

Software Engineering Report

Tourist Train Journey Management System

Group 32

Group Member

Long Fa	2013213173	130801544(Group Leader)
He Mengjing	2013212921	130799023
Huang Wenhao	2013212942	130799230
Zhu Yueran	2013213001	130799827
Han Haoran	2013213395	130803847
Wang Xinyu	2013213409	130803984

Table of Content

1	Introduction	2
2	Summary of Responsibilities and Achievements	2
2.1	Individual Member: Long Fa 2013213173 130801544	2
2.2	Individual Member: He Mengjing 2013212921 130799023	2
2.3	Individual Member: Huang Wenhao 2013212942 130799230.....	3
2.4	Individual Member: Zhu Yueran 2013213001 130799827	3
2.5	Individual Member: Han Haoran 2013213395 130803847.....	3
2.6	Individual Member: Wang Xinyu 2013213409 130803984	4
3	Project Management	4
3.1	Outline planning and architectural design.....	4
3.2	Sprint Cycle.....	5
3.3	Project Closure	6
4	Requirements.....	6
4.2	Software Prototypes	7
4.3	Non-functional Requirements.....	7
4.4	Product Backlog Change.....	8
4.5	Estimation	8
5	Analysis and Design.....	9
5.1	Class Analysis.....	9
5.2	Reusability and Extendibility	11
5.3	Design of Software	11
5.4	Design Principles	12
6	Implementation and Testing	13
6.1	Implication Strategy & Iteration Build Plan.....	13
6.2	Pair Programming.....	15
6.3	Test strategy & Test techniques.....	16
6.4	Using of TDD.....	19
6.5	Junit	20
7	Conclusion.....	24
8	Reference	24
9	Appendix	25
9.1	Main Screen Shots of the System.....	25
9.2	Detailed Class Analysis	26

1 Introduction

This coursework is about developing a tourist train journey management system using Agile methods and other techniques and principles taught in EBU5304. The system we designed would help managers manage the routes, journeys, trains and drivers of their company, drivers get control of their buses and passengers get the information they need. The whole development takes about 40 days, 4 10-day iterations in total. At last we achieved fairly satisfying results.

2 Summary of Responsibilities and Achievements

Group Number:	32		
Group Members:	He Mengjing	2013212921	130799023
	Huang Wenhao	2013212942	130799230
	Zhu Yueran	2013213001	130799827
	Han Haoran	2013213395	130803847
	Wang Xinyu	2013213409	130803984
Group Leaders Name:	Long Fa	2013213173	130801544

2.1 Individual Member: Long Fa 2013213173 130801544

As the team leader, manage group's sessions and allocate works for group members. Check each member working finished the read and write files section together with Huang Wenhao, do the Junit test and some other kind of test of our project. Help to write user manual and readme of our project

During the whole project, I've learned how to manage a group work, to use agile methods to allocate works and manage time and to code individually or cooperatively with each other.

All in all, we finished the project and report by ourselves. So I will give very high appraise to all my teammates. Because they finished almost all the work I give to them perfectly, but I think I'm not doing well as a team leader, we had some big changes during our project implementation process, which caused a lot of time. And if I could manage the work better and find the error at the very beginning, we could avoid this and save more time. Our program could be better too. Also there is some contingencies happened during our implementation process. But we could share the work due to this and fix it anyway.

2.2 Individual Member: He Mengjing 2013212921 130799023

As a group member, I mainly charged for the design and coding of GUI. And as an assistant of Han who mainly charged for report part, I provided material and modified the statements. In additional, I take participated in the whole progress with my other partners, from requirements specification to implementation. I also joined every meeting and discussion.

During this coursework, I gained the ability of teamwork with my partners, the ability of solving difficulty that I met in coding, the ability of time management and breaking the task, and the ability of communication with my teams.

In this development progress, I finished my work well. And I also help others to solving the difficulty. To be honest, what I maximum harvest is I get five precious friendship rather than getting some programming skills. I felt very happy and relax when I work with them. In fact, I also have many shortcomings. I should shared responsibility for LONG when he was busy for coding. I should gave more help to ZHU when she charging for the report.

2.3 Individual Member: Huang Wenhao 2013212942 130799230

As a member of the group, I help to finish the read and write files section of the project and help to design the construction of the whole project. Also I write file sections with group leader, including user manual and part of the report.

Through this coursework, I've gained the ability to code together with others using agile methods and the ability to search the data I need effectively and the ability to communicate and express clearly.

I'm not doing well at the beginning of the project, for the lack of essential profession skills of coding and inability of timely communication. Later, during the process of the coursework, I made up those ability step by step and finally able to finish all the work gave to me perfectly.

2.4 Individual Member: Zhu Yueran 2013213001 130799827

As a group member of relatively lower coding capability, I took the charge of all the document writing and actively joined the coding with my coding partner, Wang Xinyu. I helped her with the implementation of the management of route, journey, train and driver.

During writing backlog, I've learned how to do requirements finding and basic software design. Under the help with Wang, I improved my ability of Java coding. After most of software developed, I started to write the final report and help writing user manual. It taught me with doc techniques and most importantly, a better understanding of all Agile methods.

Working with the whole team is really a fantastic experience. It pushed me to finish the things I never thought I could've done. I fought hard to finish the work that had been assigned to me and it did achieve the expectation I supposed. There was a time when our program faced a great difficult, and I encouraged all our teammates and together we fixed it. The information gathering, assimilation of information wasn't as easy I thought it could be. It has direct impact on further design. It deserved really more time and attention. The final product, to some extent, is not complete, as it could not give an independent view for passenger. But I think it can be done after more time given for debug for the core functions.

2.5 Individual Member: Han Haoran 2013213395 130803847

As a group member, I mainly charged for collating courseware and designing the frame of the report. And as an assistant of He Mengjing who mainly charged for coding part, I was responsible for the communication between him and other programmer, I also helped him to defect the errors. In addition to this, I involved in all development progress.

I've learned a lot in this progress; teamwork, communication, time management, solving all kinds of problems, these could help me in my whole life.

Both write code and write report is the difficulty for myself, so that will be a little busy,

but finally I coordinate the time and have done my work in time. I think the good communication with your teamer is an important skill that will improve your work efficiency.

2.6 Individual Member: Wang Xinyu 2013213409 130803984

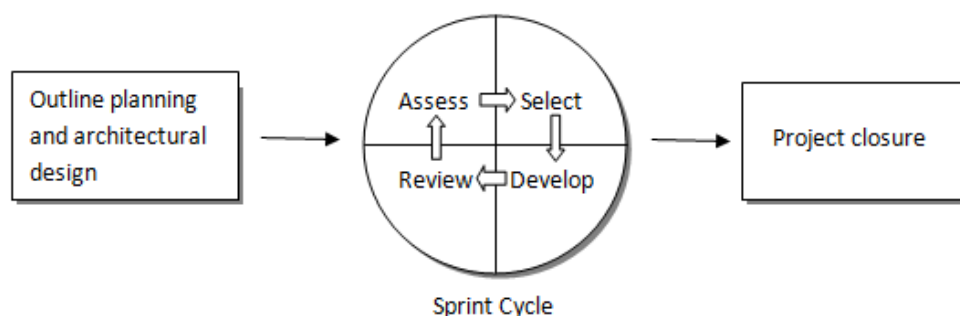
As a member of the group, I was charge for implementation of methods about the management of route, journey, train and driver. Secondly, I combined the testing result together and wrote the report on implementation and test part. Thirdly, I write the user manual with my Huang. Besides, as an assistant of Zhu who's in charge of designing the whole framework of report, I help her analysis the writing code.

After this project development, I've gained lots of abilities, the ability to finish the work together with my group members not only in coding part but the report, the ability to communicate with others, the ability to search and study the new coding methods from Internet to implement the task, the ability to use the new working software when finishing the figure of our report and more abilities I myself haven't noticed yet.

Actually I am satisfied with my appearance in this process. I finished the task given to me carefully and timely. However the process is rather difficult especially for coding part. When facing the unexpected bugs in program, I communicated with other partners in time and analyzed the logic of our coding prototypes again and again, then adjusted the implementation method. I am really appreciated for every member of our group. They helped me a lot no matter when I called them even at night. The spirit of our group touched me a lot and this was a wonderful chance for me to learn many things and improved myself.

3 Project Management

We used Agile methods all through this development. One general method of Agile project management is Scrum. There are three phases in Scrum, namely outline planning and architectural design, sprint cycles and project closure. We used it and something more. Generally, the outline planning and architectural design took about 2 weeks. The next 40 days exclude two short vacations were split into 4 sprints for each cycle. The rest are for flex and project closure.



3.1 Outline planning and architectural design

Group member: 4 Telecom students, 2 e-Commerce students in total, all of which have different levels of Java coding ability and document writing ability.

Coursework specification:

- 1) Coursework release date: 14th March 2016
 First delivery date: 25th April 2016
 Final delivery date: 23rd May 2016
 Upload needed material meeting the requests before due.
- 2) Using Agile methods in all activities.
- 3) Software requirements in function (see in 4. Requirements.)
- 4) Accept future change of software.
- 5) The system must be developed as a standalone system.
- 6) The system must be written in no net work language, namely Java, and without database.
- 7) Demonstrate necessary functions on 31st May.

3.2 Sprint Cycle

We spent 40 days in this phase as 4 sprint cycles, 10 days each and at the beginning, we prepared extra time for problems.

- 1) At the beginning of each sprint, we held group meetings to discuss the tasks in the coming sprint cycle of the whole group and each coding pairs.
- 2) Then three coding pairs started development in XP. We kept internal contact during the whole process, in case there was any need for dealing with accidents.
- 3) At the end of each sprint, we held a group meeting again to update and review the developing progress. If the development didn't go well with the plan, we would need to cope with it immediately.

In spite of regular coding cycles, we used Agile methods to run the team more efficiently.

Group Management Methods	Effect
Choose a team member to be scrum master	1. Guarantee every developing tasks finished in time. 2. Arrange meeting regularly. 3. Protect the development team from external distraction.
DSDM (Dynamic systems development methods)	1. Measure the priority of each story based on actual value and need. 2. Help to arrange the developing plan.
Story points	Help to estimate how long the work is likely to take.
Backlog	1. Show the result of requirements finding and estimating. 2. Show a clear plan list by functions. 3. Make a clear list of requirements to simplify further analysis.
Paper prototyping	Make clear plan of UI which is easy to change.
Stand up meetings	1. Keep everyone on team in track. 2. Make immediate decision or re-plan if there's anything cope with each of the members.
Weekly (sprint-ly) Meetings and Demos	1. Check the progress at the end of each sprint. 2. Make immediate changes of plan if accidents happens.
Risk Management	Prepare for the possible changes and try to find solutions.

3.2.1 Risk Management

Risk type	Possible Risk	Probability	Effects	Solution
Project Risks	Skilled staff leave or are absent due to personal reasons.	High	Serious	1.Re-plan immediately. 2.Look out next time for assign too much complex tasks to same person. 3.Encourage less skilled staff finding solutions to the task if this task is less vital.
Product Risks	Larger number of changes to requirements than anticipated.	Low	Medium	1.Assess the new requirements. 2.If necessary, add it to backlog and schedule.
	Analysis and specifications is not available on schedule.	Medium	Medium	1.Check the schedule if there's free time. 2.Schedule it before the less prior task.
	Implementation reveals the development team are lacking in necessary skills and knowledge.	High	High	1.Learn the related lesson immediately. 2.Get similar code from open source codes.
	Serious design error discovered during implementation.	High	High	1.Solve it inside the coding pair before plan. 2.Discuss it with available staff before sprint ends. 3.Discuss it in the sprint meeting and try to finish it within the next sprint.
	Serious error is not uncovered in testing -logical error, performance.	High	High	1.Check it immediately back in the main functions, interfaces and related classes. 2.If any of the staff have time, check other structure.
	User interface proves unattractive to customers.	High	Low	If the staff have time by the end of all the functions implementation, re-design the GUI.

3.3 Project Closure

After nearly two months of planning and intense sprint cycle, the program finally met the every need of customers and came to the closure. But there were still a lot of paperwork and file consolidation to do. The final work that shows our effort still needs 100% of concentration and caution.

4 Requirements

Requirements are the basis of a project, so it's vital to locate accurate, comprehensive yet detailed needs of the clients using the requirements finding techniques.

4.1.1 Background Reading

We read the coursework handout released by the module organizer carefully and got to know most of detailed enough requirements and some vague ones. Moreover, we compared this system with the current subway and bus system to gain more understanding.

4.1.2 Interviewing

To obtain more specific information of some vague requirements, we used message board to communicate with our client (the module organizer) and asked her face to face. Then we collected the information and fixed the requirement.

4.1.3 Observation

We used every version of the system as if we were managers, drivers or passengers and found the inadequacies of the system. Then we improved and fixed them in the next iteration.

4.2 Software Prototypes

The figure displays six software prototypes for the Tourist Train Journey Management System, arranged in a 3x2 grid. Each prototype includes a 'Time' label at the top.

- Main Manager Window:** Features a 'Timetable of all' section. Below it are buttons for 'Route', 'Train', 'Driver', and 'Exit'. At the bottom, there is a 'Train ID' input field, a 'Run' button with a dropdown arrow, a 'Stop' button, and an 'Order' button.
- Route/Journey Management Window:** Features a 'Timetable list by route & journey' section. Below it are dropdown menus for 'Route' and 'Journey', an 'Add' button, and a 'Delete' button.
- Driver Management Window:** Features a 'Driver Status' and 'Location' section. Below it is an 'Assign' button. At the bottom, there is a 'Driver ID' input field, an 'Add' button, a 'Run' button with a dropdown arrow, a 'Stop' button, and an 'Order' button.
- Train Management Window:** Features a 'Train Status' and 'Location' section. Below it is an 'Assign' button. At the bottom, there is a 'Train ID' input field, an 'Add' button, a 'Run' button with a dropdown arrow, a 'Stop' button, and an 'Order' button.
- On-train Window:** Features a 'Timetable' section. Below it is a 'Train ID' input field, a 'Run' button with a dropdown arrow, and a 'Stop' button.
- Station Display:** Features a 'Timetable' section.

4.3 Non-functional Requirements

4.3.1 Product requirements

- 1) Usability requirements. The system should not only meet the requirements, but also be easy enough to use. The developing group would better deliver the program with user manual and readme file which show how to install and use this program.
- 2) Reliability requirements. The system should be robust and not breakdown normally.
- 3) Portability requirements. The system should be able to run from the command line with no requirements to install extra software.

4.3.2 Organizational requirements

- 1) Delivery requirements. The system should be delivered through QMplus before the due

date and in the required format.

- 2) Implementation requirements. The system must be developed as a stand-alone Java application. No networking and No database implementation.

4.4 Product Backlog Change

In our previous backlog, we divided the requirements into different little tasks. The content of the story didn't change. But when we go ahead with the operation of the program, we found that the developing order of some functions was not supposed as we thought initially. So we ranked the sprint number again in the programming process.

Story ID	Story Name	Iteration	New Iteration
1	Station display	1	4
2	On-train display	1	4
3	Driver control	3	3
4	Train synchronization	1	2
5	Driver list	2	1
6	Driver Regulation	4	1
7	Driver Status	2	2
8	Train list	2	1
9	Train Regulation	4	1
10	Train Status	2	2
11	Remote control	3	3
12	Journey list	1	1
13	Journey Regulation	4	1
14	Timetable	2	2
15	Route List	2	1
16	Route Regulation	1	1
17	Station Regulation	4	1
18	Synchronization	1	2
19	Time	1	1

4.5 Estimation

Story ID	Story Name	Story points	Actual Days	Priority	Iteration
19	Time	1	1	4	1
17	Station Regulation	1	3	4	3
5	Driver list	2	3	3	1
6	Driver Regulation	2	5	5	1
8	Train list	2	3	3	1
16	Route Regulation	2	5	2	1
11	Remote control	2	5	2	3
9	Train Regulation	3	5	5	1
12	Journey list	3	4	1	1
13	Journey Regulation	3	5	4	1
15	Route List	3	4	3	1
14	Timetable	3	5	1	2
1	Station display	3	6	3	4
2	On-train display	3	6	1	4
4	Train synchronization	5	10	1	2
7	Driver Status	5	8	2	2
10	Train Status	5	8	2	2
3	Driver control	5	8	3	3
18	Synchronization	8	10	1	2

5 Analysis and Design

5.1 Class Analysis

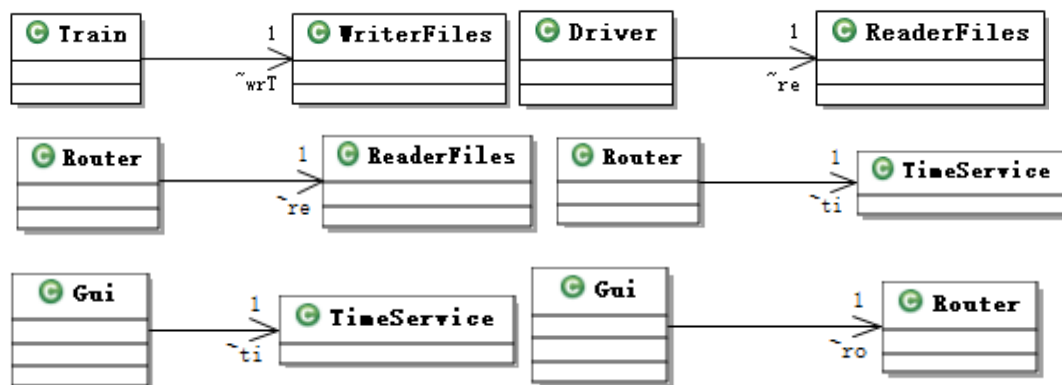
5.1.1 Identity the Classes

A good program must have different analysis classes. Boundary classes to model the interaction to between the program with external systems or users, Entity classes to model information that is persistent and the system is dependent on, Control classes to encapsulate control and coordination of the main actions and objects. Our program also has these classes to fit the method:

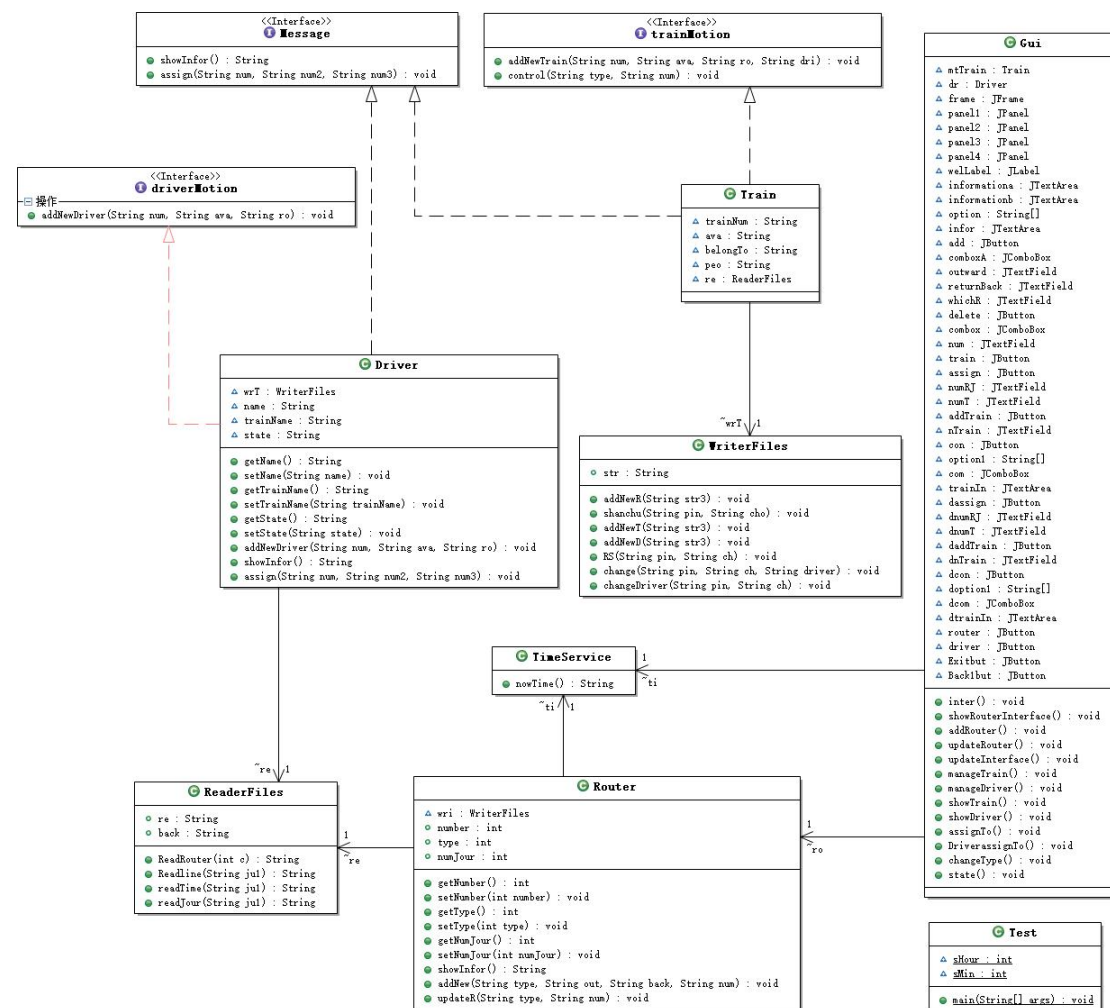
- Boundary classes: class GUI
- Entity classes: class Driver, class Route, class Train, class TimeService
- Control classes: class ReaderFiles, class WriterFiles, class Test

5.1.2 Identity class relationship

A program is made up of many classes and objects. There may be some relationship among some classes which provide the pathway for communication. Two main relationships existed at normal programs, association and inheritance. However, our program did not recognize the inheritance as we replaced it by interfaces. We pictured the classes' pair that has the relationship of association below:



5.1.3 A conceptual class diagram



5.1.4 Identity attributes

Driver	Route	Train	TimeService
wrT, WriterFiles name, String trainName, String state, String	wri, WriterFiles number, int type, int numJour, int	trainNum, String ava, String belongTo, String peo, String re, ReaderFiles	(none)

5.1.5 Add constraints

- 1) Standalone system.** The system must be developed as a standalone Java application with SIMPLE Java GUIs (AWT, Swing).
- 2) NO network programming.** The program must not write with HTML, JavaScript, JSPs, sockets, ASPs, PHPs, etc.
- 3) NO database implementation.** Students should develop this application without using a database.

5.2 Reusability and Extensibility

Our program has lots of parts can be reused, which can not only adapt to the requirements changed in the future, but also provide a template for other systems.

1) If there is a system that needs to store the newly created object when it closed every time, or you need to read the information you created in the past. Then you can reuse our class WriterFiles and ReaderFiles. In our program, the following command realized this function.

```
writer = new FileWriter("F:/information/driver.txt", true);
writer.write(str);
writer.write("\r\n");
writer.flush();
writer.close();
```

2) If there is a system that needs to display the time with specific format, then it can use the class TimeService of our program. The only thing need to do is to change the description in the command.

```
Date date=new Date();
DateFormat format=new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
String Rname=format.format(date).replace("-", "").replace(" ", "").replace(":", "");
return Rname;
```

Then the system time can be any format you want.

3) Another reusability of our program can be realized when a system need to display one txt format file on the computer or store the information users input on the GUI by using the class WriterFiles and ReaderFiles combine with class GUI.

5.3 Design of Software

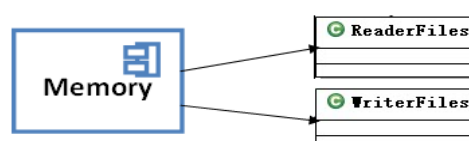
5.3.1 Design Quality Guideline

We will transform the analysis model into design model. And the design guidelines are listed as follow:

- 1) Meet the requirement which has been decided.
- 2) Be well structured: Our type is object-oriented architecture.
- 3) Be modular.
- 4) Be efficient and error free.
- 5) Be maintainable.
- 6) Be well documented.

5.3.2 Component Design

Component Based Software Engineering (CBSE) is a more advanced way to realize the reuse of code than Object Orient. In fact those two methods can be used in combine to make our program reusable. Our program contained an important component which realized the function of input and output the information, and we called the component Memory. The mapping of Memory to the classes showing as follow:



5.3.3 Interface Design

Only a component can realize any function when with one or more interfaces to other classes. These interfaces provide data or commands for component while do not change the architecture of it. In our program, the Memory component receives these data from class Train, Driver, and Route. And it also provides the object to class Gui and Test.



5.3.4 Architecture Design

1) Repository Architecture

Our program conformed to the Repository architecture; it realized the function that the user can take over the whole journey information of past time by building two memory files similarly as repository. And the class WriterFiles and ReaderFiles are the subsystem to share those data.



2) Object-Oriented Architecture

Java itself is an OO language, so our program also fit the Object-Oriented Architecture.

5.3.5 Class Design (This point can refer to the class diagram)

5.4 Design Principles

5.4.1 Single Responsibility Principle (SRP)

Part of our program designed to meet the SRP, which means that some Classes or objects of our program are only responsible for one responsibility. This increased the robustness of the program. For example, Class Driver in our program has only one method, addNewDriver(), this defined the number and state for the new Driver object. This didn't involve the matching to the Train or Route, and also didn't describe the motion of Driver, while we realized these functions by calling the interface DriverMotion or Class WriterFiles, ReaderFlies, which means that we need not change other parameters when we want to change the driver number. Similarly, Class Train and Class Route are also consistent with the SRP principle.

5.4.2 Open-Close Principle (OCP)

The most classic usage of OCP is the abstracted classes and interfaces. There are two interfaces in our program, trainMotion and driverMotion. When other objects or classes try to implement one of these two interfaces, i.e. Train to trainMotion, they don't need to modify the methods. They simply need to add the needed functions in methods implementation. So when we do the regression testing, we don't have to test the interfaces again and again, but only need to guarantee the methods that implement interfaces passing the test.

5.4.3 Don't Repeat Yourself (DRY)

DRY has some similarity with OCP. Both of the principles ask us to build an independent class/method that we may call from other modules to reduce repeating the same codes. It may improve the modularity but DRY uses the much similar codes than we wrote. If we have to use it, we need to add a large amount of code to modify it to use this principle and highly decrease the readability of our code so that it will bring trouble for modifying. So we failed to use it everywhere, as in the two alike classes, WriterFiles and ReaderFiles, we repeat a lot of similar codes to implement similar functions.

5.4.4 Dependency Inversion Principle (DIP)

DIP will not bring large influence to the upper class layer, such as GUI class and test class, when we change the codes of lower class layer, such as WriterFiles class and ReaderFiles, by adding the abstract class of the lower between the layers, which can decrease the risk brought by modifying. However we did not recognize this principle considering the practical application of our program. And we may not change the function of the entire program in the future. In fact, we written some method directly in the GUI class to reduce the risk caused by the change.

5.4.5 Interface Segregation Principle (ISP)

Have to say, our program do well with the ISP. Firstly, the driverMotion and trainMotion interfaces are simple, which both are only have one method. Therefore, users need not to consider other interfaces when interacting with an exact interface, which not only simplifies the operation of users, also makes our program more intuitive. As a matter of fact, our program only has two interfaces. This is our comprehensive consideration with the user interaction and the code implementation. The increasing number of interface means that we need more error tolerance. So we minimize the user interface as possible as we can, while each interface must be simple enough.

5.4.6 Liskov Substitution Principle (LSP)

Part of our program followed the LSP. In fact, we did not do that deliberately. We just wrote one method in some father class, such as method addNewTrain() in class Train and method nowTime() in class TimeService, as we did not want to realize any other functions in those classes. Therefore other class, such as class WriterFiles, need not change the code when calling them to realize the function.

6 Implementation and Testing

6.1 Implication Strategy & Iteration Build Plan

6.1.1 Assumption

The working time is form 8:00 from 18:00.

Each train can be distributed into only one journey.

Each driver can be distributed into only one train.

6.1.2 Integration Build Plan

We divide our software into incrementally management steps as a sequence of build plan and make each step come to a test.

Build Plan	Functionality	Implementation
1	Regulation of the route and journey	Router.java; WriterFiles.java; ReaderFiles.java; TimeService.java
2	Regulation of the train	Train.java trainMotion.java WriterFiles.java;
3	Regulation of the driver	Driver.java driverMotion.java WriterFiles.java;
4	Dynamic display the arrival location of the train	Gui.java ReadFiles.java
5	Implementation	Integration

We finish the integration build plan according to the sequence of builds. The fundamental functions are firstly to be complete such as regulation of the routes, train and driver. Then gradually extend to the distribution of the train and driver, to make these three parameters one-one correspondence. Finally accomplish the dynamic display of the location of the train which is the most difficult for our group because the GPS cannot be used, so we have to design the specific algorithm by ourselves.

6.1.3 Mapping Design to Code

Our group implements our software according to the build plan strictly. For instance, as for the mapping process of regulation of train:

We write 2 interfaces **Message** and **trainMotion** first which are as follows.

```
public interface Message {
    public String showInfor();
    public void assign(String num, String num2, String num3);
}

public interface trainMotion {
    public void addNewTrain(String num,String ava,String ro,String dri);
    public void control(String type,String num);
}
```

The common functional methods are written into the **Message** interface, it is responsible for display of train information which have been successfully added and assign the train to the specific route. The interface **trainMotion** is responsible for the exclusive attributes of the train itself that it in charge of adding the new train to the system and control the status of the train.

Then name another class **Train** to implement the two interfaces and finish the detailed function that would like to be realized.

```
public class Train implements trainMotion, Message{
    String trainNum;
    String ava;//1是available,1是没有运行
    String belongTo;
    String peo;
    WriterFiles wrT=new WriterFiles();
    ReaderFiles re=new ReaderFiles();
}
```

6.1.4 Realize Design Principle

In our programming process, we thought about the six principles all the time. For instance, as for the Do not Repeat Yourself Principle,

```
public void showTrain(){ //显示train信息
    trainIn.setText("Train number    Type(1 is available)    Router    Driver"+"\\n"+rea.ReadRouter(2));
}
```

We create the object and construct the method for each function only for once. And when reuses, we could revoke the method avoiding abundant repeating coding work.

```
class addTrainbutListener implements ActionListener {
    public void actionPerformed(ActionEvent event){
        //先暂时没有加判断，不健壮
        if(!nTrain.getText().equals("")){
            Train t=new Train();
            t.addNewTrain(nTrain.getText(),"1", "1", "1");
            nTrain.setText("");
            //showDriver();
            showTrain();
        }
    }
}

class assignbutListener implements ActionListener {
    public void actionPerformed(ActionEvent event){
        //先暂时没有加判断，不健壮
        assignTo();
        showTrain();
    }
}

class conbutListener implements ActionListener {
    public void actionPerformed(ActionEvent event){
        //先暂时没有加判断，不健壮
        changeType();
        showTrain();
    }
}
```

Besides, as for ISP principle, we write all methods in interface **Message** which will be overridden for many times and used in subclass.

Eg1:

```
public interface driverMotion {
}

public class Driver implements driverMotion {
```

Eg2:

```
public interface Message {
}

public class Driver implements driverMotion, Message {
```

Most of the principles are applied in our software programming; however, we still don't make use of all the Six Design Principles finally. But we will do better next time.

6.1.5 Language \ Tools

Language \ Tools: Java \ Eclipse.

6.1.6 Perform Unit Test

With the complement of the whole implementation process, our group is able to test the components. More details are in the following testing part.

6.2 Pair Programming

During the developing process, all of us programmers worked in pairs, which did do benefit

to the development. This helped us know coding better and reduced mistakes since two of us were staring at the same line of codes. What's more, it improved the effectiveness of coding since people coding alone seemed easy to be distracted. But only one disadvantage that occurred to us was that pair coding spent us more time on discussing the way of implement one function.

Subgroup 1	Subgroup 2	Subgroup 3
Long Fa Huang Wenhao	He Mengjing Han Haoran	Xinyu Wang Zhu Yueran



6.3 Test strategy & Test techniques

6.3.1 Test Strategy

Each component of our software should be tested for its normal and alternative flows and the behaviors, which should cover the incorrect user input and mistake behaviors. Each test should not be redundant and has a high probability of finding an error.

1) How to run them

Unit test: We choose to use Object-oriented Testing to test each component. With Agile method, we use Junit to test every one of the unit.

2) How to determine whether the test is successful

System test: When testing the integration version process, we choose to use White-box testing to test the internal logic and Regression Test to promise every process is right.

6.3.2 Techniques

6.3.2.1 Object-oriented Testing

Our testing began prior to the existence of source code. After the code has been generated, we tested individual classes and subsystem by using Junit in TDD, which can be seen from the following part in 6.3.

6.3.2.2 Regression Testing

Test cases will be created to test the functionality of the build from each stage. With the development's increment, the functionality will be increased, too. We test each stage in time to follow the iteration. Our group had made many versions, here we only show you the test we made for one of our mid-term version project.

As mentioned above, we make full use of regression techniques of finish the whole process. The obviously different version has two ones.

The first version of our software we achieve the story id number 5,6, 8,9, 12,13, 15,16, 19, and we write 3 simple tests to test our project.

Test1: test the driver list, train list and route list.

This class includes a lot of methods, in our first version, we consider manager be the core class of our project, every time we initialize the project, we create an object of it, and it'll read in all the information we need from train.java, router.java and driver.java respectively, we save the information inside 3 Array list<string>, in each string we use "" as a split symbol to differentiate the different attributes of a driver/route/train. So by arrange the information in this way we could visit any rows and columns (any attributes of any specific driver/train/route) respectively. So we added some testing code to help us prove our idea. We create 3 txt files, named them driver.txt, route.txt, train.txt, and adding some information inside it. Then we adding a print message to print the second column of each file, if we success, we could get: xline12/n xlin22/n xlin32/n respectively. (x is different depends on the file name)

```
driver.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
dlin11 dlin12 dlin13 dlin14
dlin21 dlin22 dlin23 dlin24
dlin31 dlin32 dlin33 dlin34

train.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
tlin11 tlin12 tlin13 tlin14
tlin21 tlin22 tlin23 tlin24
tlin31 tlin22 tlin23 tlin24

route.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
jline11 jline12 jline13 jline14
jline21 jline22 jline23 jline24
jline31 jline32 jline33 jline34
```

```
C:\Windows\System32\cmd.exe
Microsoft Windows [版本 10.0.10586]
(c) 2015 Microsoft Corporation. 保留所有权利。

C:\robert\软工\software engineering\bin>java main
jline12
jline22
jline32
dlin12
dlin22
dlin32
tlin12
tlin22
tlin22
```

Test succeeded. So we could use this code block to display route list. Train list and driver list, user can access to any particular parameter of any particular driver/train/route. Since we haven't combined the GUI with it, we could not present the exact outlook.

And we also achieved the time display and recording in our display screen, the GUI could work individually.

Test2: test the regulation of driver list, train list, route list.

In this test we called the Jchange(), Rchange(), and Dchange() methods to change the row2 column 2 of each files to "change", and see the print out results.

```
C:\robert\软工\software engineering\bin>java main
jline12
change
jline32
dlin12
change
dlin32
tlin12
change
tlin22
```

So these three methods did achieve their function.

Test3: GUI& current time display

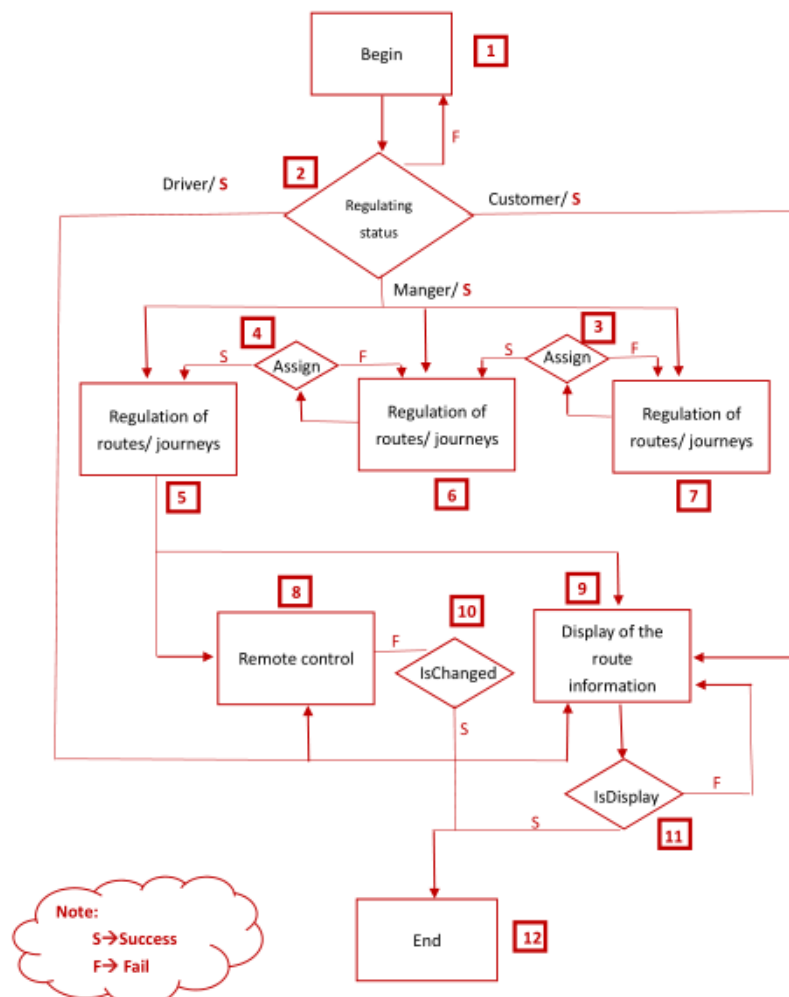


However, the design of the first version is too complex and we find it difficult to connect the front-stage and back-stage together. So this version has to be announced failed.

The Junit test part which is testing our final project has shown in 6.5.

6.3.2.3 White-box Testing

According to the requirements of our software, we make the figure of **Basis of Path Testing** which is as followings:



Path 1: 1→2→7→3→6 Test the validation of regulation of driver and train which including add/delete train/driver, list the related information by the manager. Besides, whether each driver can be assigned to the specific train, when the manager input the correct train name, driver name, and assign the driver to the train in right way as indicated by software, then the system will check and display the updated information.

Path 2: 1→2→7→3→6→4→5 Test the validation of regulation of route and journey which including add/delete route/journey, list the related information by the manager whether each train can be assigned to the specific train. Then by regression, we can also align the journey, train, driver into linear. If the user input non-exist name, assignment, the system will return a warning.

Path3: 5→8→10→12 Test the validation of remote control by manager after correct assign the information of journey, train and train.

Path4: 5→9→11→12 Test the validation of dynamic display broad of location of the running train information and the related journeys by manager. If the manager gives the wrong command, the system will give a warning.

Path5: 1→2→8→10→12 Test the validation of remote control by driver. When the driver gives the correct command, the status will change. If not, the system will give a wrong warning.

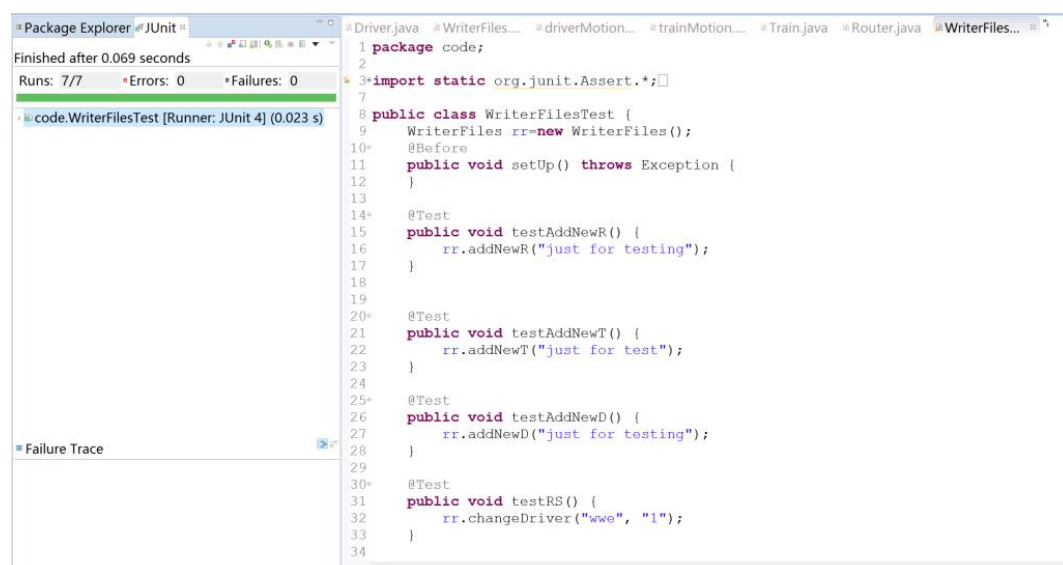
Path6: 1→2→9→11→12 Test the validation of dynamic display broad of location of the running train information and the related journeys by driver. When the driver can see the correct information, then let it run continually, otherwise, the system will give a warning.

Path7: 1→9→12 Test the validation of dynamic display broad of location of the running train information and the related journeys by customer. When the customers can see the correct information, then let it run continually, otherwise, the system will give a warning.

6.4 Using of TDD

TDD is called Test Driven Development which is a simple, short-cycled mechanism in Agile process, which contains firstly writing test to check the functional or non-functional requirement of a module, than writing codes to pass the test, finally refactoring the new code acceptable standards.

We did not use TDD all through the development. We tried it in some classes. Take writefile.java as example. We tried to make this class to write the information we input in certain .txt file. So we wrote the following testing Junit code first.



The screenshot shows an IDE with a JUnit test runner on the left and the source code of the `WriterFilesTest` class on the right. The test runner indicates that the tests passed successfully after 0.069 seconds. The source code includes imports for JUnit and the `WriterFiles` class, followed by several test methods: `testAddNewR()`, `testAddNewT()`, `testAddNewD()`, and `testRS()`.

```

1 package code;
2
3 import static org.junit.Assert.*;
4
5 public class WriterFilesTest {
6     WriterFiles rr=new WriterFiles();
7     @Before
8     public void setUp() throws Exception {
9     }
10
11     @Test
12     public void testAddNewR() {
13         rr.addNewR("just for testing");
14     }
15
16     @Test
17     public void testAddNewT() {
18         rr.addNewT("just for test");
19     }
20
21     @Test
22     public void testAddNewD() {
23         rr.addNewD("just for testing");
24     }
25
26     @Test
27     public void testRS() {
28         rr.changeDriver("wee", "1");
29     }
30
31 }

```

Then we coded the classes.

```

public class WriterFiles {
    public String str;
    public void addNewR(String str3) {
        str=str3;

        FileWriter writer;
        try {
            writer = new FileWriter("C:/information/router.txt",true);
            writer.write(str);
            writer.write("\r\n");
            writer.flush();
            writer.close();

        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void addNewT(String str3) {
        str=str3;

        FileWriter writer;
        try {
            writer = new FileWriter("C:/information/train.txt",true);
            writer.write(str);
            writer.write("\r\n");
            writer.flush();
            writer.close();

        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void addNewD(String str3) {
        str=str3;

        FileWriter writer;
        try {
            writer = new FileWriter("C:/information/driver.txt",true);
            writer.write(str);
            writer.write("\r\n");
            writer.flush();
            writer.close();

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

Run the test.



It appeared successful. Then we got it into use.

6.5 Junit

In our second version, which is also the final version, we give up some of the obsolete ideas in the first version, we separate the manager.java to 2 different class, and optimize the saving/writing methods inside it, we also give up some of the classes and methods(although this cuts out some of the additional functions we want to include at the very beginning).

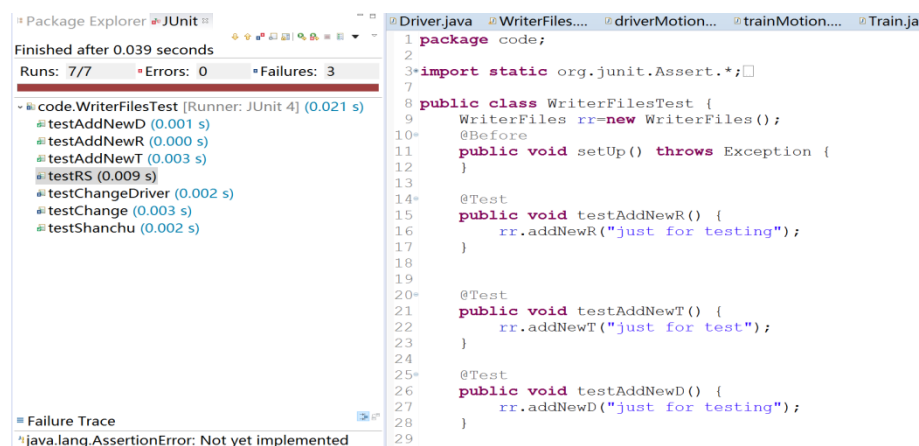
First we test the read and write file part of our program. It works well

Then we start to test the program by the unit of class.

WRITEFILES.java

We first test the class called writefiles.java, which includes a lot of methods used to add, delete and change the methods in different files, we use Junit to call the methods and see if

it works.



We didn't include anything in other method's test, so there are failures. If the methods work, we will have the last line be "just for testing".



But the test message has been writing into the file successfully.

Then we use the same logic to test the `change()` methods, which is used to change the specific information in the files.



Therefore, all the methods in `WRITEFILE.java` have passed the tests.

READMEFILE.java

There is a plenty of methods included in this class, including `readRouter()`, `readLine()`, `readTime()`, `readJour()`, except the `readRouter()` methods, all the other routers read messages from `router.java`, the differences is that they only care about the specific information. We use the Junit to testify them. We'll first test the method called `Readerfiles()`.

```

1 package code;
2
3 import static org.junit.Assert.*;
4
5
6
7
8 public class ReaderFilesReadRouterTest {
9     ReaderFiles rr=new ReaderFiles();
10
11     @Before
12     public void setUp() throws Exception {
13
14
15
16     @Test
17     public void testReadRouter() {
18         assertEquals("wwe;122;2\n\n",rr.ReadRouter(3));
19         assertEquals("111;2;20160515002509;qw\n122;2;20160515004351;as\n123;2;1;az\n",
20             assertEquals("20160515002509;1;Outward,0920:0940,Back;,1000:1010\n201605150043
21

```

As we could see, the method passes the test.

The readLine() methods reads information from router.txt and chose the unique line number from it as its information, same as the readLine(), readTime(), readJour() do the similar thing. So the basic testing logic is the same as Readfiles();

```

1 package code;
2
3 import static org.junit.Assert.*;
4
5
6
7
8 public class ReaderFilesReadRouterTest {
9     ReaderFiles rr=new ReaderFiles();
10
11     @Before
12     public void setUp() throws Exception {
13
14
15
16     @Test
17     public void testReadRouter() {
18         assertEquals("wwe;122;2\n\n",rr.ReadRouter(3));
19         assertEquals("111;2;20160515002509;qw\n122;2;20160515004351;as\n123;2;1;az\n",
20             assertEquals("20160515002509;1;Outward,0920:0940,Back;,1000:1010\n201605150043
21

```

All the four methods have passed the test.

Driver.java

Then we could test the write file methods, since writing methods is almost the same, so we just have to test one of them.

We chose the driver write files to do the test.

```

1 package code;
2 import static org.junit.Assert.*;
3
4
5 import org.junit.Before;
6 import org.junit.Test;
7
8 public class DriverTest {
9     Driver dd=new Driver();
10     ReaderFiles rr=new ReaderFiles();
11
12
13     @Before
14     public void setUp() throws Exception {
15
16
17     @Test
18     public void testAddNewDriver() {
19         dd.addNewDriver("test", "1", "1");
20         assertEquals("wwe;122;2\n\n",rr.ReadRouter(3));
21

```

We could conjecture the result after do the writing and compare it with the input. Result is showed in the picture.

So we achieve the basic reading and writing functions.

timeService.java

timeservice.java includes the method to present time and generate a string for a unique route number from the current time.


```

1 package code;
2
3 import static org.junit.Assert.*;
4
5 public class TimeServiceTest {
6     TimeService rr=new TimeService();
7     @Before
8     public void setUp() throws Exception {
9     }
10
11     @Test
12     public void testNowTime() {
13         System.out.println(rr.nowTime());
14     }
15 }
16
17
18
19
20

```

Train.java

```

1 package code;
2
3 import static org.junit.Assert.*;
4
5 public class TrainTest {
6     Train hh=new Train("num","ava","belongto","peo");
7     @Before
8     public void setUp() throws Exception {
9     }
10
11     @Test
12     public void testGetTrainNum() {
13         assertEquals("num",hh.getTrainNum());
14     }
15
16     @Test
17     public void testSetTrainNum() {
18         hh.setTrainNum("123");
19         assertEquals("123",hh.getTrainNum());
20     }
21
22     @Test
23     public void testGetAva() {
24         assertEquals("ava",hh.getAva());
25     }
26 }
27
28

```

We test the setters and getters in this class, here is the result. We also used the same logic to test the getters and setters in router.java.

```

1 package code;
2
3 import static org.junit.Assert.*;
4
5 public class RouterTest {
6     Router aa=new Router(1,2,3);
7     @Before
8     public void setUp() throws Exception {
9     }
10
11     @Test
12     public void testGetNumber() {
13         assertEquals(1,aa.getNumber());
14     }
15
16     @Test
17     public void testGetType() {
18         assertEquals(2,aa.getType());
19     }
20
21     @Test
22     public void testGetNumJour() {
23         assertEquals(3,aa.getNumJour());
24     }
25
26     @Test
27     public void testSetNumJour() {
28         aa.setNumJour(5);
29     }
30 }
31

```

So far, we have finished testing all the methods in our program. We start to test our program by their functions and running at as a whole.

GUI AND FUNCTIONAL TESTING

We start run our software using “run as java application”, finding that the result has become what we want it to be.

Time:08:04

Out: busA 20160515002509: Train-1-2-	Back: busA 20160515002509: Train-1-2-
Out: busB 20160515004351: Train-1-2-3-4-	Back: busB 20160515004351: Train-1-2-
Out: busC 20160515100122: Train-1-2-	Back: busC 20160515100122: Train-1-2-
Out: busD 20160516204025: 1-Train-2-	Back: busD 20160516204025: Train-1-2-

Route Train Driver Exit

Run

This is the site of which trains is running and which station the train is at. After print the button Route, Train, Driver, we can successfully regulate the information of route, journey, train and driver and accomplish all the functions in the story of backlogs.

7 Conclusion

We've been get through many difficulties and failures in the whole process. The first time of using Agile method to develop software is challenging but rewarding. We've learned how to build a project rather than a simple java program from requirements finding, files analyzing, project analyzing to general designing, planning and then implementing module by module, testing each by each and repeating. We've learned how to get up, get away and learn from failures mentally and technically.

After two-month development, we gained more experience of design and develop a system and understanding of software engineering strategies, techniques and principles. The development process is mainly about coding but far more than codes itself. It also taught us about communicating, managing and working effectively as a team.

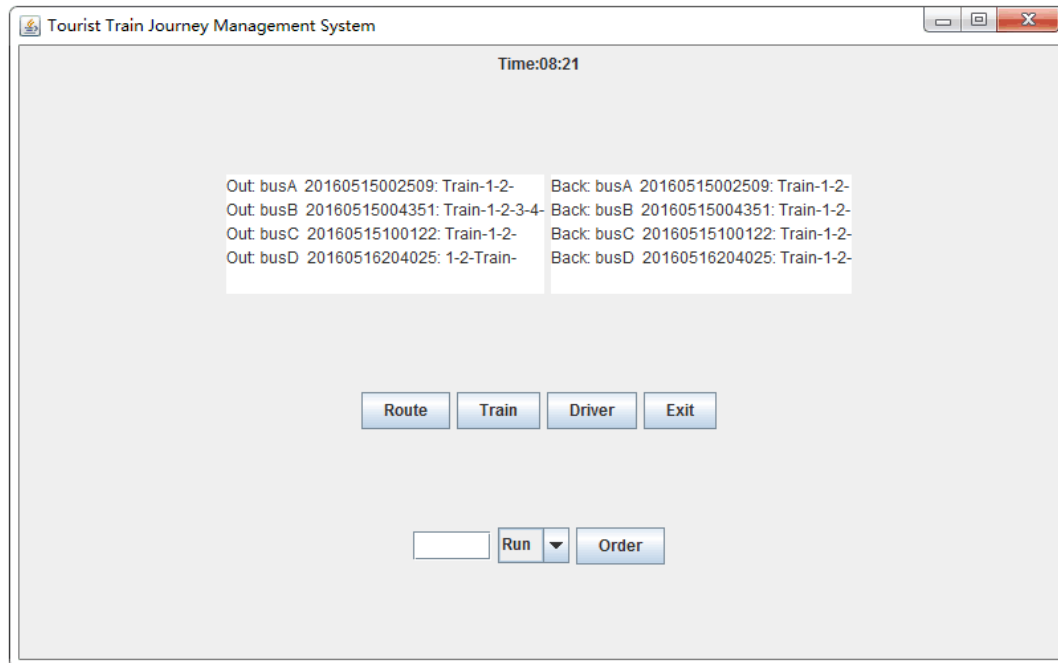
8 Reference

1. "Software Engineering" by Ian Sommerville
2. "Head First Object Oriented Analysis & Design" by Brett McLaughlin et al
3. " Agile Software Development: Principles, Patterns and Practices" by Martin, Robert
4. <https://en.wikipedia.org/>

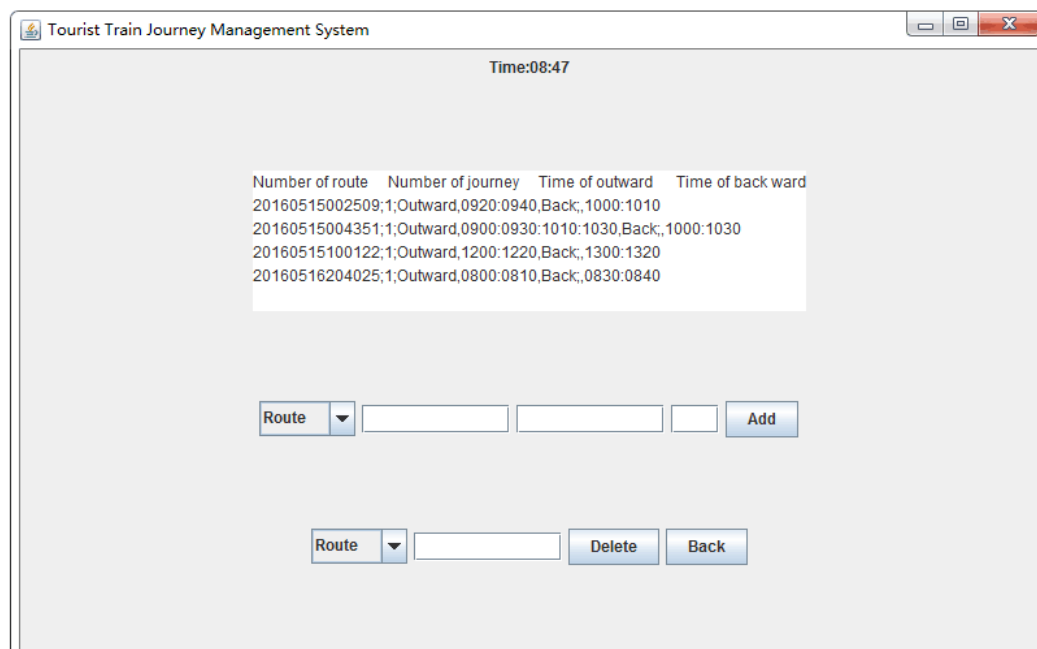
9 Appendix

9.1 Main Screen Shots of the System

Main Interface



Route/Journey Interface



Train Interface

Time:09:06

Train number	Type(1 is available)	Route	Driver
busA;2;20160515002509;	Jone	Out: busA 20160515002509: Train-1-2-	Back: busA 20160515002509: Train-1-2-
busB;2;20160515004351;	Alice	Out: busB 20160515004351: 1-Train-2-3-4-	Back: busB 20160515004351: Train-1-2-
busC;2;20160515100122;	Davie	Out: busC 20160515100122: Train-1-2-	Back: busC 20160515100122: Train-1-2-
busD;2;20160516204025;	LILY	Out: busD 20160516204025: 1-2-Train-	Back: busD 20160516204025: 1-2-Train-
busE;1;1;			

Assign

Add train Run Order Back

Driver Interface

Time:09:34

Driver	Train	Type(1 is unassigned, 2 is assigned)
Jone;	busA;2	
Alice;	busB;2	
Frank;	busC;2	
LILY;	busD;2	
Davie;	busC;2	
Wang;	1;1	

Assign

Add Driver Run Order Back

9.2 Detailed Class Analysis

Train Class(Entity class)

Operations:

Responsibility: Create a Train object of the system.

Realization: Implement the functions of train, and allocate Number, Available information

Attributes:

-train Num: int: give train a number

-ava: int: explain the state of the train, 1 on behalf of free, 2 on behalf of working

-belongsTo: int: define the train belongs to which route

-peo: String: allocate a driver to the train

Association and Aggregation: WriterFiles (String Buffer)

Describing Methods:

+void get TrainNum(): get the train number

+void set TrainNum(): give a number to the train

+void get Ava (): get the state of the train

+void set Ava (): set the state of the train

+void get BelongTo (): know the train belongs to which route

+void set BelongTo (): arrange the trains to which route

+void get Peo (): know who will drive the train

+void set Peo (): arrange a driver to the train

+void addNewTrain (): add a new train

Driver Class

Operations:

Responsibility: Create a Driver object of the system.

Realization: Implement the functions of allocating a driver to the train

Attributes:

-driverNum: int: give driver a number

-driverava: int: explain the state of the driver, 1 on behalf of free, 2 on behalf of working

-DBelongTo: int: define the driver belongs to which train

Association and Aggregation: WriterFiles (String Buffer)

Describing Methods:

+void get DriverNum(): get the Driver number

+void set DriverNum(): give a number to the Driver

+void get DriverAva (): get the state of the Driver

+void set DriverAva (): set the state of the Driver

+void get DBelongTo (): know the Driver belongs to which train

+void set DBelongTo (): arrange the Drivers to which train

+void addNewDriver (): add a new Driver

Route Class

Operations:

Responsibility: Create a Route object of the system.

Realization: Implement the functions of allocating a Route to the system

Attributes:

-RouteNum: int: give Route a number

-type: int:

-numJour: int: give a number to the new Journey

Association and Aggregation: None

Describing Methods:

+void get Num(): get the Route number

+void set Num(): give a number to the Route

+void get type (): get the type of the Route

+void set type (): set the type of the Route
 +void get NumJour (): get the number of the new Journey
 +void set NumJour (): set the number of the new Journey

TineService Class(boundary)

Operations:

Responsibility: Get the time of the system; alter the display format of time
 Realization: Display the time of system, and indicate the travel information

Attributes:

-date: Date: get the time of the system
 -format: DateFormat: the format of the time on the system
 -Rname: String: new time of the system

Association and Aggregation: None

Describing Methods:

+void nowTime(): get a new format to the system time

WriterFiles Class(control)

Operations:

Responsibility: Input parameters to other classes and give them to the integrated classes, include TrainNum, DriverNum etc. And store the objects in library files.

Realization: Create a train has complete information

Attributes:

-str: String: extract the information of Driver, Route, and Train in instances

Association and Aggregation: Driver, Train, GUI

Describing Methods:

+void addNewR(): create a new Route
 +void shanchu(): delete a new route
 +void addNewT(): create a new train
 +void addNewD(): create a new driver
 +void RS(): change the state of train, run or stop
 +void change (): change the attribute of train object
 +void changeDriver (): change the attribute of driver object

ReaderFiles Class(control)

Operations:

Responsibility: Get the information of objects

Realization: Implement the functions of Journey keep working

Attributes:

-re: String: express the information of route in library files
 -back: String: express the information of route and train in library files

Association and Aggregation: None

Describing Methods:

+void ReadRoute(): read the number of route in library files
 +void ReadLine(): read the information of route in library files
 +void ReadTime(): read the time of route in library files
 +void ReadJour(): read the information of Journey in library files

Test Class(control)

Operations:

Responsibility: Get the information of objects

Realization: Implement the functions of Journey keep working

Attributes:

-re: String: express the information of route in library files

-back: String: express the information of route and train in library files

Association and Aggregation: None

Describing Methods:

+void ReadRoute(): read the number of route in library files

+void ReadLine(): read the information of route in library files

+void ReadTime(): read the time of route in library files

+void ReadJour(): read the information of Journey in library files