C22716399 Aaron Baggot

[Company name] | [Company address]

[Document title]

[Document subtitle]

Table of Contents

[Introduction 3](#_Toc163908378)

[Restaurant Selection and Use Case Scenarios 5](#_Toc163908379)

[Class Diagram with soil implementation 6](#_Toc163908380)

[Object Diagram 7](#_Toc163908381)

[Sequence Diagram 8](#_Toc163908382)

[Booking System changeTable() 9](#_Toc163908383)

[Booking System SelectBooking() 11](#_Toc163908384)

[Booking System unselectBooking() 12](#_Toc163908385)

[BookingSystem makeReservation() 13](#_Toc163908386)

[BookingSystem cancelReservation() 15](#_Toc163908387)

[Booking System recordArrival() 16](#_Toc163908388)

[Original Sequence Diagram 19](#_Toc163908389)

[Initial Diagrams 21](#_Toc163908390)

[USE Model Overview 24](#_Toc163908391)

[Booking System 24](#_Toc163908392)

[State Machines 25](#_Toc163908393)

[Booking System state machine 25](#_Toc163908394)

[selectBooking() 26](#_Toc163908395)

[unselectBooking() 26](#_Toc163908396)

[cancelBooking() 27](#_Toc163908397)

[Record arrival state machine 27](#_Toc163908398)

[Booking and Reservation Class 28](#_Toc163908399)

[Implementation 28](#_Toc163908400)

[Overall Implementation 28](#_Toc163908401)

[Customer and Table Class 29](#_Toc163908402)

[Testing and Validation 29](#_Toc163908403)

[Pre and Post Conditions 29](#_Toc163908404)

[Booking System Record Arriva 29](#_Toc163908405)

[use> !bookSys.selectBooking(2,20,0) 29](#_Toc163908406)

[Booking System Cancel Conditions 30](#_Toc163908407)

[Cancel Pre-Condition Success 30](#_Toc163908408)

[Cancel post success 31](#_Toc163908409)

[Cancel pre success 31](#_Toc163908410)

[Cancel pre fail 32](#_Toc163908411)

[Booking System Change Table 32](#_Toc163908412)

[overNoOfCovers() 32](#_Toc163908413)

[Make Reservation – Underage Condition 36](#_Toc163908414)

[Test Case 37](#_Toc163908415)

[Adaptation of Menu Ordering System 38](#_Toc163908416)

[Adaptation of Card and Cash Payment System 41](#_Toc163908417)

[Conclusion 45](#_Toc163908418)

# Introduction

In the realm of software engineering, the efficacy and elegance of a system's design held paramount importance. It was within this context that we embarked on extending and refining the existing USE model for a restaurant management system. The restaurant domain, with its intricate interplay of customer interactions, table allocations, and payment processing, provided fertile ground for exploring various software engineering principles and methodologies.

The primary objective of this endeavour was to augment the functionality of the restaurant management system while meticulously adhering to the principles of coupling and cohesion. By seamlessly integrating new use cases, such as table reservation and dynamic table allocation, we aimed to enhance the system's versatility and utility. Moreover, we endeavoured to implement a robust payment system, ensuring secure and efficient transaction processing.

At the core of our approach lay the concept of design by contract, wherein every component of the system was imbued with preconditions, postconditions, and invariants. These contractual specifications not only served as a blueprint for the system's behaviour but also facilitated rigorous testing and validation. By employing OCL (Object Constraint Language) contracts, we articulated the precise conditions under which operations could be invoked, thereby ensuring the system's reliability and robustness.

In our pursuit of excellence, we employed a multifaceted methodology encompassing class diagrams, sequence diagrams, state machines, and object diagrams. Each of these artifacts offered unique insights into the system's structure, behaviour, and interactions, enabling comprehensive analysis and refinement. Furthermore, the integration of testing mechanisms, including !openter and !opexit procedures, enabled meticulous validation of system constraints and operation behaviour.

Throughout this report, we delved into the intricacies of the extended USE model, elucidating the rationale behind each design decision and presenting a comprehensive analysis of the system's functionality. By adhering to industry best practices and leveraging advanced software engineering techniques, we endeavoured to deliver robust, scalable, and user-centric restaurant management systems that exemplified the pinnacle of software engineering excellence.

# Restaurant Selection and Use Case Scenarios

In the realm of software engineering, creating systems that are both efficient and robust is essential. This report outlines the process of enhancing and refining the current USE model for a restaurant management system. The focus is on key functionalities such as booking reservations, handling walk-ins, and managing cancellations, all while integrating state machines and pre/post conditions. Through the exploration of various software engineering principles and methodologies, the aim is to improve the system's functionality and user experience.

As part of this project, additional use case scenarios have been incorporated to broaden the system's capabilities. These include features such as table reservation, dynamic table allocation, and payment processing. These scenarios offer a comprehensive view of how the system interacts with users and enhances its overall usability.

Make Reservation

This scenario involves customers reserving tables for dining at the restaurant. Customers provide personal details, select the date, time, and number of guests, and either choose a preferred table or allow the system to assign one. Upon confirmation, a reservation is created, and the customer receives a confirmation message.

Record Walk-In

In this scenario, customers arrive at the restaurant without prior reservations. The waitstaff checks table availability, assigns an available table to the customer, and seats them. This walk-in is then recorded in the system for tracking purposes.

Cancel Reservation

This scenario allows customers to cancel their existing reservations. After logging into the system, customers navigate to the reservation management section, select the reservation to cancel, and confirm the cancellation. Once cancelled, the reserved table becomes available for other customers.

Use Case Diagram

These scenarios are depicted in a use case diagram, illustrating how customers interact with the system to make reservations, record walk-ins, and cancel reservations. This diagram aligns with the functionalities implemented in the system, ensuring effective user interaction and management of restaurant bookings.

## Class Diagram with soil implementation

***-- Script generated by USE 6.0.0***

***!new BookingSystem('bookSys')***

***!new WalkIn('w1')***

***!new WalkIn('w2')***

***!new Reservation('r1')***

***!new Reservation('r2')***

***!new Reservation('r3')***

***!w1.covers := 3***

***!w2.covers := 4***

***!r1.covers := 4***

***!r2.covers := 2***

***!r3.covers := 2***

***!insert(bookSys, w1) into Contains***

***!insert(bookSys, w2) into Contains***

***!insert(bookSys, r1) into Contains***

***!insert(bookSys, r2) into Contains***

***!insert(bookSys, r3) into Contains***

***!new Customer('c1')***

***!new Customer('c2')***

***!new Customer('c3')***

***!c1.name := 'Aniket Bedade'***

***!c1.phoneNumber := 'C22448826'***

***!c1.age := 17***

***!c2.name := 'Aaron Baggot!'***

***!c2.phoneNumber := 'C22716399'***

***!c2.age := 48***

***!c3.name := 'John Smith'***

***!c3.phoneNumber := '086 123 1234'***

***!c3.age := 25***

***!r1.setCustomer(c1)***

***!r2.setCustomer(c2)***

***!r3.setCustomer(c3)***

***!new Table('tb1')***

***!new Table('tb2')***

***!new Table('tb3')***

***!new Table('tb4')***

***!tb1.tno := 1***

***!tb2.tno := 2***

***!tb3.tno := 3***

***!tb4.tno := 4***

***!tb1.covers := 4***

***!tb2.covers := 2***

***!tb3.covers := 2***

***!tb4.covers := 6***

***!insert (w1,tb1) into IsAt***

***!insert (r1,tb3) into IsAt***

***!insert (r2,tb2) into IsAt***

***!insert (w2,tb2) into IsAt***

***!new Time('t1')***

***!new Time('t2')***

***!new Time('t3')***

***!new Time('t4')***

***!new Time('t5')***

***!t2.min := 0***

***!t2.hour := 18***

***!t1.min := 30***

***!t1.hour := 20***

***!t3.min := 30***

***!t3.hour := 18***

***!t4.min := 0***

***!t4.hour := 20***

***!t5.min := 0***

***!t5.hour := 21***

***!w1.setTime(t1)***

***!w2.setTime(t2)***

***!r1.setTime(t3)***

***!r2.setTime(t4)***

***!r3.setTime(t5)***

***!insert(bookSys, c1) into Uses***

***!insert(bookSys, c2) into Uses***

***!insert(bookSys, c3) into Uses***

A screenshot of a computer

Description automatically generated

The provided SOIL code essentially creates and links different objects to form a booking system for a restaurant. Creating various objects representing different aspects of the booking system, such as the BookingSystem itself, different types of bookings like WalkIn and Reservation, as well as entities like Customer, Table, and Time. These objects are like instances of classes and hold specific information related to bookings, customers, tables, and time.

Once these objects are created, the code sets specific attributes for each of them. For example, attributes like covers (number of people), time, name, age, phone number, and table number are assigned values to represent the characteristics of each instance.

After creating and configuring these objects, the code establishes connections or relationships between them. For instance, it links bookings with tables using the IsAt association, connects customers with reservations through the Makes association, records walk-ins with customers via the Arrive association, and links the booking system with its contained bookings using the Contains association.

The provided code constructs a functional booking system for a restaurant. It allows for the creation and management of various bookings, both reservations and walk-ins, enables customers to interact with the system, assigns tables to bookings, and organizes the overall booking process. This system is structured with different components like classes, operations, and constraints, ensuring a robust and comprehensive solution for managing restaurant bookings.

## Object Diagram

An object diagram in USE represents a snapshot of the system at a specific moment, showing instances of classes and their relationships. It illustrates how objects interact and collaborate within the system, providing a visual depiction of runtime behaviour. Object diagrams help in understanding the runtime structure of the system and can be used to verify system design and implementation.

A screenshot of a computer

Description automatically generated

# Sequence Diagram

A sequence diagram in USE illustrates the interactions between different objects or components of a system over time. It shows the flow of messages between objects, indicating the order in which interactions occur. Sequence diagrams help visualize the dynamic behavior of a system, depicting how objects collaborate to achieve specific functionalities or scenarios.

A screenshot of a computer screen

Description automatically generated

# BookingSystem

Represents the main control class responsible for managing bookings, including selecting, unselecting, recording arrivals, cancelling reservations, and changing tables.

Booking:

Represents a generic booking with attributes such as the number of covers and time.

WalkIn:

Represents a walk-in booking, which inherits from Booking and includes additional operations specific to walk-ins.

Reservation:

Represents a reservation booking, which also inherits from Booking and includes operations specific to reservations.

Customer:

Represents a customer with attributes such as name, age, and phone number, and operations for making reservations and walk-ins.

Table:

Represents a restaurant table with attributes such as table number and covers, and a query operation to check availability.

## Booking System SelectBooking()

selectBooking: Selects a booking based on the table number and time provided.

A screenshot of a computer

Description automatically generated

## Booking System unSelectBooking()

unSelectBooking: Unselects the currently selected booking.

A screenshot of a computer

Description automatically generated

## Booking System changeTable()

changeTable: Changes the table for a given reservation if the new table is available.

A screenshot of a computer

Description automatically generated

A screenshot of a computer screen

Description automatically generated

Change table condition number of covers

A screen shot of a computer

Description automatically generated

## BookingSystem makeReservation()

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

No overlap when making reservation

A screenshot of a computer

Description automatically generated

## BookingSystem cancelReservation()

Sequence Diagram cancel reservation

**A screenshot of a computer screen

Description automatically generated**

Object Diagram cancel reservation

A screenshot of a computer

Description automatically generated

## Booking System recordArrival()

**A screenshot of a computer

Description automatically generated**

newReservation: Initial state when a new reservation is created.

waiting: Indicates that a reservation exists but the customers are yet to be seated.

seated: Represents a reservation where the customers have been seated.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Arrival before the booking

A computer screen shot of a black screen

Description automatically generated

## Original Sequence Diagram

A screenshot of a diagram

Description automatically generated

unSelected

A screenshot of a computer

Description automatically generated



Selected

A screenshot of a computer

Description automatically generated



A screenshot of a computer

Description automatically generated

# Initial Diagrams

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

# USE Model Overview

Overview of the Unified Software Engineering (USE) modelling approach.

Explanation of how USE models will be utilized to design and test the adapted systems.

The extended USE model encompasses class diagrams, sequence diagrams, state machines, and object diagrams. Each artifact offers insights into the system's structure, behaviour, and interactions. We have meticulously defined preconditions, postconditions, and invariants for each component, ensuring a clear understanding of the system's behaviour.

# Booking System

The Booking System class manages booking operations such as selecting, unselecting, recording arrivals, and cancelling bookings. The state machine diagram illustrates the transitions between different states, providing a visual representation of the booking process.

The BookingSystem class serves as the main control unit, orchestrating various operations such as selecting bookings, recording arrivals, canceling reservations, and managing the state transitions of bookings.  
SelectBooking()

Booking Management system allows customers to make reservations or walk-ins. Reservations are associated with specific customers and tables, with functionalities to set covers, record arrival times, and manage reservations.

Table Management are represented as objects with attributes such as table number and covers. The system ensures that tables are appropriately managed, transitioning between states of availability, reservation, and occupation.

Time Management class facilitates time-related operations, allowing the system to track booking times, record arrival times, and enforce constraints related to timing.

Customer Management allows customers can make reservations or walk-ins, with functionalities to specify covers, booking times, and contact information.

The Booking System class manages booking operations such as selecting, unselecting, recording arrivals, and cancelling bookings. The state machine diagram illustrates the transitions between different states, providing a visual representation of the booking process.

## State Machines

State machines play a crucial role in depicting the behavior of various components within the system. They provide a visual representation of the states that objects can occupy and the transitions between these states. In our restaurant management system, state machines are used to model the lifecycle of bookings, tables, and other entities. By delineating the possible states and transitions, we gain a deeper understanding of how these entities behave in response to different operations and events.

### Booking System state machine

newBooking: This is the initial state when a new booking is created.

notSelected: Indicates that a booking exists but has not been selected.

selected: Represents a booking that has been selected.

newBooking -> notSelected { create }

selected -> selected { [self.selected.table.covers >= self.selected.covers] recordArrival() }

selected -> notSelected { unSelectBooking() }

### selectBooking()

selected -> selected { selectBooking() }

A screenshot of a computer

Description automatically generated

### unselectBooking()

notSelected -> selected { selectBooking() }

A screenshot of a computer

Description automatically generated

### cancelBooking()

selected -> notSelected { cancelReservation() }

notSelected -> notSelected { cancel() }

A screenshot of a computer

Description automatically generated

### Record arrival state machine

A screenshot of a computer

Description automatically generated

# Booking and Reservation Class

The Booking and Reservation classes represent different types of bookings, each with specific attributes and operations. State machines depict the transitions between booking states, such as new booking, waiting, and seated, providing a clear overview of the booking lifecycle.

create: Transition from the newReservation state to the waiting state when a new reservation is created.

setArrivalTime(): Transition from the waiting state to the seated state when the arrival time for the reservation is set.

### Implementation

The Reservation class contains attributes to represent each state.

Operations such as create and setArrivalTime() trigger transitions between states.

When a new reservation is created, the system transitions from the newReservation state to the waiting state.

Setting the arrival time for the reservation transitions it to the seated state, indicating that the customers have been seated.

### Overall Implementation

State machines are implemented using a combination of attributes, operations, and conditional statements within the respective classes.

Each state transition is carefully defined to ensure the correct behaviour of the system.

By modelling the booking and reservation processes using state machines, the software provides a clear and structured representation of the system's behaviour over time, aiding in understanding and debugging.

# Customer and Table Class

The Customer class stores customer information, while the Table class represents restaurant tables with attributes such as table number and capacity. These classes are essential for managing bookings and seating arrangements effectively.

# Testing and Validation

## Pre and Post Conditions

### Booking System Record Arriva

### use> !bookSys.selectBooking(2,20,0)

use> !bookSys.recordArrival(17,00)

[Error] 1 precondition in operation call `BookingSystem::recordArrival(self:bookSys, hr:17, mn:0)' does not hold:

A screenshot of a computer

Description automatically generated

### Booking System Cancel Conditions

context BookingSystem::cancel(r : Reservation)

pre Pre1: current->includes(r)

post Post1: current->excludes(r)

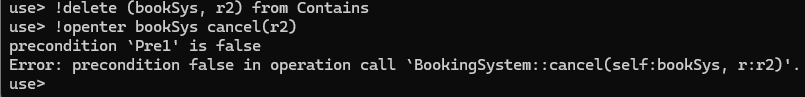
A screenshot of a computer

Description automatically generated

### Cancel Pre-Condition Success

A computer screen with white text

Description automatically generated



### Cancel post success

A black background with white text

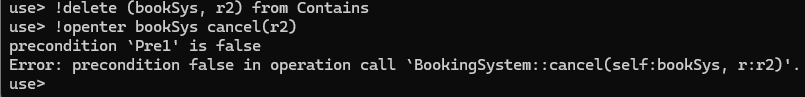
Description automatically generated

### Cancel pre success

A computer screen with white text

Description automatically generated

### Cancel pre fail



### Booking System Change Table

### overNoOfCovers()

pre-condition

A screen shot of a computer

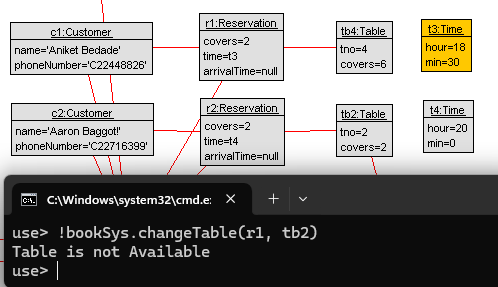
Description automatically generated

Object diagram before implementation of conditions

A screenshot of a computer

Description automatically generated

Post condition Fail – Table is not available



A screenshot of a computer

Description automatically generated

*Error: postcondition false in operation call `BookingSystem::changeTable(self:bookSys, r:r2, table:tb3)'.*

Table is not Available

[Error] 1 postcondition in operation call `BookingSystem::changeTable(self:bookSys, r:r2, table:tb3)' does not hold:

Post1: r.table->includes(table)

r : Reservation = r2

r.table : Table = tb4

r.table : Set(Table) = Set{tb4}

table : Table = tb3

r.table->includes(table) : Boolean = false

call stack at the time of evaluation:

1. BookingSystem::changeTable(self:bookSys, r:r2, table:tb3) [caller: bookSys.changeTable(r2, tb3)@<input>:1:0]

Post condition Success

A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

### Make Reservation – Underage Condition

A computer screen shot of a black screen

Description automatically generated

### Test Case

If number of people that book for the table exceeds the table cover amount. It fails

**Screenshot Record arrival Fail**

# Adaptation of Menu Ordering System

Description of the proposed changes to the menu ordering system.

Use case scenarios for menu customization, table change requests, and other relevant functionalities.

Preconditions, postconditions, and invariants for the adapted menu ordering system.

Class diagrams depicting the relationships between different components of the system.

Sequence diagrams illustrating the flow of interactions during menu ordering processes.

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

# Adaptation of Card and Cash Payment System

Explanation of the modifications to the payment system to support both card and cash transactions.

Use case scenarios for payment processing, including contactless payments and cash handling.

Preconditions, postconditions, and invariants for the adapted payment system.

Object diagrams showcasing the various objects involved in payment transactions.

Sequence diagrams demonstrating the sequence of interactions during payment processing.

The Payment System class handles payment operations, including cash and card payments. By associating payment objects with bookings, we ensure seamless payment processing. The select Payment operation validates payment details and processes payments accordingly.

A computer screen shot of a black screen

Description automatically generated

A screenshot of a computer code

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screen shot of a computer

Description automatically generated

# Conclusion

Overall, the project to extend and refine the restaurant management software was a rewarding and enriching experience. It provided a platform to apply theoretical knowledge to real-world problems, collaborate with peers, hone technical and interpersonal skills, and foster personal and professional growth. By embracing challenges, learning from experiences, and reflecting on lessons learned, I have developed a deeper appreciation for the complexities and possibilities inherent in the field of software engineering.

Software Project Notes

To improve the design of your system, particularly regarding payment and menu handling, you can make several adjustments for better modularity, efficiency, and ease of maintenance. Here are some suggestions:

Payment System:

Decouple Payment from Customer:

Instead of directly connecting the Cash attribute to the Customer, create a more generic association between Payment and Booking. This allows for more flexibility in handling payments.

Consider having separate associations for Cash and Card payments with bookings, allowing customers to choose their preferred payment method.

Refactor Payment Handling:

Implement a more flexible payment system that can handle various payment methods (Cash, Card, etc.) and can process payments for different types of bookings (WalkIn, Reservation, etc.).

Consider creating a separate class for payments (Payment), which can then be associated with bookings. This decouples the payment process from specific booking types and allows for easier extension in the future.

Error Handling and Logging:

Implement error handling mechanisms within the payment system to handle invalid payments, insufficient funds, etc.

Introduce logging mechanisms to keep track of payment transactions for auditing and debugging purposes.

Menu Handling:

Menu Representation:

Create a class to represent the menu items (MenuItem), with attributes such as name, description, price, etc.

Consider using associations to represent the relationship between menu items and bookings. For example, a booking may have associated menu items that the customer has ordered.

Menu Management:

Implement operations for adding, removing, and updating menu items dynamically. This allows for easy modification of the menu without directly modifying the code.

Consider implementing menu categories or sections to organize menu items more effectively.

Integration with Orders:

Integrate the menu system with the ordering process (Order class). When a customer places an order, the system should retrieve the selected menu items and add them to the order.

Localization and Internationalization:

If your restaurant operates in a multilingual environment, consider adding support for multiple languages in the menu system.

By implementing these improvements, you can create a more robust and maintainable system for handling payments and managing the restaurant menu. Additionally, by decoupling components and following object-oriented principles, you ensure that your system remains flexible and extensible for future enhancements.

**Software Engineering Assignment 2 – Designing a Restaurant Ordering System: A UML Modeling Challenge**

For this assignment expanding the restaurant booking system used in the lab test was selected.

The requirements include

* class diagrams
* sequence diagrams
* state machines charts
* object diagrams

SelectBooking()

Record booking() are already included

*Reservation Status*

* Pending -> Confirmed: When the reservation is confirmed by the restaurant.
* Pending -> Cancelled: When the reservation is cancelled by either the customer or the restaurant.
* Confirmed -> Cancelled: When the reservation is cancelled after being confirmed.

*Table availability*

* Available -> Reserved: When a table is reserved by a customer.
* Reserved -> Available: When a reservation for a table is cancelled.
* Available -> Occupied: When a customer is seated at a table.
* Occupied -> Available: When the customer finishes dining and leaves the table.

Booking Time window

Open

Closed

* Open -> Closed: When the booking time window for a particular day closes.

*Reservation Payment State machine*

Unpaid

Paid

Unpaid -> Paid: When the reservation payment is successfully processed.

Customer Arrival State Machine

Not Arrived

Arrived

* Not Arrived -> Arrived: When the customer arrives at the restaurant for their reservation.

Reservation Modification State Machine:

States: Unmodified, Modified

Transitions:

Unmodified -> Modified: When the reservation details are modified (e.g., change in party size, time, or special requests).

Waitlist Status State Machine:

States: Not Waitlisted, Waitlisted

Transitions:

Not Waitlisted -> Waitlisted: When the customer opts to be added to the waitlist due to unavailability of tables.

Restaurant Closing Status State Machine:

States: Open, Closing, Closed

Transitions:

Open -> Closing: When the restaurant is preparing to close for the day.

Closing -> Closed: When the restaurant completes closing procedures.

Reservation Feedback State Machine:

States: Feedback Pending, Feedback Submitted

Transitions:

Feedback Pending -> Feedback Submitted: When the customer submits feedback for their dining experience.

* **Add/Modify tables:** To add, remove, or modify a table in the system.
* **Search tables:** To search for available tables for reservation.
* **Place order:** Add a new order in the system for a table.
* **Update order:** Modify an already placed order, which can include adding/modifying meals or meal items.
* **Create a reservation:** To create a table reservation for a certain date/time for an available table.
* **Cancel reservation:** To cancel an existing reservation.
* **Check-in:** To let the guest check in for their reservation.
* **Make payment:** Pay the check for the food.