COMP 354 Intro to Software Development
Fall 2023

Final Project

Quality Control for Concordia Cart App

Weilun Zhang 40190549
Jingchao Song 40159533
ZhaoYang Liu 40157299
Haoran Sun 40159349
Qianrui Tao 40162182
Xu Zhang 40169981

Due date: Nov 30, 2023

[Final Project GitHub Link]

1. Schedule

We create an effective strategy for project execution involving meticulously breaking down tasks and timelines. By thoroughly segmenting the project into smaller, manageable components, we accurately estimated the time needed for each task. These tasks were then thoughtfully assigned to team members based on their expertise and capabilities. This thorough method helped us have a clear plan for progress, making sure the team worked efficiently.

Task Description	Task Assignment	Planned Time Frame
Display Most and Least Selling Items for Customers (Adding extension code and creating UI)	Haoran Sun and Xu Zhang	180 minutes
Display Discounts on Products for Customers (Adding extension code and creating UI)	Weilun Zhang	200 minutes
Add Low-Cost Used Products for Students (Adding extension code and creating UI)	Haoran Sun	160 minutes
Suggest Discounts on Most and Least Selling Products for Admin (Adding extension code and creating UI)	ZhaoYang Liu	160 minutes
Suggest Restocking for Admin (Adding extension code and creating UI)	Qianrui Tao	180 minutes
Project Report	Development Team	150 minutes

2. Risk Analysis

Risk Description	Criticality	Mitigation Plan
Project design pattern	High	We follow our established design patterns for a structured and maintainable codebase strictly and avoid major change of existing codes.
Test-driven Development	High	We use test-driven development to identify and prevent potential mistakes in new functions of the shopping cart website.This helps ensure the correctness of the new website.
Performance Issue	Medium	We modify the web content part to make additional functions more suitable to the new website.This allows people to identify the improvement of the new shopping cart website.
User experience Issue	Medium	We try to implement as easy as possible to help users who use new functions of this website. We provide clear guidance for users to use new functions of this website.
Reliability of modified code achievement	High	We review the code after completing it . We also follow the general pattern design and try not to make any difference as we planned.
Code comment	Medium	We write comments to all new added code to help

		other team members can handle the code and continue the rest of task

3. Materialized Risks

Communication Challenges:

- If team members misunderstand functional requirements, it could lead to discrepancies in the understanding of what needs to be developed. We conducted regular meetings to clarify expectations and ensure everyone was on the same page

Skill Misalignment:

- Inadequate programming skills and experience within the team could lead to technical obstacles or limitations throughout the development process. We assigned tasks based on each member's abilities.

Time Pressure and Limited Technical Support:

- Limited technical support and strict timelines may compromise quality and hinder project progress due to urgent delivery deadlines and insufficient resources to address technical challenges. We adjusted task assignments according to the strengths of group members and could optimize resources and address technical challenges efficiently. Moreover, We started working as soon as we received the task.

4. Implementation & Testing

In the project's implementation, we've employed the MVC (Model-View-Controller) pattern to compartmentalize the components. The classes are organized as follows:

Customer Extension 1

Model

ProductBean.java

- We added a new private variable (private int prodUsedQuantity;private double prodUsedPrice;private double prodDiscountPrice;private String

Condition) and getter & setter method to be able to display the most selling and least selling product information.

- **ProductBean.java** has these variables to display the information of products:
    1. private String prodId
    2. private String prodName
    3. private String prodType
    4. private String prodInfo
    5. private double prodPrice
    6. private int prodQuantity
    7. private InputStream prodImage
    8. private int prodUsedQuantity (added for extension)
    9. private double prodUsedPrice (added for extension)
    10. private double prodDiscountPrice (added for extension)
    11. private String Condition (added for extension)
- **ProductBean.java** has a constructor and getter & setter methods used to access and modify the private variables of a class.

ProductService.java

- We added some new methods to implement the discounts based on the user's interests and most selling items.
    1. **public** boolean sellNUsedProduct(String prodId, int n): This method is used to compare products that users have purchased in the past.
    2. **public** List<ProductBean> getLowStockProducts(): This method is used to obtain products with less stock in stock.
    3. public List<ProductBean> getUsedProductsByType(String type): it is designed to determine the discount percentage for a given ProductBean
    4. public List<ProductBean> getDiscountedProductsByType(String type): This method is to get all discounted products by entering a product type

View

userHome.jsp

- In this part we add the most popular products and the least popular products and display the information we need to obtain by calling the added variable in the productbean.

cartDetails.jsp

- In this jsp file, we add used and currPrice two variables to display the difference between the used price and current price information.

Controller

AddProductSrv.java

1. We can compare the number of current products with the number of past products to know whether the product is selling well, and then get information about whether the product is a best-selling product.

UpdateProductSrv.java

1. In this class we add three more variables which are: prodUsedQuantity, prodUsedPrice, prodDiscountPrice.
2. Then we update these three values
    - Integer prodUsedQuantity = Integer.parseInt(request.getParameter("used"))
    - Double prodUsedPrice = Double.parseDouble(request.getParameter("usedprice"))
    - Double prodDiscountPrice = Double.parseDouble(request.getParameter("discountprice"))

Customer Extension 2

Model

ProductBean.java

- We added a new private variable (discount) and getter & setter method to be able to display the discount information.
- **ProductBean.java** has these variables to display the information of products:

    1. private String prodId
    2. private String prodName
    3. private String prodType
    4. private String prodInfo
    5. private double prodPrice
    6. private int prodQuantity
    7. private InputStream prodImage
    8. private double discount (added for extension)

- **ProductBean.java** has a constructor and getter & setter methods used to access and modify the private variables of a class.

ProductService.java

- We added some new methods to implement the discounts based on the user's interests and most selling items.

1. public List<ProductBean> getDiscountedProductsForUser(String userEmailId): it aims to retrieve a list of discounted products based on a user's interests.
2. private List<String> getUserInterests(String userEmailId): it is used to retrieve a list of interests associated with a user's email ID from a database table named.
3. private double calculateDiscount(ProductBean product): it is designed to determine the discount percentage for a given ProductBean
4. private int countSoldItem(String prodId): it retrieves the total count of sales for a specific product ID from a table named sales in a database.

View

userHome.jsp

- In this jsp file, we add two discount and discountedPrice variables to display the discount information.
- We also modify the user Home to display the discounted price: <%= discountedPrice %>

cartDetails.jsp

- In this jsp file, we add two discount and discountedPrice variables to display the discount information.
- We also modify the cart page to display the discounted price: <td><%= discountedPrice %></td> <td><%= item.getQuantity() %></td>

Controller

AddProductSrv.java

1. In this AddProductSrv class we apply discounts based on user interests and popular items and we add a new method calculateDiscount(String prodType, double prodPrice) to calculate the discount.

UpdateProductSrv.java

1. In this class we Introduced a field prodDiscount to handle the discount information for the product being updated.
    - Double prodDiscount = Double.parseDouble(request.getParameter("discount"));
2. We modified the retrieval of discount information from the request parameters.
3. We updated the ProductBean with a prodDiscount field and set the discount value.
    - ProductBean product = new ProductBean();

Customer Extension 3

Model

ProductBean.java

- add boolean isUsed and double usedProductPrice
- add these new variables to display the information of products in **ProductBean.java**:
  this.usedProduct = usedProduct;
  this.usedProductPrice = usedProductPrice;
- **ProductBean.java** has a constructor and getter & setter methods used to access and modify the private variables of a class.

ProductServiceImpl.java

- Add a few methods to implement the used products based on the user's interests.
  1. public List<ProductBean> getUsedProducts(): retrieve a list of used products based on the user's interests.
  2. public String updateUsedProductInfo(String prodId, String updatedInfo): show the updated used products
  3. private ProductBean mapResultSetToProductBean(ResultSet rs) throws SQLException: Used to query database results
  4. public String addProduct(ProductBean product):it is used to modify addProduct method to handle used products

View

UpdateProduct.jsp

- In this file, add a user interest block to let customers choose their interests and then the system will display the popular and used products depending on the user's choice.
- Adding a few items to allow customers to choose.
  <option value="textbook">TEXTBOOKS</option>
  <option value="laptop">LAPTOP</option>
  <option value="mobile">MOBILE</option>
  <option value="Camera">CAMERA</option>
  <option value="Tablet">TABLET</option>
  <option value="Speaker">SPEAKER</option>
  <option value="Others">OTHER</option>

addProduct.jsp

- Adding a new field, usedproducts, let customer to choose the used products
  <label for="usedpRroduct">New Field</label> <input type="text"
  placeholder="Choose used product" name="usedproduct"
  class="form-control" id="usedproduct" required>

Controller

UpdateProductSrv.java

- Add new import com.shashi.beans.ProductBean and com.shashi.service.impl.ProductServiceImpl in the **UpdateProductSrv.java** file
- add `boolean usedProduct` and `Double usedProductPrice`

Admin Extension 1

Model:

ProductBean.java

- add boolean isUsed and double usedProductPrice
- add these new variables to display the information of products in **ProductBean.java**:
  this.usedProduct = usedProduct;
  this.usedProductPrice = usedProductPrice;
- **ProductBean.java** has a constructor and getter & setter methods used to access and modify the private variables of a class.

ProductServiceImpl.java

- Add a few methods to make the admin decide if they have to give a discount to a certain product.

private void applyDiscounts():get all the products information and set discount for a product .

private Map<String, Integer> getSalesData():retrieve data from web to obtain sales information and use it for review.

userHome.jsp

- In this jsp file, we add two discount and discountedPrice variables to display the discount information.
- We also modify the user Home to display the discounted price: <%= discountedPrice %>

cartDetails.jsp

- In this jsp file, we add two discount and discountedPrice variables to display the discount information.
- We also modify the cart page to display the discounted price: <td><%= discountedPrice %></td> <td><%= item.getQuantity() %></td>

Controller

AddProductSrv.java

2. In this AddProductSrv class we apply discounts based on user interests and popular items and we add a new method calculateDiscount(String prodType, double prodPrice) to calculate the discount.

UpdateProductSrv.java

4. In this class we Introduced a field prodDiscount to handle the discount information for the product being updated.
    - Double prodDiscount = Double.parseDouble(request.getParameter("discount"));
5. We modified the retrieval of discount information from the request parameters.
6. We updated the ProductBean with a prodDiscount field and set the discount value.
    - ProductBean product = new ProductBean();

Admin Extension 2

Model

ProductBean.java

- Add a new field to store the quantity of each product available in stock.

ProductServiceImpl.java

- This code is a method named getLowStockProducts() that aims to retrieve a list of products having a low stock quantity.
    1. List<ProductBean> lowStockProducts ： store products with low stock quantity.
    2. DBUtil.provideConnection()：It establishes a connection to the database
    3. ps = con.prepareStatement()：to select products from the databasewhere the quantity (pquantity) is less than or equal to 3.
    4. while (rs.next())：It iterates through the result set and retrieves each product row that matches the condition.

View

adminHome.jsp

- Use JavaScript in this file to check inventory via AJAX request and display a popup prompt.
- The Bootstrap modal structure is defined with an ID lowStockModal, and within the modal's body, it iterates through the lowStockProducts list to display the details of the products with low stock.

5. Unit Test

The development of unit tests within the system relies on the Junit testing framework. Its popularity within the Java ecosystem stems from its robust features, ease of use, and capability to facilitate thorough and reliable unit testing procedures, thereby ensuring the stability and integrity of the software under development.

Customer Extension 1

ProductBeanTest

ProductBeanTest has tested one method:

- private int prodUsedQuantity: Check the past quantity of this item.
- private double prodUsedPrice: Check the past price of this item.
- private double prodDiscountPrice: View the discounted price of this item.
- private String Condition: Enter a condition you want to check.

ProductServiceTest

ProductServiceTest has tested these methods:

- public boolean sellNUsedProduct(String prodId, int n): Compare the quantity sold compared to before and in the past.

- public List<ProductBean> getLowStockProducts():Get information about low quantity products in stock.

- public List<ProductBean> getUsedProductsByType(String type):Enter a type to retrieve past product information.

- public List<ProductBean> getDiscountedProductsByType(String type):Get information about discounted products by entering a type.

Customer Extension 2

ProductBeanTest

ProductBeanTest has tested one method:

- setDiscount: to check if the set discount matches the expected discount.

ProductServiceTest

ProductServiceTest has tested these methods:

- public List<ProductBean> getDiscountedProductsForUser(String userEmailId): to verify that the method correctly filters products based on a user's interests and applies discounts accordingly.
- private List<String> getUserInterests(String userEmailId): Checks if the method fetches user interests correctly from the database.

- private int countSoldItem(String prodId): Ensures that the method correctly counts sold items for a given product ID.
- private double calculateDiscount(ProductBean product): Verifies that the method calculates discounts accurately based on given criteria.

Customer Extension 3

ProductBeanTest

ProductBeanTest has tested one method:

- setUsedProductst: to check if the set used products matches the expected used products.

ProductServiceTest

ProductServiceTest has tested these methods:

- public List<ProductBean> getUsedProducts(): to verify the method correctly filters products based on a user's interests and displays the used products correctly.
- public String updateUsedProductInfo(String prodId, String updatedInfo): Checks if the system can update the used products based on the user's interests.
- private ProductBean mapResultSetToProductBean(ResultSet rs) throws SQLException: Ensures that the method correctly query database results.
- public String addProduct(ProductBean product): the system displays products that have already been added, and handles the used products.

Admin Extension 1

ProductServiceTest

ProductServiceTest has tested these methods:

- Private void applydiscount: retrieve data if it satisfies the condition to apply discount and which discount it is going to be.
- Private Map<String integer> getSalesData: check it gets data from website successfully and it is the correct value as we wish to get.

Admin Extension 2

ProductBeanTest

ProductBeanTest has tested one method:

- List<ProductBean> getLowStockProducts()： The purpose of this method is to retrieve a list of products with low stock and return a collection containing objects of type ProductBean.

ProductServiceImpl Test

ProductServiclmpl Test has tested one method:

- public List<ProductBean> getLowStockProducts()：The name of this method indicates its purpose to retrieve a list of products where the stock is lower than or equal to 3.
- lowStockProducts：Create an ArrayList object to store products with stock lower than or equal to 3.
- DBUtil.provideConnection()：Get database connection.
- ResultSet：Execute the query, iterate through the result set, and extract the product information from each row.
- SQLException：Print the exception information.
- finally：Close the database connection and related resources to ensure proper resource release.

6. Quality Control

To align with the conversation from the final project meeting, we have changed the names of the methods and variables in our Extension. This adjustment can reduce the risk of confusion.

Customer Extension 1

- Changes in Requirements:

Previous Requirements:
Focus on fundamental cart functionalities without specific features designed to showcase product popularity based on user interests and brands.

New Requirements:
Implement a feature enabling customers to access the most and least selling items aligned with their interests and preferred brands.
Based on user interactions, purchases, and preferences to ascertain user interests and favored brands.

- Changes in Design Strategies:

Previous Design:
The initial design lacked modules for analyzing user preferences or presenting popularity based on brand interests.

New Design:
Creating modules to monitor user interactions and purchases, facilitating an understanding of user interests, and favored brands. Develop algorithms that identify

and display the most and least selling items according to user preferences. Create a user interface within the shopping platform for customers to access and view this data.

- Changes in Test Case Approach:

Previous Test Cases:
The previous test cases have addressed standard cart operations, authentication processes, and fundamental product functionalities.

New Test Cases:
Incorporate test scenarios simulating diverse user preferences, brand interests, and sales performances across different products. Verify the precision of the displayed most and least selling items derived from user interactions and brand preferences.

Customer Extension 2

To maintain quality control for the extension that showcases discounts on products tailored to user interests and top-selling items, various crucial aspects require thorough consideration: defining requirements, outlining design strategies, devising test cases, and integration plans.

- Changes in Requirements:

Previous Requirements:
The initial requirements lacked details on tailoring discounts to individual users using browsing and purchase history.
The emphasis appeared to be on fundamental cart functions without personalized discount capabilities.

New Requirements:
Implementing personalized discounts derived from user interests and purchase history. Utilizing browsing data to discern user preferences. Incorporating top-selling items within user interest categories for discount allocation.

- Changes in Design Strategies:

Previous Design:
The initial design only prioritized generic cart functions.
New Design:
Modify modules specifically for scrutinizing user behavior and preferences.
Craft algorithms to pinpoint popular items within user interest segments.
Establish a system to compute and exhibit personalized discounts.

- Changes in Test Case Approach:

New Test Cases:

Incorporate test scenarios simulating diverse user behaviors, encompassing browsing and purchase histories. Integrate scenarios to validate the precision of personalized discount computations. Evaluate user-specific promotions, discount presentation, and their alignment with user activities.

- Changes in Software Quality:

Junit Testing: Safeguard existing functionalities against impacts from new implementations.
Code Reviews and Testing: Conduct thorough code reviews and extensive testing to ensure accuracy and reliability.
Modular Design: Uphold modularity to facilitate scalability and ease of maintenance.
Simulate User Feedback: Integrate simulate user feedback to refine features, optimizing performance.

Customer Extension 3

- Changes in Requirements:

Previous Requirements:
The original design specifications possibly overlooked educational requirements or didn't aim for affordable products tailored to students. The emphasis was on fundamental shopping cart functions, neglecting features specifically beneficial to students.

New Requirements:
Incorporating a functionality to comprehend students' educational needs and preferences.
Recognizing available used products, such as textbooks or laptops, offered at reduced prices.
Displaying these affordable, pre-owned items to students interested in related products to enhance the affordability of education.

- Changes in Design Strategies:

New Design:
Introduce modules designed to identify and categorize pre-owned products suitable for students. Develop algorithms that analyze user interests and pair them with accessible, cost-effective educational items. Implement a user-friendly interface to effectively showcase these items to students who express interest in them.

- Changes in Test Cases:

New Test Scenarios:

Incorporate test cases simulating student interactions, gauging their engagement with educational products, and the presentation of affordable items. Construct test scenarios focusing on user preferences concerning educational materials and their affordability. Validate the precision of suggesting low-cost, pre-owned products to students.

Admin Extension 1

- Changes in Requirements:

Previous Requirements:
The initial requirements have centered on core admin functionalities, lacking specific features for recommending discounts based on sales data.

New Requirements:
Implement a feature enabling Admin to propose discounts on best-selling and least-selling products. Utilize sales data to identify popular and less popular items within designated categories or across the entire inventory.

- Changes in Design Strategies:

Previous Design:
The initial design lacked modules for analyzing sales data or enabling the implementation of discount strategies for Admin use.

New Design:
Incorporate modules to gather and analyze sales data, distinguishing between popular and less popular products. Create an admin interface facilitating the addition of discounts.

- Changes in Test Cases:

Previous Test Cases:
Previous test cases have encompassed basic functionalities and user interactions without specific scenarios related to suggested discounts.

New Test Cases:
Verify the precision of suggested discount rates for both top-selling and least-selling products.
Test the integration of Admin interface functionalities concerning suggesting and implementing discounts.

Admin Extension 2

- Changes in Requirements:

Previous Requirements:
The initial requirements have emphasized basic cart functionalities without restocking features with warnings.

New Requirements:
Introduce a feature that recommends restocking for items with quantities of three or fewer. Prevent popular products from going out of stock to avoid potential sales loss.

- Changes in Design Strategies:

Previous Design:
The original design lacked modules to monitor item quantities or recommend restocking with warning.

New Design:
Incorporate modules for real-time tracking of product quantities. Develop algorithms that detect products with low quantities and recommend restocking. Create an interface for admin users to access and review restocking suggestions.

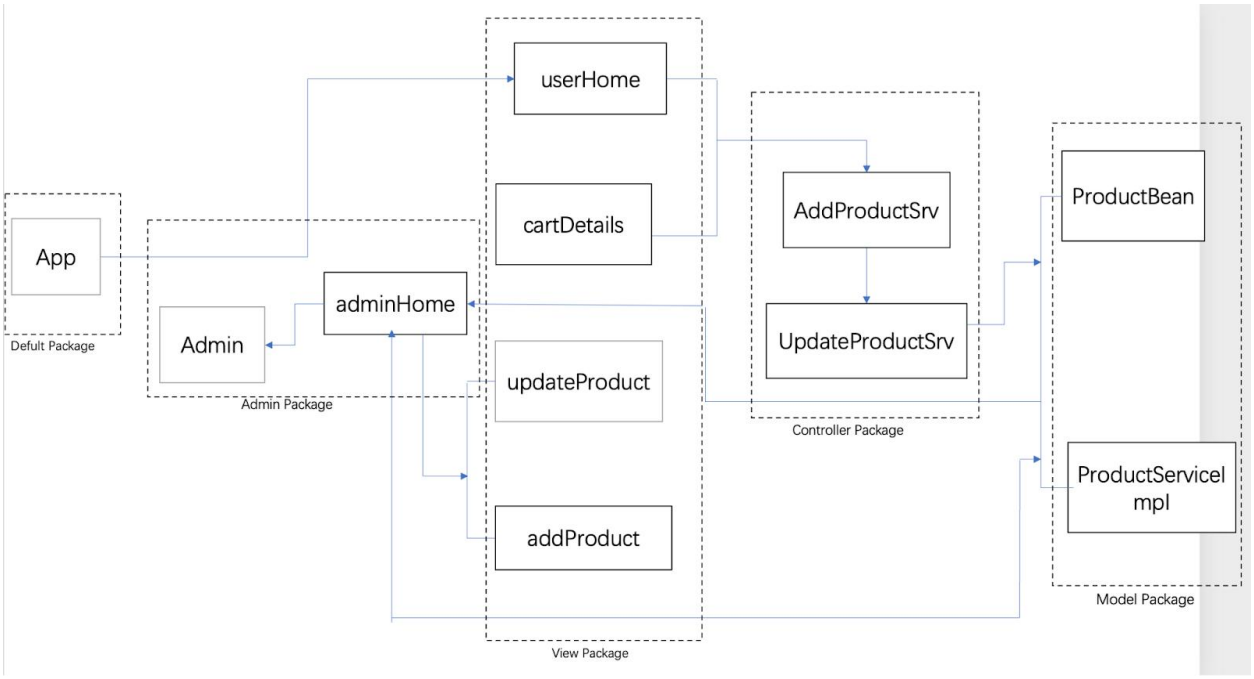
- Changes in Test Cases:

Previous Test Cases:
Test cases have addressed basic functionalities, cart operations, and user interactions but might not have covered scenarios related to low quantity restocking.

New Test Cases:
Incorporate test cases to simulate scenarios where products experience low quantities. Verify the precision of restocking recommendations for products with quantities of three or fewer. Test the functionality of the admin interface concerning viewing and acting upon restocking suggestions.

Components Integration

To complete the software, the components must be integrated with each other



In this program at first the driver class, APP class,acting as the entry point for the application. Upon execution, it initializes the main components of the application, which could be the main window or setup configuration.

ProductBean class is a model class that holds product data and has been extended with new variables to track additional product attributes like used quantity, price, discount, and condition.
ProductService class and its implementation ProductServiceImpl class contain business logic to handle product-related operations such as selling used products, retrieving low stock products, applying discounts, and more. These services are updated to handle the new attributes added to ProductBean.

UserHome and CartDetail are part of the View layer, presenting the user interface. They are updated to display new product information like most and least selling products, used product prices, and discounts. They interact with the Model layer to present the data processed by the services.

AddProductSrv class and UpdateProductSrv are part of the Controller layer, managing user input and model interactions. They handle adding new products, updating existing products with new attributes like used quantity and discount prices, and applying business logic to calculate discounts or compare sales data.

Throughout the extensions, new methods and variables are integrated into existing classes to extend their functionality. For example, ProductBean gains new attributes across extensions, while ProductService implements new logic to support these attributes.

In the View layer, JSP files are augmented to utilize the new Model attributes, displaying updated information to the user.

Controllers gain additional responsibilities to handle the new business rules, such as applying discounts or managing used product data, ensuring that user actions are correctly processed and reflected in the View.

The ProductServiceImpl interacts with the database to retrieve and update product information, including handling low stock alerts that are displayed using AJAX calls in the adminHome.
When the application runs, the App class initializes the View, which presents the user interface. User actions in the View trigger Controller actions, which utilize Services to process business logic and interact with the Model to get or set data. The Model, updated with new product information, sends data back to the View to be displayed to the user.

7. Revision History and Contribution

Revision History

Version	1.0
Complete on	Nov 26, 2023
Reason for changes	First technical review to our main function
Primary authors	Weilun Zhang, Jingchao Song, ZhaoYang Liu, Haoran Sun, Qianrui Tao, and Xu Zhang

Contribution

Weilun Zhang	Customer Extension 2, Junit Testing, and Report
Jingchao Song	Customer Extension 3, Junit Testing, and Report
ZhaoYang Liu	Admin Extension 1, Junit Testing, and Report
Haoran Sun	Customer Extension 1, Junit Testing, and Report
Qianrui Tao	Asmin Extension 2, Junit Testing, and Report
Xu Zhang	Customer Extension 1, Junit Testing, and Report