南京信息工程大学
Nanjing University of Information Science & Technology

Software Engineering Project

PoS System Using Agile

Member 1: Jinru YAO 202283890008

Member 2: Yao YAO 202283890009

Member 3: Yuchen ZHANG 202283890014

Lab ID: Practice Assignment

Date: 13/1/26

GitHub URL:

https://github.com/HarukaYao0514/PoS_System.git

Content

1. Part 1 – The Agile Process 3
    1.1 Project Overview 3
    1.2 Core Business Processes 3
        1.2.1 Process Sale 3
        1.2.2 Handle Returns 3
    1.3 Agile Core Principles 4
        1.3.1 Iteration and Incremental Development 4
        1.3.2 Collaboration 4
        1.3.3 Feedback 4
        1.3.4 Prioritization 4
        1.3.5 Time-boxing: 5
    1.4 Agile Framework 5
        1.4.1 Sprint 1: (2 Days) 5
        1.4.2 Sprint 2: (2 Days) 5
        1.4.3 Sprint 3: (3 Days) 6
    1.5 Tools 7

2. Part 2 – The point-of-sale (POS) system 8

    2.1 Use Case Model 8
    2.2 OO Design / Interaction Diagrams 9
    2.3 Domain Model / OOA 9
    2.4 System Logical Architecture 9

1. Part 1 – The Agile Process

1.1 Project Overview

This project employs an agile development approach based on the Scrum framework to build a simplified supermarket checkout system tailored for small retail settings. The system primarily supports three core functions. The first is checkout functionality: when customers bring items to the register, cashiers input each product code into the system, which displays selected items and the running total in real time. For cash payments, the system verifies sufficient funds, automatically calculates change, and deducts the corresponding inventory. The second is returns processing. When customers bring items for return, cashiers input the returned items and quantities. The system refunds the corresponding amount and restores the item's inventory count. The third is inventory viewing. Users can check the current stock status of all items at any time. Inventory data updates in real-time with each successful payment or valid return, ensuring accurate and consistent information.

1.2 Core Business Processes

1.2.1 Process Sale

The customer arrives at the checkout counter with the items to be purchased. The cashier enters each purchased item into the POS system, which displays real-time information for each item and the cumulative total. The customer makes payment. The system verifies if the payment amount is sufficient, records the payment details, and calculates the change amount. It then automatically deducts the corresponding inventory from the items, generates a receipt for the customer to collect, and the customer leaves with the items and receipt.

1.2.2 Handle Returns

Customers bring the items they wish to return to the checkout counter. The cashier enters each returned item and its quantity into the POS system. The system calculates the refund amount based on this information, processes the refund, and simultaneously restocks the corresponding quantities back into inventory.

1.3 Agile Core Principles

1.3.1 Iteration and Incremental Development

The project adopts an iterative development approach, where functionality is decomposed into multiple small, manageable units. Each unit is completed within an independent iteration cycle, forming a runnable, testable functional module. This methodology avoids the complexity associated with developing all features simultaneously and reduces risks during system integration and debugging.

1.3.2 Collaboration

This project was completed collaboratively by team members, emphasizing effective communication and division of labor throughout development. The team utilized Git for version control, preventing conflicts through feature branching, regular merges, and code reviews. Each member was assigned specific modules—such as checkout logic, return processing, or inventory management — and documented implementation details with clear comments. The code employed a modular design with unified interface definitions to ensure seamless integration. Daily briefings aligned progress, promptly resolved dependency issues, and maintained both development efficiency and system consistency.

1.3.3 Feedback

The project prioritizes feedback collection and responsiveness to changes throughout development. Validation testing is conducted immediately after each functional module is completed. For instance, after implementing the sales functionality, the system checks whether the corresponding product inventory quantities are correctly reduced. Following the implementation of the return functionality, it confirms whether the inventory of returned products is accurately increased. Issues identified during testing are promptly documented, and the relevant logic is adjusted accordingly. This continuous validation and refinement mechanism ensures the system's behavior progressively aligns with actual business requirements.

1.3.4 Prioritization

Task prioritization is based on business value and criticality. High-value features are developed first, such as point-of-sale processing and real-time inventory updates, which directly impact the availability of core system processes. Lower-priority tasks like report generation and demo video production are scheduled for later development phases. This sequencing ensures critical functionality is completed and validated early on.

1.3.5 Time-boxing:

Project development is strictly confined to a seven-day completion window, with the entire cycle divided into three time boxes—Sprints—of two days, two days, and three days respectively. Each Sprint is assigned clear and specific development objectives. Once the time boundaries are established, they cannot be extended. This constraint compels the team to concentrate resources on high-priority tasks within the limited timeframe, preventing scope creep or schedule delays. This approach maintains a steady development rhythm and ensures timely delivery of phased results.

1.4 Agile Framework

1.4.1 Sprint 1: (2 Days)

- **Goal:** Set up project foundation and implement inventory display functionality.
- **User Stories:**
  - As a developer, I aim to establish a clear project structure to efficiently organize and manage code.
  - As a developer, I aim to initialize product information and inventory to provide foundational data for system operation.
  - As a store manager, I aim to view the current inventory status of all products at any time to maintain control over stock levels.
- **Tasks:**
  - Set up the local project structure
  - Create a GitHub remote repository
  - Design the product data model and initialize inventory
  - Implement inventory display functionality

1.4.2 Sprint 2: (2 Days)

- **Goal:** Implement core transaction functions, including cashier operations, return processing, and inventory display.
- User Stories:
  - As a cashier, I want to add items to orders by entering their product IDs to complete sales quickly.
  - As a cashier, I want the system to automatically calculate change based on the amount paid by the customer to ensure transaction accuracy.
  - As a cashier, I want to process customer returns by entering the product ID and quantity to complete refunds.
  - As a cashier, I want the system to automatically update inventory after each sale or return, ensuring real-time data consistency.
- Tasks:
- Implement product order creation functionality

- Implement payment processing and change calculation
- Automatically update inventory post-sale
- Implement return processing and inventory restoration
- Integrate all features and conduct end-to-end testing

1.4.3 Sprint 3: (3 Days)

- **Goal:** Documentation writing and presentation video production.
- **User Stories:**
  - As a developer, I aim to produce agile development process reports that clearly document the methodologies and procedures employed.
  - As a developer, I aim to produce system analysis and implementation reports that detail system design and key technical specifications.
  - As a project team member, I aim to create a 5-minute demonstration video that comprehensively showcases the system's three core functionalities.
- **Tasks:**
  - Prepare an Agile Development Process Report
  - Prepare a System Analysis and Implementation Report
  - Produce a 5-minute demonstration video

Using Agile with PoS System									
Duration: 7 days									
Task ID	**Tasks**	**Estimate (Hours)**	**D1**	**D2**	**D3**	**D4**	**D5**	**D6**	**D7**
1	Set Up Project Structure	0.5	0.5	0	0	0	0	0	0
2	Set Up GitHub Remote Repository	0.5	0.5	0	0	0	0	0	0
3	Design product data and initialize inventory	0.5	0.5	0	0	0	0	0	0
4	Implement inventory display functionality	3	2	1	0	0	0	0	0
5	Implement cashier functions	3	0	3	0	0	0	0	0
6	Implement change calculation	1	0	0	1	0	0	0	0
7	Automatically update inventory after sale	2	0	0	2	0	0	0	0
8	Implement the return processing function and update inventory	3	0	0	0	3	0	0	0
9	Integrate all features and test	1	0	0	0	1	0	0	0
10	Prepare an Agile Development Process Report	2	0	0	0	0	2	0	0
11	Prepare a System Analysis and Implementation Report	4	0	0	0	0	1	2	1
12	Produce a 5-Minute Presentation Video	2	0	0	0	0	0	0	2
	Remaining Effort (Actual)	22.5	19	15	12	8	5	3	0
	Ideal Trend (Estimate)	22.5	19	17	14	11	7	4	0

Fig. 1 Task Log

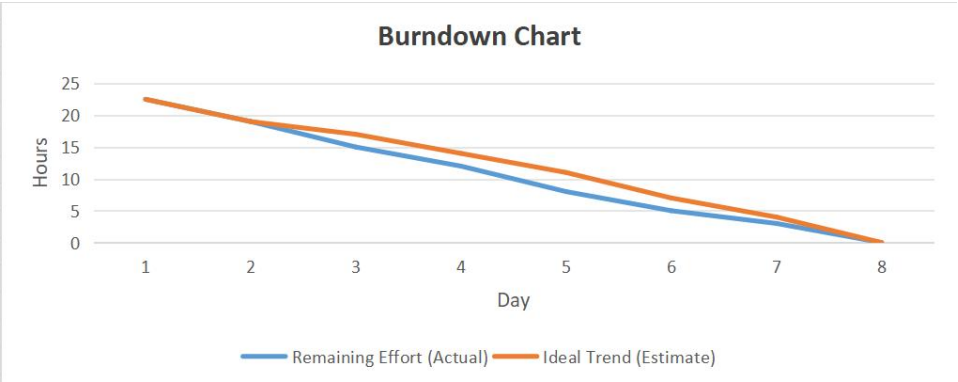

Fig. 2 Burndown Chart

1.5 Tools

- **PyCharm:** As the primary integrated development environment, used for writing, debugging, and running Python code.
- **GitHub:** Utilizes Git for version control, hosts project code repositories, supports team collaboration, branch management, and commit history tracking, ensuring traceability throughout the development process.
- Excel: Manages user stories, task assignments, and Sprint planning. Tracks progress through spreadsheets and generates burn-down charts to visualize iteration advancement.
- Word: Used for drafting project documentation, including agile development process reports and system analysis and implementation specifications, supporting structured formatting and content organization.
- Bilibili: Serves as a video publishing platform for uploading and sharing project demonstration videos, facilitating the presentation of core system functionalities and meeting course delivery requirements.

2. Part 2 – The point-of-sale (POS) system

2.1 Use Case Model

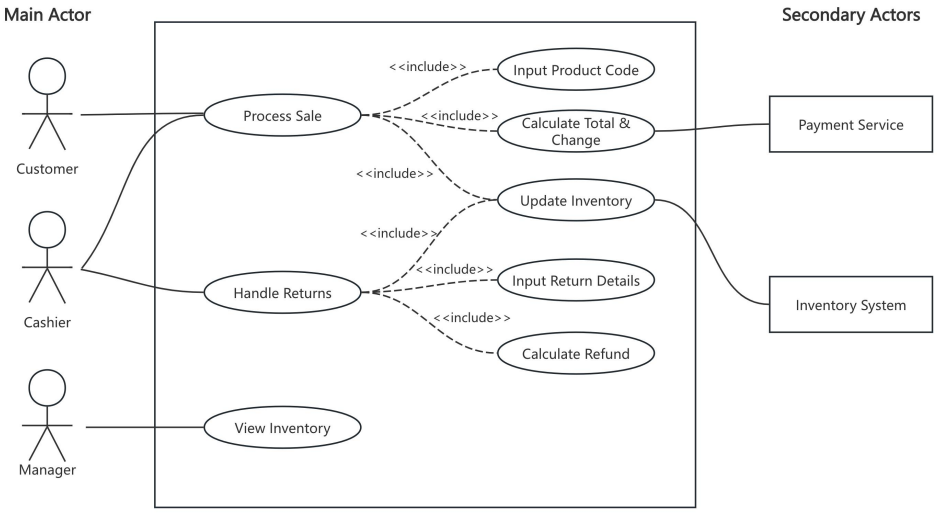

Fig. 3 Use Case Diagram

The use case model for this system defines the interaction boundaries between external participants and the system's functional requirements. The use case diagram illustrates how primary participants (Customer, Cashier, Manager) interact with the system through core functions such as ‘Process Sale’, ‘Handle Returns’, and ‘View Inventory’, whilst also involving secondary participants including the Payment Service and Inventory System. Within the specific sales process use case text, the procedure is detailed where the cashier initiates a new sale and cycles through entering items. The system utilises the SaleService to invoke Product.update_stock in real-time, thereby deducting stock. The transaction is ultimately completed through payment amount validation. Should an exception occur where the payment amount is insufficient, the system captures and signals the anomaly via the ValueError mechanism, ensuring the robustness of the transaction logic.

2.2 Domain Model / OOA

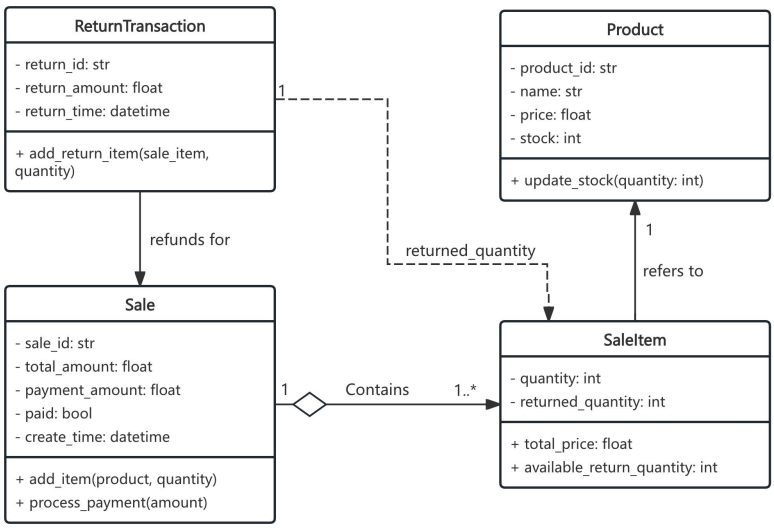

Fig. 6 UML Class Diagram - domain model

The system's domain model illustrates the static structure and intrinsic relationships of core business entities through a UML class diagram. This diagram comprises four principal classes: Product, Sale, SaleItem, and ReturnTransaction. The Sale and SaleItem classes maintain a one-to-many (1..*) composite relationship, reflecting the lifecycle management of sales items within an order. SaleItem references Product via an association (refers to), thereby acquiring product attributes during transactions and executing inventory updates. Furthermore, ReturnTransaction associates with the original Sale order (refunds for), implementing the business logic for returns processing through operations on specific SaleItems.

2.3 System Logical Architecture

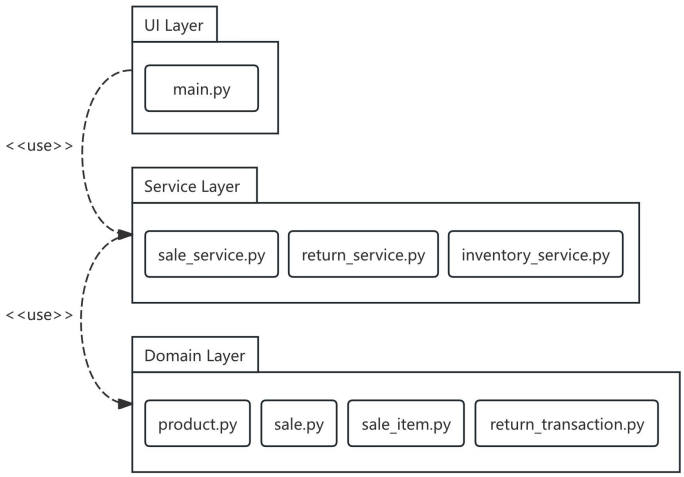

Fig. 7 Logical Architecture-UML Package Diagrams

The system’s logical architecture adopts a typical three-tier layered pattern, with UML package diagrams clearly illustrating module dependencies and separation of concerns. The topmost UI Layer contains main.py, responsible for handling all user

input and view presentation. The intermediate Service Layer integrates sales, returns, and inventory services, acting as the system's central brain to coordinate operations across domain objects. The bottom-most Domain Layer stores all core entity classes, forming the foundational data logic for the entire system. This unidirectional dependency structure—'UI → Service → Domain'—effectively achieves layer decoupling, enhancing the system's maintainability.

2.4 OO Design / Interaction Diagrams

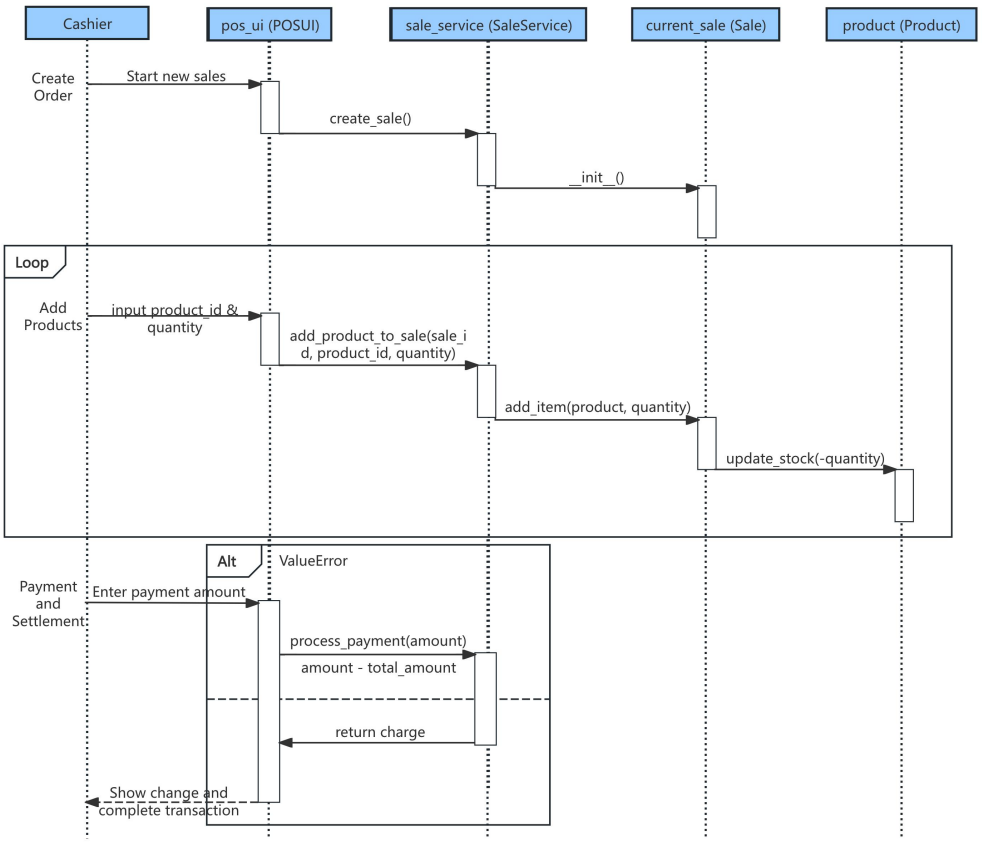

Fig. 4 System Sequence Diagram - Purchase Process

Process Sale: This use case describes the process by which a cashier handles a customer's purchase of goods and completes the settlement.

- Use Case Name: Process Sale
- Primary Participant: Cashier
- Prerequisites: The cashier has logged into the POS system, and the system has been initialised.
- Primary Success Scenario:

  a. The cashier selects 'Start New Sale' on the interface, and the system creates a new Sale instance via the SaleService.

  b. [Loop] The cashier enters the product ID and quantity.

c. The system retrieves the product information based on the ID and adds it to the order as a SaleItem.
d. The system automatically calls Product.update_stock(-quantity) to deduct the stock in real time.
e. The cashier enters the customer's payment amount.
f. The system calls process_payment to validate the amount and calculates the change using the formula amount - total_amount.
g. The system displays the change and marks the order as 'Paid', completing the transaction.

- Extension/Exception Paths:
  - 2a. Item does not exist or stock is insufficient: The system raises a ValueError, and the cashier cancels the entry.
  - 6a. [Alt Branch] Payment amount is insufficient: The system catches the exception, prompts 'Insufficient funds', and remains on the payment page awaiting re-entry.

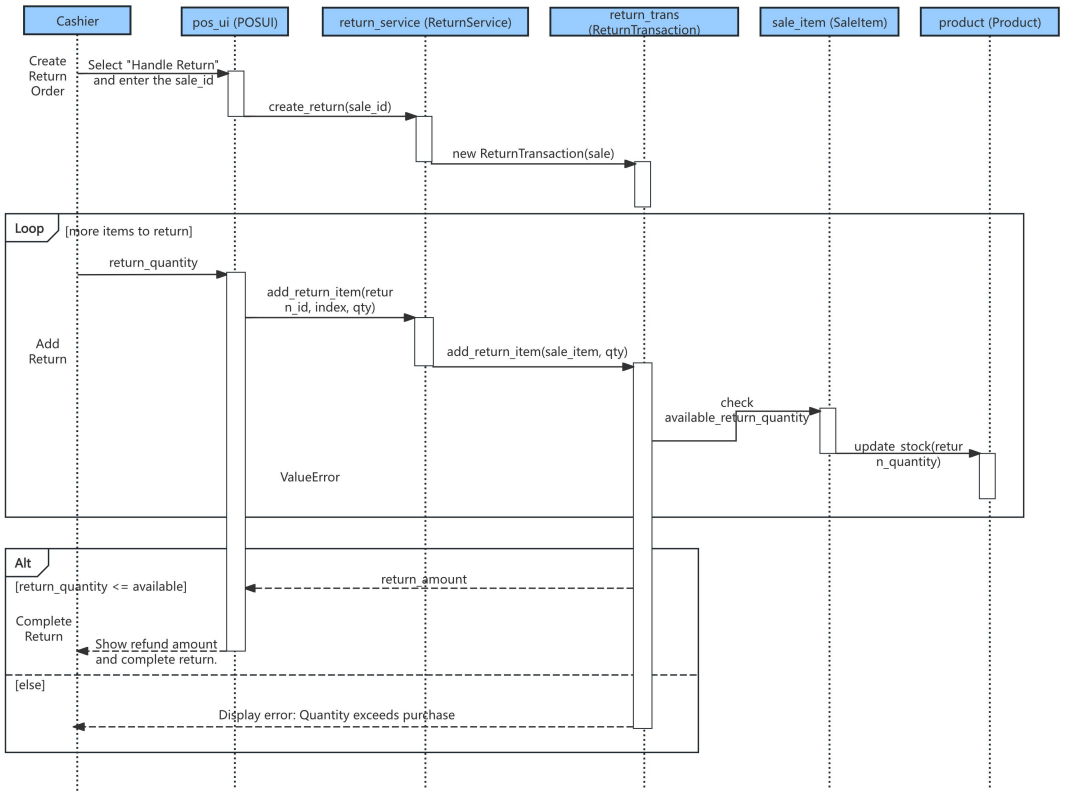

Fig. 5 System Sequence Diagram - Refund Process

Handle Returns: This use case describes the process by which a cashier handles customer returns and issues refunds.

- Use Case Name: Handle Returns
- Primary Participant: Cashier
- Preconditions: The original sales order has been paid in full.
- Primary Success Scenario:

a. The cashier enters the original order ID. The system creates a ReturnTransaction object via the ReturnService.
b. [Loop] The cashier inputs the items to be returned and their quantities.
c. The system verifies whether the return quantity exceeds the available return allowance for the original order (check available_return_quantity).
d. Upon successful verification, the system calls Product.update_stock to increase inventory for real-time replenishment.
e. The system accumulates the total refund amount (return_amount).
f. Cashier confirms completion of return; system displays total amount to be refunded to customer.

- Extension/Exception Path:
  - 3a. [Alt Branch] Invalid return quantity: If quantity exceeds original purchase quantity or is negative, system displays ‘Quantity exceeds purchase’ and rejects update.
  - 4a. Order status anomaly: If original order remains unpaid, system refuses to create return order.