Biyani's Think Tank

Concept Based Notes

Software Engineering

BCA III Sem.

Ms. Neha Tiwari

Asst. Professor (Dept. of IT)

Biyani Girls College, Jaipur

BIYANI
Where you can trust

Published by :
Think Tanks
Biyani Group of Colleges

Concept & Copyright :
© **Biyani Shikshan Samiti**
Sector-3, Vidhyadhar Nagar,
Jaipur-302 023 (Rajasthan)
Ph : 0141-2338371, 2338591-95 Fax : 0141-2338007
E-mail : acad@biyanicolleges.org
Website :www.gurukpo.com; www.biyanicolleges.org

ISBN : 978-93-83462-35-3

Edition: 2025

Leaser Type Setted by :
Biyani College Printing Department

Preface

I am glad to present this book, especially designed to serve the needs of the students. The book has been written keeping in mind the general weakness in understanding the fundamental concepts of the topics. The book is self- explanatory and adopts the "Teach Yourself" style. It is based on question- answer pattern. The language of book is quite easy and understandable basedon scientific approach.

Any further improvement in the contents of the book by making corrections, omission and inclusion is keen to be achieved based on suggestions from the readers for which the author shall be obliged.

I acknowledge special thanks to Mr. Rajeev Biyani, *Chairman* & Dr. Sanjay Biyani, *Director* (*Acad.*) Biyani Group of Colleges, who are the backbones and main concept provider and also have been constant source of motivation throughout this Endeavour. They played an active role in coordinating the various stages of this Endeavour and spearheaded the publishing work.

I look forward to receiving valuable suggestions from professors of various educational institutions, other faculty members and students for improvement of the quality of the book. The reader may feel free to send in their comments and suggestions to the under mentioned address.

Author

Syllabus

Unit-I

Software Engineering Fundamentals: Software, Problem Domain, Software Engineering Challenges, Software Processes (processes, projects & products, component), Software Requirement Analysis & Specification. Software Development Process Models: Waterfall Model, Prototyping, Iterative Enhancement Model, Spiral Model. Introduction to Agile Model: Principles, Steps, Various Agile Process Models.

Unit-II

Software Project Planning: Cost Estimation- Uncertainties in Cost Estimation, Building Cost Estimation Models, On Size Estimation, COCOMO Model. Project Scheduling: Average Duration Estimation, Project Scheduling & Milestones. Quality Assurance Plans: Verification & Validation, Inspection & Reviews.

Unit-III

Design Engineering: Design Process & Design Quality, Design Concepts (abstraction, architecture, modularity, functional independence, refinement, and design classes), The Design Model (data design elements, architectural design elements, interface design elements, component-level design elements, deployment-level design elements). Testing Strategies & Tactics: A strategic approach to software testing, Strategic issues, Software testing fundamentals, Test characteristics, Test Strategies for conventional software: Unit Testing, Integration testing, Validation Testing, System testing, Black-Box testing, White Box testing.

Unit-IV

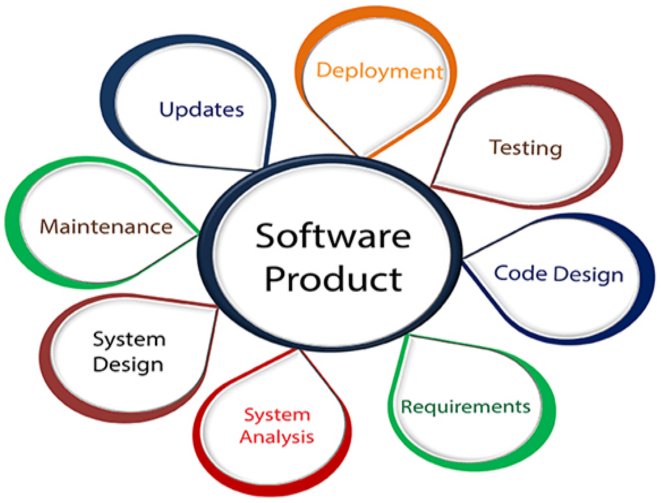
Software Reliability: Risk Management,Measures of Reliability & Availability, Software Safety. Maintenance and Reengineering: Introduction to: Software Maintenance, Software Supportability, Reengineering, Reverse Engineering, Restructuring, and Forward Engineering.

Chapter-1

Introduction to Software Engineering

Q1. What is Software Engineering?

The term Software engineering **is the product of two words,** software**, and** engineering**. The** software **is a collection of integrated programs. Software subsists of carefully-organized instructions and code written by developers on any of various particular computer languages. Computer programs and related documentation such as requirements, design models and user manuals.** Engineering **is the application of** scientific **and** practical **knowledge to** invent, design, build, maintain**, and** improve frameworks, processes, etc**.**



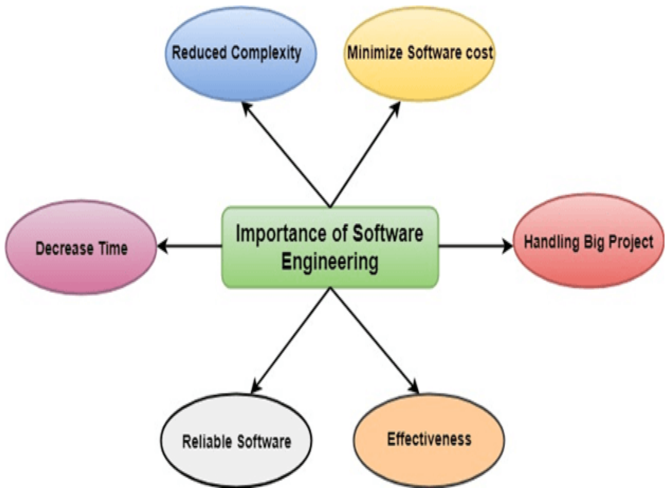
Q2. What are the needs of Software Engineering?

The necessity of software engineering appears because of a higher rate of progress in user requirements and the environment on which the program is working.

- **Huge Programming:** It is simpler to manufacture a wall than to a house or building, similarly, as the measure of programming become extensive engineering has to step to give it a scientific process.
- **Adaptability:** If the software procedure were not based on scientific and engineering ideas, it would be simpler to re-create new software than to scale an existing one.
- **Cost:** As the hardware industry has demonstrated its skills and huge manufacturing has let down the cost of computer and electronic hardware. But the cost of programming remains high if the proper process is not adapted.

- **Dynamic Nature:** The continually growing and adapting nature of programming hugely depends upon the environment in which the client works. If the quality of the software is continually changing, new upgrades need to be done in the existing one.
- **Quality Management:** Better procedure of software development provides a better and quality software product.

Q3. What are the importances of Software Engineering?



The importance of Software engineering is as follows:

1. **Reduces complexity:** Big software is always complicated and challenging to progress. Software engineering has a great solution to reduce the complication of any project. Software engineering divides big problems into various small issues. And then start solving each small issue one by one. All these small problems are solved independently to each other.
2. **To minimize software cost:** Software needs a lot of hardwork and software engineers are highly paid experts. A lot of manpower is required to develop software with a large number of codes. But in software engineering, programmers project everything and decrease all those things that are not needed. In turn, the cost for software productions becomes less as compared to any software that does not use software engineering method.
3. **To decrease time:** Anything that is not made according to the project always wastes time. And if you are making great software, then you may need to run many codes to get the definitive running code. This is a very time-consuming procedure, and if it is not well handled, then this can take a lot of time. So if you are making your software according to the software engineering method, then it will decrease a lot of time.
4. **Handling big projects:** Big projects are not done in a couple of days, and they need lots of patience, planning, and management. And to invest six and seven months of any company, it requires heaps of planning, direction, testing, and maintenance. No one can say that he has given four months of a company to the task, and the project is still in its first stage. Because the company has provided many resources to the plan and it should be completed. So to handle a big project without any problem, the company has to go for a software engineering method.

5. **Reliable software:** Software should be secure, means if you have delivered the software, then it should work for at least its given time or subscription. And if any bugs come in the software, the company is responsible for solving all these bugs. Because in software engineering, testing and maintenance are given, so there is no worry of its reliability.
6. **Effectiveness:** Effectiveness comes if anything has made according to the standards. Software standards are the big target of companies to make it more effective. So Software becomes more effective in the act with the help of software engineering.

Q4. What are the Software Processes?

The term **software** specifies to the set of computer programs, procedures and associated documents (Flowcharts, manuals, etc.) that describe the program and how they are to be used.

A software process is the set of activities and associated outcome that produce a software product. Software engineers mostly carry out these activities. These are four key process activities, which are common to all software processes. These activities are:

1. **Software specifications:** The functionality of the software and constraints on its operation must be defined.
2. **Software development:** The software to meet the requirement must be produced.
3. **Software validation:** The software must be validated to ensure that it does what the customer wants.
4. **Software evolution:** The software must evolve to meet changing client needs.

Chapter 2

Software Development Life cycle

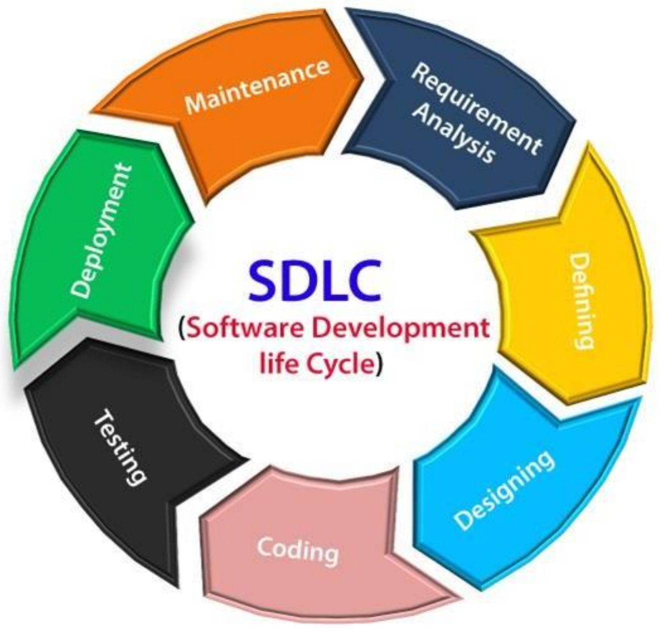
Q1.Describe the term of Software Development Life Cycle.

A software life cycle model (also termed process model) is a pictorial and diagrammatic representation of the software life cycle. A life cycle model represents all the methods required to make a software product transit through its life cycle stages. It also captures the structure in which these methods are to be undertaken.

In other words, a life cycle model maps the various activities performed on a software product from its inception to retirement. Different life cycle models may plan the necessary development activities to phases in different ways. Thus, no element which life cycle model is followed, the essential activities are contained in all life cycle models though the action may be carried out in distinct orders in different life cycle models. During any life cycle stage, more than one activity may also be carried out.

SDLC Cycle

SDLC Cycle represents the process of developing software. SDLC framework includes the following steps:



The stages of SDLC are as follows:

Stage1: Planning and requirement analysis

Requirement Analysis is the most important and necessary stage in SDLC.The senior members of the team perform it with inputs from all the stakeholders and domain experts or SMEs in the industry. Planning for the quality assurance requirements and identifications of the risks associated with the projects is also done at this stage.

Business analyst and Project organizer set up a meeting with the client to gather all the data like what the customer wants to build, who will be the end user, what is the objective of the product. Before creating a product, a core understanding or knowledge of the product is very necessary.

For Example, A client wants to have an application which concerns money transactions. In this method, the requirement has to be precise like what kind of operations will be done, how it will be done, in which currency it will be done, etc.Once the required function is done, an analysis is complete with auditing the feasibility of the growth of a product. In case of any ambiguity, a signal is set up for further discussion.

Once the requirement is understood, the SRS (Software Requirement Specification) document is created. The developers should thoroughly follow this document and also should be reviewed by the customer for future reference.

Stage2: Defining Requirements

Once the requirement analysis is done, the next stage is to certainly represent and document the software requirements and get them accepted from the project stakeholders.This is accomplished through "SRS"- Software Requirement Specification document which contains all the product requirements to be constructed and developed during the project life cycle.

Stage3: Designing the Software

The next phase is about to bring down all the knowledge of requirements, analysis, and design of the software project. This phase is the product of the last two, like inputs from the customer and requirement gathering.

Stage4: Developing the project

In this phase of SDLC, the actual development begins, and the programming is built. The implementation of design begins concerning writing code. Developers have to follow the coding guidelines described by their management and programming tools like compilers, interpreters, debuggers, etc. are used to develop and implement the code.

Stage5: Testing

After the code is generated, it is tested against the requirements to make sure that the products are solving the needs addressed and gathered during the requirements stage.During this stage, unit testing, integration testing, system testing, acceptance testing are done.

Stage6: Deployment

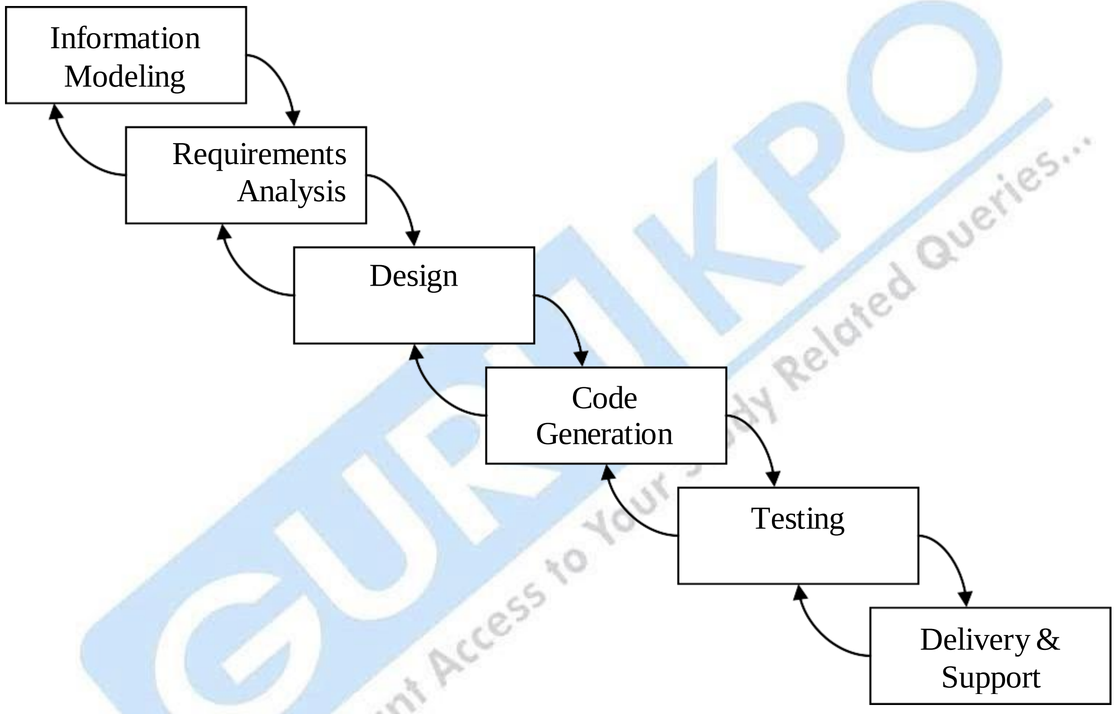
Once the software is certified, and no bugs or errors are stated, then it is deployed.Then based on the assessment, the software may be released as it is or with suggested enhancement in the object segment.After the software is deployed, then its maintenance begins.

Stage7: Maintenance

Once when the client starts using the developed systems, then the real issues come up and requirements to be solved from time to time.This procedure where the care is taken for the developed product is known as maintenance.

Q2. Describe the Waterfall Model / Linear Sequential Model.

Ans.: Sometimes called the ***classic life cycle or the linear sequential model,*** **the** ***waterfall model*** is a systematic, sequential approach to software development in whichdevelopment is seen as flowing downwards (like a waterfall) that begins at the system level and progresses through analysis, design, coding, testing and support. To follow the waterfall model, one proceeds from one phase to the next in a sequential manner. For example, one first completes "requirements specification". When the requirements are fully completed, one proceeds to design. The software is designed (on paper) and this design should be a plan for implementing the requirements given. When the design is fully completed, an implementation of that design, i.e. coding of the design is made by programmers. After the implementation phases are complete, the software product is tested and debugged; any faults introduced in earlier phases are removed here. Then the software product is installed, and later maintained to add any new functions that the user needs and remove bugs. Thus in a waterfall model, we can move to the next step only when the previous step is completed and removed of all errors. There is no jumping back and forth or overlap between the steps in a waterfall model.



The model consists of six distinct stages, namely :

(1) In the *Information Modelling* phase

(a) Work begins by gathering information related to the existing system. This will consists of all items consisting of hardware, people, databases etc.

(2) In the *requirements analysis* phase

(a) The problem is specified along with the desired objectives (goals).

(b) The constraints are identified.

(a) All information about the functions, behaviour, and performance are documented and checked by the customers.

(3) In the *design phase*, all inputs, computations and outputs of the system should be converted into a software model so that it can be coded by programmers. The hardware requirements are also determined at this stage along with a picture of the overall system architecture.

(4) In the *code generation* phase, the design has to be translated into a machine-readable form using any of the programming languages available that is suitable for the project.

(5) In the *testing* phase stage

(a) Once code is generated, testing begins.

(b) It focuses on all the statements of the software and removes all errors.

(c) It ensures that proper input will produce actual results.

(d) Detailed documentation from the design phase can significantlyreduce the coding effort.

(6) The *delivery and support* phase consists of delivering the final product to the customer and then taking care of the maintenance of the product. In this phase the software is updated to :

(a) Meet the changing customer needs

(b) Adapted to accommodate changes in the external environment

(c) Correct errors that were not previously known in the testing phases

(d) Enhancing the efficiency of the software

Q3.Explain the Prototyping Process Model.

Ans.: The prototyping model begins with the requirements gathering. The developer and the customer meet and define the objectives for the software, identify the needs, etc. A „ quick design‟ is then created. This design focuses on those aspects of the software that will be visible to the customer. It then leads to the construction of a prototype. The prototype is then checked by the customer and any modifications or changes that are required are made to the prototype. Looping takes place in this process and better versions of the prototype are created. These are continuously shown to the user so that any new changes can be updated in the prototype. This process continues till the user is satisfied with the system. Once a user is satisfied, the prototype is converted to the actual system with all considerations for quality and security.

The prototype is considered as the „first system‟. It is advantageous because both the customers and the developers get a feel of the actual system. But there are certain problems with the prototyping model too.

(1) The prototype is usually created without taking into consideration overall software quality.

(2) When the customer sees a working model in the form of a prototype,and then is told that the actual software is not created, the customer can get irritated.

(3) Since the prototype is to be created quickly, the developer will use whatever choices he has at that particular time (eg, he may not know a good programming language, but later may learn. He then cannot change the whole system for the new programming language). Thus the prototype may be created with less-than-ideal choices.

Q4. Describe the Rapid Application Development Model. State itsdisadvantages.

Ans.: Rapid Application Development (RAD) is an incremental software development process model that focuses on a very short development cycle. The RAD model is a „high-speed" version of the linear sequential model. It enables a development team to create a fully functional system within a very short time period (e.g. 60 to 90 days).

Business Modeling : The information flow among business functions is modeled in a way that answers the following questions :

What information drives the business process?
What information is generated?
Who generates it?

Where does the information go?
Who processes it?

Data Modeling : It gives all the details about what data is to be used in the project. All the information found in the business modeling phase is refined into a set of data objects and the characteristics and the relationships between these objects are defined.

Process Modeling : Here all the processes are defined that are needed to usethe data objects to create the system. Processing descriptions are created for adding, modifying, deleting, or retrieving a data object.

Application Generation : RAD makes use of the fourth generation techniques and tools like VB, VC++, Delphi etc rather than creating software using conventional third generation programming languages. The RAD reuses existing program components (when possible) or creates reusable components (when necessary). In all cases, automated tools (CASE tools) are used to facilitate construction of the software.

Testing and Turnover : Since the RAD process emphasizes reuse, many of the program components have already been tested. This minimizes the testing and development time.

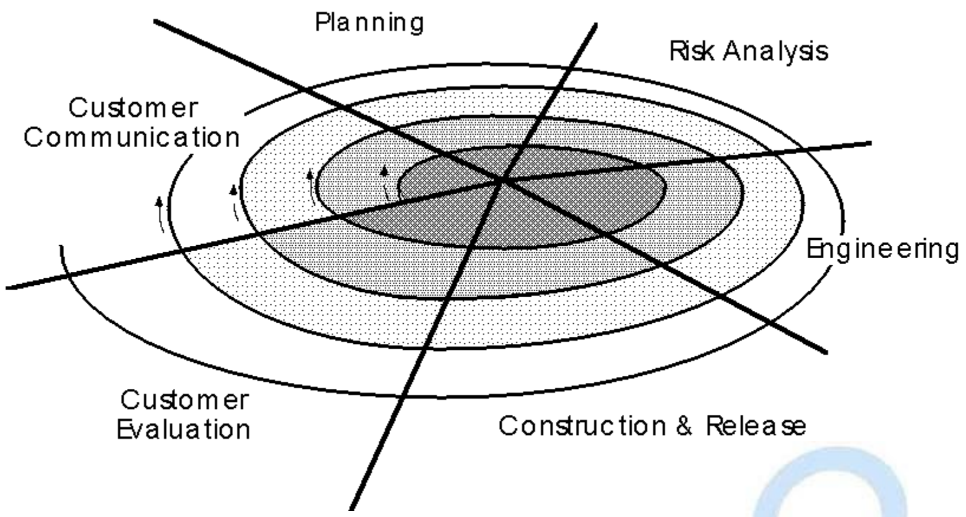
If a business application can be divided into modules, so that each major function can be completed within the development cycle, then it is a candidate for the RAD model. In this case, each team can be assigned a model, which is then integrated to form a whole.

Disadvantages :

- For Large projects, RAD requires sufficient resources to create the right number of RAD teams.
- If a system cannot be properly divided into modules, building components for RAD will be problematic
- RAD is not appropriate when technical risks are high, e.g. this occurs when anew application makes heavy use of new technology.

Q5. Explain the Spiral Model. What are the advantages of this model?

Ans.: The spiral model, combines the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model, therein providing the potential for rapid development of incremental versions of the software. In this model the software is developed in a series of incremental releases with the early stages being either paper models or prototypes. Later iterations become increasingly more complete versions of the product.



As illustrated, the model is divided into a number of task regions.

These regions are :

(1) The **customer communication** task – to establish effective communication between developer and customer.

(2) The **planning** task – to define resources, time lines and other project related information..

(3) The **risk analysis** task – to assess both technical and management risks.

(4) The **engineering** task – to build one or more representations (prototypes) of the application.

(5) The **construction and release** task – to construct, test, install and provide user support (e.g., documentation and training).

(6) The **customer evaluation** task – to obtain customer feedback based on the evaluation of the software representation created during the engineering stage and implemented during the install stage.

The evolutionary process begins at the centre position and moves in a clockwise direction. Each traversal of the spiral typically results in a deliverable. For example, the first and second spiral traversals may result in the production of a product specification and a prototype, respectively. Subsequent traversals may then produce more sophisticated versions of the software.

An important distinction between the spiral model and other software models is the explicit consideration of risk. There are no fixed phases such as specification or design phases in the model and it encompasses other process models. For example, prototyping may be used in one spiral to resolve requirement uncertainties and hence reduce risks. This may then be followed by a conventional waterfall development.

Advantages of the Spiral Model :

- The spiral model is a realistic approach to the development of large- scale software products because the software evolves as the process progresses. In addition, the developer and the client better understand and react to risks at each evolutionary level.
- The model uses prototyping as a risk reduction mechanism and allows for the development of prototypes at any stage of the evolutionary development.
- It maintains a systematic stepwise approach, like the classic life cycle model, but incorporates it into an iterative framework that more reflect the real world.
- If employed correctly, this model should reduce risks before they become problematic, as consideration of technical risks are considered at all stages.

Q6. What is Feasibility? Describe the different types of Feasibility.

Ans.: Feasibility is the determination of whether or not a project is worth doing. The process followed in making this determination is called feasibility study. A feasibility study is carried out to select the best system that meets performance requirements. When conducting feasibility study, an analyst can consider 7 types of feasibility:

- **Technical Feasibility :** It is concerned with specifying the equipment and the computer system that will satisfy and support the proposed user requirements. Here we need to consider the configuration of the system which tells the analyst how many work stations are required, how the units are interconnected so that they can operate and communicate smoothly.
- **Operation Feasibility :** It is related to human organizational aspects. The points to be considered here are – what changes will be brought with the system?, what new skills will be required?, do the existing staff members have these skills and can they be trained?
- **Economic Feasibility :** It is the most frequently used technique for evaluating a proposed system. It is also called Cost/Benefit Analysis. It is used to determine the benefits and savings that are expected from the proposed system and compare them with the costs. If benefits are more than the cost, the proposed system is given an OK.

- **Social Feasibility :** It is a determination of whether the proposed system will be acceptable to the people or not. It finds out the probability of the project being accepted by the group of people who are directly affected by the changed system.
- **Management Feasibility :** It is a determination of whether the proposed system is acceptable to the management of the organization. The project may be rejected, if the management does not accept the proposed system.
- **Legal Feasibility :** It is a determination of whether the proposed project is under legal obligation of known Acts, Statutes, etc.
- **Time Feasibility :** It is a determination of whether the project will be completed within a specified time period. If the project takes too much time, it is likely to be rejected.

Chapter-3

Software Requirement Analysis

Q1. What is Software Requirement Specification - [SRS]?

Ans A **Software requirements specification** (SRS) is a description of a software system to be developed. It lays out functional and non-functional requirements, and may include a set of use cases that describe user interactions that the software must provide.

Software requirements specification establishes the basis for an agreement between customers and contractors or suppliers (in market-driven projects, these roles may be played by the marketing and development divisions) onwhat the software product is to do as well as what it is not expected to do. Software requirements specification permits a rigorous assessment of requirements before design can begin and reduces later redesign. It should also provide a realistic basis for estimating product costs, risks, and schedules. Used appropriately, software requirements specifications can help prevent software project failure

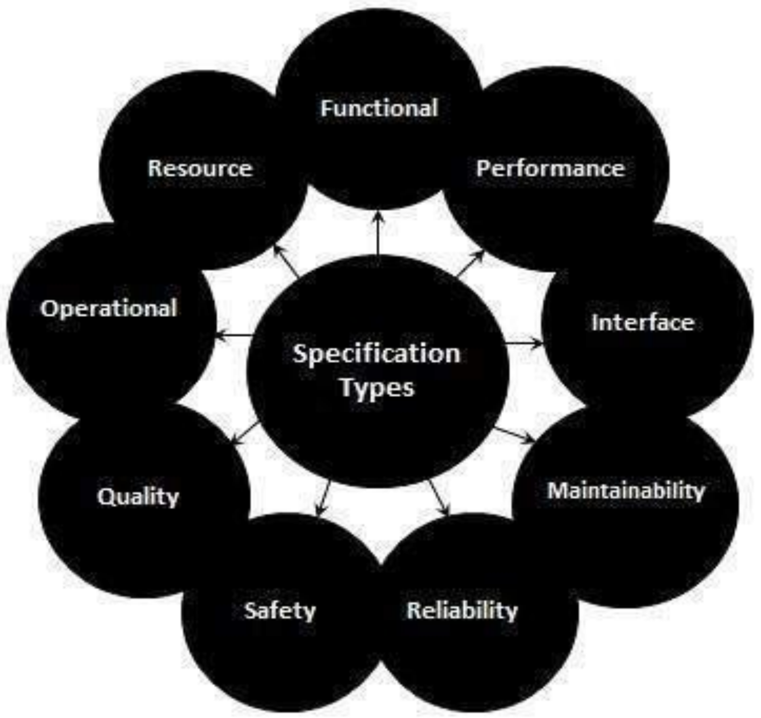
A software requirements specification (SRS) is a document that captures complete description about how the system is expected to perform. It is usually signed off at the end of requirements engineering phase.

Qualities of SRS:

- Correct
- Unambiguous
- Complete
- Consistent
- Ranked for importance and/or stability
- Verifiable
- Modifiable
- Traceable

Types of Requirements:

The below diagram depicts the various types of requirements that are capturedduring SRS.



Q2. What are the properties of a good SRS document.

Ans. Properties of a good SRS document. The essential properties of a good SRS document are the following:

Concise: The SRS report should be concise and at the same time, unambiguous, consistent, and complete. Verbose and irrelevant descriptions decrease readability and also increase error possibilities.

Structured: It should be well-structured. A well-structured document is simple to understand and modify. In practice, the SRS document undergoes several revisions to cope up with the user requirements. Often, user requirements evolve over a period of time. Therefore, to make the modifications to the SRS document easy, it is vital to make the report well-structured.

Black-box view: It should only define what the system should do and refrain from stating how to do these. This means that the SRS document should define the external behavior of the system and not discuss the implementation issues. The SRS report should view the system to be developed as a black box and should define the externally visible behavior of the system. For this reason, the SRS report is also known as the black-box specification of a system.

Conceptual integrity: It should show conceptual integrity so that the reader can merely understand it. Response to undesired events: It should characterize acceptable responses to unwanted events. These are called system response to exceptional conditions.

Verifiable: All requirements of the system, as documented in the SRS document, should be correct. This means that it should be possible to decide whether or not requirements have been met in an implementation.

Q3. Describe Data Flow Diagrams

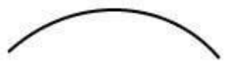
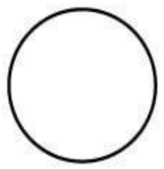
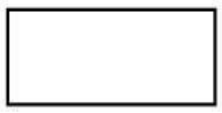
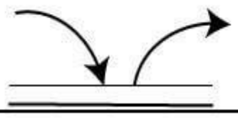
A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It can be manual, automated, or a combination of both.

It shows how data enters and leaves the system, what changes the information, and where data is stored.

The objective of a DFD is to show the scope and boundaries of a system as a whole. It may be used as a communication tool between a system analyst and any person who plays a part in the order that acts as a starting point for redesigning a system. The DFD is also called as a data flow graph or bubble chart.

The following observations about DFDs are essential:

1. All names should be unique. This makes it easier to refer to elements in the DFD.
2. Remember that DFD is not a flow chart. Arrows is a flow chart that represents the order of events; arrows in DFD represents flowing data. A DFD does not involve any order of events.
3. Suppress logical decisions. If we ever have the urge to draw a diamond-shaped box in a DFD, suppress that urge! A diamond-shaped box is used in flow charts to represents decision points with multiple exists paths of which the only one is taken. This implies an ordering of events, which makes no sense in a DFD.
4. Do not become bogged down with details. Defer error conditions and error handling until the end of the analysis.

Symbol	Name	Function
	Data flow	Used to Connect Processes to each , other , to sources or Sinks; te arrow head indicates direction of data flow.
	Process	Perfroms Some transformation of Input data to yield output data.
	Source of Sink (External Entity)	A Source of System inputs or Sink of System outputs.
	Data Store	A repository of data; the arrow heads indicate net inputs and net outputs to store.

Symbols for Data Flow Diagrams

Circle: A circle (bubble) shows a process that transforms data inputs into data outputs.

Data Flow: A curved line shows the flow of data into or out of a process or data store.

Data Store: A set of parallel lines shows a place for the collection of data items. A data store indicates that the data is stored which can be used at a later stage or by the other processes in a different order. The data store can have an element or group of elements.

Source or Sink: Source or Sink is an external entity and acts as a source of system inputs or sink of system outputs.

Levels in Data Flow Diagrams (DFD)

The DFD may be used to perform a system or software at any level of abstraction. Infact, DFDs may be partitioned into levels that represent increasing information flow and functional detail. Levels in DFD are numbered 0, 1, 2 or beyond. Here, we will see primarily three levels in the data flow diagram, which are: 0-level DFD, 1-level DFD, and 2-level DFD.

0-level DFDM

It is also known as fundamental system model, or context diagram represents the entire software requirement as a single bubble with input and output data denoted by incoming and outgoing arrows. Then the system is decomposed and described as a DFD with multiple bubbles. Parts of the system represented by each of these bubbles are then decomposed and documented as more and more detailed DFDs. This process may be repeated at as many levels as necessary until the program at hand is well understood. It is essential to preserve the number of inputs and outputs between levels, this concept is called leveling by DeMacro. Thus, if bubble "A" has two inputs x_1 and x_2 and one output y, then the expanded DFD, that represents "A" should have exactly two external inputs and one external output as shown in fig:

The Level-0 DFD, also called context diagram of the result management system is shown in fig. As the bubbles are decomposed into less and less abstract bubbles, the corresponding data flow may also be needed to be decomposed.

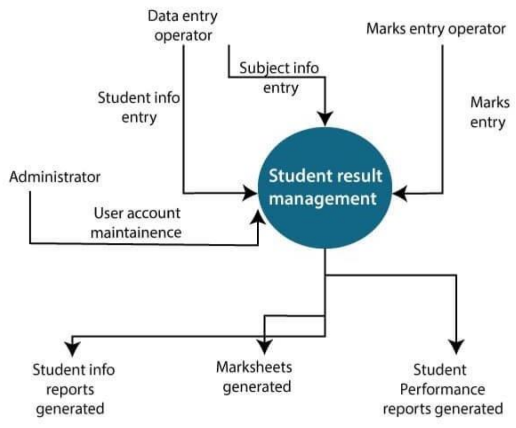


Fig: Level-0 DFD of result management system

1-level DFD

In 1-level DFD, a context diagram is decomposed into multiple bubbles/processes. In this level, we highlight the main objectives of the system and breakdown the high-level process of 0-level DFD into sub processes.

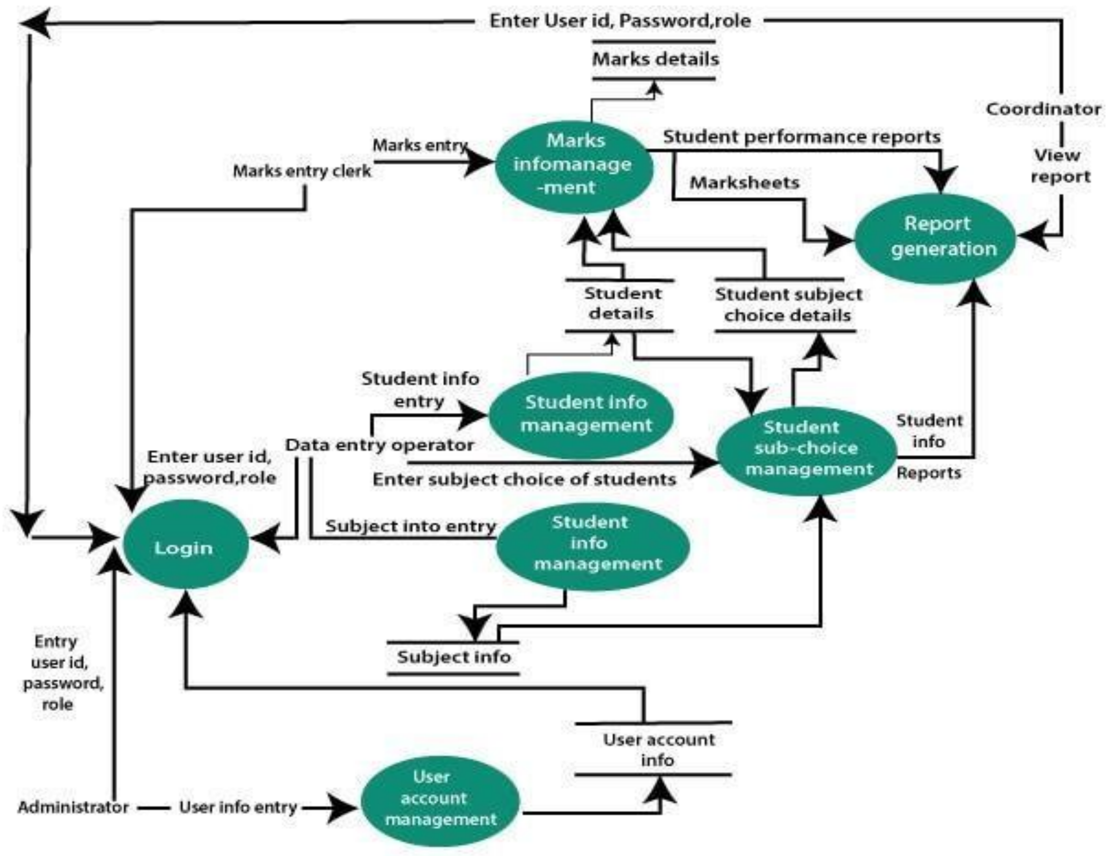
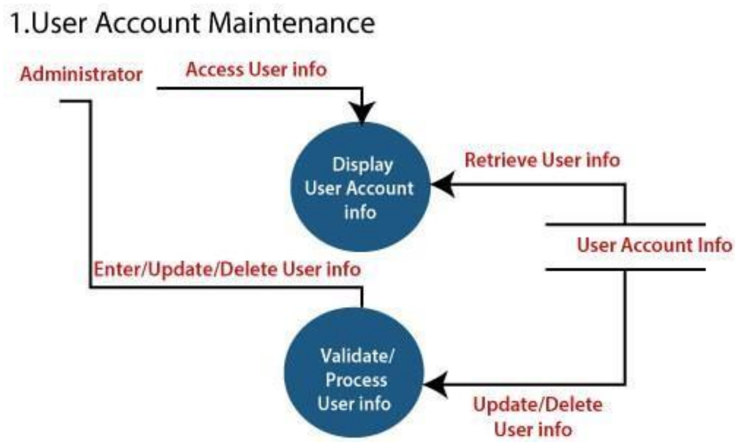


Fig: Level-1 DFD of result management system

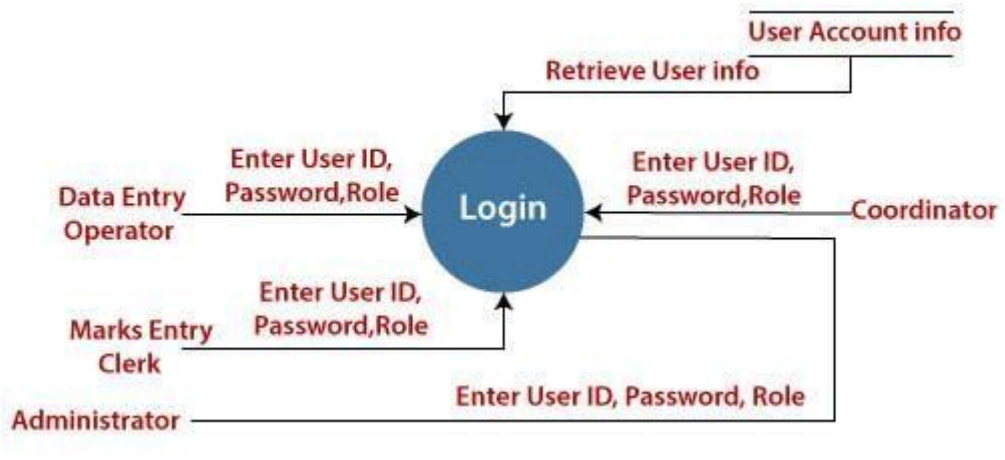
2-Level DFD

2-level DFD goes one process deeper into parts of 1-level DFD. It can be used to project or record the specific/necessary detail about the system's functioning.

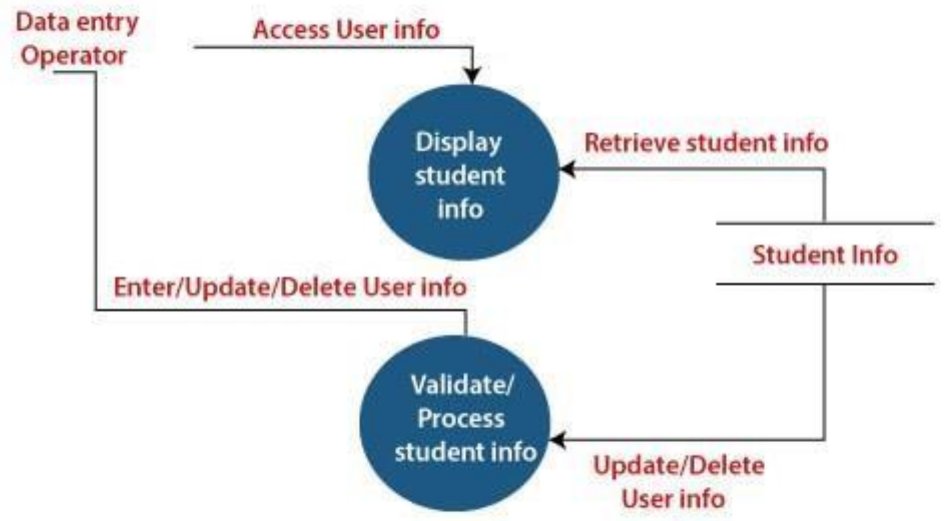

1.User Account Maintenance
Administrator
Access User info
Display User Account info
Retrieve User info
User Account Info
Enter/Update/Delete User info
Validate/ Process User info
Update/Delete User info

2. Login

The level 2 DFD of this process is given below:


User Account info
Retrieve User info
Enter User ID, Password,Role
Data Entry Operator
Login
Enter User ID, Password,Role
Coordinator
Enter User ID, Password,Role
Marks Entry Clerk
Enter User ID, Password, Role
Administrator

3. Student Information Management



Chapter-4

Software Project Planning

Q1. What is Software project planning. Describe the need of Project management?

A Software Project is the complete methodology of programming advancement from requirement gathering to testing and support, completed by the execution procedures, in a specified period to achieve intended software product.

Need of Software Project Management:-

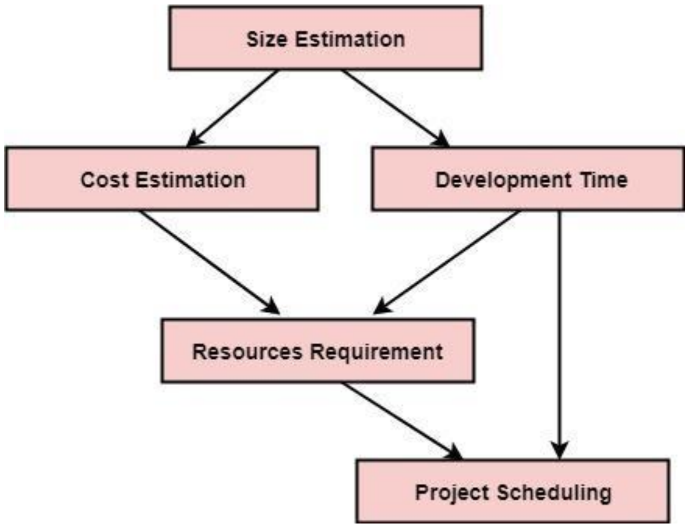
Software development is a sort of all new streams in world business, and there's next to no involvement in structure programming items. Most programming items are customized to accommodate customer's necessities. The most significant is that the underlying technology changes and advances so generally and rapidly that experience of one element may not be connected to the other one. All such business and ecological imperatives bring risk in software development; hence, it is fundamental to manage software projects efficiently.

Q2. What are the work of Software Project Manager?

Software manager is responsible for planning and scheduling project development. They manage the work to ensure that it is completed to the required standard. They monitor the progress to check that the event is on time and within budget. The project planning must incorporate the major issues like size & cost estimation scheduling, project monitoring, personnel selection evaluation & risk management. To plan a successful software project, we must understand:

- Scope of work to be completed
- Risk analysis
- The resources mandatory
- The project to be accomplished
- Record of being followed

Software Project planning starts before technical work start. The various steps of planning activities are:



Q3. Define the Software Cost Estimation.

For any new software project, it is necessary to know how much it will cost to develop and how much development time will it take. These estimates are needed before development is initiated, but how is this done? Several estimation procedures have been developed and are having the following attributes in common.

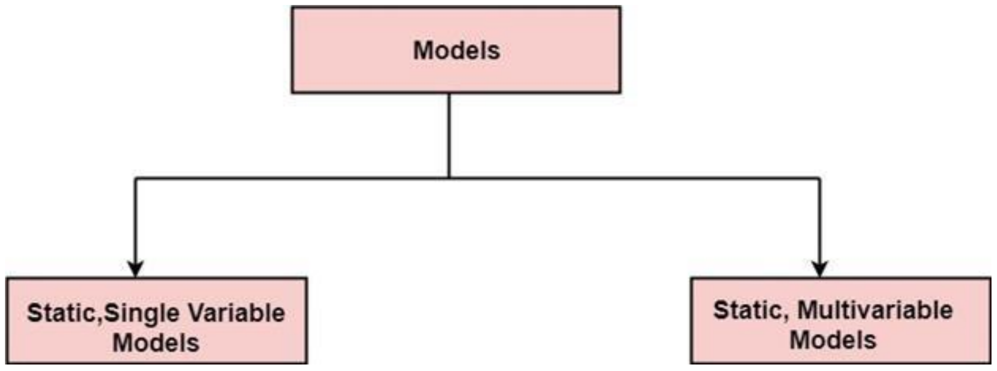
1. Project scope must be established in advanced.
2. Software metrics are used as a support from which evaluation is made.
3. The project is broken into small PCs which are estimated individually. To achieve true cost & schedule estimate, several option arise.
4. Delay estimation
5. Used symbol decomposition techniques to generate project cost and schedule estimates.
6. Acquire one or more automated estimation tools.

Uses of Cost Estimation

1. During the planning stage, one needs to choose how many engineers are required for the project and to develop a schedule.
2. In monitoring the project's progress, one needs to access whether the project is progressing according to the procedure and takes corrective action, if necessary.

Cost Estimation Models

A model may be static or dynamic. In a static model, a single variable is taken as a key element for calculating cost and time. In a dynamic model, all variable are interdependent, and there is no basic variable.



Static, Single Variable Models: When a model makes use of single variables to calculate desired values such as cost, time, efforts, etc. is said to be a single variable model. The most common equation is:

$C=aL^b$

Where C = Costs
L= size
a and b are constants

The Software Engineering Laboratory established a model called SEL model, for estimating its software production. This model is an example of the static, single variable model.

$E=1.4L^{0.93}$
$DOC=30.4L^{0.90}$
$D=4.6L^{0.26}$

Where E= Efforts (Person Per Month)
DOC=Documentation (Number of Pages)
D = Duration (D, in months)
L = Number of Lines per code

Static, Multivariable Models: These models are based on method (1), they depend on several variables describing various aspects of the software development environment. In some model, several variables are needed to describe the software development process, and selected equation combined these variables to give the estimate of time & cost. These models are called multivariable models.

WALSTON and FELIX develop the models at IBM provide the following equation gives a relationship between lines of source code and effort:

$$\mathbf{E=5.2L^{0.91}}$$

In the same manner duration of development is given by

$$\mathbf{D=4.1L^{0.36}}$$

The productivity index uses 29 variables which are found to be highly correlated productivity as follows:

$$\mathbf{I = \sum_{i=1}^{29} W_i X_i}$$

Where $\mathbf{W_i}$ is the weight factor for the $\mathbf{i^{th}}$variable and $\mathbf{X_i=\{-1,0,+1\}}$ the estimator gives $\mathbf{X_i}$one of the values **-1, 0 or +1** depending on the variable decreases, has no effect or increases the productivity.

Example: Compare the Walston-Felix Model with the SEL model on a software development expected to involve 8 person-years of effort.

a. Calculate the number of lines of source code that can be produced.
b. Calculate the duration of the development.
c. Calculate the productivity in LOC/PY
d. Calculate the average manning

Solution:

The amount of manpower involved = 8PY=96persons-months

(a) Number of lines of source code can be obtained by reversing equation to give:

$$L = \left(\frac{E}{a}\right) 1/b$$

Then

L (SEL) = (96/1.4)1⁄0.93=94264 LOC
L (SEL) = (96/5.2)1⁄0.91=24632 LOC

(b) Duration in months can be calculated by means of equation

D (SEL) = 4.6 (L) 0.26
= 4.6 (94.264)0.26 = 15 months
D (W-F) = 4.1 $L^{0.36}$
= 4.1 (24.632)0.36 = 13 months

(c) Productivity is the lines of code produced per persons/month (year)

$$P\ (SEL) = \frac{94264}{8} = 11783\ \frac{LOC}{Person}\ -Years$$

$$P\ (Years) = \frac{24632}{8} = 3079\ \frac{LOC}{Person}\ -Years$$

(d) Average manning is the average number of persons required per month in the project

$$M\ (SEL) = \frac{96P-M}{15M} = 6.4Persons$$

$$M\ (W\text{-}F) = \frac{96P-M}{13M} = 7.4Persons$$

Q4.What is a Data Dictionary? Give an example.

Ans.: A Data Dictionary (DD) is a structured repository of data about data. It is a set of accurate definitions of all DFD data elements and data structures. A data dictionary defines each term encountered during the analysis and designof a new system. Data dictionary is the place where we keep the details of the contents of data flows, data stores & processes.

Without a data dictionary the development of large systems becomes difficult. The data dictionary is an effective solution to the problem of complicated nature. The main purpose of a data dictionary is to provide a source of reference in which the analyst, the user, the designer can look up & find out its content and any other relevant information.

The main advantage of a DD is the documentation. It is a valuable reference to the organization which helps in communication between the analyst and the user. It is also important in building a database.

The Data Dictionary notations are

= is composed of

+ AND

( ) Optional value[

] Either/Or

{} iteration

** comment

@ identifier (key field)

separates alternative choices in the [] construct

Examples of Data dictionary –

Name = Courtesy-Title + First-Name + (Middle-Name) + Last-Name

Courtesy-Title = [Mr. | Miss | Mrs. | Ms. | Dr. | Prof.]

First-Name = { Legal-Character }

Last-Name = { Legal-Character }

Legal-Character = [A-Z | a-z |0-9| ' | - | |]

Q5.Briefly describe a Decision Tree with example.

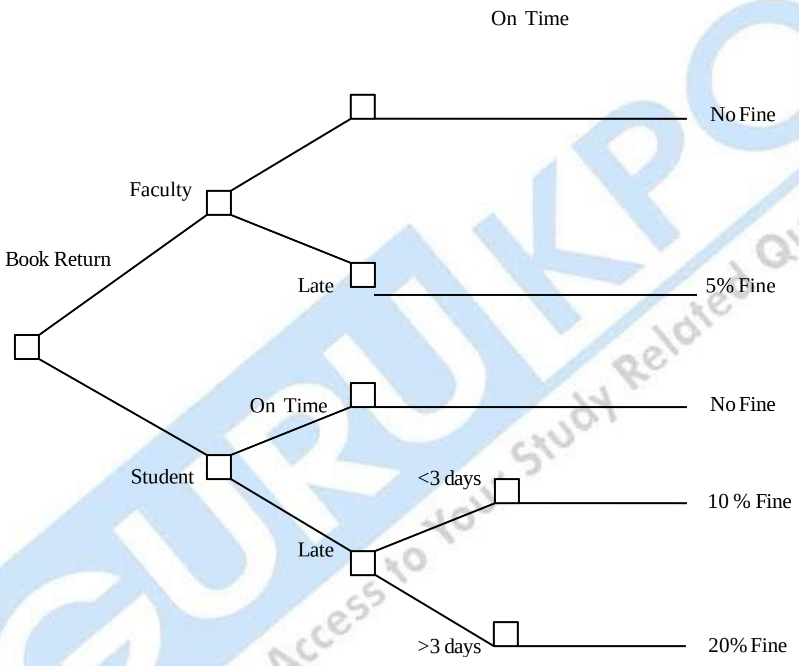
Ans.: Decision tree are graphical representation methods of representing a sequence of logical decisions. It is mainly used when decisions need to be taken or for defining policies. A decision tree has as many branches as there are logical alternatives. It is easy to construct, easy to read and easy to update. A decision tree is used to identify the strategy most likely to reach a goal. It isalso used as a means for calculating probabilities or making financial or number based decisions. A decision making tree is essentially a diagram that represents, in a specially organized way, the decisions, the main external or other events that introduce uncertainty, as well as possible outcomes of all those decisions and events.

Q6.How to draw a Decision Tree?

Ans.: You start a decision tree with a decision that needs to be made. This decision is represented by a small square towards the left of a large piece of paper. From this box draw out lines towards the right for each possible solution, and write that solution along the line. At the end of each solution line, considerthe results. If the result of taking that decision is uncertain, draw a small circle. If the result is another decision that needs to be made, draw another square. **Squares represent decisions; circles represent uncertainty or random factors.** Write the decision or factor to be considered above the square or circle. If you have completed the solution at the end of the line, just leave it blank. Starting from the new decision squares on your diagram, drawout lines representing the options that could be taken. From the circles, drawout lines representing possible outcomes. Again mark a brief note on the line saying what it means. Keep on doing this until you have drawn down as many of the possible outcomes and decisions as you can see leading on from your original decision.

Example : Book return policy in library

If a Faculty returns a book late, a fine of 5% of the book rate is charged. If a Student returns a book late by 3 days, fine is 10%, else 20% of book rate.



Q7. **What are Decision Tables? Explain with example.**

Ans.: Decision tables are a precise yet compact way to model complicated logic. Decision tables, like if-then-else and switch-case statements, associate conditions with actions to perform. But, unlike the control structures found in traditional programming languages, decision tables can associate many independent conditions with several actions in an elegant way. Decision tables are typically divided into four quadrants, as shown below.

The four quadrants	
Conditions	Condition alternatives
Actions	Action entries

Each decision corresponds to a variable, relation or predicate whose possible values are listed among the condition alternatives. Each action is a procedureor operation to perform, and the entries specify whether (or in what order) the action is to be performed for the set of condition alternatives the entrycorresponds to. Many decision tables include in their condition alternatives the **don't care** symbol, a hyphen. Using don't cares can simplify decision tables, especially when a given condition has little influence on the actions to be performed. In some cases, entire conditions thought to be important initially are found to be irrelevant when none of the conditions influence which actions are performed. The limited-entry decision table is the simplestto describe. The condition alternatives are simple boolean values, and the action entries are check-marks, representing which of the actions in a given column are to be performed.

A technical support company writes a decision table to diagnose printer problems based upon symptoms described to them over the phone from theirclients.

Printer troubleshooter									
		Rules							
Conditions	Printer does not print	Y	Y	Y	Y	N	N	N	N
	A red light is flashing	Y	Y	N	N	Y	Y	N	N
	Printer is unrecognized	Y	N	Y	N	Y	N	Y	N
Actions	Check the power cable			X					
	Check the printer-computer cable	X		X					

	Ensure printer software is installed	X		X		X		X	
	Check/replace ink	X	X			X	X		
	Check for paper jam		X		X				

Decision tables make it easy to observe that all possible conditions are accounted for. In the example above, every possible combination of the three conditions is given. In decision tables, when conditions are omitted, it is obvious even at a glance that logic is missing. Compare this to traditional control structures, where it is not easy to notice gaps in program logic with a mere glance --- sometimes it is difficult to follow which conditions correspond to which actions!

Just as decision tables make it easy to audit control logic, decision tables demand that a programmer think of all possible conditions. With traditional control structures, it is easy to forget about corner cases, especially when the else statement is optional. Since logic is so important to programming, decision tables are an excellent tool for designing control logic.

Chapter-5

Testing Strategies& Maintenance

Q1.Explain System Testing.

Ans.: Once source code has been generated, software must be tested to remove and correct as many errors as possible before delivery to the customer. The goal of system testing is to design a series of test cases that have a high likelihood of finding errors. Testing is the process of examining a product to determine what defects it contains. An information system is an integrated collection of software components. Components can be tested individually or in groups, orthe entire system can be tested as a whole. Testing is necessary for the successof the system. A small system error can explode into a much larger problem.

The proper choice of test data is as important as the test itself. If the test data that is inputted is not valid or according to the requirements, the reliability of the output will be low. Test data may be artificial or live. Artificial data is created only for testing purposes. Live data on the other hand, is taken from the users actual files. So there can be bias toward correct values. The design of tests for software products is also a very important topic. The designs may be White Box testing or Black Box testing.

Q2.What is Unit Testing?

Ans.: A strategy for software testing may be viewed as a spiral. Unit testing begins at the center of the spiral. Testing progresses by moving outward to integration testing, then towards validation testing and finally system testing.

Unit testing is the process of testing individual code modules before they are integrated with other modules. The unit being testing can be a function, subroutine, procedure or method. Units can also be very small groups of interrelated modules that are always executed as a group. The goal of unit testing is to identify and fix as many errors as possible before modules are combined into large units. Errors become more difficult and expensive to locate and fix when many modules are combined. Here the module interface is tested to see that information flows in and out of the program unit properly. It makes use of white box testing. Because a component is not a stand-alone program, a driver and/or stub software must be developed for each unit test. A driver is like a main program that accepts test case data, passes the data to the component and prints the results. A stub replaces modules that are subordinate the component to be tested. It uses the subordinate modules interface, does data manipulation, prints the result of entry and then returns control to the module undergoing the test.

Q3.Breifly describe what is Software Quality Assurance.

Ans.: Quality is a characteristic and attribute of something, which is measurable.There can be two types of quality: *quality of design* – it is the characteristics that the designers specify which will include the materials used, performance specifications, etc. and *quality of conformance* – which is the degree to whichthe design specifications are followed during the manufacturing process. Software Quality Assurance (SQA) consists of a means of monitoring the software engineering processes to ensure quality. It provides management with the data necessary to be informed about product quality. Software today is being developed in rapid speeds and this affects its quality. Software that is developed needs to meet certain standards for it to be certified and used by users. Software quality assurance is thus useful to keep the software development process in check and see that quality products are created forthe market. Just as a team of members that are used for the development process, a SQA group is a group that assists the software team in achieving a high quality end product.

The software life cycle includes various stages of development, and each stage has a goal of quality assurance. Several factors determine the quality ofa system. Among them are correctness, reliability, efficiency, usability, accuracy, etc. There are three levels of quality assurance: testing, validation and certification.

In system testing, the goal is to remove the errors in the software. This is extremely difficult and time-consuming. The system needs to be put through a "fail-test" so that we know what will make the system fail. A successful test is one that can uncover the errors so that the system can then be corrected toreach a good level of quality.

System validation checks the quality of the software in both simulated and live environments. First the software is passed through the simulated environment (not live) where the errors and failures are checked based on artificial data and user requirements. This is also known as *alpha testing*. The software is tested and verified and all changes are then made to the software. This modified software is them sent through the second phase that is the live environment. This is called *beta testing* where the software is sent to the user‟s site. Here the system will go through actual user data and requirements. After a scheduled time, failures and errors are documented and final correction and enhancements are made before the software is released for use.

The third level is to certify that the program or software package is correct and conforms to all standards. Nowadays, there is trend towards buying of ready-to-use software. So certification is of utmost importance. A package that is certified goes through a team of specialists who test, review, and determine how well it meets the vendor‟s claims. Certification is actually issued after the package passes the test.

Q4.Explain Software Maintenance. Describe its classification.

Ans.: The last part of the system development life cycle is system maintenance which is actually the implementation of the post-implementation plan. When systems are installed, they are generally used for long periods. This period ofuse brings with it the need to continually maintain the system. Maintenance accounts for 50-80% if the total system development. Maintenance is not as rewarding and exciting as developing systems.

Maintenance can be classified as :

(1) **Corrective :** It means repairing, processing or performance failures or making changes because of previously uncorrected problems.

(2) **Adaptive :** It means changing the program functions.

(3) **Perfective :** It means enhancing the performance or modifying the programs to respond to the users additional or changing needs

The greatest amount of time is spent on perfective. Maintenance covers a wide range of activities including correcting coding and design errors, updating documentation and test data.

Q5.Diffrence between validation & Verification?

Ans : Verification and Validation example is also given just below to this table.

Verification	**Validation**
1. Verification is a static practice of verifying documents, design, code and program.	1. Validation is a dynamic mechanism of validating and testing the actual product.
2. It does not involve executing the code.	2. It always involves executing the code.
3. It is human based checking of documents and files.	3. It is computer based execution of program.
4. Verification uses methods like inspections, reviews, walkthroughs, and Desk-checking etc.	4. Validation uses methods like black box (functional) testing, gray box testing, and white box (structural) testing etc.
5. **Verification** is to check whether the software conforms to specifications.	5. **Validation** is to check whether software meets the customer expectations and requirements.
6. It can catch errors that validation cannot catch. It is low level exercise.	6. It can catch errors that verification cannot catch. It is High Level Exercise.
7. Target is requirements specification, application and software architecture, high level, complete design, and database design etc.	7. Target is actual product-a unit, a module, a bent of integrated modules, and effective final product.
8. Verification is done by QA team to ensure that the software is as per the specifications in the SRS document.	8. Validation is carried out with the involvement of testing team.
9. It generally comes first-done before validation.	9. It generally follows after **verification**.

Q.6. Differences Between Black Box Testing and White Box Testing?

Ans The Differences Between Black Box Testing and White Box Testing are listed below.

Criteria	**Black Box Testing**	**White Box Testing**
Definition	Black Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is NOT known to the tester	White Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is known to the tester.
Levels Applicable To	Mainly applicable to higher levels of testing:Acceptance Testing System Testing	Mainly applicable to lower levels of testing: Unit Testing Integration Testing
Responsibility	Generally, independent Software Testers	Generally, Software Developers
Programming Knowledge	Not Required	Required
Implementation Knowledge	Not Required	Required
Basis for Test Cases	Requirement Specifications	Detail Design

Q7.What are the differences between Alpha Testing and Beta Testing?

Alpha testing is a type of acceptance testing, which is performed to identify all possible bugs/issues before releasing the product to the end-user. Alpha test is a preliminary software field test carried out by a team of users to find out the bugs that were not found previously by other tests. Alpha testing is to simulate a real user environment by carrying out tasks and operations that actual user might perform. Alpha testing implies a meeting with a software vendor and client to ensure that the developers appropriately meet the client's requirements in terms of the performance, functionality, and durability of the software.

Alpha testing needs lab environment, and usually, the testers are an internal employee of the organization. This testing is called alpha because it is done early on, near the end of the software development, but before beta testing.

Beta Testing is a type of acceptance testing; it is the final test before shipping a product to the customers. Beta testing of a product is implemented by "real users "of the software application in a "real environment." In this phase of testing, the software is released to a limited number of end-users of the product to obtain feedback on the product quality. It allows the real customers an opportunity to provide inputs into the design, functionality, and usability of the product. These inputs are essential for the success of the product. Beta testing reduces product failure risks and increases the quality of the product through customer validation. Direct feedback from customers is a significant advantage of beta testing. This testing helps to tests the software in a real environment. The experiences of the previous users are forwarded back to the developers who make final changes before releasing the software product.

KEY TERMS

Abstract Class	A class that has no direct instances, but whose descendants may have direct instances.
Abstract operation	Defines the form or protocol of the operation, but not its implementation.
Acceptance testing	The process whereby actual users test a completed information system, the end result of which is the users acceptance of the system.
Access method	An operating system algorithm for storing and locating data in secondary memory.
Action stubs	That part of a decision table that lists the actions that result for a given set of conditions.
Activation	The time period during which an object performs an operation.
Actor	An external entity that interacts with the system (similar to an external entity in data flow diagramming).
Adaptive maintenance	Changes made to a system to evolve its functionality to changing business needs or technologies.
Afferent module	A module of a structure chart related to input to the system.
Affinity clustering	The process of arranging planning matrix information so that clusters of information with some predetermined level or type of affinity are placed next to each other on a matrix report.
Aggregation	A part-of relationship between a component object and an aggregate object.
Alias	An alternative name given to an attribute.
Alpha testing	User testing of a completed information system using simulated data.
Analysis	The third phase of the SDLC in which the current system is studied and alternative replacement systems are proposed.
Analysis tools	CASE tools that enable automatic checking for incomplete, inconsistent, or incorrect specifications in diagrams, forms, and reports.
Anomalies	Errors or inconsistencies that may result when a user attempts to update a table that contains redundant data. There are three

	types of anomalies: insertion, deletion, and modification anomalies.
Application independence	The separation of data and the definition of data from the applications that use these data.
Application program interface (API)	Software which allows a specific front-end program development platform to communicate with a particular back-end database engine, even when the front-end and back-end were not built to be compatible.
Application server	A computing server where data analysis functions primarily reside.
Application software	Computer software designed to support organizational functions or processes.
Association	A relationship between object classes
Association class	An association that has attributes or operations of its own, or that participates in relationships with other classes.
Associationrole	The end of an association which connects it to a class.
Associative entity	An entity type that associates the instances of one or more entity types and contains attributes that are peculiar to the relationship between those entity instances. Also called a gerund.
Asynchronous message	A message in which the sender does not have to wait for the recipient to handle the message.
Attribute	A named property or characteristic of an entity that is of interest to the organization.
Audit trail	A list of changes to a data file which allows business transactions to be traced. Both the updating and use of data should be recorded in the audit trail, since the consequences of bad data should be discovered and corrected.
Authorization rules	Controls incorporated to restrict access to systems and data and also to restrict the actions that people may take once in the system.
Backward recovery (rollback)	An approach to rebuilding a file in which before images of changed records are restored to the file in reverse order until some earlier state is achieved.
Balancing	The conservation of inputs and outputs to a data flow diagram

	process when that process is decomposed to a lower level.
Baseline modules	Software modules that have been tested, documented, and approved to be included in the most recently created version of a system.
Baseline Project Plan	A major outcome and deliverable from the project initiation and planning phase which contains the best estimate of a projectâs scope, benefits, costs, risks, and resource requirements.
Batch processing	Information that is collected or generated at some predetermined time interval and can be accessed via hard copy or on-line devices.
Behavior	Represents how an object acts and reacts.
Beta testing	User testing of a completed information system using real data in the real user environment.
Binary relationship	A relationship between instances of two entity types. This is the most common type of relationship encountered in data modeling.
Biometric device	An instrument that detects personal characteristics such as fingerprints, voice prints, retina prints, or signature dynamics.
Blocking factor	The number of physical records per page.
Bottom-up planning	A generic information systems planning methodology that identifies and defines IS development projects based upon solving operational business problems or taking advantage of some business opportunities.
Boundary	The line that marks the inside and outside of a system and which sets off the system from its environment.
Build routines	Guidelines that list the instructions to construct an executable system from the baseline source code.
Business case	The justification for an information system, presented in terms of the tangible and intangible economic benefits and costs, and the technical and organizational feasibility of the proposed system.
Business Process Reengineering (BPR)	The search for, and implementation of, radical change in business processes to achieve breakthrough improvements in products and services.

Business rules	Specifications that preserve the integrity of a conceptual or logical data model.
Calculated field	A field which can be derived from other database fields. Also called computed or derived field.
Candidate key	An attribute (or combination of attributes) that uniquely. identifies each instance of an entity type.
Cardinality	The number of instances of entity B that can (or must) beassociated with each instance of entity A.
Central transform	The area of a transform-centered information system where the most important derivation of new information takes place.
Class diagram	Shows the static structure of an object-oriented model: the object classes, their internal structure, and the relationships in which they participate.
Class-scope attribute	An attribute of a class that specifies a value common to an entire class, rather than a specific value for an instance.
Client	The (front-end) portion of the client/server database system that provides the user interface and data manipulation functions.
Client/server architecture	A LAN-based computing environment in which a central database server or engine performs all database commands sent to it from client workstations, and application programs on each client concentrate on user interface functions.
Closed-ended questions	Questions in interviews and on questionnaires that ask those responding to choose from among a set of prespecified responses.
Closed system	A system that is cut off from its environment and does not interact with it.
Code generators	CASE tools that enable the automatic generation of program and database definition code directly from the design documents, diagrams, forms, and reports stored in the reposito
Cohesion	The extent to which a system or a subsystem performs a single function.
Command language interaction	A human-computer interaction method where users enter explicit statements into a system to invoke operations.

Competitive strategy	The method by which an organization attempts to achieve its mission and objectives.
Component	An irreducible part or aggregation of parts that make up a system, also called a subsystem.
Component diagram	Shows the software components or modules and their dependencies.
Composition	A part object that belongs to only one whole object and lives and dies with the whole.
Computer-aided software engineering (CASE)	Software tools that provide automated support for some portion of the systems development process.
Computing infrastructure	All the resources and practices required to help people adequately use computer systems to do their primary work.
Conceptual data model	A detailed model that captures the overall structure of organizational data while being independent of any database management system or other implementation considerations.
Concrete class	A class that can have direct instances.
Concurrency control	A method for preventing loss of data integrity due to interference between users in a multiuser environment.
Condition stubs	That part of a decision table that lists the conditions relevant to the decision.
Configuration management	The process of assuring that only authorized changes are made to a system.
Constraint	A limit to what a system can accomplish.
Constructor operation	An operation that creates a new instance of a class.
Context diagram	An overview of an organizational system that shows the system boundary, external entities that interact with the system, and the major information flows between the entities and the system.
Corporate strategic planning	An ongoing process that defines the mission, objectives, and strategies of an organization.
Corrective maintenance	Changes made to a system to repair flaws in its design, coding, or implementation.
Coupling	The extent to which subsystems depend on each other.

Critical path scheduling	A scheduling technique where the order and duration of a sequence of activities directly affect the completion date of a project.
Cross life cycle CASE	CASE tools designed to support activities that occur across multiple phases of the systems development life cycle.
Cross referencing	A feature performed by a data dictionary that enables one description of a data item to be stored and accessed by all individuals so that a single definition for a data item is established and used.
Data	Raw facts about people, objects, and events in an organization.
Data compression technique	Pattern matching and other methods which replace repeating strings of characters with codes of shorter length.
Data couple	A diagrammatic representation of the data exchanged between two modules in a structure chart.
Data dictionary	The repository of all data definitions for all organizational applications.
Data flow	Data in motion, moving from one place in a system to another.
Data flow diagram	A picture of the movement of data between external entities and the processes and data stores within a system.
Data-oriented approach	An overall strategy of information systems development that focuses on the ideal organization of data rather than where and how data are used.
Data store	Data at rest, which may take the form of many different physical representations.
Data type	A detailed coding scheme recognized by system software for representing organizational data.
Database	A shared collection of logically related data designed to meet the information needs of multiple users in an organization.
Database engine	The (back-end) portion of the client/server database system running on the server and providing database processing and shared access functions.
Database management system (DBMS)	Software that is used to create, maintain, and provide controlled access to user databases.

Decision support systems (DSS)	Computer-based systems designed to help organization members make decisions; usually composed of a database, model base, and dialogue system.
Decision table	A matrix representation of the logic of a decision, which specifies the possible conditions for the decision and the resulting actions.
Decision tree	A graphical representation of a decision situation in which decision points (nodes) are connected together by arcs (one for each alternative on a decision) and terminate in ovals (the action which is the result of all of the decisions made on the path that leads to that oval).
Default value	A value a field will assume unless an explicit value is entered for that field.
Degree	The number of entity types that participate in a relationship.
Design strategy	A high-level statement about the approach to developing an information system. It includes statements on the systemâs functionality, hardware and system software platform, and method for acquisition.
Desk checking	A testing technique in which the program code is sequentially executed manually by the reviewer.
DFD completeness	The extent to which all necessary components of a data flow diagram have been included and fully described.
DFD consistency	The extent to which information contained on one level of a set of nested data flow diagrams is also included on other levels.
Diagramming tools	CASE tools that support the creation of graphical representations of various system elements such as process flow, data relationships, and program structures.
Dialogue	The sequence of interaction between a user and a system.
Dialogue diagramming	A formal method for designing and representing human-computer dialogues using box and line diagrams.
Direct installation	Changing over from the old information system to a new one by turning off the old system when the new one is turned on.
Discount rate	The rate of return used to compute the present value of future cash flows.

Disruptive technologies	Technologies that enable the breaking of long-held business rules that inhibit organizations from making radical business changes.
Distributed database	A single logical database that is spread across computers in multiple locations which are connected by a data communications link.
Documentation	*See* External documentation, Internal documentation, System documentation, User documentation.
Documentation generators	CASE tools that enable the easy production of both technical and user documentation in standard formats.
Domain	The set of all data types and values that an attribute can assume.
Drop-down menu	A menu positioning method that places the access point of the menu near the top line of the display; when accessed, menus open by dropping down onto the display.
DSS generators	General purpose computer-based tools used to develop specific decision support systems.
Economic feasibility	A process of identifying the financial benefits and costs associated with a development project.
Efferent module	A module of a structure chart related to output from the system.
Electronic performance support system (EPSS)	Component of a software package or application in which training and educational information is embedded. An EPSS can take several forms, including a tutorial, an expert system shell, and hypertext jumps to reference material.
Encapsulation	The technique of hiding the internal implementation details of an object from its external view.
Encryption	The coding (or scrambling) of data so that they cannot be read by humans.
End users	Non-information-system professionals in an organization who specify the business requirements for and use software applications. End users often request new or modified applications, test and approve applications, and may serve on project teams as business experts.
End-user development	An approach to systems development in which users who are not computer experts satisfy their own computing needs

	through the use of high-level software and languages such as electronic spreadsheets and relational database management systems.
Entity instance (instance)	A single occurrence of an entity type.
Entity-relationship data model (E-R model)	A detailed, logical representation of the entities, associations, and data elements for an organization or business area.
Entity-relationshipÊdiagramÊ(E-RÊdiagram)	A graphical representation of an E-R model.
Entity type	A collection of entities that share common properties or characteristics.
Environment	Everything external to a system which interacts with the system.
Event	Something that takes place at a certain point in time; a noteworthy occurrence that triggers a state transition.
Exclusive relationships	A set of relationships for which an entity instance can participate in only one of the relationships at a time.
Executive support systems	Computer-based systems developed to support the information-intensive but limited-time decision making of executives (also referred to as executive information systems).
Expert systems	Computer-based systems designed to mimic the performance of human experts.
External documentation	System documentation that includes the outcome of structured diagramming techniques such as data flow and entity-relationship diagrams.
External information	Information that is collected from or created for individuals and groups external to an organization.
Feasibility	*See* Economic feasibility, Legal and contractual feasibility, Operational feasibility, Political feasibility, Schedule feasibility, Technical feasibility.
Field	The smallest unit of named application data recognized by system software.
File organization	A technique for physically arranging the records of a file on secondary storage devices.

File server	A device that manages file operations and is shared by each client PC attached to a LAN.
First normal form (1NF)	A relation that contains no repeating data.
Flag	A diagrammatic representation of a message passed between two modules.
Foreign key	An attribute that appears as a nonkey attribute in one relation and as a primary key attribute (or part of a primary key) in another relation.
Form	A business document that contains some pre-defined data and may include some areas where additional data are to be filled in. An instance of a form is typically based on one database record.
Form and report generators	CASE tools that support the creation of system forms and reports in order to prototype how systems will "look and feel" to users.
Form interaction	A highly intuitive human-computer interaction method whereby data fields are formatted in a manner similar to paper-based forms.
Formal system	The official way a system works as described in organizational documentation.
Forward recovery (roll forward)	An approach to rebuilding a file in which one starts with an earlier version of the file and either reruns prior transactions or replaces a record with its image after each transaction.
Functional decomposition	An iterative process of breaking the description of a system down into finer and finer detail which creates a set of charts in which one process on a given chart is explained in greater detail on another chart.
Functional dependency	A particular relationship between two attributes. For any relation R, attribute B is functionally dependent on attribute A if, for every valid instance of A, that value of A uniquely determines the value of B. The functional dependence of B on A is represented as A > B.
Gantt chart	A graphical representation of a project that shows each task activity as a horizontal bar whose length is proportional to its time for completion.
Hashed file organization	The address for each record is determined using a hashing

	algorithm.
Hashing algorithm	A routine that converts a primary key value into a relative record number (or relative file address).
Help desk	A single point of contact for all user inquiries and problems about a particular information system or for all users in a particular department.
Homonym	A single name that is used for two or more different attributes (for example, the term invoice to refer to both a customer invoice and a supplier invoice).
Horizontal partitioning	Distributing the rows of a table into several separate tables.
I-CASE	An automated systems development environment that provides numerous tools to create diagrams, forms, and reports; provides analysis, reporting, and code generation facilities; and seamlessly shares and integrates data across and between tools.
Icon	Graphical pictures that represent specific functions within a system.
Identifier	A candidate key that has been selected as the unique, identifying characteristic for an entity type.
Implementation	The sixth phase of the SDLC in which the information system is coded, tested, installed, and supported in the organization.
Incremental commitment	A strategy in systems analysis and design in which the project is reviewed after each phase and continuation of the project is rejustified in each of these reviews.
Index	A table or other data structure used to determine the location of rows in a file that satisfy some condition.
Indexed file organization	The records are either stored sequentially or non sequentially and an index is created that allows software to locate individual records.
Indifferent condition	In a decision table, a condition whose value does not affect which actions are taken for two or more rules.
Informal system	The way a system actually works.
Information	Data that have been processed and presented in a form suitable for human interpretation, often with the purpose of revealing trends or patterns.

Information center	An organizational unit whose mission is to support users in exploiting information technology.
Information repository	Automated tools to manage and control access to organizational business information and application portfolios as components within a comprehensive repository.
Information systems analysis and design	The complex organizational process whereby computer-based information systems are developed and maintained.
Information systems planning (ISP)	An orderly means of assessing the information needs of an organization and defining the systems, databases, and technologies that will best satisfy those needs.
Inheritance	The property that occurs when entity types or object classes are arranged in a hierarchy and each entity type or object class assumes the attributes and methods of its ancestors; that is, those higher up in the hierarchy. Inheritance allows new but related classes to be derived from existing classes.
Input	Whatever a system takes from its environment in order tofulfill its purpose.
Inspections	A testing technique in which participants examine program code for predictable language-specific errors.
Installation	The organizational process of changing over from the current information system to a new one.
Intangible benefit	A benefit derived from the creation of an information system that cannot be easily measured in dollars or with certainty. (6) *See also* Tangible benefit.
Intangible cost	A cost associated with an information system that cannot be easily measured in terms of dollars or with certainty.
Integration testing	The process of bringing together all of the modules that a program comprises for testing purposes. Modules are typically integrated in a top-down, incremental fashion.
Interface	In systems theory, the point of contact where a system meets its environment or where subsystems meet each other.
Internal documentation	System documentation that is part of the program source code or is generated at compile time.
Internal information	Information that is collected, generated, or consumed within an organization.

Interrelated components	Dependence of one subsystem on one or more subsystems.
JAD session leader	The trained individual who plans and leads Joint Application Design sessions.
Joint Application Design (JAD)	A structured process in which users, managers, and analysts work together for several days in a series of intensive meetings to specify or review system requirements.
Key business processes	The structured, measured set of activities designed to produce a specific output for a particular customer or market.
Knowledge engineersÊ	Computer professionals whose job it is to elicit knowledge from domain experts in order to develop expert systems. (Website)
Legal and contractual feasibility	The process of assessing potential legal and contractual ramifications due to the construction of a system.
Level-0 diagram	A data flow diagram that represents a systems major processes, data flows, and data stores at a high level of detail.
Level-n diagram	A DFD that is the result of n nested decompositions of a series of subprocesses from a process on a level-0 diagram.
Local area network (LAN)	The cabling, hardware, and software used to connect workstations, computers, and file servers located in a confined geographical area (typically within one building or campus).
Location transparency	A design goal for a distributed database which says that a user (or user program) requesting data need not know at which site those data are located.
Logical database model	A description of data using a notation that corresponds to an organization of data used by database management systems.
Logical design	The fourth phase of the SDLC in which all functional features of the system chosen for development in analysis are described independently of any computer platform.
Logical system description	Description of a system that focuses on the systems function and purpose without regard to how the system will be physically implemented.
Lower CASE	CASE tools designed to support the implementation and maintenance phases of the systems development life cycle.

Maintainability	The ease with which software can be understood, corrected, adapted, and enhanced.
Maintenance	The final phase of the SDLC in which an information system is systematically repaired and improved; or changes made to a system to fix or enhance its functionality.
Management information systems (MIS)	Computer-based systems designed to provide standard reports for managers about transaction data.
Mean time between failures (MTBF)	A measurement of error occurrences that can be tracked over time to indicate the quality of a system.
Menu interaction	A human-computer interaction method where a list of system options is provided and a specific command is invoked by user selection of a menu option.
Method	The implementation of an operation.
Middleware	A combination of hardware, software, and communication technologies that bring together data management, presentation, and analysis into a three-tiered client/server environment.
Mission statement	A statement that makes it clear what business a company is in.
Modularity	Dividing a system up into chunks or modules of a relatively uniform size.
Module	A self-contained component of a system, defined by function.
	Shows that an object is an instance of more than one class
Multivalued attribute	An attribute that may take on more than one value for each entity instance.
Natural language interaction	A human-computer interaction method where inputs to and outputs from a computer-based application are in a conventional speaking language such as English.
Normal form	A state of a relation that can be determined by applying simple rules regarding dependencies to that relation.
Normalization	The process of converting com-plex data structures into simple, stable data structures.

Null value	A special field value, distinct from 0, blank, or any other value, that indicates that the value for the field is missing or otherwise unknown.
Object	An entity that has a well-defined role in the application domain and has state, behavior, and identity.
Object-based interaction	A human-computer interaction method where symbols are used to represent commands or functions.
Object class (class)	A set of objects that share a common structure and a common behavior.
Object diagram	A graph of instances that are compatible with a given class diagram.
Object-oriented analysis and design (OOAD)	Systems development methodologies and techniques based on objects rather than data or processes.
Objective statements	A series of statements that express organizations qualitative and quantitative goals for reaching a desired future position.
On-line processing	The collection and delivery of the most recent available information, typically through an on-line workstation. (14)
One-time cost	A cost associated with project start-up and development, or system start-up. (6)
Open-ended questions	Questions in interviews and on questionnaires that have no prespecified answers.
Open system	A system that interacts freely with its environment, taking input and returning output.
Operation	A function or a service that is provided by all the instances of a class.
Operational feasibility	The process of assessing the degree to which a proposed system solves business problems or takes advantage of business opportunities.
Output	Whatever a system returns to its environment in order to fulfill its purpose.
Outsourcing	The practice of turning over responsibility of some to all of an organizationâs information systems applications and operations to an outside firm.
Overriding	The process of replacing a method inherited from a super class by a more specific implementation of that method in a

	subclass.
Package	A set of cohesive, tightly coupled classes representing a subsystem.
Page	The amount of data read or written in one secondary memory (disk) input or output operation. For I/O with a magnetic tape, the equivalent term is record block.
Parallel installation	Running the old information system and the new one at the same time until management decides the old system can be turned off.
Partial functional dependency	A dependency in which one or more nonkey attributes are functionally dependent on part, but not all, of the primary key.
Participatory Design (PD)	A systems development approach that originated in Northern Europe in which users and the improvement in their work lives are the central focus.
Perfective maintenance	Changes made to a system to add new features or to improve performance.
PERT chart	A diagram that depicts project activities and their inter-relationships. PERT stands for Program Evaluation Review Technique.
Phased installation	Changing from the old information system to the new one incrementally, starting with one or a few functional components and then gradually extending the installation to cover the whole new system.
Physical design	The fifth phase of the SDLC in which the logical specifications of the system from logical design are transformed into technology-specific details from which all programming and system construction can be accomplished.
Physical file	A named set of contiguous records.
Physical record	A group of fields stored in adjacent memory locations and retrieved together as a unit.
Physical system description	Description of a system that focuses on how the system will be materially constructed.
Picture (or template)	A pattern of codes that restricts the width and possible values for each position of a field.

Pointer	A field of data that can be used to locate a related field or record of data.
Political feasibility	The process of evaluating how key stakeholders within the organization view the proposed system.
Polymorphism	The same operation may apply to two or more classes in different ways.
Pop-up menu	A menu positioning method that places a menu near the current cursor position.
Present value	The current value of a future cash flow.
Preventive maintenance	Changes made to a system to avoid possible future problems.
Primitive DFD	The lowest level of decomposition for a data flow diagram.
Process	The work or actions performed on data so that they are transformed, stored, or distributed.
Process-oriented approach	An overall strategy to information systems development that focuses on how and when data are moved through and changed by an information system.
Processing logic	The steps by which data are transformed or moved and a description of the events that trigger these steps.
Project	A planned undertaking of related activities to reach an objective that has a beginning and an end.
Project close-down	The final phase of the project management process that focuses on bringing a project to an end.
Project execution	The third phase of the project management process in which the plans created in the prior phases (project initiation and planning) are put into action.
Project identification and selection	The first phase of the SDLC in which an organizations total information system needs are identified, analyzed, prioritized, and arranged.
Project initiation	The first phase of the project management process in which activities are performed to assess the size, scope, and complexity of the project and to establish procedures to support later project activities.
Project initiation and planning	The second phase of the SDLC in which a potential information systems project is explained and an argument for

	continuing or not continuing with the project is presented; a detailed plan is also developed for conducting the remaining phases of the SDLC for the proposed system.
Project management	A controlled process of initiating, planning, executing, and closing down a project.
Project manager	An individual with a diverse set of skills--management, leadership, technical, conflict management, and customer relationship--who is responsible for initiating, planning, executing, and closing down a project.
Project planning	The second phase of the project management process which focuses on defining clear, discrete activities and the work needed to complete each activity within a single project.
Project workbook	An on-line or hard copy repository for all project correspondence, inputs, outputs, deliverables, procedures, and standards that is used for performing project audits, orientation of new team members, communication with management and customers, scoping future projects, and performing post-project reviews.
Prototyping	An iterative process of systems development in which requirements are converted to a working system which is continually revised through close work between an analyst and users.
Pseudocode	A method for representing the instructions in a module with language very similar to computer programming code.
Purpose	The overall goal or function of a system.
Query operation	An operation that accesses the state of an object but does not alter the state.
Rapid Application Development (RAD)	Systems development methodology created to radically decrease the time needed to design and implement information systems. RAD relies on extensive user involvement, Joint Application Design sessions, prototyping, integrated CASE tools, and code generators.
Record partitioning	The process of splitting logical records into separate physical segments based on affinity of use.
Recurring cost	A cost resulting from the ongoing evolution and use of a system.
Recursive foreign key	A foreign key in a relation that references the primary key

	values of that same relation.
Reengineering	Automated tools that read program source code as input, perform an analysis of the programs data and logic, and then automatically, or interactively with a systems analyst, alter an existing system in an effort to improve its quality or performance.
Referential integrity	An integrity constraint specifying that the value (or existence) of an attribute in one relation depends on the value (or existence) of an attribute in the same or another relation.
Relation	A named, two-dimensional table of data. Each relation consists of a set of named columns and an arbitrary number of unnamed rows.
Relational database model	A data model that represents data in the form of tables or relations.
Relationship	An association between the instances of one or more entity types that is of interest to the organization.
Repeating group	A set of two or more multi valued attributes that are logically related.
Report	A business document that contains only pre-defined data; that is, it is a passive document used solely for reading or viewing. A report typically contains data from many unrelated records or transactions.
Repository	A centralized database that contains all diagrams, form and report definitions, data structure, data definitions, process flows and logic, and definitions of other organizational and system components; it provides a set of mechanisms and structures to achieve seamless data-to-tool and data-to-data integration.
Resource	Any person, group of people, piece of equipment, or material used in accomplishing an activity.
Reusability	The ability to design software modules in a manner so that they can be used again and again in different systems without significant modification.
Reverse engineering	Automated tools that read program source code as input and create graphical and textual representations of program design-level information such as program control structures, data structures, logical flow, and data flow.

Rules	That part of a decision table that specifies which actions are to be followed for a given set of conditions.
Schedule feasibility	The process of assessing the degree to which the potential timeframe and completion dates for all major activities within a project meet organizational deadlines and constraints for affecting change.
Scribe	The person who makes detailed notes of the happenings at a Joint Application Design session.
Second normal form (2NF)	A relation is in second normal form if it is in first normal form and every non key attribute is fully functionally dependent on the primary key. Thus no non key attribute is functionally dependent on part (but not all) of the primary key.
Secondary key	One or a combination of fields for which more than one record may have the same combination of values.
Sequence diagram	Depicts the interactions among objects during a certain period of time.
Sequential file organization	The records in the file are stored in sequence according to a primary key value.
Single location installation	Trying out a new information system at one site and using the experience to decide if and how the new system should be deployed throughout the organization.
Slack time	The amount of time that an activity can be delayed without delaying the project.
Smart card	A thin plastic card the size of a credit card with an embedded microprocessor and memory.
Source/sink	The origin and/or destination of data, sometimes referred to as external entities.
Stakeholder	A person who has an interest in an existing or new information system. A stakeholder is someone who is involved in the development of a system, in the use of a system, or someone who has authority over the parts of the organization affected by the system.
State	Encompasses an objects properties (attributes and relationships) and the values those properties have.

State diagram	A model of the states of an object and the events that cause the object to change from one state to another.
State transition	Changes in the attributes of an object or in the links an object has with other objects.
Statement of Work (SOW)	Document prepared for the customer during project initiation and planning that describes what the project will deliver and outlines generally at a high level all work required to complete the project.
Structure chart	Hierarchical diagram that shows how an information system is organized.
Structured English	Modified form of the English language used to specify the logic of information system processes. Although there is no single standard, Structured English typically relies on action verbs and noun phrases and contains no adjectives or adverbs.
Stub testing	A technique used in testing modules, especially where modules are written and tested in a top-down fashion, where a few lines of code are used to substitute for subordinate modules.
Support	Providing ongoing educational and problem solving assistance to information system users. For in-house developed systems, support materials and jobs will have to be prepared or designed as part of the implementation process.
Synchronous message	A type of message in which the caller has to wait for the receiving object to finish executing the called operation before it can resume execution itself.
Synonyms	Two different names that are used to refer to the same data item (for example, car and automobile).
System	An inter-related set of components, with an identifiable boundary, working together for some purpose.
System documentation	Detailed information about a systems design specifications, its internal workings, and its functionality.
System librarian	A person responsible for controlling the checking-out and checking-in of baseline modules for a system when a system is being developed or maintained.
System testing	The bringing together of all the programs that a system comprises for testing purposes. Programs are typically integrated in a top-down, incremental fashion.

Systems analyst	The organizational role most responsible for the analysis and design of information systems.
Systems development life cycle (SDLC)	The traditional methodology used to develop, maintain, and replace information systems.
Systems development methodology	A standard process followed in an organization to conduct all the steps necessary to analyze, design, implement, and maintain information systems.
Tangible benefit	A benefit derived from the creation of an information system that can be measured in dollars and with certainty.
Tangible cost	A cost associated with an information system that can be measured in terms of dollars and with certainty.
Technical feasibility	A process of assessing the development organizations ability to construct a proposed system.
Ternary relationship	A simultaneous relationship among instances of three entity types.
Third normal form (3NF)	A relation is in third normal form if it is in second normal form and no transitive dependencies exist.
Three-tiered client/server	Advanced client/server architectures in which there are three logical and distinct applications--data management, presentation, and analysis--which are combined to create a single information system.
Top-down planning	A generic information systems planning methodology that attempts to gain a broad understanding of the information system needs of the entire organization.
Transaction analysis	The process of turning data flow diagrams of a transaction-centered system into structure charts.
Transaction-centered system	An information system that has as its focus the dispatch of data to their appropriate locations for processing.
Transaction processing systems (TPS)	Computer-based versions of manual organization systems dedicated to handling the organizations transactions; e.g., payroll.
Transactions	Individual, simple events in the life of an organization that contain data about organizational activity.
Transform analysis	The process of turning data flow diagrams of a transform-centered system into structure charts.

Transform-centered system	An information system that has as its focus the derivation of new information from existing data.
Transitive dependency	A functional dependency between two (or more) non key attributes in a relation.
Triggering operation (trigger)	An assertion or rule that governs the validity of data manipulation operations such as insert, update, and delete.
Turnaround document	Information that is delivered to an external customer as an output that can be returned to provide new information as an input to an information system.
Unary relationship (recursive relationship)	A relationship between the instances of one entity type.
Unit testing	Method in which each module is tested alone in an attempt to discover any errors in its code.
Update operation	An operation that alters the state of an object.
Upper CASE	CASE tools designed to support information planning and the project identification and selection, project initiation and planning, analysis, and design phases of the systems development life cycle.
Usability	An overall evaluation of how a system performs in supporting a particular user for a particular task.
Use case	A complete sequence of related actions initiated by an actor, it represents a specific way of using the system.
Use-case diagram	A diagram that depicts the use cases and actors for a system.
User documentation	Written or other visual information about an application system, how it works, and how to use it.
Value chain analysis	The process of analyzing an organizations activities to determine where value is added to products and/or services and the cost are incurred for doing so; usually also includes a comparison with the activities, added value, and costs of other organizations for the purpose of making improvements in the organizations operations and performance.
Vertical partitioning	Distributing the columns of a table into several separate tables.
View	A subset of the database that is presented to one or more users.

Walkthrough	A peer group review of any product created during the systems development process. Also called structured walkthrough.
Well-structured relation	A relation that contains a minimum amount of redundancy and allows users to insert, modify, and delete the rows in a table without errors or inconsistencies.
Work breakdown structure	The process of dividing the project into manageable tasks and logically ordering them to ensure a smooth evolution between tasks.

CASE STUDY

CASE 1: A Railway reservation system functions as follows:

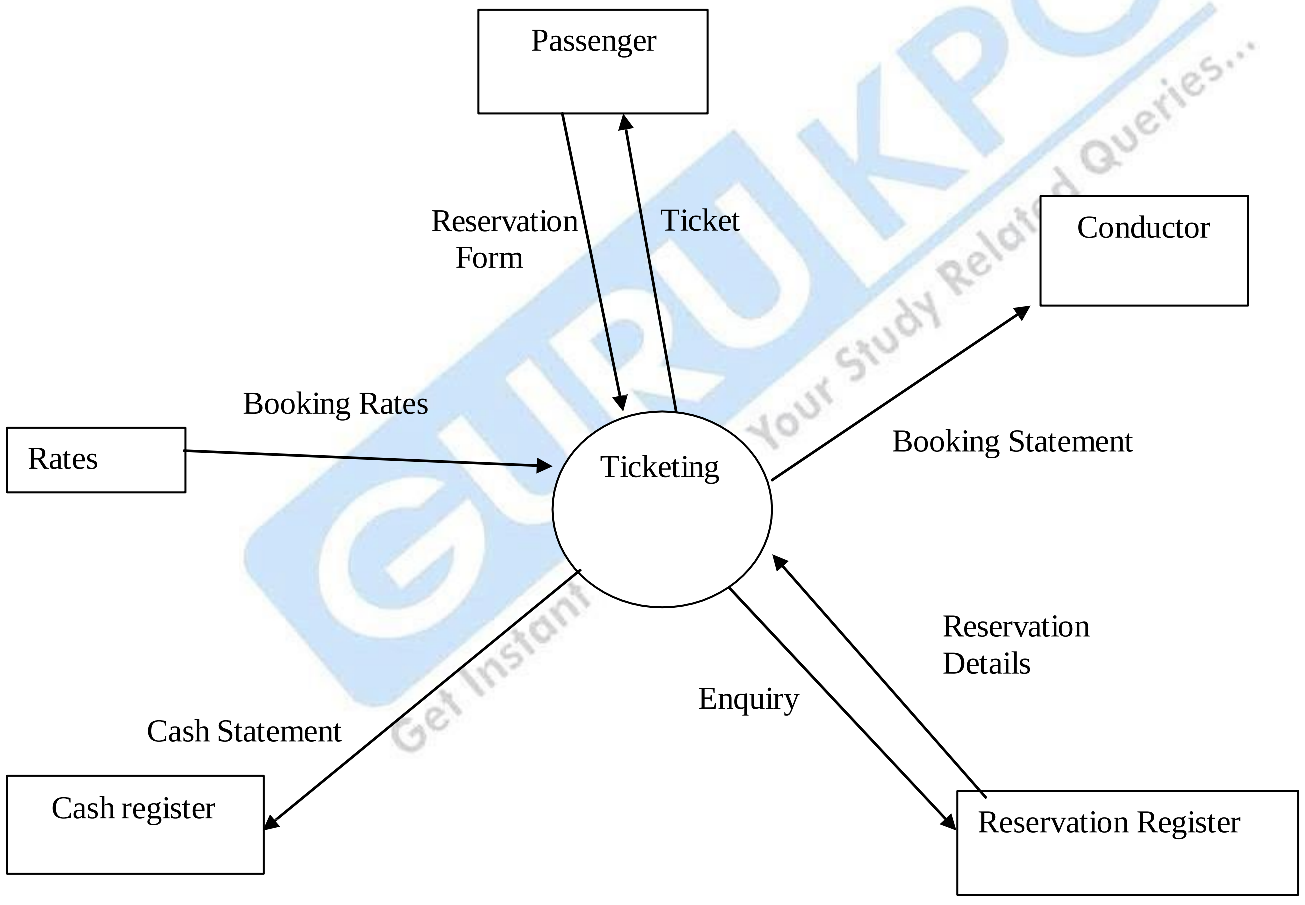
The passenger fills in a reservation form giving his/her particulars and source and destination details. The counter clerk ensures whether seats is available or not from the reservation register. if seat is not available ,the form is returned back to the passenger. Otherwise the clerk will prepare the tickets, compute the charges for the tickets and a booking statement is composed. One copy of the booking statement is retained as office copy, one is given to the train conductor and one copy is pasted on the compartment. A cash statement is prepared at the end of each shift.

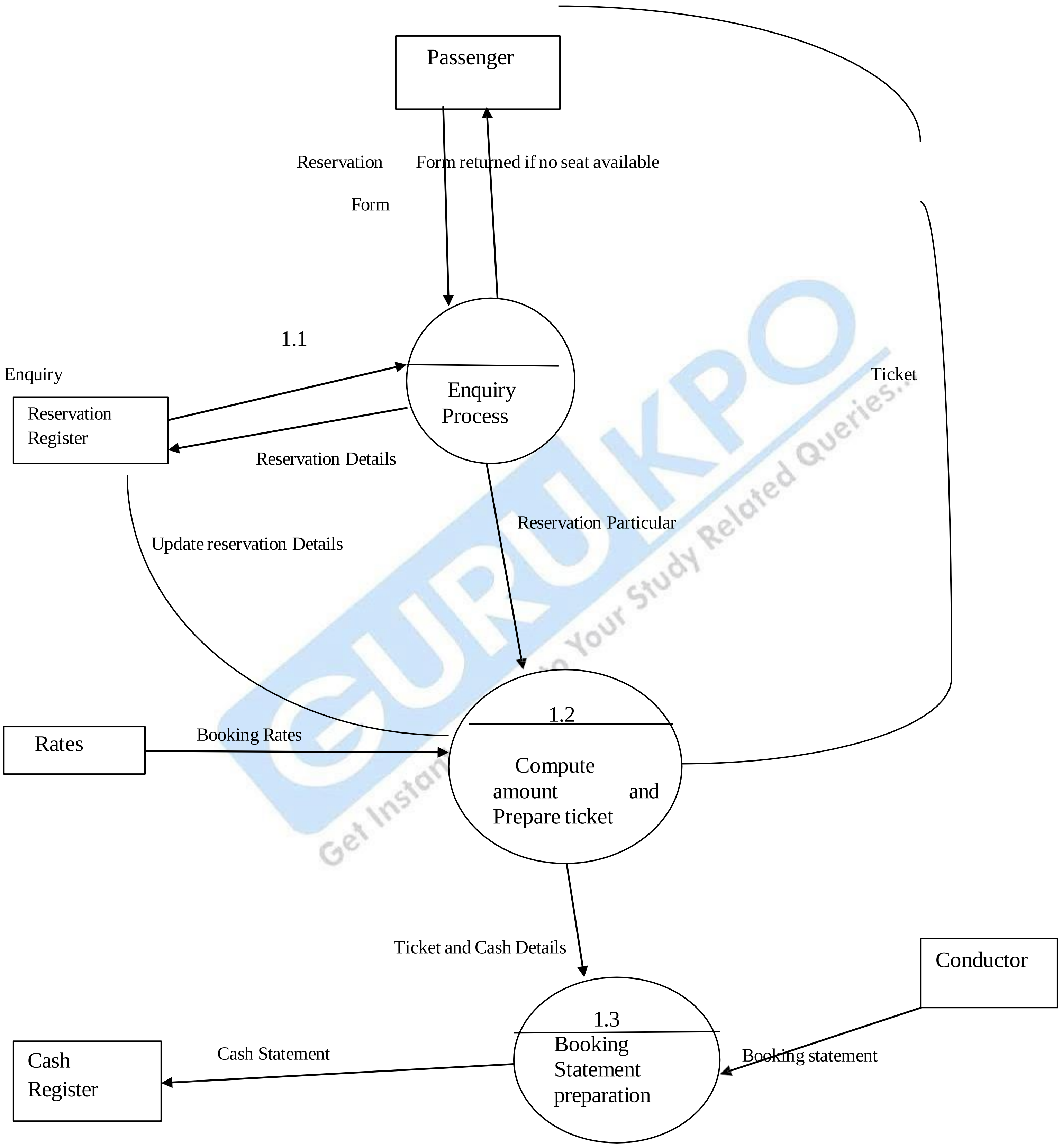
PREPARE A DATAFLOW DIAGRAM FOR THE ABOVE SYSTEM

Context Diagram for Railway Reservation System

First Level Data Flow Diagram

SOLUTION:




Passenger
Reservation Form
Form returned if no seat available
1.1
Enquiry
Enquiry Process
Reservation Register
Reservation Details
Ticket
Update reservation Details
Reservation Particular
1.2
Rates
Booking Rates
Compute amount and Prepare ticket
Ticket and Cash Details
Conductor
1.3
Booking Statement preparation
Cash Register
Cash Statement
Booking statement