Course Code	**:**	**BCS-051**
Course Title	**:**	**Introduction to Software Engineering**
Assignment Number	**:**	**BCA(V)051/Assignment/2024-25**
Maximum Marks	**:**	**100**
Weightage	**:**	**25%**
Last Date of Submission	**:**	**31st October,2024(For July, Session)** **30th April, 2025(For January, Session)**

This assignment has eight questions for a total of 80 marks. Answer all the questions. Each question carries 10 marks. Rest 20 marks are for viva voce. You may use illustrations and diagrams to enhance explanations. Please go through the guidelines regarding assignments given in the Programme Guide for the format of presentation.

Q1. What is SRS? Develop SRS for "Railway Reservation System". Make necessary assumptions. Follow IEEE SRS format. Briefly explain the characteristics of a good SRS.

Q2. Draw first three levels of DFDs for a "Railway Reservation System". Make assumptions, wherever necessary. Briefly explain the all the DFDs with respect to Railway Reservation System.

Q3. Develop a test case for any testing technique for "Railway Reservation System". Briefly explain the all the test cases with respect to Railway Reservation System.

Q4. What are application logic objects? Explain with the help of an example.

Q5. What is Spiral model for software development? Explain the types of software systems developed using this model.

Q6.

a) Explain the different categories of Software Maintenance.
b) Draw GANTT chart for the development of "Railway Reservation System". Briefly explain the chart with respect to Railway Reservation System.

Q7. What is Software Configuration Management (SCM) ? Explain the need of SCM with the help of an example.

Q8. Write short notes on the following:
(a) Object Oriented Metrics
(b) Coupling
(c) Software Quality Assurance
(d) Capability Maturitys Model

Q1. What is SRS? Develop 5RS for "Railway Reservation System". Make necessary assumptions. Follow IEEE SRS format. Briefly explain the characteristics of a good SRS.

Answer:

What is SRS?

Software Requirement Specification (SRS) Format as the name suggests, is a complete specification and description of requirements of the software that need to be fulfilled for the successful development of the software system. These requirements can be functional as well as non-functional depending upon the type of requirement. The interaction between different customers and contractors is done because it is necessary to fully understand the needs of customers.

SRS for Railway Reservation System

1.INTORDUCTION

The introduction of the Software Requirements Specification (SRS) provides an overview of the entire SRS purpose scope, definitions, acronyms, abbreviations, references and overview of SRS A Software Requirements Specification (SRS) a requirements specification for a software system is a complete description of the behaviour of a system to be developed. It includes a set of use cases that describe all that interactions the users will have with the software. Use cases are abo known as functional requirements. In addition to use cases, the SRS also contains non-functional (or supplementary) requirements. Non-factional requirements are requirements which impose constraints on the design or implementation (such as performance engineering requirements, quality standards, or design constraints). The aim of this document is to gather and analyse and give an in-depth insight of the complete Marvel Electronics and Home Entertainment software system by defining the problem statement in detail. This is a documentation of the project Railways Reservation System done sincerely and satisfactorily by my group members. 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 OBJECTIVE:-

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.

The origin of most software systems is in the need of a client, who either wants to automate the existing manual system or desires a new software system. The software system is itself created by the developer. Finally, the end user will use the completed system. Thus, there are three major parties interested in a new system: the client, the user, and the developer. Somehow the requirements for the system that will satisfy the needs of the clients and the concerns of the users have to be communicated to the developer. The problem is that the client doesn't usually design the software or the software development process and the

1.2 SCOP:-

- "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 Glossary:-

- This should define all technical terms and abbreviations used in the document
- NTES- National Train Enquiry System
- IVRS- Interactive Voice Response system
- PRS-passenger reservation system

2.Overall Description

This document contains the problem statement that the current system is facing which is hampering the growth opportunities of the company. It further contains a list of the stakeholders and users of the proposed solution. It also illustrates the needs and wants of the stakeholders that were identified in the brainstorming exercise as part of the requirements workshop. It further lists and briefly describes the major features and a brief description of each of the proposed system.

2.1 Product Perspective:

- Before the automation, the system suffered from the following DRAWBACKS:
- The existing system is highly manual involving a lot of paper work and calculation and therefore may be erroneous. This has lead to inconsistency and inaccuracy in the maintenance of data.
- The data, which is stored on the paper only, may be lost, stolen or destroyed due to natural calamity like fire and water.
- The existing system is sluggish and consumes a lot of time causing inconvenience to Ø customers and the airlines staff.
- Due to manual nature, it is difficult to update, delete, add or view the data.

- Since the number of passengers have drastically increased therefore maintaining and retrieving detailed record of passenger is extremely difficult.
- An railways has many offices around the world, an absence of a link between these offices lead to lack of coordination and communication.
- Hence the railways reservation system is proposed with the following:-
- The computerization of the reservation system will reduce a lot of paperwork and hence the load on the airline administrative staff.
- The machine performs all calculations. Hence chances of error are nil.
- EDUCATIONAL LEVEL:-At least user of the system should be comfortable with English language._
- TECHNICAL EXPERTISE:- User should be comfortable using general purpose applications on the computer system.

2.2 Project Functions:

Booking agents with varying levels of familiarity with computers will mostly use this system. With this in mind, an important feature of this software is that it be relatively simple to use. The scope of this project encompasses:

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.

Characteristics of a Good SRS

1. Clarity: The requirements should be clearly stated and easily understandable by all stakeholders.
2. Consistency: The document should be free from contradictions and should use consistent terminology.
3. Completeness: All necessary requirements should be included to avoid ambiguities.
4. Modifiability: The document should be organized in a way that allows for easy updates and changes.
5. Verifiability: The requirements should be stated in a manner that allows them to be tested and verified.
6. Traceability: Each requirement should be traceable back to its origin and should be linked to design and testing phases.

2.4 Constrains:

- Software constraints:
- The system will run under windows98 or higher platforms of operating system.

2.5 Assumptions and Dependencies:

- Booking Agents will be having a valid user name an password to access the software
- The software needs booking agent to have complete knowledge of railways reservation system.
- Software is dependent on access to internet.

3.1 Function Requirements

3.1.1 performance requirements:

- User Satisfaction: The system is such that it stands up to the user expectations.
- Response Time: The response of all the operation is good. This has been made possible by careful programming.
- Error Handling: Response to user errors and undesired situations has been taken care of to ensure that the system operates without halting.
- Safety and Robustness: The system is able to avoid or tackle disastrous action. In other words, it should be foul proof. The system safeguards against undesired events, without human intervention.
- Portable: The software should not be architecture specific. It should be easily transferable to other platforms if needed.
- User friendliness:The system is easy to learn and understand. A native user can also use the system effectively, without any difficulties.

3.1.2 Design constraint:

There are a number of factors in the client's environment that may restrict the choices of a designer. Such factors include standards that must be followed, resource limits, operating environment, reliability and security requirements and policies that may have an impact on the design of the system. An SRS (Software Requirements Analysis and Specification) should identify and specify all such constraints.

- Standard Compliance: This specifies the requirements for the standards the system must follow. The standards may include the report format and accounting properties.
- Hardware Limitations: The software may have to operate on some existing or predetermined hardware, thus imposing restrictions on the design. Hardware limitations can include the types of machines to be used, operating system available on the system, languages supported and limits on primary and secondary storage.
- Reliability and Fault Tolerance: Fault tolerance requirements can place a major constraint on how the system is to be designed. Fault tolerance requirements often make the system more complex and expensive. Requirements about system behavior in the face of certain kinds of faults are specified. Recovery requirements are often an integral part here, detailing what the system should do I some failure occurs to ensure certain properties. Reliability requirements are very important for critical applications.
- Security: Security requirements are particularly significant in defence systems and database systems. They place restrictions on the use of certain commands, control access to data, provide different kinds of access requirements for different people, require the use of passwords and cryptography techniques and maintain a log of activities in the system.

3.1.3 Hardware requirements:

For the hardware requirements the SRS specifies the logical characteristics of each interface b/w the software product and the hardware components. It specifies the hardware requirements like memory restrictions, cache size, the processor, RAM size etc... those are required for the software to run.

Minimum Hardware Requirements

Processor Pentium III

Hard disk drive 40 GB

RAM 128 MB

Cache 512 kb

Preferred Hardware Requirements

Processor Pentium IV

Hard disk drive 80 GB

RAM 256 MB

Cache 512 kb

3.1.4 Software requirements:

Any window based operating system with DOS support are primary requirements for software development. Windows XP, FrontPage and dumps are required. The systems must be connected via LAN and connection to internet is mandatory.

3.1.5 other requirements:

Software should satisfy following requirements as well:-

- SECURITY
- PORTABILITY
- CORRECTNESS
- EFFICIENCY
- FLEXIBILTY

3.2 Non-Function Requirements

3.2.1 Security:

The system use SSL (secured socket layer) in all transactions that include any confidential customer information. The system must automatically log out all customers after a period of inactivity. The system should not leave any cookies on the customer's computer containing the user's password. The system's back-end servers shall only be accessible to authenticated management.

3.2.2 Reliability:

The reliability of the overall project depends on the reliability of the separate components. The main pillar of reliability of the system is the backup of the

database which is continuously maintained and updated to reflect the most recent changes. Also the system will be functioning inside a container. Thus the overall stability of the system depends on the stability of container and its underlying operating system.

3.2.3 Availability:

The system should be available at all times, meaning the user can access it using a web browser, only restricted by the down time of the server on which the system runs. A customer friendly system which is in access of people around the world should work 24 hours. In case of a of a hardware failure or database corruption, a replacement page will be shown. Also in case of a hardware failure or database corruption, backups of the database should be retrieved from the server and saved by the Organizer. Then the service will be restarted. It means 24 x 7 availability.

Q2. Draw first three levels of DFDs for a "Railway Reservation System". Make assumptions, wherever necessary. Briefly explain the all the DFDs with respect to Railway Reservation System

Answer:

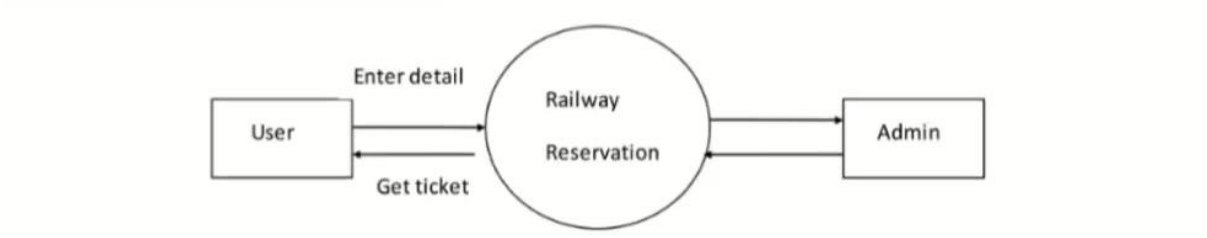
A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system. DFDs can also be used for the visualization of data processing (structured design). On a DFD, data items flow from an external data source or an internal data store to an internal data store or an external data Sink, via an internal process. A DFD provides no information about the timing of processes, or about whether processes will operate in sequence or in parallel. It is therefore quite different from a flowchart, which shows the flow of control through an algorithm, allowing a reader to determine what operations will be performed, in what order, and under what circumstances, but not what kinds of data will be input to and output from the system, nor where the data will come

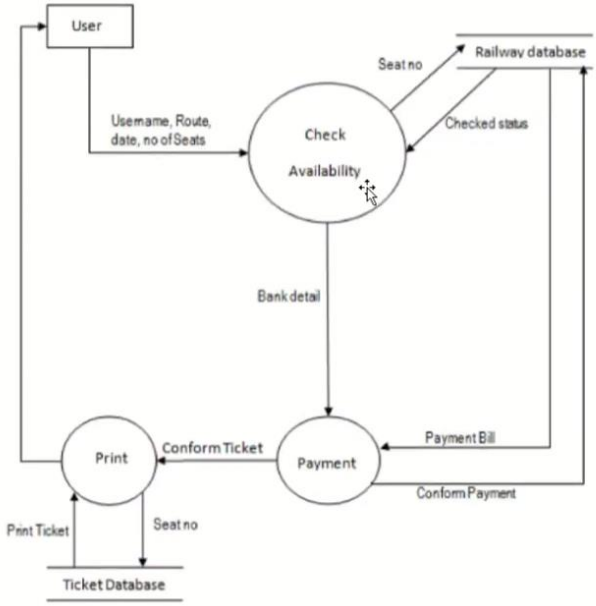
from and go to, nor where the data will be stored (all of which are shown on a DFD).

It is common practice to draw a context-level data flow diagram first, which shows the interaction between the system and external agents which act as data sources and data sinks. On the context diagram (also known as the 'Level 0 DFD) the system's interactions with the outside world are modelled purely in terms of data flows across the system boundary. The context diagram shows the entire system as a single process, and gives no clues as to its internal organization.

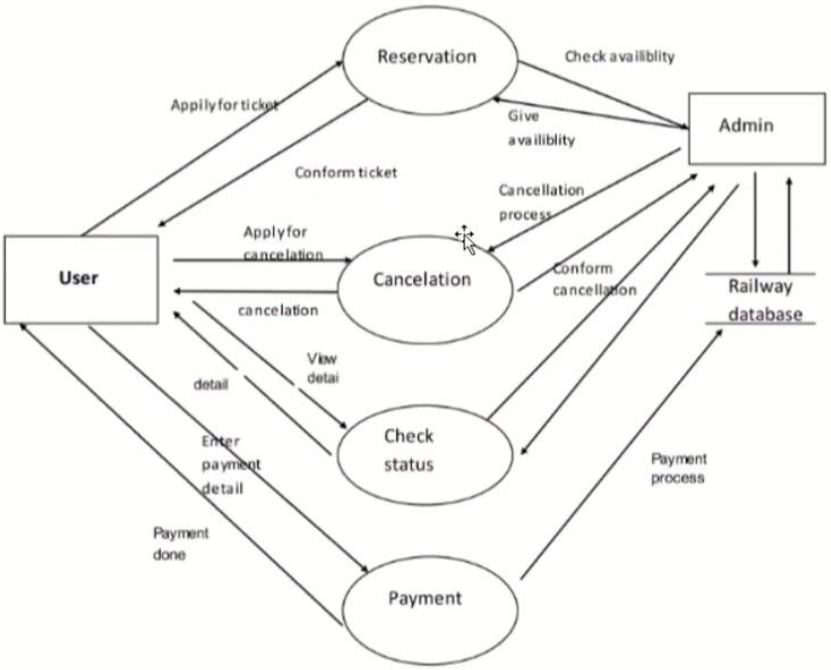
This context-level DFD is next "exploded", to produce a Level 1 DFD that shows some of the detail of the system being modelled. The Level 1 DFD shows how the system is divided into sub-systems (processes), each of which deals with one or more of the data flows to or from an external agent, and which together provide all of the functionality of the system as a whole. It also identifies internal data stores that must be present in order for the system to do its job, and shows the flow of data between the various parts of the system.

Level 0




User
Railway database
Seat no
Username, Route, date, no of Seats
Check Availability
Checked status
Bank detail
Print
Conform Ticket
Payment
Payment Bill
Conform Payment
Print Ticket
Seat no
Ticket Database

Level 2


Reservation
Check availiblity
Applly for ticket
Give availiblity
Admin
Conform ticket
Cancellation process
Apply for cancelation
User
Cancelation
Conform cancellation
Railway database
cancelation
View detai
detail
Check status
Enter payment detail
Payment process
Payment done
Payment

Question 3: Develop a test case for any testing technique for "Railway Reservation System". Briefly explain the all the test cases with respect to Railway Reservation System.

Answer:

When developing test cases for a Railway Reservation System, it's essential to cover various aspects of the system to ensure it functions correctly under different scenarios. Here, I'll outline test cases for Functional Testing a technique that validates whether the software functions as intended according to the requirements.

Test Case for Functional Testing: Booking a Train Ticket

Test Case 1: Successful Ticket Booking

Objective: To verify that a user can successfully book a train ticket when all inputs are valid.

Test Case ID: TC_Booking_001

Preconditions:

- User is registered and logged into the system.
- The train schedule is loaded with available seats.
- The payment gateway is functional.

Test Steps:

1. Navigate to the train search page.
2. Enter valid search criteria (e.g., source, destination, date).
3. Select a train from the search results.
4. Choose a seat type (e.g., economy, business).
5. Enter payment details and proceed to checkout.
6. Confirm the booking.

Expected Results:

1. The system displays available trains based on the search criteria.
2. The user can select a seat and proceed with booking.
3. The payment is processed successfully.
4. The system confirms the booking with a reference number and sends a confirmation email/SMS.

Postconditions:

- The booking details are updated in the database.
- The seat is reserved.
- The user receives a booking confirmation.

Test Case 2: Ticket Booking with Invalid Payment Details

Objective: To verify that the system handles invalid payment details appropriately.

Test Case ID: TC_Booking_002

Preconditions:

- User is registered and logged into the system.
- The train schedule is loaded with available seats.

Test Steps:

1. Navigate to the train search page.
2. Enter valid search criteria (e.g., source, destination, date).
3. Select a train from the search results.
4. Choose a seat type (e.g., economy, business).
5. Enter invalid payment details (e.g., incorrect credit card number) and proceed to checkout.
6. Confirm the booking.

Expected Results:

- The system detects invalid payment details and displays an error message.
- The booking is not processed.
- The seat remains available for other users.

Postconditions:

- No changes are made to seat reservations.
- The user is prompted to correct payment details.

Test Case 3: Canceling a Ticket

Objective: To verify that a user can successfully cancel a booked ticket.

Test Case ID: TC_Cancellation_001

Preconditions:

- User has an existing booking.
- The booking is within the cancellation period.

Test Steps:

1. Log into the user account.
2. Navigate to the "My Bookings" section.
3. Select the booking to be canceled.
4. Initiate the cancellation process.
5. Confirm the cancellation.

Expected Results:

- The system processes the cancellation request.
- The seat is released and made available for other users.
- The user receives a cancellation confirmation and refund, if applicable.

Postconditions:

The booking is removed from the user's account.

The seat availability is updated.

Test Case 4: Viewing Train Schedule

Objective: To verify that users can view the train schedules correctly.

Test Case ID: TC_ViewSchedule_001

Preconditions:

- The train schedule data is loaded in the system.

Test Steps:

1. Navigate to the train schedule page.
2. Enter valid search criteria (e.g., source, destination, date).
3. View the list of available trains.

Expected Results:

- The system displays a list of trains matching the search criteria.
- Each train entry shows the departure and arrival times, train
- number, and seat availability.

Postconditions:

1. Users can view train schedules for the specified date and route.

Test Case 5: Viewing Booking History

Objective: To ensure that users can view their booking history accurately.

Test Case ID: TC_ViewHistory_001

Preconditions:

- User has made previous bookings.

Test Steps:

1. Log into the user account.
2. Navigate to the "Booking History" section.
3. Review the list of past bookings.

Expected Results:

- The system displays a list of all past bookings with details such as booking reference, train details, and date of travel.

Postconditions:

- Users can access and review their booking history.

Q4 What are application logic objects? Explain with the help of an example

Answer:

Application logic objects, also known as business logic objects, are components in a software system that encapsulate the core functionality and business rules of the application. They manage the internal processing of data and enforce business rules, separating this logic from the user interface and data access layers. These objects are crucial in defining how the application responds to various inputs and how it processes data to meet business requirements.

Key Characteristics of Application Logic Objects

1. **Encapsulation:** They encapsulate business rules and operations, ensuring that the logic is centralized and reusable.
2. **Modularity**: They promote modularity by separating business logic from user interface (UI) and data access code.
3. **Maintainability:** They make the system easier to maintain and update by isolating changes to business rules within specific components.
4. **Testability:** They can be independently tested to verify that the business rules and logic are functioning correctly.

Example: Railway Reservation System

Let's consider a Railway Reservation System to illustrate how application logic objects work.

Scenario: Booking a Train Ticket

In this system, several application logic objects could be involved. Here's a simplified example focusing on key objects and their roles:

1. TicketBookingManager

Purpose: Manages the logic for booking tickets, including seat selection, availability checks, and payment processing.

Responsibilities:

Validate booking requests.

Check seat availability.

Calculate the cost of the ticket.

Process payments.

Confirm the booking.

2. TrainScheduleManager

Purpose: Manages train schedules and seat availability.

Responsibilities:

- Retrieve train schedules.
- Check seat availability for a given train and date.
- Reserve or release seats.

Q5. What is Spiral model for software development? Explain the types of software systems developed using this model.

Answer:

the Spiral Model is a software development methodology that combines iterative development with the principles of risk management. Developed by Barry Boehm in 1986, this model emphasizes iterative refinement through repeated cycles, or "spirals," allowing for ongoing evaluation and adjustment throughout the development process.

Key Characteristics of the Spiral Model

1. **Iterative Development:** The model follows an iterative approach, where each iteration (or spiral) involves revisiting phases of the software development lifecycle-planning, risk analysis, engineering, and evaluation.
2. **Risk Management**: Each iteration includes a risk analysis phase, which identifies and addresses potential risks early in the development process.
3. **Prototyping**: The model often involves creating prototypes to help clarify requirements and refine system design based on feedback.
4. **Customer Involvement**: Regular feedback from stakeholders is incorporated into each iteration, ensuring that the system evolves to meet user needs and expectations.

Phases of the Spiral Model

1. **Planning:** Define objectives, constraints, and requirements. Develop an initial plan and schedule.
2. R**isk Analysis**: Identify potential risks and develop strategies to mitigate them. This phase involves assessing the feasibility of the project and analyzing the technical and operational risks.
3. **Engineering:** Design and develop the system or component. Create prototypes if necessary to validate design decisions and requirements.
4. **Evaluation**: Review the progress with stakeholders. Gather feedback and make necessary adjustments. Evaluate the effectiveness of the current iteration and plan for the next one.

Types of Software Systems Developed Using the Spiral Model

The Spiral Model is particularly suited for complex and large-scale software systems where requirements are likely to evolve or where there is a high degree of uncertainty. Here are some examples of software systems where the Spiral Model is often applied:

1. Large Enterprise Systems
   - Example: Enterprise Resource Planning (ERP) systems.
   - Reason: These systems often involve complex requirements and integration with various business processes. The iterative nature of the Spiral Model helps manage the complexity and adapt to changing needs.
2. Custom Software Development
   - Example: Custom CRM (Customer Relationship Management) systems.
   - Reason: Custom solutions are tailored to specific business needs and require frequent feedback and adjustments from stakeholders. The Spiral Model's iterative approach allows for ongoing refinement.
3. Research and Development Projects
   - Example: Advanced research tools or experimental software solutions.
   - Reason: Research projects often involve uncertainty and evolving requirements. The Spiral Model provides a structured way to explore and refine new ideas.
4. High-Risk Projects
   - Example: Defense or aerospace software systems.
   - Reason: High-risk projects benefit from the Spiral Model's focus on risk analysis and mitigation. The model helps identify potential issues early and adapt to unforeseen challenges.
5. Prototyping Projects
   - Example: User interface design tools or applications requiring extensive user feedback.
   - Reason: The Spiral Model allows for the creation of prototypes and iterative testing, which is valuable for projects where user feedback is crucial for refining design.

Q6. a) Explain the different categories of Software Maintenance.

Answer:

Software is always changing and as long as it is being used, it has to be monitored and maintained properly. This is partly to adjust for the changes within an organization but is even more important because technology keeps changing.

Your software may need maintenance for any number of reasons to keep it up and running, to enhance features, to rework the system for changes into the future, to move to the Cloud, or any other changes. Whatever the motivation is for software maintenance, it is vital for the success of your business. As such, software maintenance is more than simply finding and fixing bugs. It is keeping the heart of your business up and running.

There are four types of software maintenance:

- Corrective Software Maintenance
- Adaptive Software Maintenance
- Perfective Software Maintenance
- Preventive Software Maintenance

Corrective Software Maintenance

Corrective software maintenance is what one would typically associate with the maintenance of any kind. Correct software maintenance addresses the errors and faults within software applications that could impact various parts of your software, including the design, logic, and code. These corrections usually come from bug reports that were created by users or customers- but corrective software maintenance can help to spot them before your customers do, which can help your brand's reputation.

Adaptive Software Maintenance

Adaptive software maintenance becomes important when the environment of your software changes. This can be brought on by changes to the operating system, hardware, software dependencies, Cloud storage, or even changes within the operating system. Sometimes, adaptive software maintenance reflects organizational policies or rules as well. Updating services, making modifications to vendors, or changing payment processors can all necessitate adaptive software maintenance.

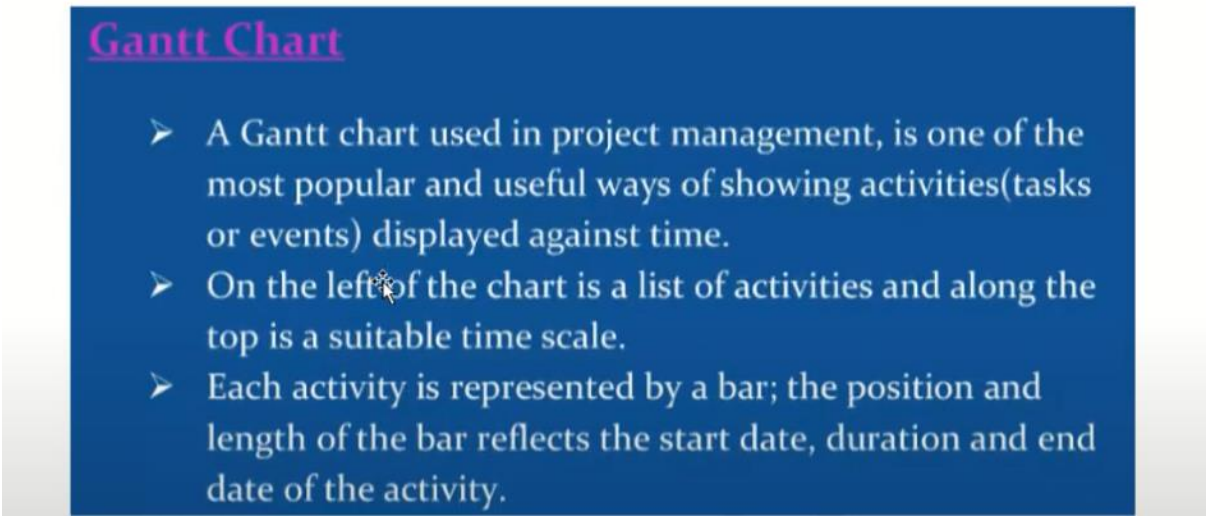
Perfective Software Maintenance

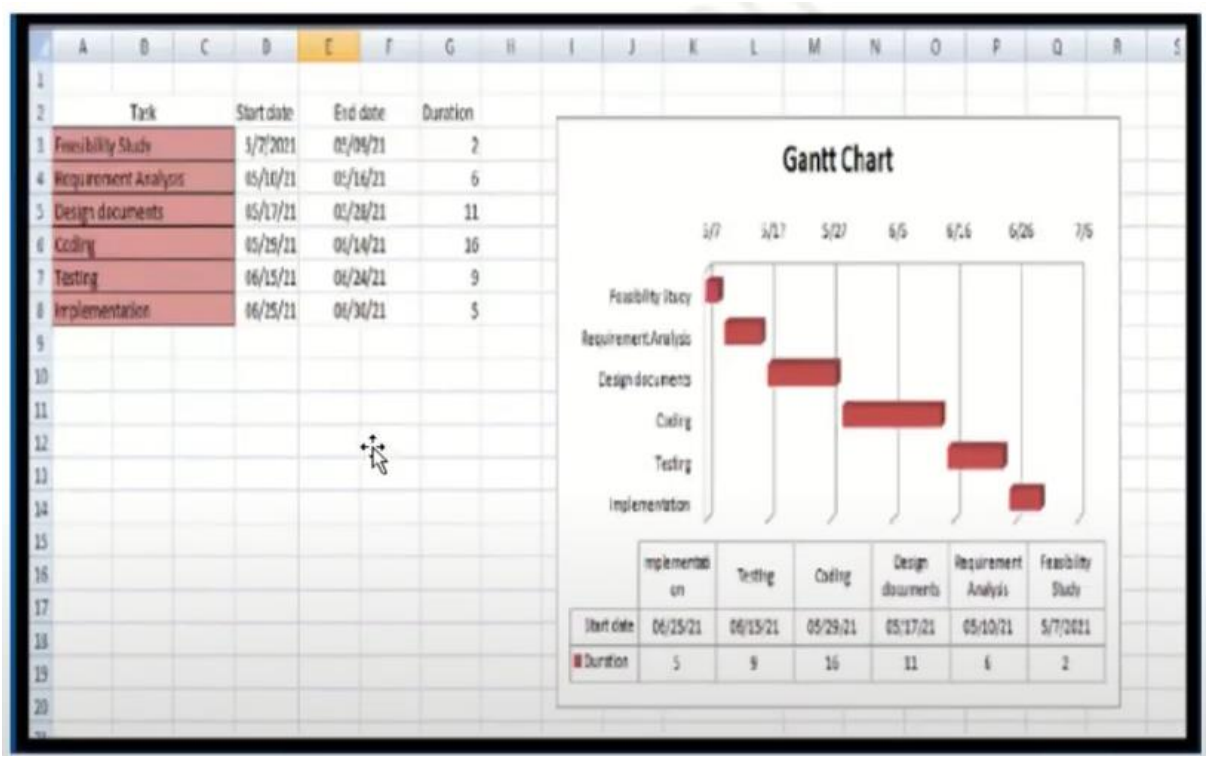
Perfective software maintenance focuses on the evolution of requirements and features that existing in your system. As users interact with your applications, they may notice things that you did not or suggest new features that they would like as part of the software, which could become future projects or enhancements. Perfective software maintenance takes over some of the work, both adding features that can enhance user experience and removing features that are not effective and functional. This can include features that are not used or those that do not help you to meet your end goals.

Preventive Software Maintenance

Preventative Software Maintenance helps to make changes and adaptations to your software so that it can work for a longer period of time. The focus of the type of maintenance is to prevent the deterioration of your software as it continues to adapt and change. These services can include optimizing code and updating documentation as needed.

b) Draw GANTT chart for the development of "Railway Reservation System". Briefly explain the chart with respect to Railway Reservation System





Q7. What is Software Configuration Management (SCM)? Explain the need of SCM with the help of an example.

Answer:

Software Configuration Management

When we develop software, the product (software) undergoes many changes in their maintenance phase; we need to handle these changes effectively.

Several individuals (programs) works together to achieve these common goals. This individual produces several work product (SC Items) e.g., Intermediate version of modules or test data used during debugging, parts of the final product.

The elements that comprise all information produced as a part of the software process are collectively called a software configuration.

As software development progresses, the number of Software Configuration elements (SCI's) grow rapidly.

Why need for System configuration management?

1. Replicability: Software version control (SCM) makes ensures that a software system can be replicated at any stage of its development. This is necessary for testing, debugging, and upholding consistent environments in production, testing, and development.
2. Identification of Configuration: Source code, documentation, and executable files are examples of configuration elements that SCM helps in locating and labeling. The management of a system's constituent parts and their interactions depend on this identification.
3. Effective Process of Development: By automating monotonous processes like managing dependencies, merging changes, and resolving disputes, SCM simplifies the development process. Error risk is decreased and efficiency is increased because of this automation.

Example of SCM Necessity:

Imagine a software development project where a team of developers is working on a web application. Each developer is assigned different features: Developer A is working on the user authentication module, Developer B is working on the payment gateway, and Developer C is focusing on the user interface enhancements.

Without SCM, coordinating these changes would be chaotic. Developer A and Developer B might inadvertently make conflicting changes to shared files, leading to integration problems.

Moreover, if Developer C introduces a bug in the user interface, it would be challenging to trace back and fix the issue without a detailed history of changes.

Using SCM tools like Git, each developer works in separate branches for their features. They can test their changes independently and only merge them into the main branch once they are stable and reviewed. If an issue arises, the team can review the history of changes, pinpoint the source of the problem, and roll back to a previous stable version if needed.

Q8. Write short notes on the following:

a. Object Oriented Metrics
b. Coupling

c. Software Quality Assurance
d. Capability Maturitys Model

Answer:

(a) Object-Oriented Metrics

Object-Oriented Metrics are measures used to evaluate the quality and complexity of object-oriented software systems. These metrics help in assessing various aspects of the system, such as design, maintainability, and performance. Common object-oriented metrics include:

- Complexity Metrics:
  - Cyclomatic Complexity: Measures the number of independent paths through a program's source code.
  - Depth of Inheritance Tree (DIT): Measures the depth of the inheritance hierarchy for a class. A deeper tree can indicate greater complexity and potential reuse.
  - Number of Children (NOC): Counts the number of immediate subclasses of a class. It provides insight into the class's influence in the hierarchy.
- Size Metrics:
  - Lines of Code (LOC): Measures the number of lines in the code. While not always indicative of complexity, it helps in understanding the scale of the codebase.
  - Number of Methods (NOM): Counts the methods in a class, which can indicate its size and potential for complexity. BO
- Coupling and Cohesion Metrics:
  - Coupling Between Objects (CBO): Measures the number of classes that are coupled with a given class. Lower coupling is generally preferred as it implies better modularity.
  - Lack of Cohesion in Methods (LCOM): Measures how well the methods of a class are related to each other. High LCOM can indicate that a class might be trying to do too many things.

These metrics assist in evaluating the design and functionality of object-oriented systems, guiding improvements and ensuring high-quality software.

1. **Large Enterprise Systems**
   - **Example**: Enterprise Resource Planning (ERP) systems.
   - **Reason**: These systems often involve complex requirements and integration with various business processes. The iterative nature of the Spiral Model helps manage the complexity and adapt to changing needs.
2. **Custom Software Development**
   - **Example**: Custom CRM (Customer Relationship Management) systems.
   - **Reason**: Custom solutions are tailored to specific business needs and require frequent feedback and adjustments from stakeholders. The Spiral Model's iterative approach allows for ongoing refinement.
3. **Research and Development Projects**
   - **Example**: Advanced research tools or experimental software solutions.
   - **Reason**: Research projects often involve uncertainty and evolving requirements. The Spiral Model provides a structured way to explore and refine new ideas.
4. **High-Risk Projects**
   - **Example**: Defense or aerospace software systems.
   - **Reason**: High-risk projects benefit from the Spiral Model's focus on risk analysis and mitigation. The model helps identify potential issues early and adapt to unforeseen challenges.
5. **Prototyping Projects**
   - **Example**: User interface design tools or applications requiring extensive user feedback.
   - **Reason**: The Spiral Model allows for the creation of prototypes and iterative testing, which is valuable for projects where user feedback is crucial for refining design.

(b) Coupling

Coupling refers to the degree of interdependence between software modules. In general, lower coupling is preferred as it indicates that modules are less dependent on each other, leading to a more modular and maintainable system. Key types of coupling include:

- Content Coupling: One module modifies or relies on the internal workings of another module. This is considered the worst form of coupling because it makes modules highly dependent on each other's implementation.
- Common Coupling: Modules share global data. Changes to the global data can affect all modules, making the system difficult to manage.
- External Coupling: Modules depend on external systems or data formats, such as databases or file systems.
- Control Coupling: One module controls the behavior of another by passing control information (e.g., flags).
- Data Coupling: Modules share data by passing data parameters, with no control information. This is considered a better practice as it limits the dependency to only necessary data.

- Message Coupling: Module Modules interact through well-defined interfaces minimizing dependencies.

Reducing coupling improves modularity, making the and test. e system easier easier to understand, maintain,

(c) Software Quality Assurance (SQA)

Software Quality Assurance (SQA) encompasses a range of activities designed to ensure that software meets specified requirements and quality standards. SQA involves:

- Process Assurance: Ensuring that the development processes are followed correctly and effectively. This includes adherence to development methodologies, standards, and best practices.
- Quality Control: Activities that monitor and evaluate the quality of the software during the development phase. This includes testing (unit, integration, system, acceptance), inspections, and reviews.
- Verification and Validation:
  - Verification: Ensuring the software meets design specifications and is built correctly.
  - Validation: Ensuring the software meets user requirements and provides the expected functionality.
- Continuous Improvement: Implementing feedback loops to continuously improve the processes and practices based on lessons learned from previous projects.

SQA aims to prevent defects in the software, improve development efficiency, and ensure the final product meets user expectations and regulatory requirements.

(d) Capability Maturity Model (CMM)

The Capability Maturity Model (CMM) is a framework used to improve and optimize software development processes. Developed by the Software Engineering Institute (SEI), it assesses an organization's maturity level in software development and provides a roadmap for process improvement. The model consists of five maturity levels:

1. Initial: Processes are ad hoc and chaotic. Success depends on individual effort rather than established practices.
2. Managed: Basic project management processes are established to track cost, schedule, and functionality. Processes are repeatable but still not fully standardized.
3. Defined: Processes are documented, standardized, and integrated into a cohesive whole. All projects follow these standard processes.
4. Quantitatively Managed: Processes are measured and controlled using quantitative data. Performance metrics are used to manage and improve process performance.
5. Optimizing: Focus is on continuous improvement of processes through incremental and innovative improvements. The organization actively seeks to improve performance through feedback and technological advances.