Table of Contents

[Summary of Responsibilities and Achievements 2](#_Toc483597416)

[I. Introduction 4](#_Toc483597417)

[II. Project Management 5](#_Toc483597418)

[1. Outline planning 5](#_Toc483597419)

[1.1. General Architecture 6](#_Toc483597420)

[2. Sprint cycle 6](#_Toc483597421)

[2.1. Sprint 1 6](#_Toc483597422)

[2.2. Sprint 2 7](#_Toc483597423)

[2.3. Sprint 3 8](#_Toc483597424)

[3. System wrap-up 8](#_Toc483597425)

[III. Requirements 8](#_Toc483597426)

[1. Techniques Finding 8](#_Toc483597427)

[1.1. Document or record sampling 9](#_Toc483597428)

[1.2. Interviewing 9](#_Toc483597429)

[1.3. Background Reading 9](#_Toc483597430)

[1.4. Conclusion 9](#_Toc483597431)

[2. Requirements Capturing Steps and Understanding of System 9](#_Toc483597432)

[2.1. Listing customer requirements 9](#_Toc483597433)

[2.2. Understanding system context 10](#_Toc483597434)

[2.3. Assumptions 10](#_Toc483597435)

[3. General Functional Requirements 10](#_Toc483597436)

[3.1. Movie Management 10](#_Toc483597437)

[3.2. Seat Management 10](#_Toc483597438)

[3.3. Administrator Management 10](#_Toc483597439)

[3.4. Ticket ID Management 11](#_Toc483597440)

[4. Non-Functional Requirements 11](#_Toc483597441)

[4.1. Privacy and safety 11](#_Toc483597442)

[4.2. Usability 11](#_Toc483597443)

[4.3. Reliability 11](#_Toc483597444)

[5. Software Prototype 12](#_Toc483597445)

[6. Changes in Backlog 12](#_Toc483597446)

[7. Iteration and Estimate in stories 12](#_Toc483597447)

[7.1. Iteration 12](#_Toc483597448)

[7.2. Estimate 13](#_Toc483597449)

[IV. Design and Analysis 14](#_Toc483597450)

[1. Design Class Diagram 14](#_Toc483597451)

[2. Sequence Diagram 14](#_Toc483597452)

[3. Identify Entity Class, Boundary Class and Control Class 15](#_Toc483597453)

[4. Add Constrains 15](#_Toc483597454)

[5. Major Operations 16](#_Toc483597455)

[6. Design Principle 17](#_Toc483597456)

[7. Architectural Design 18](#_Toc483597457)

[8. Reusability and changeable 18](#_Toc483597458)

[V. Testing 19](#_Toc483597459)

[1. Test Strategy 19](#_Toc483597460)

[2. Unit Test 19](#_Toc483597461)

[2.1. Current time test 19](#_Toc483597462)

[Test Procedure 19](#_Toc483597463)

[2.2. Admin Login test 20](#_Toc483597464)

[Test Procedure 20](#_Toc483597465)

[2.3. Choose seats test 21](#_Toc483597466)

[Test Procedure 21](#_Toc483597467)

[3. White-box testing 22](#_Toc483597468)

[VI. Implement 25](#_Toc483597469)

[Appendix: screenshot 26](#_Toc483597470)

# Summary of Responsibilities and Achievements

**Group Number**: <57>

**Group Members**: <Yiming YAN 140919341> <Yaji CUI 140922042>

<Yingxin LIN 140921399> <Yiqi DING>

<Yinuo YANG 140919949> <Tianlu ZHANG140921894>

**Group Leaders Name**: <Yiming YAN 140919341>

**Individual Member:**

**<Yiming YAN>**

* Write all code of interfaces of all data and data processing.
* Write codes of movie time choose graphical user interface
* Write user manual.
* Write one third part of analysis and test of the report.
* Write general functional requirements and non-functional requirements of report.
* Add Javadoc comments for each class and method.

I am honoured to be a group leader to lead others to complete this project, from which I better practice the knowledge learned in the software engineering course, but also let me better understand that doing a perfect software not only need a strong ability of code writing, what’s more, the whole team cooperation is the most significant. As a team leader, I not only need to complete my part of the work, but also coordinate the work of each. Meanwhile, I provide them with the necessary help. I am mainly responsible for the software required for the preparation of the necessary data format, and provide the interface to deal with data to provide the necessary data for other class construction. These data include the construction of ticket numbers, movie information, sold ticket information, and administrator user passwords. In addition to this I am also responsible for building the time of the film selection interface, and to ensure that the film has been played can not be selected. Thank you very much for the coursework of my exercise, but also to thank my team members for my support. The lack of any of them can not successfully complete the project.

**<Yaji CUI>**

* Write three screen choose codes.
* Write readme.txt.
* Beautify all graphic user interfaces.
* Write one third of analysis and design part including UML and sequence diagram.

My work is to optimized interface(GUI) of the project, making it more attractive, and design some of the diagram in the final report. I learned a lot through the coursework, from discussing what the key use case are to thinking by my own. I did a fairly good job in my point of view because I finished the job with a whole view of the program and deeper understanding. When I set about doing the sequence diagram, I can draw it very fast with a prospect of the whole program.

**<Yingxin LIN>**

* Write the code of the main movie choose graphic user interface.
* Write the report of the management part.
* Write the second half of requirement part of report.

**<Yiqi DING>**

* Write codes of two confirm windows,
* Provide test strategy and test the whole software.
* Write test section of report.

Firstly, the biggest issue was when I was assigned to handle with writing about test strategy, I had no idea how to write. Even after a throughout reading of documents, I wasn’t even capable for coming out an unshaped draft. After asking teammates for help and referring to the previous’ work, I finished the work elaborately. Furthermore, when I was cooperating with code, some bugs occurred out of expectation. I tried solving it on my own, notwithstanding hardness. However, it turns out that asking teammates for help is more beneficial. After all, it is a comfort that the software runs successfully, and our team work turns out to be extraordinary.

**<Yinuo YANG>**

* Design backlog.
* Write administrator code.
* Write one third part of analysis and design.

I am good at programming writing, and performs well in communicate with the other team members, which makes the team members work together and stick together. He works with a deep warm heart to listen to others` advice and provide proper help.

In this process of software development, I made contributions to finish part of the entire cinema system, and also draw the relative icons in the report. I realize the power of teamwork. When I get into troubles about code writing, they always provide constructive advice and solve these problems. The outcome of the project is the process of overcoming difficulties, and I think I will be useful for my future.

**<Tianlu ZHANG>**

* Write code of welcome and success.
* Write front half of requirement part of the report.

In this process, I learnt a lot from my group and be glad to improve my programming ability which is a very weak thing for me before. When using agile method in the software development process, I have a very nice experience on programming and software process. During the part of the report, I responsible for the requirement part after discussion with my group. This also improves my conclusion ability.

# 

# Introduction

Cinema self-service ticket system is based on java. The ticketing system includes features such as movie selection, selection of schedules, and seat selection, as well as an interface for administrators to facilitate viewing and analysing today's ticket sales. This report describes in detail the whole process of project development. First introduced our engineering requirements, and introduced the project management related methods. Next, we introduce the design principles of each class, and offer a clear analysis of the quality of the code. Finally, our team code for a complete test to ensure that the system can work properly.

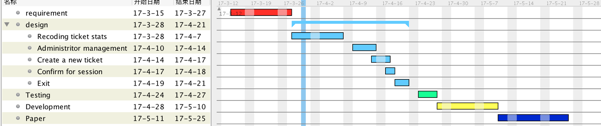
# Project Management

In this project, we are going to develop a self-service ticketing kiosk and we apply an agile project management method – Scrum to manage iterative development rather than specific technical approaches to agile software engineering.

We divided this project into three phases a. outline planning b. sprint cycle c. project wrap-up.

## Outline planning

This system is designed for cinema to save labour cost. After reading the need document, we firstly divided this project by the role. For customer, they can choose the seat and film they want to see on the screen. Besides, customers at different age can buy tickets at different discount. Because system should satisfy a re-usable principle, an administrator is necessary. For administrator, they are able to add different film information to the system after logging into administrator account.



### General Architecture

The first step is building general target and system architecture.

Details about planning estimating, decision making and adapting to changes are mentioned in the next sprint cycle part.

## Sprint cycle

This project has THREE sprint cycles, and each cycle last about two weeks or so.

### Sprint 1

In sprint 1, the main architecture of this system is finished, this system can operate the main function fluently.

|  |  |  |
| --- | --- | --- |
| Story Name | Iteration | Acceptance Criteria |
| Recording tickets states | 1 | 1. File can be read or written and record information rightly without IO exception.  2. Contents can be recognise and send information to each class constructor. |
| Administrator password protection | 1 | The background interface can enter only if the username and the password are both correct. |
| Go back to welcome interface | 1 | After clicking the button, customers can return former interface. |
| Get brief information | 1 | Interface display correct information of each movie. |
| Choose a movie | 1 | 1. Customers can choose movie through scoller without errors.  2. Interface can jump to the next time choosing interface. |
| Choose time | 1 | 1. Customers can choose time through clicking button without errors.  2. Interface can jump to the next seat choosing interface. |
| Input student ID | 1 | 1. Windows can be input student ID.  2. ID can be get by class.F16 |
| Verify student ID | 1 | ID can be verified within the student ID rule. |
| Confirm after choosing time | 1 | 1. Confirm window shows correctly.  2. Click 'yes' to continue the purchasing process.  3. Click 'no' to return this interface. |
| Admin exit | 1 | Exit the program successfully |

### Sprint 2

In sprint 1, we finish the infrastructure of the system. Thus in sprint 2, we mainly focused on details of the system and make it more stable.

|  |  |  |
| --- | --- | --- |
| Story Name | Iteration | Acceptance Criteria |
| Administrator login | 2 | 1. Administrator can sign in.  2. There is no error occurring during the login process. |
| Get the sale report of tickets | 2 | Once the manager enter the system successfully, the information including the total sale of each film, total number of tickets sold and each type of ticket sold should be printed out. |
| Create ticket ID | 2 | Software can generate ID correctly, including:  1. Exact number of ID.  2. Correct movie name.  3. Correct movie time.  4. Correct seat number. |
| Buying multiple tickets | 2 | 1. Tickets ID numbers can generate correctly.  2. Tickets stats can be written into TicketFile.txt.  3. Interface can be return to the start interface. |
| Confirm after choosing the seat | 2 | 1. Confirm window shows correctly.  2. Click 'yes' to continue the purchasing process.  3. Click 'no' to return this interface. |

### Sprint 3

Sprint 3’s target is the most complicated function of the system

|  |  |  |
| --- | --- | --- |
| Story Name | Iteration | Acceptance Criteria |
| Choose seats | 3 | 1. Customers can choose seat through clicking button without errors. |

## System wrap-up

After finishing the different part of the system, we finally integrate different functions into a whole complete system and finish the required documentation like the user manual and development doc.

# Requirements

## Techniques Finding

In techniques finding process, we use fact-finding techniques to find the functional and non-functional requirements of our system.

### Document or record sampling

We first search the internet about what a real ticket booking system looks like, and compare the searching result with the handout. According to the result mentioned above, we figure out several necessary function of the system – choose seat, movie and movie time and confirm purchasing. The existing patterns may not be the best way to model the new system, thus we change some non-functional or functional requirements to meet our need.

### Interviewing

A good way to understand user requirements is to interview different user. A generic system needs feedback of generic people.

Using this method, we get more specific feedback and information about the system and what functions a user really need and which part needs to be changed. Since the roles involved in the system are customer and administrator, we also need to interview the user experience of administrator, and make changes according to their feedback. User feedback is one of the most important part in system’s iteration.

### Background Reading

* Collect information about theater seat arrangement and tickets’ price list and what function a self-service ticket kiosk should have.
* A self-service ticket kiosk must be user-friendly.
* Reusability is an important part of the code, thus external function can be added to the system.

### Conclusion

Concluded from the survey above, we get the requirement below.

|  |  |
| --- | --- |
| **Role** | Requirements |
| Customer | 1. Choose seat 2. Get brief movies information 3. Select movie time 4. Change the selection |
| Administrator | 1. Generate statistic report 2. System safety 3. Confirm user’s purchasing |

## Requirements Capturing Steps and Understanding of System

### Listing customer requirements

* According to customers
  + Customers has an easy access to self-service.
  + Customers can choose the movie which they want to see.
  + If they choose a wrong movie, customers can return to re-select movie
  + Customers cannot buy a ticket which the time is before current time.
  + Customers can select one or more tickets at one time.
  + Customers can easily choose what kind of ticket they want to buy.
  + Customers cannot choose seats which are ordered by others to avoid conflicts with other customers.
  + Customers can pay for tickets at a convenient way.
* According to administrator
  + Administrator has an easy access to see report.
  + Administrator can create a new account or change the password.
  + Administrator wants that only he can login to the system and see the report, meanwhile, get a report from his e-mail.
  + Report should include more detain of sales today.
* According to basic function
  + Tickets cannot be generated if customers input an incomplete information.
  + Different information including movie information, tickets sold information and administrator account information should be stored at different level of secuirity if possible.
  + Ticket ID can be generated automatic and all information about this ticket are stored at ticket ID.

### Understanding system context

This system is not Internet-based, thus it should be optimized for offline operation and there is no database link to the system.

### Assumptions

* Every customer should obey the rules of ticket buying.
* No external attack.

## General Functional Requirements

After the requirements capturing steps, we figure out that general functional requirements.

### Movie Management

* Movie information should be easy edit in Movie.txt
* Only the administrator is able to change it.

### Seat Management

* Seats states can be read from files.
* Seat interface should be display differently.
* Users and administrator should not edit the states of the seats.

### Administrator Management

* Administrator has right to see the report.
* Administrator can create new account or change the password.

### Ticket ID Management

* Ticket ID should be managed by system.
* Ticket ID should include all information of it.
* Ticket ID should be generated following the given rules.

## Non-Functional Requirements

After the requirements capturing steps, we figure out that general non-functional requirements.

### Privacy and safety

* Only administrator can get know of the sales of today.
* Customers cannot see others tickets ID and information.

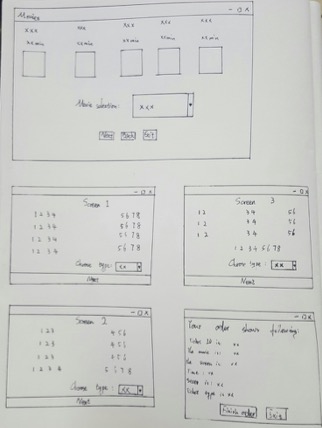
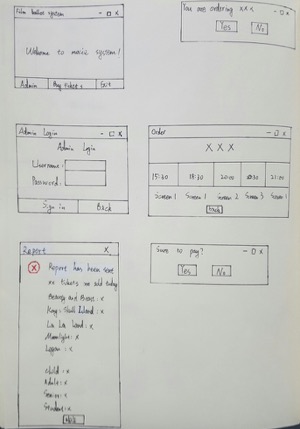
### Usability

* In ticket-buying interface, all the buttons should be triggered as the next step. They should not trigger certain functions that beyond the user’s authority.
* Navigation buttons such as NEXT should be clear for all users.
* The system can output if the information is not complete, however, output is null. That is used for avoid exceptions.

### Reliability

* No other environment except JAVA JDK is needed.
* No unprivileged action is allowed.

## Software Prototype



## Changes in Backlog

After a period of working time, we find the original backlog is not very suitable. Not only for the time planning aspect but also the stories itself, so we need to change to cope with the coursework demonstration in June 4th. The changes of product backlog is in several aspects as follows.

1. We add the discount price.
2. We add a time control to read the real time so that overdue can not buy tickets.
3. We add a screening option to the report so that it can show ticket ordered by movie or by ticket type.
4. We add a background to the GUI interface to make it more attractive.
5. We make the administrator’s account and password can read from the file.
6. We also put the code deadline on May 23th.

## Iteration and Estimate in stories

After combining the above and a period of working time, we get the final product backlog.

### Iteration

After changing backlog, we only have 4 iterations. Because of time-limited, we shorten each iteration and decrease the number of iterations.

1. Iteration 1: At this iteration, we have 11 story functions to implement. These functions are basic for film ballot system. We finished movie choosing function, exit function, back to the last interface function and so on from March 30 to April 22. Since we have so much stories in this iteration, we start it at a different time.
2. Iteration 2: Most of this iteration is about the movie ticket. At this iteration, we have 4 story functions to implement. Excluding the administrator login, others are all about ticket. Such as buy multiple tickets, confirm after choose seats and so on. We start it from March 31 and in April 18.
3. Iteration 3: From April 4 to April 8, we enter into the third iteration. This iteration we maintenance the previous function that we already finished and add the seats choose function. The customer can see what seat can be chose in this interface
4. Iteration 4: Finally, we implement more extra functions from May 11 to May 23. Since we already implement the basic functions, we start extend our functions in this stage, such as the student discount ticket and the ticket sales report that administrator can see after log in.

### Estimate

When we estimate the time, we take the story point method to make the estimation more accurately. When we have designed all stories and finished priory rank, we start to estimate the time from the first function. We don’t know how much time we will need, so we arrange quite a lot stories in the first iteration which contains a long time. After we finished the first one, we found that the story we planned in one iteration is too much and we don’t need to do so much work in the same time (about 11 stories). So we change the plans and time. Concentrating on a small amount of work at the same time improves our efficiency. After we almost finish the system, we find that we still have enough time to complete the details we deleted before to reduce the time. So we add some extend functions and extend the iteration again.

At last, we finished the system at May 23. Comparing with the original backlog, we change a lot which mostly are concentrated on the time because of the uncertain demonstration and original wrong estimation. Fortunately, we finished all thing step by step.

# Design and Analysis

## newfile.cld.jpgDesign Class Diagram

## WechatIMG68.jpegSequence Diagram

## Identify Entity Class, Boundary Class and Control Class

|  |  |  |
| --- | --- | --- |
| Entity Class | Boundary Class | Control Class |
| Admin | Confirm | FileContentHandling |
| Movie |  | Ticket |
| Time |  |  |
| Screen1 |  |  |
| Screen2 |  |  |
| Screen3 |  |  |
| Success |  |  |

Entity classed form the graphical interface of the entire cinema system, which satisfies all the functional requirements. In each entity class has a button connected with another interface. The boundary class Warning is used to warn users if they want to confirm their needs. For example, when the user chooses the wrong choice, the system will give the user enough tolerance to let them return to the previous interface to re-select, and Confirm have a warning function for users whether they are sure to pay for. In the control class, FileContentHanding is used to read and write files and split the contents to find what system needs.

## Add Constrains

* **Welcome**: This boundary class is used for everyone who use this system will enter this interface to choose whether to data management as a staff member or tickets purchasing as a customer by click two buttons.
* **Admin**: This boundary class is used for the user to enter the passwords and username. If they match the relative information in the system file, the selling number can be printed out.
* **Movie**: This is a boundary class. The movie posters extracted from file (jpg) will be printed in the screen. There is a combo box for customer to choose the prefer movie.
* **FileContentHandling**: Be used to get the file path, extract information in the txt file, and make it accessible.
* **Time**: This is a boundary class. The relative time and screen of the chosen film will be shown in the screen.
* **Confirm**: This boundary class is used to confirm your check and alert you to make the right chooses. Unless you make sure it’s your order, the system won’t let you go to the next page.
* **Success:** This boundary class is used to print out how much money a customer should pay for and show all the details about your booking tickets, including the students number, the money, the seats, the name of the movie and so on.

## Major Operations

* **adminLogin()**

**Responsibility:** Administrator can sign in and see the static report of sales of tickets sorted by ticket type and movie kind.

**References:** Login using case.

**Exceptions:** If the administrator input a wrong username or password, he cannot login successfully.

**Output:** Login successfully and sent email to mailbox.

**Pre-condition:** Username and password should be consistent with files Administrator.

**Post-condition:** Get the report.

* **chooseMovie()**

**Responsibility:** Customers can choose the movie.

**References:** Choose movie with case.

**Exceptions:** Customers choose a wrong movie will enter first movie time interface.

**Output:** Specific movie timetable

**Pre-condition:** Customers can choose a movie which they want and movies should follow the timetable.

**Post-condition:** Get into time choice interface and system memory the movie used for generating ticket ID.

* **chooseTime()**

**Responsibility:** Customers can choose a specific movie’s time

**References:** Choose time with case.

**Exceptions:** Customers choose a time which has started, the warning will let them know and re-select a time.

**Output:** Screen 1/2/3 seats figure

**Pre-condition:** Customers choose a correct time of the movie.

**Post-condition:** Get into seat choice interface and system memory the time used for generating ticket ID.

* **chooseSeat()**

**Responsibility:** Customers can choose seats whatever they want

**References:** Assign seat states by using case.

**Exceptions:** If the seat is reserved, seat will show specific symbol to show customers.

**Output:** Purchase success.

**Pre-condition:** Choose a right movie and a correct time.

**Post-condition:** Get into seat success interface and system memory the seats numbers used for generating ticket ID.

* **purchaseSuccess()**

**Responsibility:** Show information of what customers have purchased successfully.

**References:** Show information using a case.

**Exceptions:** If customers choose nothing, it also shows nothing.

**Output:** Information of tickets.

**Pre-condition:** Choose a right movie, time and seats.

**Post-condition:** Return to movie choice interface.

* **generateReport()**

**Responsibility:** a

**References:** Get report information from handling ticket file.

**Exceptions:** If nothing sold, system display nothing. If file does not exist, system can generate a new file.

**Output:** E-mail has been sent to administrator mailbox and display a message box on the screen

**Pre-condition:** Correct username and password.

**Post-condition:** E-mail sent and report display.

## Design Principle

1. **Interface Segregation Principle (ISP)**

Clients should not be forced to depend upon interfaces that they do not use. When we design the interface, we talk about the interface as much as possible and each interface is very small.

For example, when the movie information is displayed, the information interface offer information such as movie name, movie time and other information and there is no need to obtain information such as cinema seating (useless information relative to this interface). The advantage of this is that if we need to change the interface information such as getting the movie name, we only need to modify him. There is no need to force customers to adapt to the interfaces they do not need.

1. **Single Responsibility Principle (SRP)**

The principle raises an ideal expectation of the object's responsibility. The object should not take too much responsibility to ensure that the object is highly cohesive. The principle of a single duty is also conducive to the stability of the object. The less the responsibility of the object, the less the dependency between the objects, the less the degree of coupling, the less the restraint and containment of the other objects, thus ensuring the scalability of the system.

In our system, each object only assumes that its own interface generation and monitored for the associated action. Each class has only a few relationships with other classes, and the dependency between each class is low. For example, if we delete an interface, the other interface is still able to run properly. Or that we want to add a new interface, which is also easy.

1. **Open-Closed Principle (OCP)**

Software modules should be open for extension and closed for modification. It is useful that class can be extended. There are many class in our program can implement the kind of principle, for example there are many kinds of interface and they are essentially similar. We can modify a little to distinguish them but their parent classes cannot be modified by children classes.

1. **Don’t Repeat Yourself (DRY)**

During software development, we followed the principle of DRY to ensure the readability of software code and maintainability. For example, some of the functions of the button is very similar, so we only need to write a method can be called multiple calls. In addition, some of the statements generated by the interface is also very similar, we can also use the same method to reduce the amount of code

1. **Liskov Substitution Principle (LSP)**

We’ve designed our subclasses with extreme caution so that methods are not overridden in a way that changes assumptions about their behavior, which would cause the code to behave in an unexpected way.

## Architectural Design

Defines the relationship between major structural elements of the software. The overall structure of the software and ways in which structure provides conceptual integrity for a system.

## Reusability and changeable

Our project maximizes the degree of cohesion and minimizes the coupling degree.

**Coupling**: is the degree of interdependence between software modules; a measure of how closely connected two routines or modules are the strength of the relationships between modules. All these functions in this project are grouped by the relationship of design pattern, connected to each other by the referencing method.

Each entity class has a few simple, stable, effective interfaces connect with other. For example, in the project, each class has button, which users can click it and construct a new GUI and close current GUI. There is no information interaction. In addition, each entity class only use one or two methods in file content class to get information.

This low-coupling design makes program modifications very simple, especially when we modify the logic between the interface and changing the function required to add or subtract some items for the log file. And if any other class problems occurring, they will not affect the activities of others.

**Cohesion:** refers to the degree to which the elements inside a [module](https://en.wikipedia.org/wiki/Module_(programming)) belong together. Thus, cohesion measures the strength of relationship between pieces of functionality within a given module. Each class has highly relevant methods and variables. For example, in the file processing class, there are ways to read specific content of a file, and there are ways to write specific content to a file. In other GUI generation class, each class is all in the service to create the interface and the way to monitor the action.

High cohesion reduces module complexity and increases system maintainability, for logical changes in the domain affect fewer modules. Increased module reusability, because application developers will find the component they need more easily among the cohesive set of operations provided by the module.

Hence, the system is convenient to change any code and the programme is robust.

# Testing

## Test Strategy

This software is providing convenience for both audiences and administers of the cinemas, by functions of selecting movies, screens, time, seats etc. The strategy defines the approach for software testing and result evaluating. Most of the tests are automated. Regression and unit test are used for the process of blocks. And system test for whole software.

## Unit Test

We use Unit Test to test the validity of the program on the minimum function.

As requested in the handout, the software should be capable for taking current time into account. Then we choose to use unit test with input of time to determine it.

### Current time test

|  |  |  |
| --- | --- | --- |
| Test cases | Test case 1 | Test case 2 |
| Input(time) | Different time before the start of the movie | Different times which are after the start time of the movie |
| Output | Return true | Return false |
| Condition | There are available movies | There is message box warn users |

### Test Procedure

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test case:( time) | Test description | Test case | Pass/Fail | Numbers of bugs | Bug# | comments |
| 1.1 | Test the time of 10:40 | 1.1 | p | 0 | 0 | At 10:40, screen 1 for BEAUTY AND THE BEAST at 10:00 is not available. |
| 1.2 | Test the time of 11:00 | 1.2 | p | 0 | 0 | At 11:00, screen 3 for BEAUTY AND THE BEAST at 13:00 is available. |
| 1.3 | Test the time of 13:30 | 1.3 | p | 0 | 0 | At 13:30, screen 2 for MOONLIGHT at 16:00 is available. |
| 1.4 | Test the time of 13:30 | 1.4 | p | 0 | 0 | At 13:30, screen 2 for LA LA LAND at 13:00 is not available. |
| 1.5 | Test the time of 22:00 | 1.5 | p | 0 | 0 | At 22:00, no movie is available |

### Admin Login test

|  |  |  |
| --- | --- | --- |
| Test cases | Test case 1 (Wrong input) | Test case2 (Correct input) |
| Input | Wrong username or password | Correct username and password |
| Output | Return false | Return true |
| Condition | Login failed | Login successfully |

### Test Procedure

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test case:( time) | Test description | Test case | Pass/Fail | Numbers of bugs | Bug# | comments |
| 1.1 | Input wrong username and wrong password | 1.1 | p | 0 | 0 | A dialog box shows to tell the user the wrong username. |
| 1.2 | Input correct username and wrong password | 1.2 | p | 0 | 0 | A dialog box shows to tell the user the wrong password. |
| 1.3 | Input wrong username and correct password | 1.3 | p | 0 | 0 | A dialog box shows to tell the user the wrong username. |
| 1.4 | Input correct username and correct password | 1.4 | p | 0 | 0 | Report display and is sent to e-mail. |
| 1.5 | AdminFile does not exist | 1.5 | p | 0 | 0 | A dialog box shows to tell the user the wrong username. |

### Choose seats test

|  |  |  |
| --- | --- | --- |
| Test cases | Test case 1 (Wrong input) | Test case2 (Correct input) |
| Input | Choose nothing | Choose one or more seats |
| Output | No ticket is generated | Return true |
| Condition | Order tickets failed | Order successfully |

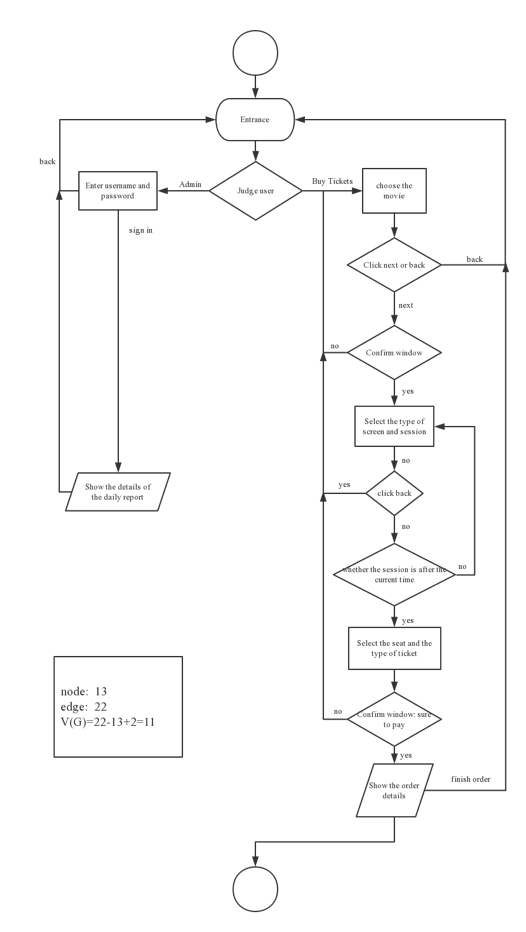
### Test Procedure

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test case:( time) | Test description | Test case | Pass/Fail | Numbers of bugs | Bug# | comments |
| 1.1 | Rightly choose a movie and a time but choose none of the seats. | 1.1 | p | 0 | 0 | Tickets generate failed and shows nothing has been ordered because users should choose one or more seats. |
| 1.2 | Rightly choose a movie and a time and one seat. | 1.2 | p | 0 | 0 | One ticket is generated successfully and shows information of it. |
| 1.3 | Rightly choose a movie and a time and three seats. | 1.3 | p | 0 | 0 | Three tickets are generated successfully and shows information of them. |
| 1.4 | Lack of Ticket.txt | 1.4 | F | 0 | 0 | System cannot shows seats which has been order correctly. |

## White-box testing

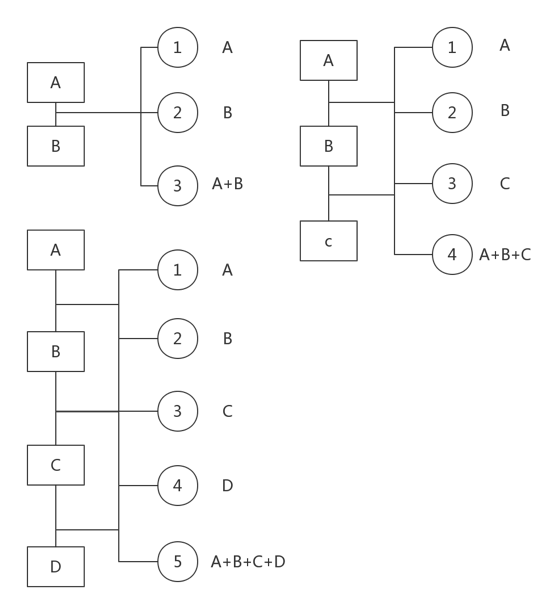
We use white-box testing technique to test the internal program logic. Using this method, an elaborate graph can well organize the whole process for further understanding of this software.

In this graph, we include the main process of the software. We also take the confirm window in the graph. Some small blocks are not presented.



Due to the page limit, we only discuss the situation of audiences in different aspects below.

We suppose A=select movie, B=select the type of screen and session, C=current time, D=select the seat and the type of ticket



In audience angle, we test the function of selecting movie first, and then test with screens and time selection together. We also include two different cases of time. One is that sessions are available, the other -the cinema has no available session. And then we add seat and the type of ticket, to find whether there is discount for child, student and senior. During those actions, we are going to find whether errors or bugs exit.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test case | Test description | Pass/Fail | Numbers of bugs | Bug# | comments |
| A | Write Junit to test | p | 0 | 0 | Successful running as expected |
| A+B | Test A+B when there are available movies | p | 0 | 0 | Successful running as expected |
| A+B+C | Test A+B+C when there are available with different types of movies | p | 0 | 0 | Successful running as expected |
| A+B+C | Test A+B+C when there are no available sessions | P | 0 | 0 | It is coordinate with the request of current time. Since there are no movies available, user can’t select sessions |
| A+B+C+D | Test A+B+C+D when there are available seats | p | 0 | 0 | Successful running as expected |
| A+B+C+D | Test A+B+C+D by users in different ages | p | 0 | 0 | Successful running as expected |

We use regression test to make sure the capability of each component by using audience angle. And all the blocks are running with expected and coordinate with request from handout.

# Implement

We implement the system's java language and apply the agile software development model. The system code is pasted in the appendix. The management approach exists in the second part of the analysis in the third part, the design and analysis are located in the fourth part. The last part of the discussion we discussed the strategy and methods.

# Appendix: screenshot

