check.



(Figure 15)

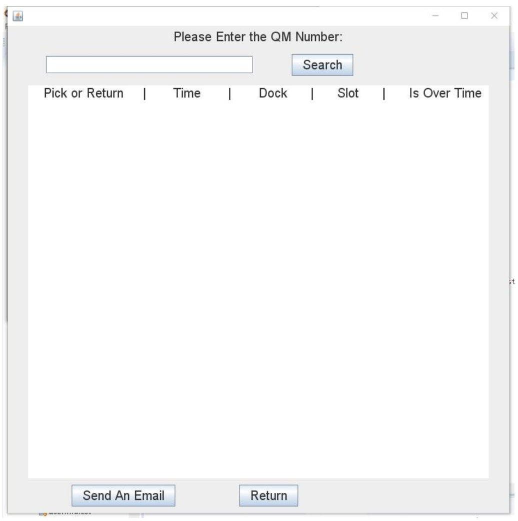
16. Assuming that you choose dock A, the current state of dock A will be displayed in figure 16.



```
Slot1: Empty
Slot2: Parked
Slot3: Parked
Slot4: Parked
Slot5: Parked
Slot6: Parked
Slot7: Parked
Slot8: Parked
```

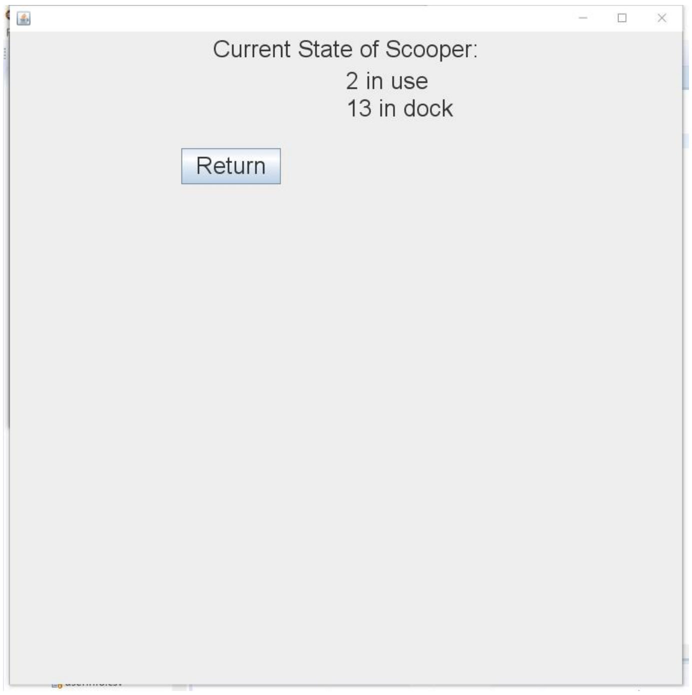
(Figure 16)

17. Pressing "send email" button and enter figure 17, you will input the user's QM number who you want to send email to. And pressing send button to send email.


Please Enter the QM Number:
Search
Pick or Return | Time | Dock | Slot | Is Over Time
Send An Email
Return

(Figure 17)

18. Pressing "scooter state" will enter figure 18. To show the current of scooter usage.


Current State of Scooper:
2 in use
13 in dock

(Figure 18)