|  |
| --- |
|  |

|  |
| --- |
| REVISION HISTORY |

|  |  |  |  |
| --- | --- | --- | --- |
| **VERSION** | **DATE** | **DESCRIPTION** | **AUTHOR** |
| 1.0 | 22.09.2025 | Initial Version |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |
| --- |
| CONTENTS |

[1 SW System Overview 4](#_Toc208787546)

[1.1 Purpose 4](#_Toc208787547)

[1.2 Scope 4](#_Toc208787548)

[1.3 Use-Case Diagram 4](#_Toc208787549)

[1.4 General Constraints 4](#_Toc208787550)

[1.5 Assumptions and Dependencies 4](#_Toc208787551)

[1.6 Acronyms and Abbreviations 4](#_Toc208787552)

[2 SW Functional Requirements 5](#_Toc208787553)

[2.1 Features / Functions to be Implemented 5](#_Toc208787554)

[2.1 Acceptance Criteria 5](#_Toc208787555)

[2.2 Implementation Requirements 5](#_Toc208787556)

[3 SW Non-Functional Requirements 6](#_Toc208787557)

[3.1 Resource Consumption 6](#_Toc208787558)

[3.2 License Issues 6](#_Toc208787559)

[3.3 Coding Standard 6](#_Toc208787560)

[3.4 Modular Design 6](#_Toc208787561)

[3.5 Reliability 6](#_Toc208787562)

[3.6 Portability 6](#_Toc208787563)

[3.7 General Operational Guidelines 6](#_Toc208787564)

[4 SW Design Artifacts 7](#_Toc208787565)

[4.1 CRC Cards (Class–Responsibility–Collaboration) 7](#_Toc208787566)

[4.2 Conceptual UML Diagram (entities & relationships) 7](#_Toc208787567)

# SW System Overview

*Specify the purpose and the overview of the SRS.*

## Purpose

This project simulates the operation of a small photo studio. The system manages orders placed by clients, records customer details, processes photos, tracks material consumption, and generates reports. It supports three main roles: receptionist, photographer, and administrator.

## Scope

The system will include: order management, pricing, reporting, consumables tracking.

The system will not handle direct payment processing, employee payroll, or scheduling.

Benefits: The software will increase order processing speed, will lead to less manual errors in pricing and reporting.

Key-features: Digital order tracking, automated pricing, daily report generation

## Use-Case Diagram

|  |
| --- |
|  |

## General Constraints

## 1. Implementation language: C++

## 2. Platform: Windows/Linux desktop

## 3. Data storage: file system (text or CSV files)

## 4. Standards: UML notation for diagrams

## 5. Performance: each exchange transaction must complete in ≤ 2 seconds

6. User interface: console only (CLI)

## Assumptions and Dependencies

1. The studio has a stable internet connection and a reliable power source.
2. All users (receptionist, photographer, and administrator) have access to a computer running the program.
3. The system depends on the local file system being accessible for storing transactions and reports
4. No internet connection or external APIs are required.

## Acronyms and Abbreviations

*List all acronyms and abbreviations used in the document along with their explanations.*

|  |  |
| --- | --- |
| **Terms Used** | **Description of terms** |
| **cli** | Command Line Interface |
| **API** | Application Programming Interface |
| **CSV** | Comma-separated Values |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# SW Functional Requirements

## 2.1 Features / Functions to be Implemented

User stories:

1. As a Client, I want to place an order so that I can get my photos printed or film developed.
2. As a Receptionist, I want to record client details (surname) and desired completion time when taking an order so that the order is accurately registered.
3. As a Receptionist, I want to specify if an order is "express" so that the correct surcharge is applied.
4. As a Receptionist, I want to print an order form for the client and for the photographer so that both parties have a physical record.
5. As a Receptionist, I want to view active orders so that I can monitor their status and manage client expectations.
6. As a Receptionist, I want to mark an order as completed and record payment so that the order lifecycle is correctly updated and revenue is accounted for.
7. As a Receptionist, I want to generate a daily report on completed orders and revenue so that I can submit it to management.
8. As a Photographer, I want to view my assigned orders so that I know what tasks I need to complete.
9. As a Photographer, I want to update the status of an order (e.g., in progress, ready for pickup) so that the receptionist is aware of the progress.
10. As a Photographer, I want to report the consumables (paper, developer, etc.) used for my tasks so that stock levels can be accurately tracked.
11. As a Studio Administrator, I want to manage consumables stock levels so that I can ensure the studio is always supplied.
12. As a Studio Administrator, I want to review reports on consumed materials so that I can make informed purchasing decisions.

## Acceptance Criteria

FR1: The system must allow the receptionist to register new client orders.

Validation: Acceptance test – create an order with client surname, order type, urgency, and completion time; verify it is stored correctly.

FR2: The system must calculate prices, applying a 25% surcharge for express orders.

Validation: Test case – register two identical orders (one urgent, one normal); check that urgent price = normal price × 1.25.

FR3: The system must allow the photographer to process orders.

Validation: Test case – after registering an order, pass it to the photographer; system should display “Processing order for [surname].”

FR4: The system must track consumables usage per processed order.

Validation: Test case – process at least one order of each type (photo printing, film developing); system should output consumables used.

FR5: The system must generate a daily report of completed orders and revenue.

Validation: Acceptance test – process multiple orders, run generateRevenueReport(), and verify total revenue equals sum of all order prices.

FR6: The system must generate a daily report of consumed materials.

Validation: Acceptance test – after processing orders, run generateMaterialReport(); verify materials are reported correctly.

Non-Functional Requirements

NFR1: The system should be simple and console-based.

Validation: Quality metric – program runs entirely in the console without GUI.

NFR2: The system should provide readable text output for reports.

Validation: Acceptance test – reports are generated with clear formatting (totals, client names, order types).

NFR3: The system should be modular, with classes representing roles.

Validation: Code inspection – confirm that Receptionist, Photographer, Client, Administrator are implemented separately.

NFR4: The system should allow easy extension (e.g., adding new services).

Validation: Code review – confirm that new order types can be added by extending OrderType enum without modifying core logic.

## Implementation Requirements

1. The system must be implemented in C++11.
2. Standard library (iostream, string, vector) must be used for input/output and data structures.
3. The code should follow object-oriented design principles (encapsulation, inheritance, separation of concerns).
4. Each class should implement only its assigned responsibilities from the CRC cards.
5. Methods should be simple, with clear console output demonstrating system behavior.

# SW Non-Functional Requirements

## Resource Consumption

- Response time for any operation: ≤ 1 second

- Maximum memory usage: ≤ 100 MB

- Maximum file size for daily logs: ≤ 5 MB

## License Issues

- Only standard C++ STL libraries are allowed

- No proprietary third-party libraries are permitted.

- External libraries may only be used if they have permissive open-source licenses (MIT, Apache-

2.0).

## Coding Standard

- Each function and class must include descriptive comments.

- Unit tests must cover all critical components (e.g., calculation of exchanged amount).

## Modular Design

Modularity:

* The system must be organized into separate classes with clear responsibilities
* Modules should interact through well-defined interfaces
* Business logic must be separated from data storage structures

Extensibility:

* The initial architecture should allow possibilities to extend services in future (adding digital scanning service)
* Enums (like OrderType) should support adding more order types without breaking existing code.

Maintainability:

* The code should be easy to read and follow standard naming conventions, so future developers (or students) can understand responsibilities quickly.
* By keeping classes loosely coupled (e.g., Receptionist doesn’t need to know how the photographer processes orders, only that it does), changes in one module won’t break others.

## Reliability

- The system must reject invalid input without crashing.

- File writes must be atomic to avoid corruption.

- Error messages must be logged in a text file for troubleshooting.

## Portability

- The system must compile and run on Windows 10+ and Ubuntu Linux.

- Identical inputs must produce identical outputs on both platforms.

## General Operational Guidelines

- The system must be robust, easy to maintain, and simple to use.

- Daily reset functionality must be provided to start each workday with a clean state.

# SW Design Artifacts

## CRC Cards (Class–Responsibility–Collaboration)

|  |  |
| --- | --- |
| Class: Client | |
| Responsibilities | Collaborators |
| * Provide surname and order details * Place orders for services | * Receptionist * Order |

|  |  |
| --- | --- |
| Class: Receptionist (inherits Employee) | |
| Responsibilities | Collaborators |
| * Register new client orders * Pass orders to the photographer * Generate revenue report | * Client * Order * Photographer * Report |

|  |  |
| --- | --- |
| Class: Photographer (inherits Employee) | |
| Responsibilities | Collaborators |
| * Process orders * Track consumables used * Generate material usage report | * Order * ConsumableUsage * Administrator * Receptionist |

|  |  |
| --- | --- |
| Class: Administrator | |
| Responsibilities | Collaborators |
| * Manage consumable inventory * Account for materials reported by the photographer | * Photographer * Consumable * ConsumableUsage |

|  |  |
| --- | --- |
| Class: Consumable | |
| Responsibility | Collaborators |
| * Represent a consumable material * Store available quantity | * ConsumableUsage * Administrator |

|  |  |
| --- | --- |
| Class: ConsumableUsage | |
| Responsibility | Collaborators |
| * Record how much of each consumable was used for a given service/order | * Consumable * Photographer * Order |

|  |  |
| --- | --- |
| Class: Employee (abstract / parent) | |
| Responsibility | Collaborators |
| * Represent shared attributes of employees (name, ID) * Define base responsibilities for staff roles | * Receptionist * Photograpgher * Administrator |

|  |  |
| --- | --- |
| Class: ExpressOrder (inherits Order) | |
| Responsibility | Collaborators |
| * Represent urgent orders * Apply 25% surcharge to base price | * Order * Receptionist |

|  |  |
| --- | --- |
| Class: Order | |
| Responsibility | Collaborators |
| * Represent a customer’s request * Stores client’s information * Contain order status * Calculate the final price * Contain a list of order items | * Client * Receptionist * Photographer * OrderItem |

|  |  |
| --- | --- |
| Class: OrderItem | |
| Responsibility | Collaborators |
| * Represent a single service request within an order * Store service type and cost | * Order * Service |

|  |  |
| --- | --- |
| Class: Report | |
| Responsibility | Collaborators |
| * Represent reports * Store report content and allow output/printing | * Receptionist * Photographer * Administrator |

|  |  |
| --- | --- |
| Class: Service | |
| Responsibility | Collaborators |
| * Represent a service offered by the studio * Define base price and consumables required | * OrderItem * Photographer * ConsumableUsage |

## Conceptual UML Diagram (entities & relationships)

*Draw a conceptual class diagram with key entities and their relationships; focus on nouns from User Stories/Use Cases, omit methods and low-level details.*