Guru Gobind Singh Indraprastha University

University School of Information and Communication Technology

Software Engineering Lab

IT-665

Submitted by:

Adarsh Raj

Enrollment No: 02616404523

MCA (SE) 1st Year

Submitted to:

Ms. Neha Roy

INDEX

S.No.	Name of Program	Date	Remarks
1	Write a case study on any five Software Crisis.		
2	Pick up a case study of your choice which you want to automate and write its problem statement. draw use case diagram and Data flow diagram of the study selected in practical no 1 sample project.		
3	Write a complete SRS document of the sample project as per IEEE std.		
4	Draw following UML diagram for any sample project with description: Entity Relationship Diagram Class Diagram Object Diagram Activity Diagram State Chart Diagram Sequence Diagram Collaboration Diagram Deployment Diagram		
5	WAP to finds maximum of three numbers and calculate LOC of a given program.		
6	WAP to implement the function point method of software estimation.		
7	WAP to implement COCOMO-1 model of software size estimation.		
8	WAP to calculate Cyclomatic Complexity.		
9	Draw Gantt Chart and Pert chart for any sample project with description.		
10	Perform Boundary value analysis for the given program.		
11	Perform Robust Testing and Worst case testing for Triangle and Month problem.		
12	Perform Robust Worst case testing for Triangle and Month problem.		
13	Perform Equivalence class testing on Triangle Problem, Month Problem and Student Division Problem		
14	Perform Limited Entry Decision Table testing on Triangle and Student Division Problem		

1. Write a case study on any five Software Crisis.

Introduction: In the ever-evolving landscape of technology and software development, the adage "To err is human" applies perhaps more poignantly than ever before. Despite the tremendous advancements in software engineering, unexpected and, at times, catastrophic software crises have continued to surface, challenging the very foundations of reliability and trust in digital systems. While some software crises have achieved notoriety on a global scale, others have remained relatively obscure but are equally illuminating in their lessons.

This case study delves into the annals of software engineering to uncover five unique instances where software-related crises disrupted industries, jeopardized lives, and exposed vulnerabilities that often lay hidden beneath lines of code. These case studies serve as cautionary tales, shedding light on the multifaceted nature of software crises, the complexities of their origins, and the profound consequences they can yield.

Each of these case studies offers valuable insights into the world of software development and its inherent challenges. From the dramatic radiation therapy mishaps caused by the Therac-25 to the algorithmic turmoil that brought financial giant Knight Capital Group to its knees, these crises reveal the critical importance of software quality assurance, testing, and rigorous development practices.

As we explore these unique software crises, we invite you to consider the profound impact that software has on our daily lives and the imperative of continual improvement in the field of software engineering. With these lessons in mind, we embark on a journey to dissect these extraordinary cases, hoping to extract wisdom that can guide us towards a future where software crises become increasingly rare and less devastating.

1. Therac-25 Radiation Therapy Machine (1985-1987):

Background: The Therac-25 was a radiation therapy machine used for cancer treatment.

Crisis Description: Several patients received lethal radiation overdoses due to software and hardware malfunctions. The software controlling the

machine had race conditions, leading to incorrect dosages being administered.

Impact: Multiple lawsuits were filed against the manufacturer, AECL, and the incidents raised concerns about the safety of medical devices and the need for rigorous software testing in healthcare equipment.

2. Knight Capital Group Trading Glitch (2012):

Background: Knight Capital Group was a financial services firm involved in high-frequency trading.

Crisis Description: Due to a software deployment error, Knight Capital's trading algorithm went haywire, causing it to execute thousands of erroneous trades in just a few minutes. This resulted in a $440 million loss for the company.

Impact: Knight Capital was forced to seek financial assistance, and its reputation was severely damaged. The incident highlighted the risks associated with high-frequency trading and the need for robust software deployment procedures.

3. The Denver International Airport Baggage System (1995):

Background: Denver International Airport (DIA) built an automated baggage handling system, intending to improve efficiency.

Crisis Description: The software controlling the baggage system was plagued by software bugs and integration issues. The system frequently misrouted luggage, causing delays and baggage handling problems.

Impact: DIA incurred massive cost overruns and was forced to delay the airport's opening by over a year. The baggage system became a symbol of failed project management and software engineering.

4. The Norwegian National Lottery's Random Number Generator (2003):

Background: The Norwegian National Lottery used a software-based random number generator for selecting winning lottery numbers.

Crisis Description: It was discovered that the random number generator wasn't truly random and exhibited patterns. Players exploited this flaw to predict winning numbers and won significant prizes.

Impact: The lottery operator had to suspend the game temporarily and faced public backlash. This incident highlighted the importance of rigorously testing random number generators in gaming and gambling systems.

5. The Mars Climate Orbiter (1999):

Background: NASA's Mars Climate Orbiter was a spacecraft designed to study the Martian climate.

Crisis Description: The spacecraft crashed into Mars due to a software error. Lockheed Martin, the spacecraft's manufacturer, used English units for thruster data, while NASA's software assumed metric units.

Impact: The loss of the Mars Climate Orbiter was not only a financial setback but also a significant setback to NASA's Mars exploration program. It emphasized the importance of consistent units in software and communication.

These unique case studies demonstrate that software-related crises can occur in various domains, from healthcare to finance to aerospace, underscoring the critical need for robust software engineering practices and thorough testing.

2. Pick up a case study of your choice which you want to automate and write its problem statement. Draw use case diagram and Data flow diagram of the study selected in practical no 1 sample project.

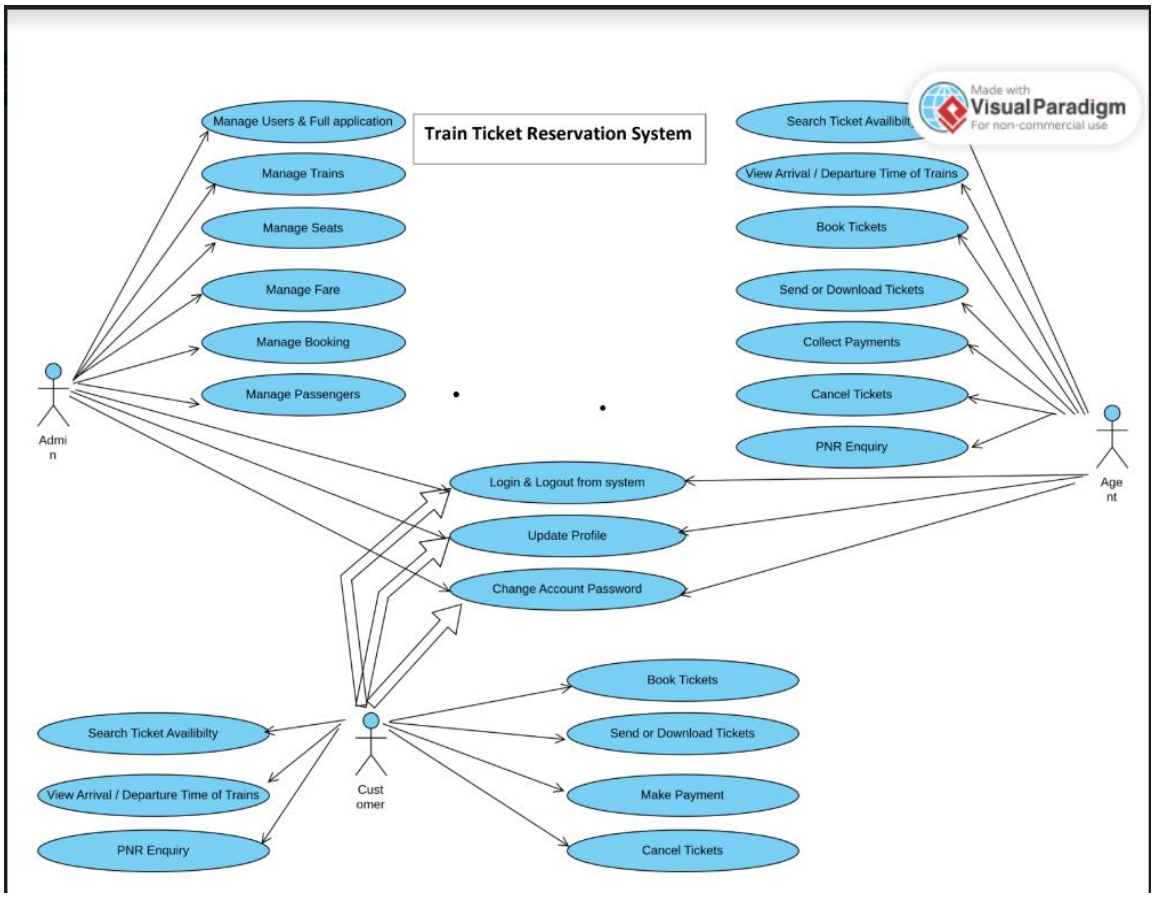
Problem Statement: Railway Train Ticket Reservation Development

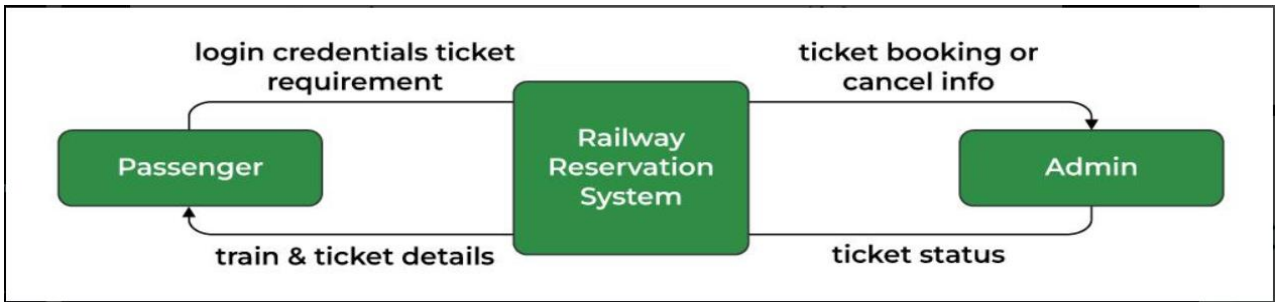
The online reservation system is a very good feature, because standing in Queues all the time and jostling away wasting your time is not the best thing to do, so the IRCTC website does help a great deal to book your tickets.

Tatkal: This facility is a great boon, if you consider the cases of emergency, but a lot of people do use it in a wrongful manner and the entry of agents into the reservation system is nothing less then a cancer for the whole system. The Site goes down at exactly 8:01 a.m and you have about 0.1% chance of getting a ticket, that too if lucky.

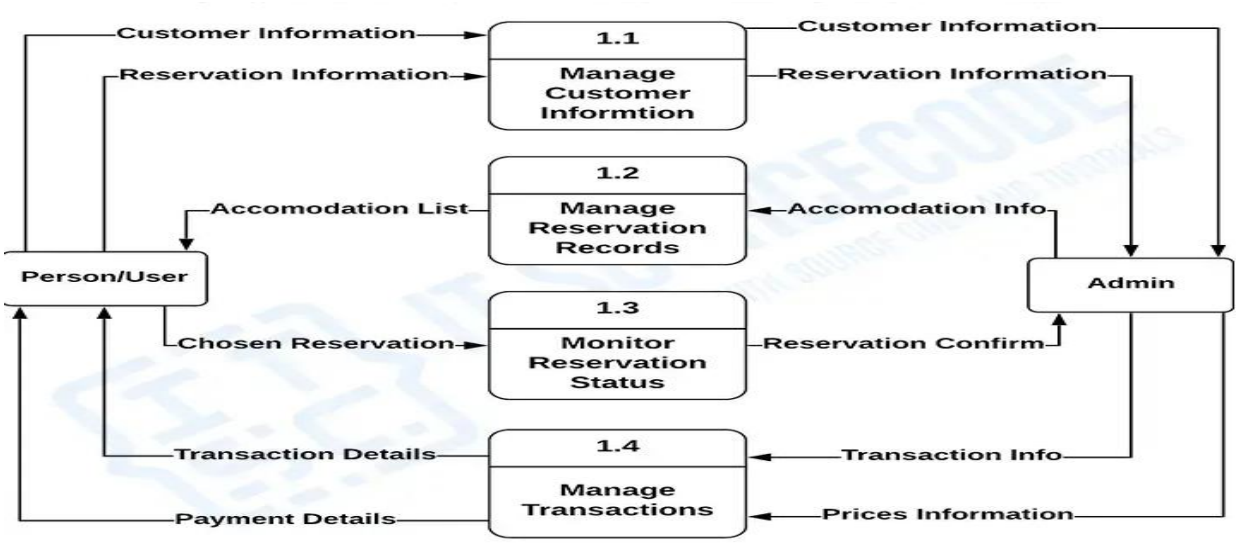
PNR status: The status enquiry feature is a lot better than before, when you had to dial your PNR number over and over again to no avail. The website does give your status fairly easily.

USE CASE DIAGRAM

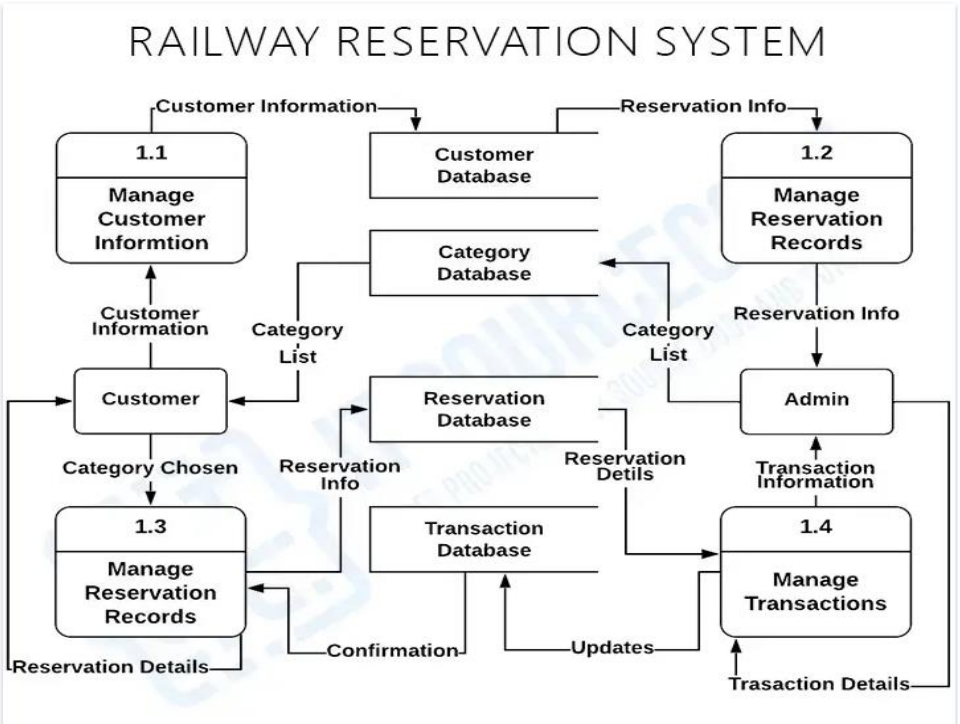




DFD LEVEL 0



DFD LEVEL 1



DFD LEVEL 2

3. Write a complete SRS document of the sample project as per IEEE std.

Software Requirements Specification (SRS) Document

1. Introduction

A Software has to be developed for automating the manual Railway Reservation System.

- RESERVE SEATS – Reservation form has to be filled by passenger. If seats are available entries like train name, number, destination are made.
- CANCEL RESERVATION- The clerk deletes the entry in the System and changes in the Reservation Status.
- VIEW RESERVATION STATUS-The user need to enter the PIN number printed on ticket.

1.1 Purpose

The purpose of this source is to describe the railway reservation system which provides the train timing details, reservation, billing and cancellation on various types of reservation namely,

- Confirm Reservation for confirm Seat.
- Reservation against Cancellation.
- Waiting list Reservation.
- Online Reservation.
- Tatkal Reservation

1.2 Scope

The basic scope of the project is given as under. “Railways Reservation System” is an attempt to simulate the basic concepts of an online Reservation system. The system enables to perform the following functions:

- SEARCH FOR TRAIN
- BOOKING OF A SELECTED FLIGHT
- PAYMENT
- CANCELLATION
- Freight Revenue enhancement
- Passenger Revenue enhancement
- Improved & optimized service

1.3 Definitions, Acronyms, and Abbreviations

- HTML (Hyper Text Markup Language): It is used to create Static web pages.
- JSP (Java Server pages): It is used to create dynamic web content.

- J2EE (Java 2 Enterprise Edition): It is a programming platform, belonging to the Java platform. which is used for developing and running distributed java applications.
- WASCE (WebSphere Application Server Community Edition): It is an application server that runs supports the J2EE and the web service appliction.
- WSAO (WebSphere Studio Application Developer It is a designer toolkit which designed to develop more complex projects by providing a complete dynamic web service.
- DB2 (IBM Database 2): It is a database management System that provides a flexible and efficient database platform to raise a Strong on demand" business applications.
- HTTP (Hyper Text Transfer Protocol): It is a transaction oriented client/ server protocol between a web browser and a web server.

1.4 References

The websites are referred https://www.w3schools.com

2. Overall Description

2.1 Product Perspective

A Software has to be developed for automating the manual Railway Reservation System.
• RESERVE SEATS – Reservation form has to be filled by passenger. If seats are available entries like train name, number, destination are made.
• CANCEL RESERVATION- The clerk deletes the entry in the System and changes in the Reservation Status.
• VIEW RESERVATION STATUS-The user need to enter the PIN number printed on ticket.

The main users are users**: Admin, Passengers who are enquirying.**

2.2 Software Interface

- Front End Client: Html • Web Server: WASCE
- Data Base Server: DB2
- Back End: Java

2.3 Hardware Interface

- Client Side: pc (Monitor)
- Server Side: PC

2.4 Product Function

Search: This function allows the booking agent to search for train that are available between the two travel cities, namely the "Departure city" and "Arrival city" as desired by the traveller. The system initially prompts the agent for the departure and arrival city, the date of departure, preferred time slot and the number of passengers. It then displays a list of train available with

different airlines between the designated cities on the specified date and time.

Selection: This function allows a particular train to be selected from the displayed list. All the details of the train are shown :-

1. train Number
2. Date, time and place of departure
3. Date, time and place of arrival
4. TRAIN Duration
5. Fare per head
6. Number of stoppages – 0, 1, 2…

Review: If the seats are available, then the software prompts for the booking of train. The train information is shown. The total fare including taxes is shown and flight details are reviewed. ¨ Traveller Information: It asks for the details of all the passengers supposed to travel including name, address, telephone number and e-mail id.

Payment: It asks the agent to enter the various credit card details of the person making the reservation.

1. Credit card type
2. Credit card number
3. CVC number of the card
4. Expiration date of the card
5. The name on the card

Cancellation : The system also allows the passenger to cancel an existing reservation. This function registers the information regarding a passenger who has requested for a cancellation of his/her ticket. It includes entries pertaining to the train No., Confirmation No., Name, Date of Journey, Fare deducted.

2.5 User Characteristics

The users of the system are members and the admin who maintain the system. The members are assumed to have basic knowledge of the computers and Internet browsing.

2.6 Constraints

The users access the Online Reservation System from any computer that has Internet browsing capabilities and an Internet connection.

3. Specific Requirements:

3.1 Supplementary Requirements

The user must be agreed with an the term and conditions that have provided by the System Administrator. local authority and Should Obey to the all Government standards andProtocols. Licensing Require

- The usage is restricted to SAC. Adarsh Raj is developing the Online Reservation Systemand Signs the maintenance contract.
- Legal, Copyright. and Other Notices
- Online Reservation System is a trademark and cannot be used without consent.
- Applicable Standards.
- The ISO/ IEC 6592 guidelines for the documentation Of computer based Systems Willbe followed.

4. Draw following UML diagram for any sample project with description:

a) Entity Relationship Diagram

b) Class Diagram

c) Object Diagram

d) Activity Diagram

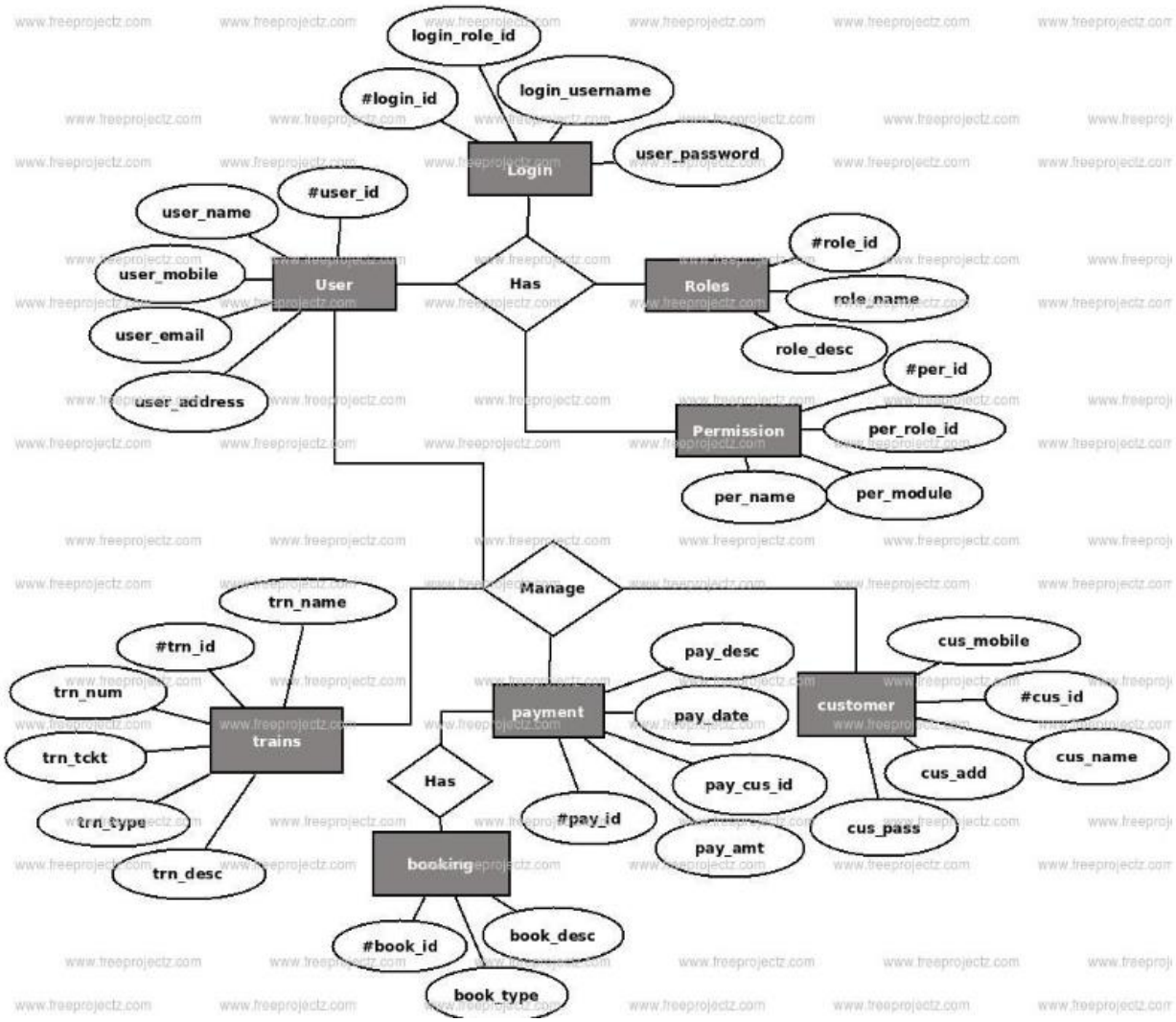
e) State Chart Diagram

f) Sequence Diagram

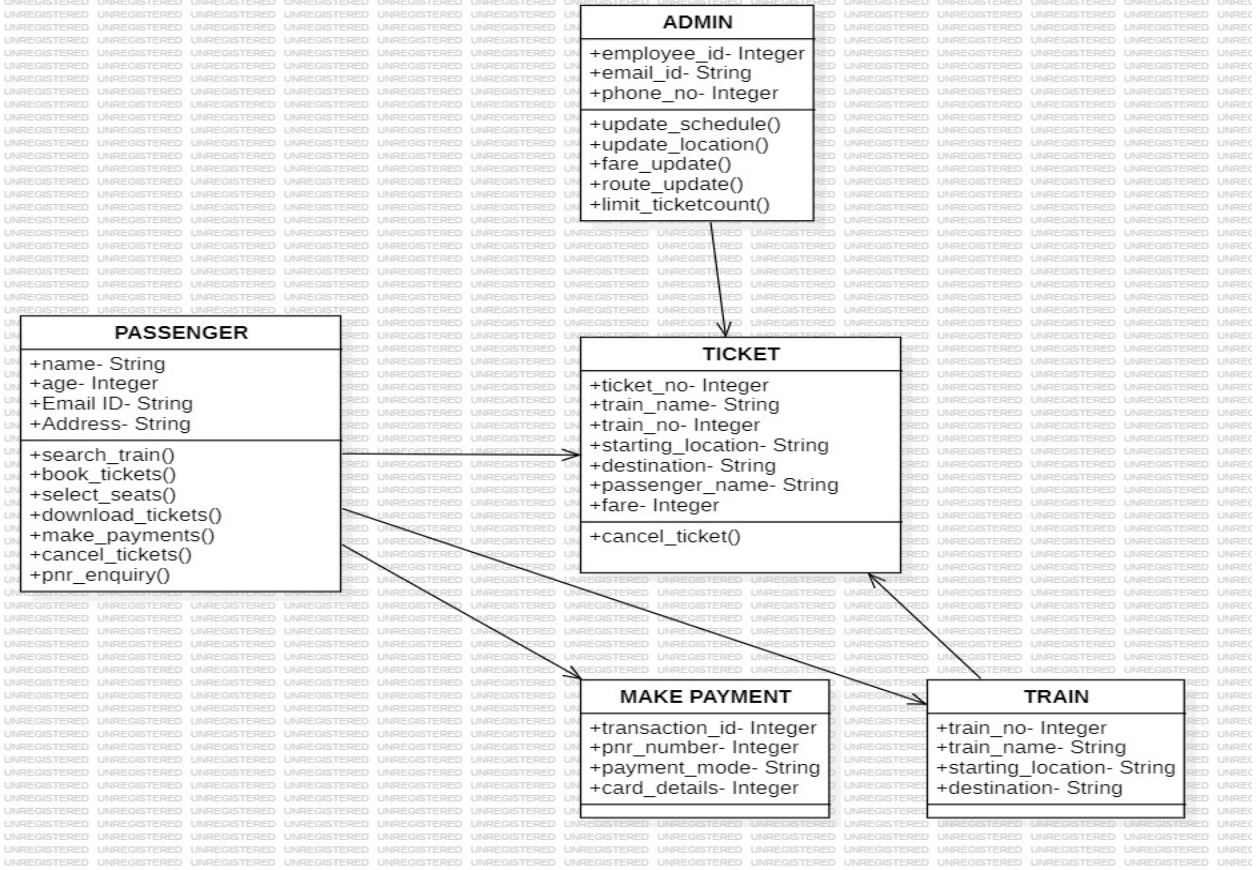
g) Collaboration Diagram

h) Deployment Diagram

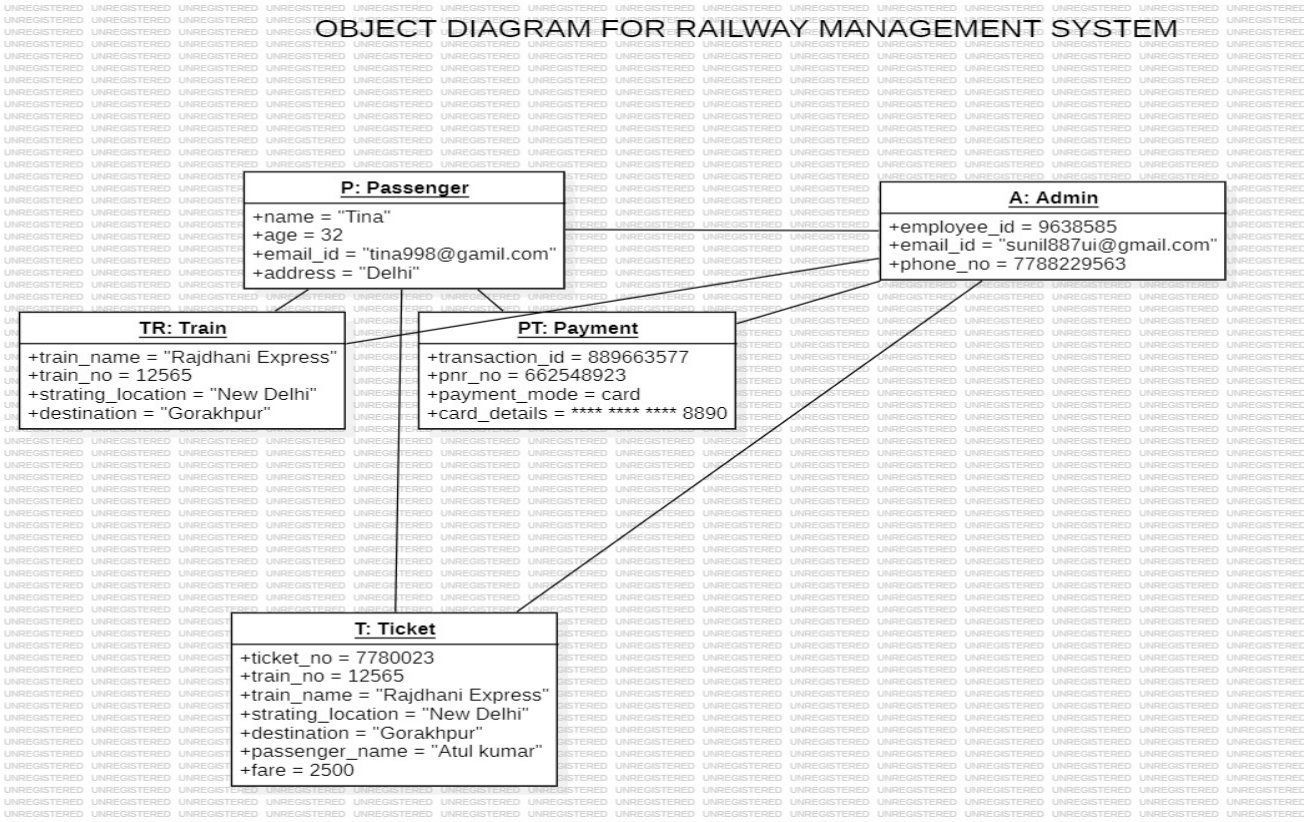
Entity Relationship Diagram



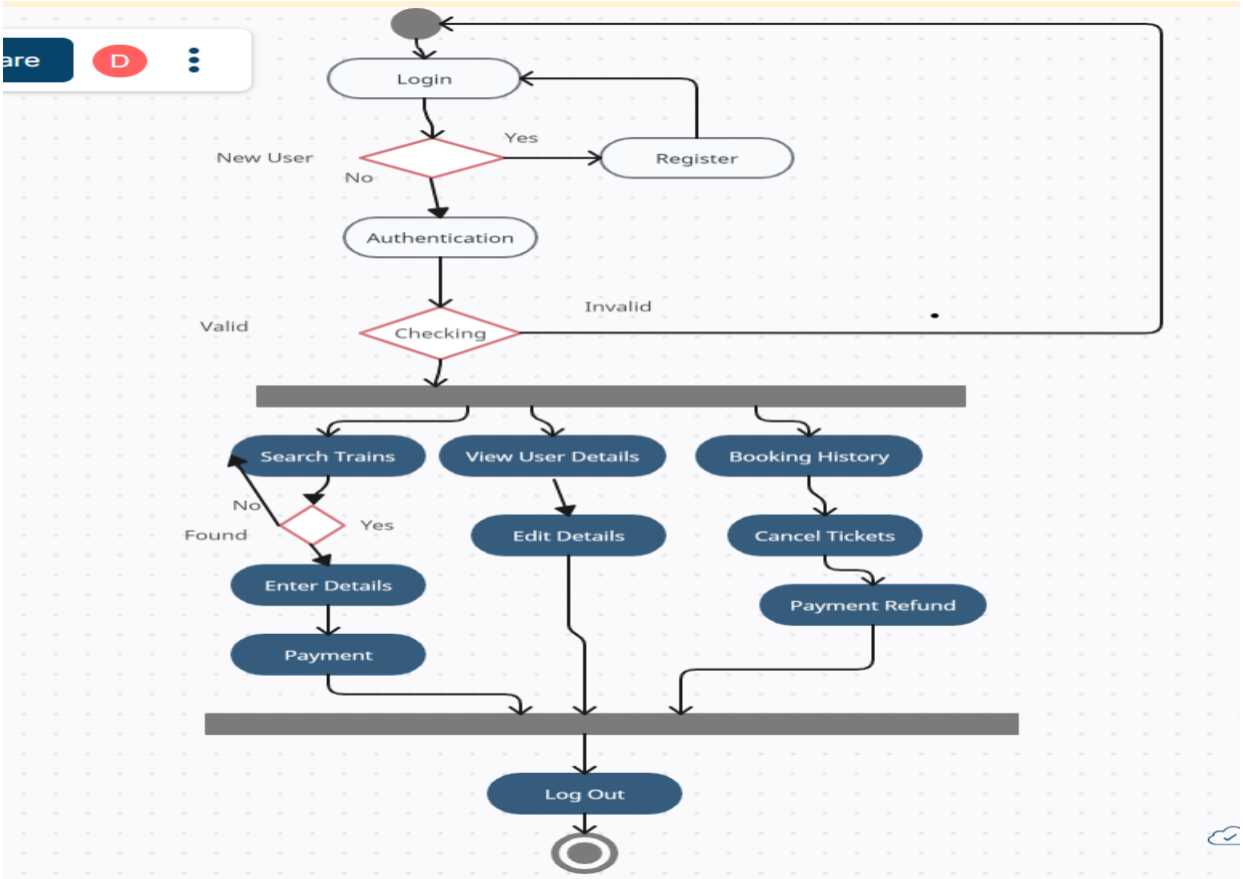
Class Diagram



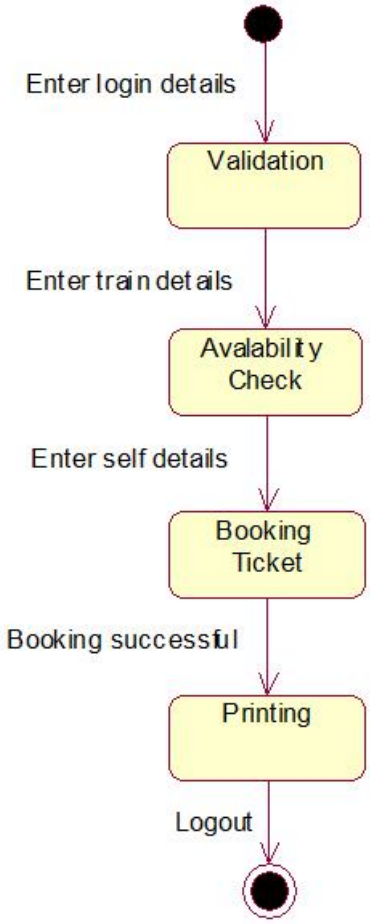
Object Diagram



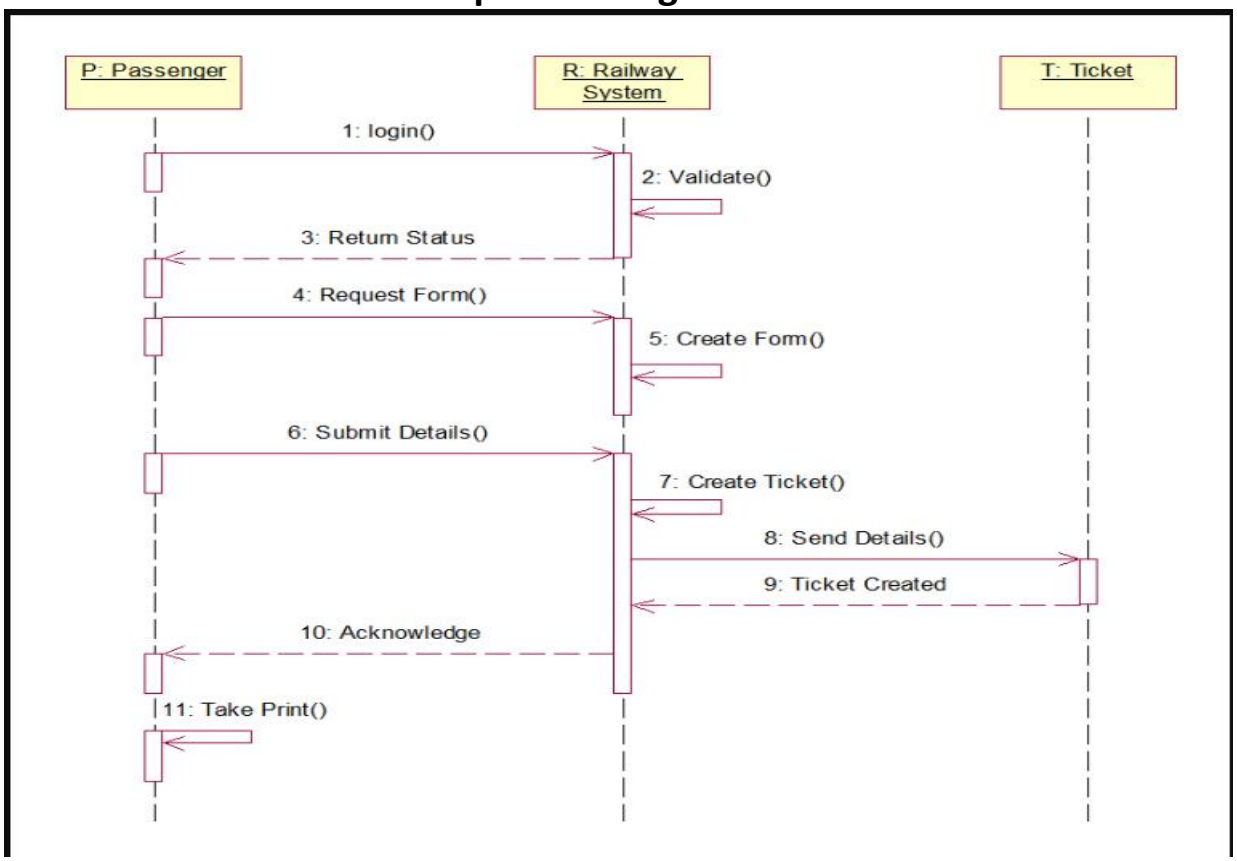
Activity Diagram



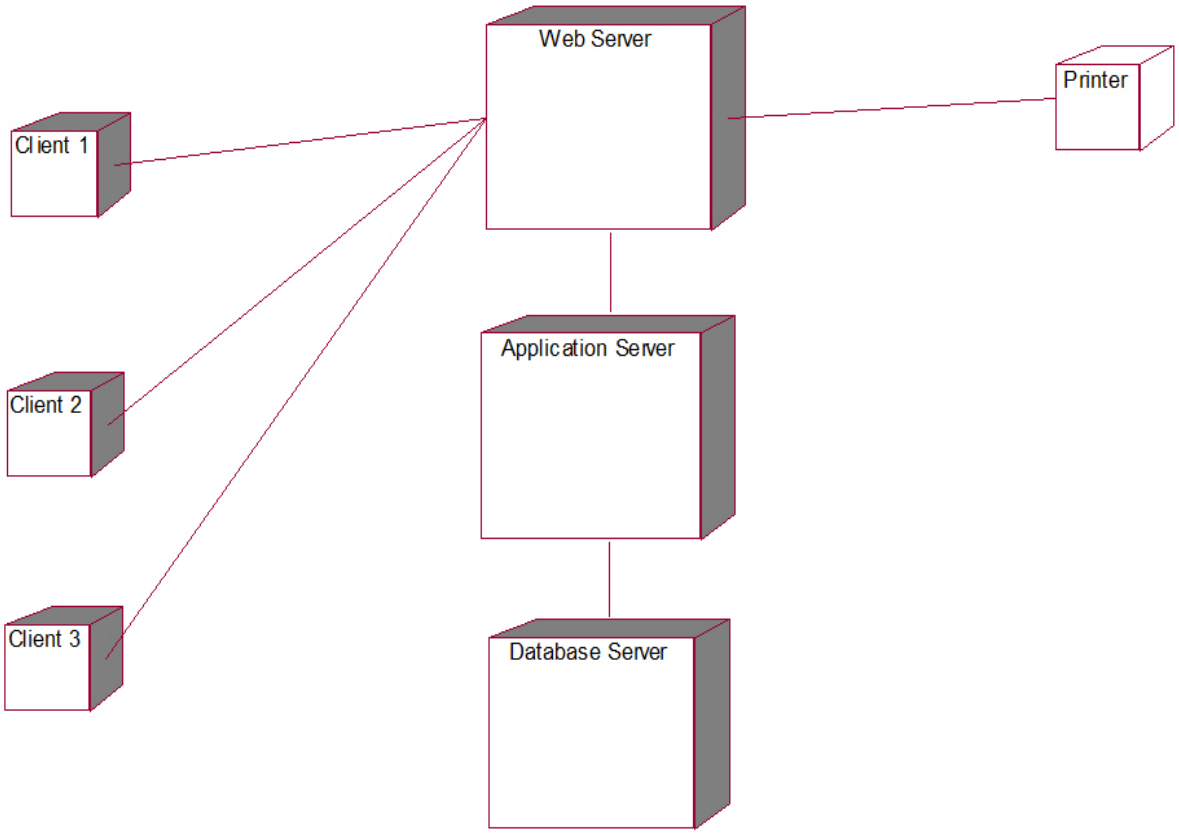
State Chart Diagram



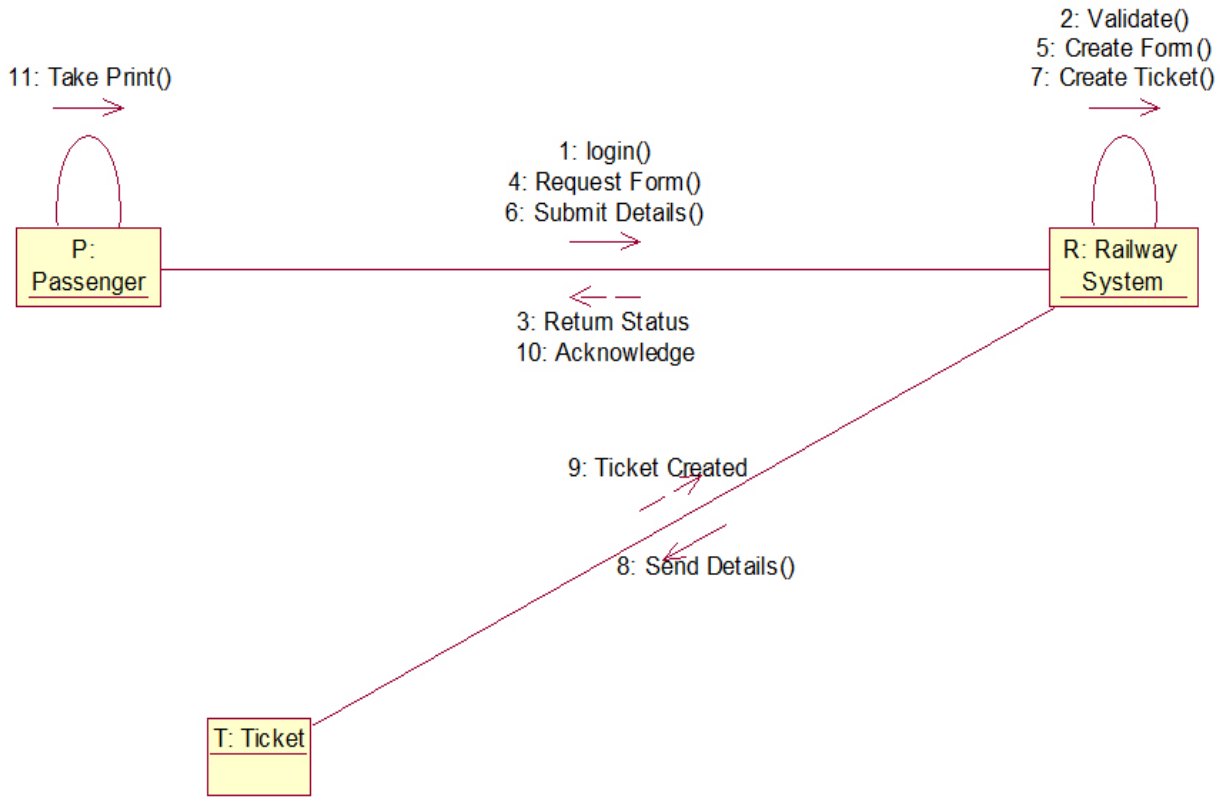
Sequence Diagram



Deployment Diagram



Collaboration Diagram



5. WAP to finds maximum of three numbers and calculate LOC of a given program.

Python program that finds the maximum of three numbers:

```
def maximum_of_three(a, b, c): max_num
  = a
  if b > max_num:
    max_num = b
  if c > max_num:
    max_num = c
  return max_num #
Example usage:
num1 = 10
num2 = 25
num3 = 15
result = maximum_of_three(num1, num2, num3)
print("The maximum of the three numbers is:", result)
```

The lines of code (LOC) calculation, the LOC count will vary depending on how we define it. In a general sense,we can count the non-empty lines of code (excluding comments and blank lines) as follows:

```
def maximum_of_three(a, b, c):  # 1 LOC
  max_num = a  # 1 LOC
  if b > max_num:  # 1 LOC
      max_num = b  # 1 LOC
  if c > max_num:  # 1 LOC
    max_num = c  # 1 LOC
  return max_num  # 1 LOC #
Example usage: # 1 LOC
num1 = 10  # 1 LOC
num2 = 25  # 1 LOC
num3 = 15  # 1 LOC
result = maximum_of_three(num1, num2, num3)  # 1 LOC
print("The maximum of the three numbers is:", result)  # 1 LOC
```

In this case, excluding comments, blank lines, and considering each line with actual code as 1 LOC, the total LOC for this program is 13.

6. WAP to implement the function point method of software estimation.

The Function Point method is a technique used in software engineering for estimating the size and complexity of software development projects. It involves assessing different aspects of the system based on function points, which are units representing functionality provided to the user.

Here's a simplified Python program that performs a basic estimation using the Function Point method:

```
def calculate_function_points(inputs, outputs, inquiries, files, interfaces):
    total_function_points = inputs + outputs + inquiries + files + interfaces
    return total_function_points

# Example estimation
num_inputs = 5
num_outputs = 4
num_inquiries = 3
num_files = 6
num_interfaces = 2
result = calculate_function_points(num_inputs, num_outputs, num_inquiries, num_files, num_interfaces)
print("Estimated Function Points:", result)
```

The function calculate_function_points simply sums up the inputs, outputs, inquiries, files, and interfaces to estimate the total function points.

7. WAP to implement COCOMO-1 model of software size estimation.

The COCOMO (Constructive Cost Model) is a classic software cost estimation model. COCOMO-1 is the original version and provides a basic estimation based on three modes: Organic, Semi-Detached, and Embedded.

Here's a simple Python program that implements the COCOMO-1 model for software size estimation:

```
def cocomo_1(LOC, mode):
    if mode.lower() == "organic":
        effort = 2.4 * (LOC ** 1.05)
    elif mode.lower() == "semi-detached":
        effort = 3.0 * (LOC ** 1.12)
    elif mode.lower() == "embedded":
        effort = 3.6 * (LOC ** 1.20)
    else:
        return "Invalid mode. Please choose 'Organic', 'Semi-Detached', or 'Embedded'."
    return f"Estimated Effort: {effort:.2f} Person-Months"
# Example estimation
lines_of_code = 10000  # Number of Lines of Code (LOC)
project_mode = "organic"  # Mode: Organic, Semi-Detached, Embedded
result = cocomo_1(lines_of_code, project_mode)
print("COCOMO-1 Estimation:")
print(result)
```

This provides the estimated effort in person-months based on the number of lines of code (LOC) and the chosen project mode.

8. WAP to calculate Cyclomatic Complexity.

Cyclomatic Complexity is a software metric used to measure the complexity of a program. It's calculated based on the control flow of theprogram, counting the number of independent paths through the sourcecode.

Here's a simple Python program to calculate Cyclomatic Complexity for agiven code using a basic approach:

```
def cyclomatic_complexity(code):
    num_decision_points = code.count('if') + code.count('elif') + code.count('while') + code.count('for')
    num_edges = code.count(';') + num_decision_points
    complexity = num_edges - len(code.split('\n')) + 2
    return complexity
# Example code snippet
sample_code = """
def max_of_three(a, b, c):if a
    > b:
        if a > c:
            return a
        else:
            return c
    else:
        if b > c:
            return b
        else:
            return c
"""
result = cyclomatic_complexity(sample_code)print("Cyclomatic
Complexity:", result)
```

This estimates Cyclomatic Complexity by counting decision points (such asif, elif, while, for statements) and statements as edges in the code. Replacesample_code with your actual code to compute its Cyclomatic Complexity.

9. Draw Gantt Chart and Pert chart for any sample project with description.

Gantt Chart:

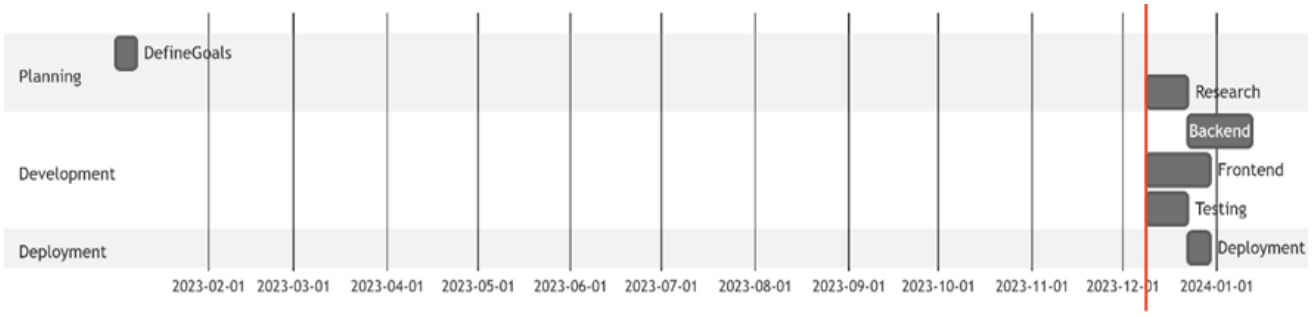
A Gantt Chart is a visual representation of project scheduling. In the context of a job search portal project, the Gantt Chart outlines various tasks and their durations, displayed against a timeline. Each task is represented by a bar, showing its start and end dates.

Description:

Planning Phase: The initial phase involves defining project goals and conducting research.

Development Phase: Subdivided into Backend Development, Frontend Development, and Testing.

Deployment Phase: This final phase includes deployment tasks.



PERT Chart:

PERT (Program Evaluation and Review Technique) Charts display the flow and dependencies between different tasks or activities within a project.

Description:

Dependencies: Arrows connect tasks showing their dependencies, for instance, research preceding backend and frontend development.

Task Flow: Shows the flow of tasks, for example, the completion of backend and frontend development before testing.

Critical Path: Indicates the longest path through the project tasks, highlighting critical activities that might impact the overall project timeline.

Both charts help project managers visualize and plan tasks, dependencies, and project timelines for effective project management and execution.

PROGRAM 1: TRIANGLE PROBLEM

Accept three integers which are supposed to be the three sides of triangle and determine if the three values represent an equilateral triangle, isosceles triangle, scalene triangle, or they do not form a triangle at all.

PRE-CONDITION: $1 \leq a \leq 10$, $1 \leq b \leq 10$ and $1 \leq c \leq 10$ and $a < b + c$, $b < a + c$ and $c < a + b$.

CODE:

```
def triangle_type(a, b, c):
    if a == b == c:
        return "Equilateral Triangle"
    elif a == b or a == c or b == c:
        return "Isosceles Triangle"
    elif a != b != c:
        return "Scalene Triangle"
    else:
        return "Not a Triangle"

# Test the triangle type function
side1 = 5
side2 = 5
side3 = 5

result = triangle_type(side1, side2, side3)
print("Triangle Type:", result)
```

PROGRAM 2: DAY IN A MONTH PROBLEM

Find number of days in a month

CODE:

```
def days_in_month(month, year):
    thirty_one_days = [1, 3, 5, 7, 8, 10, 12]
    thirty_days = [4, 6, 9, 11]

    if month in thirty_one_days:
        return 31
    elif month in thirty_days:
        return 30
    elif month == 2:
        if year % 4 == 0 and (year % 100 != 0 or year % 400 == 0):
            return 29  # Leap year
        else:
            return 28 # Non-leap year
    else:
        return -1  # Invalid month

# Test the days_in_month function
month = 2
year = 2024

result = days_in_month(month, year)
print("Number of Days in the Month:", result)
```

PROGRAM 3: STUDENT DIVISION PROBLEM

Consider the program for the determination of division of a student.Its input is a triple of positive integers (mark1, mark2, mark3) and values are from interval [0, 100]. The program is given . The output
may be one of the following words: [First division with distinction, Firstdivision, Second division, Third division, Fail, Invalid marks].

PRE-CONDITION: 0 <= mark1 <= 100, 0 <= mark2 <= 100, 0 <= mark3 <= 100.

CODE:

```
def determine_division(mark1, mark2, mark3):
    if mark1 < 0 or mark1 > 100 or mark2 < 0 or mark2 > 100 or mark3 <0 or mark3 > 100:
        return "Invalid marks"
    average_mark = (mark1 + mark2 + mark3) / 3if
    average_mark >= 70:
        if mark1 >= 70 and mark2 >= 70 and mark3 >= 70:return
            "First division with distinction"
        else:
            return "First division"elif
    average_mark >= 60:
        return "Second division"elif
    average_mark >= 50:
        return "Third division"else:
return "Fail"
# Test the determine_division functionmark1 = 85
mark2 = 90
mark3 = 75
result = determine_division(mark1, mark2, mark3)print("Student's
Division:", result)
```

10. Perform Boundary value analysis for the given programs.

Boundary Value Analysis:

Description: Boundary Value Analysis focuses on testing at the edges of input domains.

Purpose: Ensures the correct handling of values at the boundaries of valid ranges.

Test Cases: Tests include values at the lower, upper, and midpoints of valid ranges.

PROGRAM 1: TRIANGLE PROBLEM

Test Case	Side A	Side B	Side C	Expected Result
Test 1	1	1	1	Equilateral Triangle
Test 2	1	1	2	Isosceles Triangle
Test 3	1	2	2	Isosceles Triangle
Test 4	2	2	1	Isosceles Triangle
Test 5	2	3	4	Scalene Triangle
Test 6	3	3	7	Not a Triangle
Test 7	10	10	10	Equilateral Triangle
Test 8	1	2	3	Not a Triangle

This table includes boundary value test cases covering scenarios at the edges of the input domain.

PROGRAM 2: DAY IN A MONTH PROBLEM

Test Case	Month	Year	Expected Result
Test 1	1	2022	31
Test 2	2	2022	28
Test 3	2	2024	29
Test 4	4	2022	30
Test 5	5	2022	31
Test 6	12	2022	31
Test 7	13	2022	-1
Test 8	0	2022	-1
Test 9	2	1900	28
Test 10	2	2000	29

Covers tests at boundaries of valid ranges (months 1-12, leap year, non-leap year).

PROGRAM 3: STUDENT DIVISION PROBLEM

Test Case	Mark 1	Mark 2	Mark 3	Expected Result
Test 1	0	0	0	Fail
Test 2	50	50	50	Third division
Test 3	60	60	60	Second division
Test 4	70	70	70	First division with distinction
Test 5	85	90	75	First division with distinction
Test 6	101	90	75	Invalid marks
Test 7	85	-1	75	Invalid marks
Test 8	85	90	101	Invalid marks
Test 9	-1	90	75	Invalid marks
Test 10	85	101	75	Invalid marks

Tests cover scenarios at the edges of the input domain, including minimum, maximum, and midpoints.

11. Perform Robust Testing and Worst case testing for Triangle and Month problem.

PROGRAM 1: TRIANGLE PROBLEM

Robust Testing

The robust testing for the triangle problem includes tests with invalid inputs, such as negative sides, zero, and sides violating the triangle inequality.

Robust Testing Table:

Test Case	Side A	Side B	Side C	Expected Result
Test 1	-1	2	3	Not a Triangle
Test 2	1	-2	3	Not a Triangle
Test 3	1	2	-3	Not a Triangle
Test 4	0	2	3	Not a Triangle
Test 5	1	0	3	Not a Triangle
Test 6	1	2	0	Not a Triangle
Test 7	5	1	1	Not a Triangle
Test 8	1	5	1	Not a Triangle
Test 9	1	1	5	Not a Triangle

Covers tests with invalid inputs (negative sides, zero, sides violating the triangle inequality).

Worst Case Testing:

The worst-case scenario testing for the triangle problem involves the sides being at the edge values where they barely meet the triangle inequality or sides forming an invalid triangle.

Worst Case Testing Table:

Test Case	Side A	Side B	Side C	Expected Result
Test 1	1	1	1	Equilateral Triangle
Test 2	1	2	3	Not a Triangle
Test 3	1	3	2	Not a Triangle
Test 4	2	1	3	Not a Triangle
Test 5	2	3	1	Not a Triangle
Test 6	3	1	2	Not a Triangle
Test 7	3	2	1	Not a Triangle

Focuses on edge cases where the sides barely meet the triangle inequality or form an invalid triangle.

PROGRAM 2: DAY IN A MONTH PROBLEM

Robust Testing

The robust testing for the month problem involves tests with invalid inputs, such as negative months, out-of-range months, and years for leap year verification.

Robust Testing Table:

Test Case	Month	Year	Expected Result
Test 1	-1	2022	Invalid marks
Test 2	13	2022	Invalid marks
Test 3	2	1900	28
Test 4	2	2000	29

Covers invalid inputs (negative months, out-of-range months, leap year verification).

Worst Case Testing

The worst-case scenario testing for the month problem includes inputs at the edge values where leap year verification is essential.

Worst Case Testing Table:

Test Case	Month	Year	Expected Result
Test 1	2	1900	28
Test 2	2	2000	29

Focuses on edge cases where leap year verification is critical, such as February with different year scenarios.

These tests verify the behaviour of the programs under different invalid input scenarios and at the edges of the valid input ranges, ensuring robustness and coverage of edge cases.

12. Perform Robust Worst case testing for Triangle and Month problem.

PROGRAM 1: TRIANGLE PROBLEM

Robust Worst Case Testing:

For the Triangle problem, we'll combine the robust and worst-case scenarios to handle invalid inputs, edges of valid inputs, and conditions that barely meet or violate the triangle inequality.

Robust Worst Case Testing Table:

Test Case	Side A	Side B	Side C	Expected Result
Test 1	-1	2	3	Not a Triangle
Test 2	1	-2	3	Not a Triangle
Test 3	1	2	-3	Not a Triangle
Test 4	0	2	3	Not a Triangle
Test 5	1	0	3	Not a Triangle
Test 6	1	2	0	Not a Triangle
Test 7	5	1	1	Not a Triangle
Test 8	1	5	1	Not a Triangle
Test 9	1	1	5	Not a Triangle
Test 10	1	1	1	Equilateral Triangle
Test 11	1	2	3	Not a Triangle

Test Case	Side A	Side B	Side C	Expected Result
Test 12	1	3	2	Not a Triangle
Test 13	2	1	3	Not a Triangle
Test 14	2	3	1	Not a Triangle
Test 15	3	1	2	Not a Triangle
Test 16	3	2	1	Not a Triangle

Covers invalid inputs, edges of valid inputs, and conditions that barely meet or violate the triangle inequality.

PROGRAM 2: DAY IN A MONTH PROBLEM

Robust Worst Case Testing:

For the Month problem, combining robust and worst-case scenarios includes invalid inputs, edge cases for months, and leap year verification.

Robust Worst Case Testing Table:

Test Case	Month	Year	Expected Result
Test 1	-1	2022	Invalid marks
Test 2	13	2022	Invalid marks
Test 3	2	1900	28
Test 4	2	2000	29
Test 5	1	1	31
Test 6	12	9999	31
Test 7	2	1901	28
Test 8	2	2100	28

Encompasses invalid inputs, edge cases for months, and leap year verification, including extreme year scenarios.

13. Perform Equivalence class testing on Triangle Problem, Month Problem and Student Division Problem.

Equivalence class testing focuses on categorizing inputs into equivalence classes (valid and invalid) to ensure comprehensive test coverage.

PROGRAM 1: TRIANGLE PROBLEM

Equivalence Class Testing Table for Triangle Problem:

Test Case	Side A	Side B	Side C	Expected Result
Test 1	5	5	5	Equilateral Triangle
Test 2	5	5	10	Isosceles Triangle
Test 3	5	10	5	Isosceles Triangle
Test 4	10	5	5	Isosceles Triangle
Test 5	3	4	5	Scalene Triangle
Test 6	-1	2	3	Not a Triangle
Test 7	1	-2	3	Not a Triangle
Test 8	1	2	-3	Not a Triangle
Test 9	0	2	3	Not a Triangle
Test 10	1	0	3	Not a Triangle
Test 11	1	2	0	Not a Triangle
Test 12	5	1	1	Not a Triangle
Test 13	1	5	1	Not a Triangle

Test Case	Side A	Side B	Side C	Expected Result
Test 14	1	1	5	Not a Triangle

The tests cover various types of triangles (equilateral, isosceles, scalene), as well as invalid cases (not forming a triangle).

PROGRAM 2: DAY IN A MONTH PROBLEM

Equivalence Class Testing Table for Month Problem:

Test Case	Month	Year	Expected Result
Test 1	1	2022	31
Test 2	2	2022	28
Test 3	2	2024	29
Test 4	4	2022	30
Test 5	5	2022	31
Test 6	12	2022	31
Test 7	-1	2022	Invalid marks
Test 8	13	2022	Invalid marks
Test 9	2	1900	28
Test 10	2	2000	29

Covers months with different days (28, 29, 30, 31) and invalid inputs (negative month, out-of-range month).

PROGRAM 3: STUDENT DIVISION PROBLEM

Equivalence Class Testing Table for Student Division Problem:

Test Case	Mark 1	Mark 2	Mark 3	Expected Result
Test 1	0	0	0	Fail
Test 2	50	50	50	Third division
Test 3	60	60	60	Second division
Test 4	70	70	70	First division with distinction
Test 5	85	90	75	First division with distinction
Test 6	-1	90	75	Invalid marks
Test 7	85	101	75	Invalid marks

Categorizes the student's division into different classes (first, second, third, fail) and invalid cases (invalid marks).

Equivalence class testing ensures coverage of different valid and invalid equivalence classes for each problem, aiding in comprehensive testing of different scenarios.

14. Perform Limited Entry Decision Table testing on Triangle and Student Division Problem.

Limited Entry Decision Table testing involves creating a table that outlines different combinations of inputs and their expected outcomes based on specific conditions.

PROGRAM 1: TRIANGLE PROBLEM

Limited Entry Decision Table Testing for Triangle Problem:

Test Case	Side A	Side B	Side C	Conditions	Expected Result
Test 1	5	5	5	All sides equal	Equilateral Triangle
Test 2	5	5	10	Two sides equal, different from third	Isosceles Triangle
Test 3	5	10	5	Two sides equal, different from third	Isosceles Triangle
Test 4	10	5	5	Two sides equal, different from third	Isosceles Triangle
Test 5	3	4	5	All sides different	Scalene Triangle
Test 6	-1	2	3	Invalid input: Negative side	Not a Triangle
Test 7	1	-2	3	Invalid input: Negative side	Not a Triangle
Test 8	1	2	-3	Invalid input: Negative side	Not a Triangle

Test Case	Side A	Side B	Side C	Conditions	Expected Result
Test 9	0	2	3	Invalid input: Zero side	Not a Triangle
Test 10	1	0	3	Invalid input: Zero side	Not a Triangle
Test 11	1	2	0	Invalid input: Zero side	Not a Triangle
Test 12	5	1	1	Violation of triangle inequality	Not a Triangle
Test 13	1	5	1	Violation of triangle inequality	Not a Triangle
Test 14	1	1	5	Violation of triangle inequality	Not a Triangle

This decision table covers different conditions for sides of a triangle, including valid equilateral, isosceles, and scalene triangles, as well as invalid inputs violating the triangle inequality.

PROGRAM 3: STUDENT DIVISION PROBLEM

Limited Entry Decision Table Testing for Student Division Problem:

Test Case	Mark 1	Mark 2	Mark 3	Conditions	Expected Result
Test 1	0	0	0	All marks zero	Fail
Test 2	50	50	50	All marks equal and in third division	Third division
Test 3	60	60	60	All marks in second division range	Second division
Test 4	70	70	70	All marks in first division range	First division with distinction
Test 5	85	90	75	All marks in first division range	First division with distinction
Test 6	-1	90	75	Invalid mark: Negative	Invalid marks
Test 7	85	101	75	Invalid mark: Out of range	Invalid marks

The decision table categorizes different combinations of marks into different division categories and includes invalid inputs (negative marks, out-of-range marks).

Limited Entry Decision Table testing helps cover various combinations of inputs based on conditions, ensuring the system responds correctly to different scenarios and conditions.