

PROBLEM STATEMENT TO DEFINE A BOOK BANK WITH BOUNDED SCOPE OF PROJECT

AIM:

To Write a Problem Statement to define a book bank with bounded scope of project

PROBLEM STATEMENT:

Identify the product whose software requirements are specified in this document. Describe the scope of the product that is covered by this SRS, particularly if this SRS describes only part of the system or a single subsystem. Describe the different types of user that the document is intended for, such as developers, project managers, marketing staff, users, testers, and documentation writers. Describe what the rest of this SRS contains and how it is organized. Suggest a sequence for reading the document, beginning with the overview sections and proceeding through the sections that are most pertinent to each reader type.

PROJECT SCOPE

Provide a short description of the software being specified and its purpose, including relevant benefits, objectives, and goals. Relate the software to corporate goals or business strategies. If a separate vision and scope document is available, refer to it rather than duplicating its contents here. An SRS that specifies the next release of an evolving product should contain its own scope statement as a subset of the long-term strategic product vision.

PROBLEM STATEMENT:

A Book Bank lends books and magazines to member, who is registered in the system. Also it handles the purchase of new titles for the Book Bank. Popular titles are brought into multiple copies. Old books and magazines are removed when they are out of date or poor in condition. A member can reserve a book or magazine that is not currently available in the book bank, so that when it is returned or purchased by the book bank, that person is notified. The book bank can easily create, replace and delete information about the titles, members, loans and reservations from the system.

CONCLUSION:

Thus the Problem Statement to define a book bank with bounded scope of project was written successfully.

PROCESS MODEL TO DEFINE ACTIVITIES AND RELATED TASK SET FOR BOOK BANK

AIM:

To Write a process model to define activities and related task set for book bank

OVERALL DESCRIPTION

Software Processes is a coherent set of activities for specifying, designing, implementing and testing software systems. A software process model is an abstract representation of a process that presents a description of a process from some particular perspective. There are many different software processes but all involve:

- Specification – defining what the system should do;
- Design and implementation – defining the organization of the system and implementing the system;
- Validation – checking that it does what the customer wants;
- Evolution – changing the system in response to changing customer needs.

Types of Software Process Model

Software processes, methodologies and frameworks range from specific prescriptive steps that can be used directly by an organization in day-to-day work, to flexible frameworks that an organization uses to generate a custom set of steps tailored to the needs of a specific project or group. In some cases a “sponsor” or “maintenance” organization distributes an official set of documents that describe the process.

Software Process and Software Development Lifecycle Model

One of the basic notions of the software development process is SDLC models which stands for Software Development Life Cycle models. There are many development life cycle models that have been developed in order to achieve different required objectives. The models specify the various stages of the process and the order in which they are carried out. The most used, popular and important SDLC models are given below:

- Waterfall model
- V model
- Incremental model
- RAD model
- Agile model

- Iterative model
- Spiral model
- Prototype model

Waterfall Model

The process model, we have chosen to develop this software is a Linear Sequential Model (Waterfall Model). Linear Sequential Model suggests a systematic, sequential approach to software development that begins at the system level and progresses through analysis, design, coding, testing and support.

Linear Sequential Model approach has the following phases: -

Software requirements analysis: -

In this, software engineer understands the nature of a program to be built, he must understand the information domain for the software as well as required function, behavior, performance and interface. Requirements for both the system and the software are documented and reviewed with the customer.

Design: -

It has four distinct attributes of a program: data structure, software architecture, interface representations and procedural detail. It is documented and becomes part of the software.

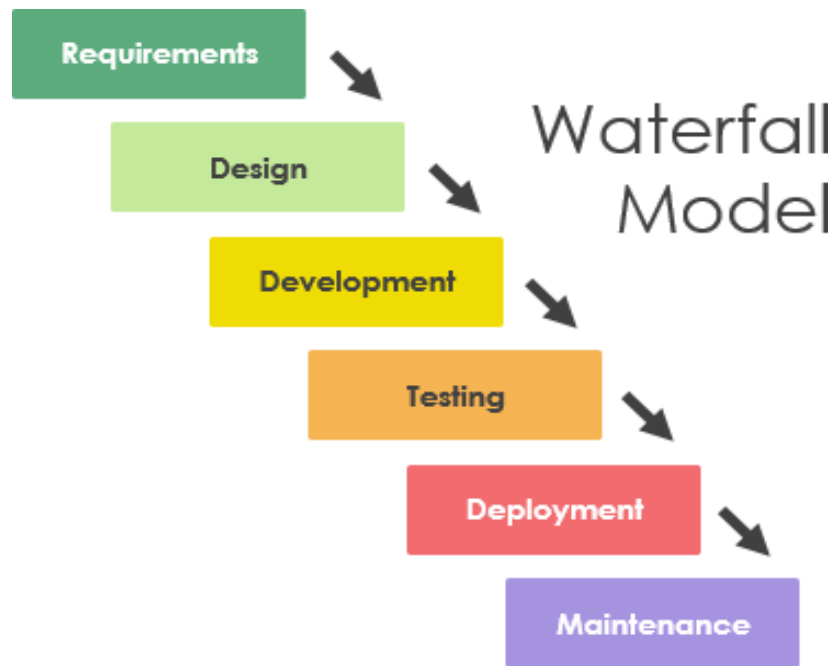
Code generation: -

Design must be translated into a machine-readable form which is done by code generation.

Testing: -

It focuses on the logical internals of the software, ensuring that all the statements have been tested, and on the functional externals; that is conducting test to uncover errors and ensure that defined input will produce actual results.

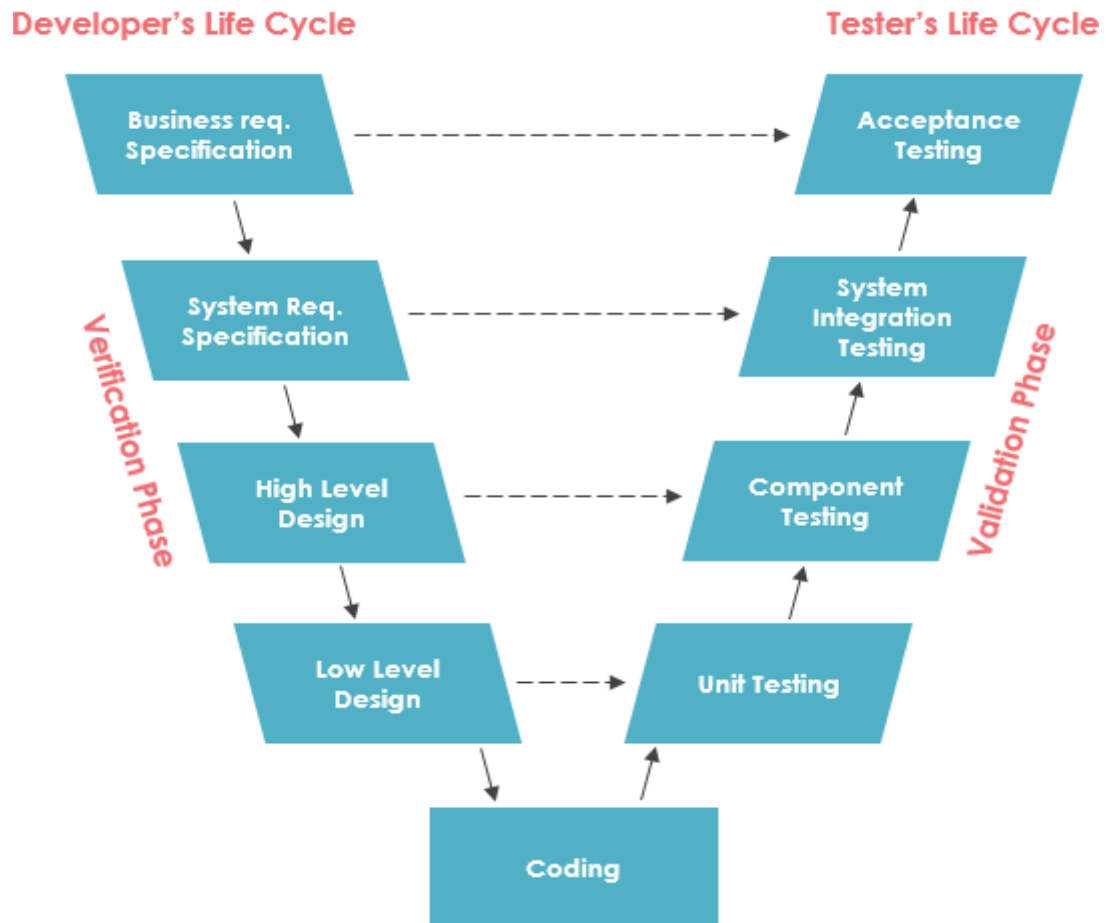
The approach is typical for certain areas of engineering design.



V Model

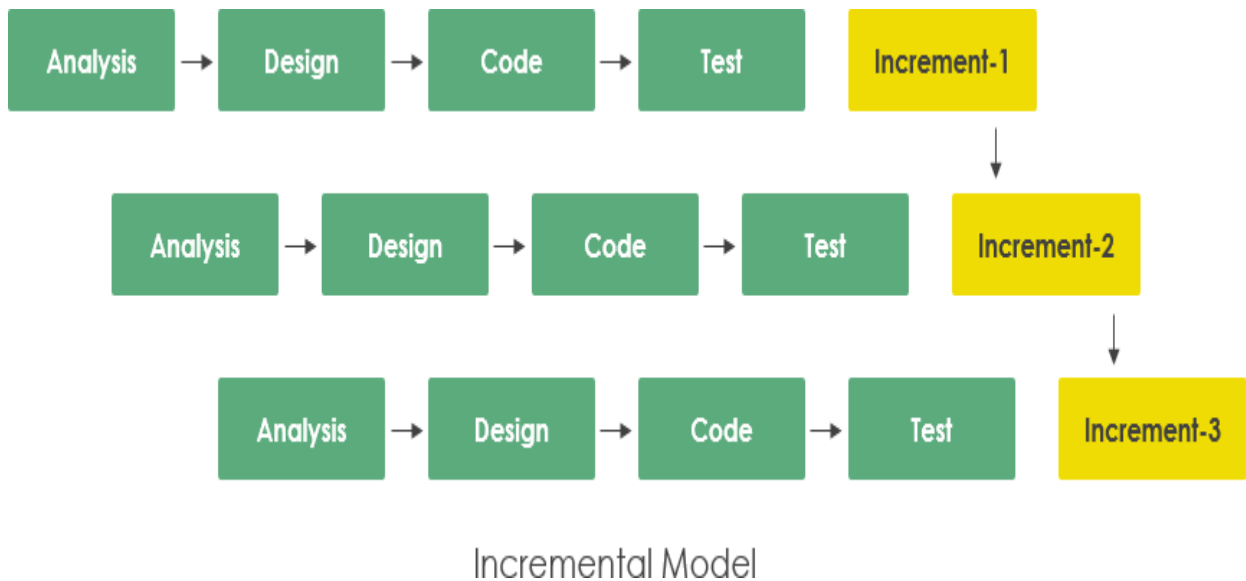
The V-model represents a development process that may be considered an extension of the waterfall model and is an example of the more general V-model. Instead of moving down in a linear way, the process steps are bent upwards after the coding phase, to form the typical V shape. The V-Model demonstrates the relationships between each phase of the development life cycle and its associated phase of testing. The horizontal and vertical axes represent time or project completeness (left-to-right) and level of abstraction (coarsest-grain abstraction uppermost), respectively.

V-Model



Incremental model

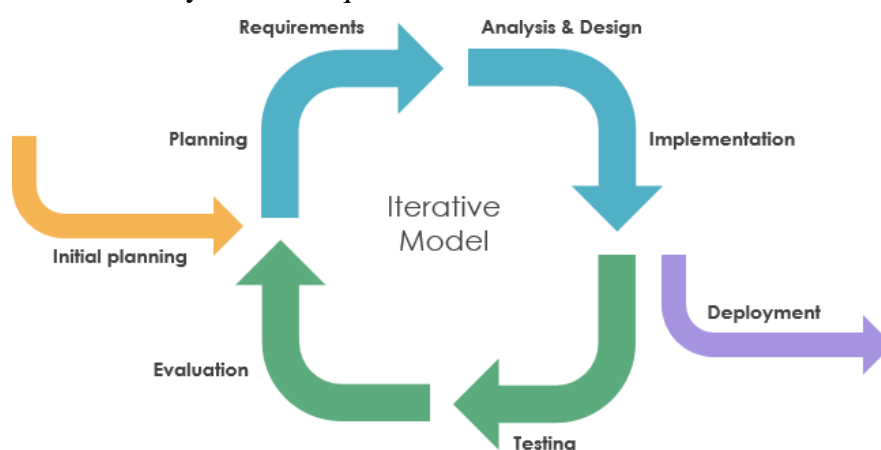
The incremental build model is a method of software development where the model is designed, implemented and tested incrementally (a little more is added each time) until the product is finished. It involves both development and maintenance. The product is defined as finished when it satisfies all of its requirements. Each iteration passes through the requirements, design, coding and testing phases. And each subsequent release of the system adds function to the previous release until all designed functionality has been implemented. This model combines the elements of the waterfall model with the iterative philosophy of prototyping.



Iterative Model

An iterative life cycle model does not attempt to start with a full specification of requirements by first focusing on an initial, simplified set user features, which then progressively gains more complexity and a broader set of features until the targeted system is complete. When adopting the iterative approach, the philosophy of incremental development will also often be used liberally and interchangeably.

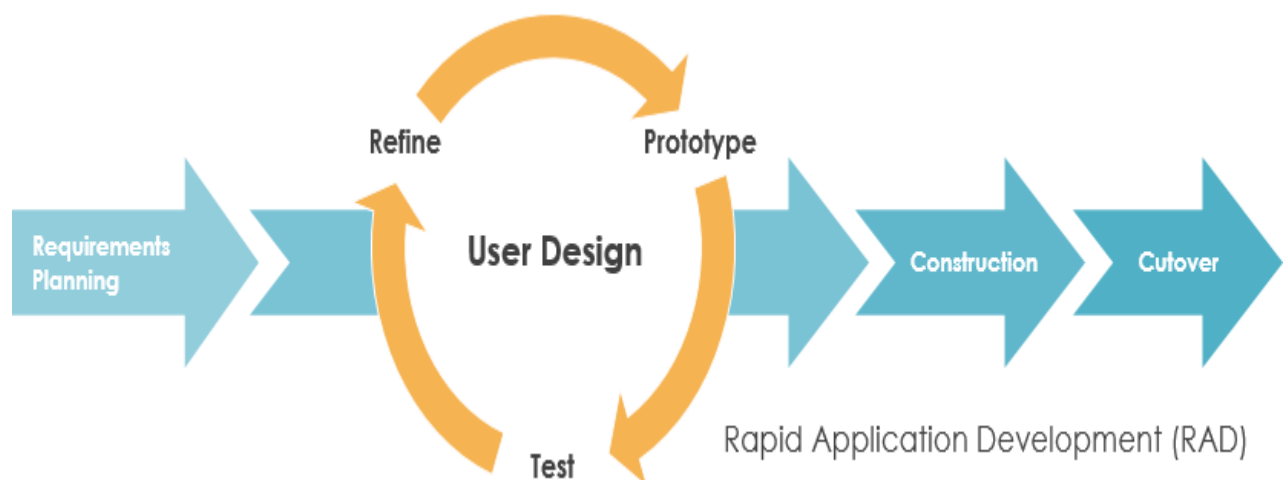
In other words, the iterative approach begins by specifying and implementing just part of the software, which can then be reviewed and prioritized in order to identify further requirements. This iterative process is then repeated by delivering a new version of the software for each iteration. In a light-weight iterative project the code may represent the major source of documentation of the system; however, in a critical iterative project a formal software specification may also be required.



RAD model

Rapid application development was a response to plan-driven waterfall processes, developed in the 1970s and 1980s, such as the Structured Systems Analysis and Design Method (SSADM). Rapid application development (RAD) is often referred as the adaptive software development. RAD is an incremental prototyping approach to software development that end users can produce better feedback when examining a live system, as opposed to working strictly with documentation. It puts less emphasis on planning and more emphasis on an adaptive process.

RAD may result in a lower level of rejection when the application is placed into production, but this success most often comes at the expense of a dramatic overrun in project costs and schedule. RAD approach is especially well suited for developing software that is driven by user interface requirements. Thus, some GUI builders are often called rapid application development tools.



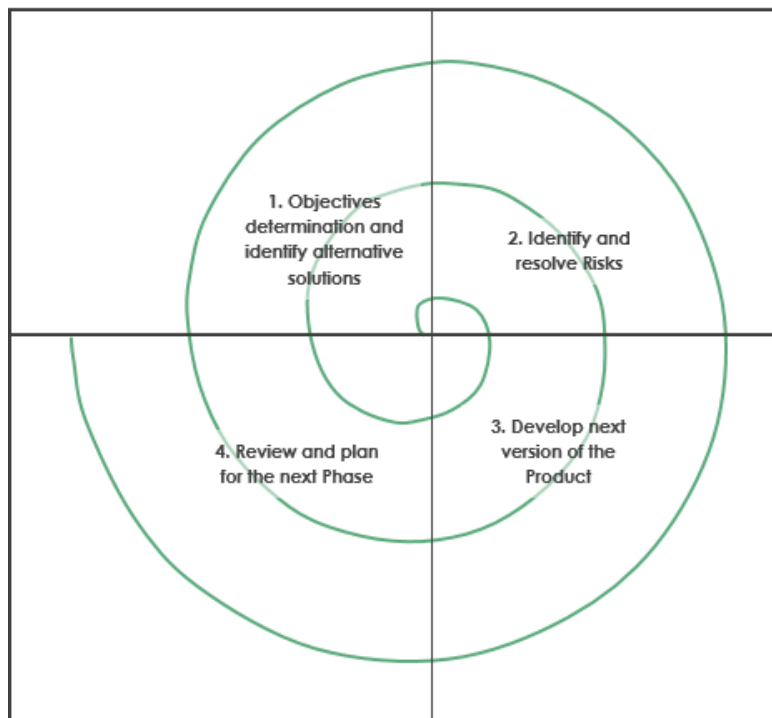
Spiral model

The spiral model, first described by Barry Boehm in 1986, is a risk-driven software development process model which was introduced for dealing with the shortcomings in the traditional waterfall model. A spiral model looks like a spiral with many loops. The exact number of loops of the spiral is unknown and can vary from project to project. This model supports risk handling, and the project is delivered in loops. Each loop of the spiral is called a Phase of the software development process.

The initial phase of the spiral model in the early stages of Waterfall Life Cycle that is needed to develop a software product. The exact number of phases needed to develop the product can be varied by the project manager depending upon the project risks. As the project

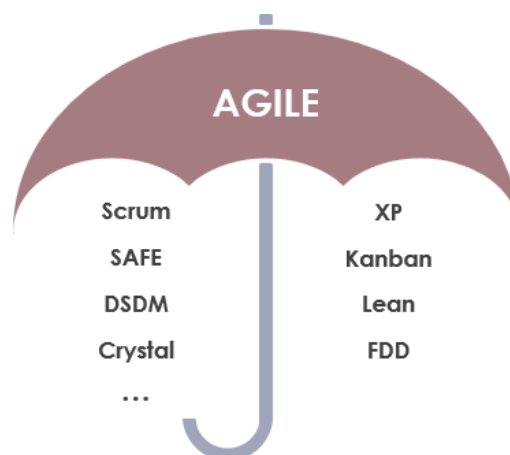
manager dynamically determines the number of phases, so the project manager has an important role to develop a product using a spiral model.

Spiral Model



Agile model

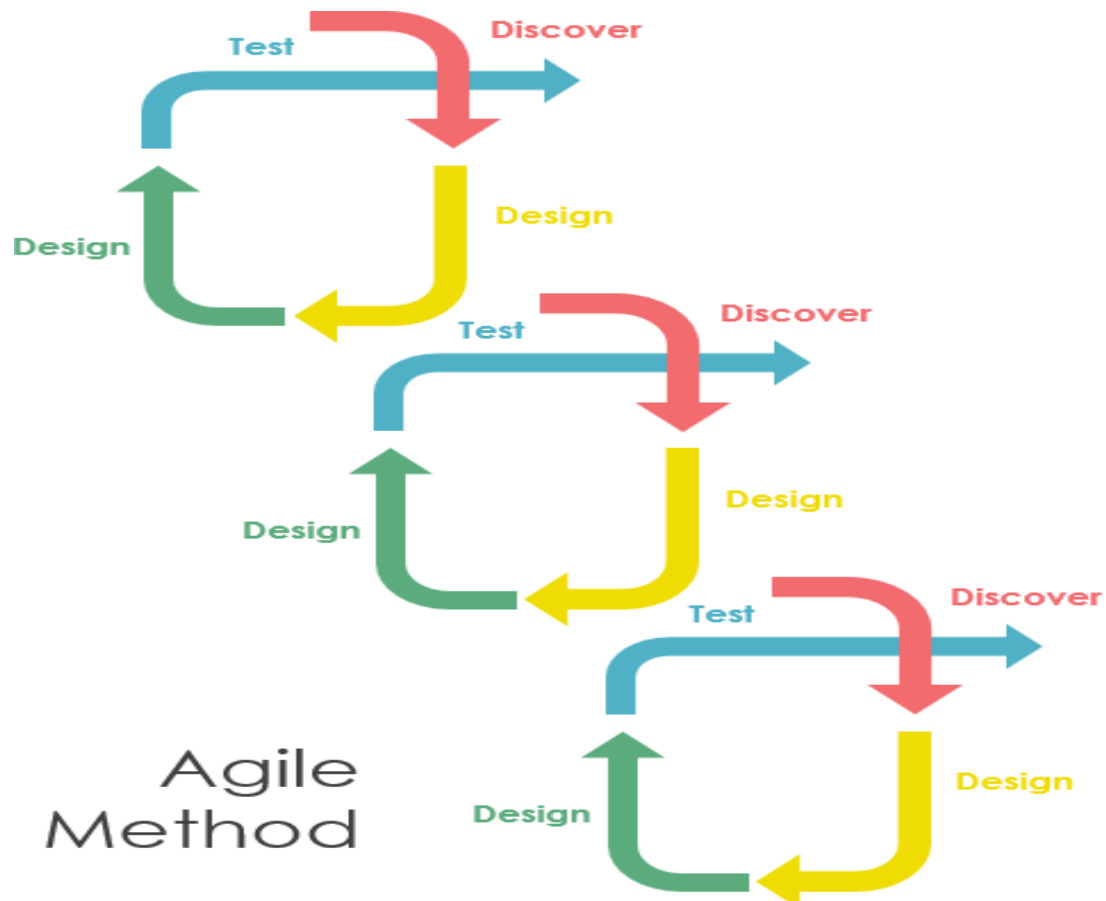
Agile is an umbrella term for a set of methods and practices based on the values and principles expressed in the Agile Manifesto that is a way of thinking that enables teams and businesses to innovate, quickly respond to changing demand, while mitigating risk. Organizations can be agile using many of the available frameworks available such as Scrum, Kanban, Lean, Extreme Programming (XP) and etc.



The Agile movement proposes alternatives to traditional project management. Agile approaches are typically used in software development to help businesses respond to unpredictability which refer to a group of software development methodologies based on

iterative development, where requirements and solutions evolve through collaboration between self-organizing cross-functional teams.

The primary goal of being Agile is empowered the development team the ability to create and respond to change in order to succeed in an uncertain and turbulent environment. Agile software development approach is typically operated in rapid and small cycles. This results in more frequent incremental releases with each release building on previous functionality. Thorough testing is done to ensure that software quality is maintained.



LINEAR SEQUENTIAL MODEL has been used because of the following reasons: -

It is relatively simple and easier to understand approach as compared to other models. The requirements are well stated and understood before in hand. In this model we have to complete one stage before proceeding to next. So, we have clearly defined stages and well understood milestones. The advancement in program does not need to be checked upon by the customer during the process. So, this model does not create problem. The requirements are

fixed and work can proceed to completion in a linear manner. The model provides a structural approach

CONCLUSION:

Thus, the process model to define activities and related task set for book bank are done.

PREPARE BROAD SRS FOR THE BOOK BANK

AIM:

To prepare broad SRS (Software Requirement Specification) for the book bank

PRODUCT PERSPECTIVE

Describe the context and origin of the product being specified in this SRS. For example, state whether this product is a follow-on member of a product family, a replacement for certain existing systems, or a new, self-contained product. If the SRS defines a component of a larger system, relate the requirements of the larger system to the functionality of this software and identify interfaces between the two. A simple diagram that shows the major components of the overall system, subsystem interconnections, and external interfaces can be helpful.

PRODUCT FEATURES

Summarize the major features the product contains or the significant functions that it performs or lets the user perform. Only a high level summary is needed here. Organize the functions to make them understandable to any reader of the SRS. A picture of the major groups of related requirements and how they relate, such as a top level data flow diagram or a class diagram, is often effective.

USER CLASSES AND CHARACTERISTICS

Identify the various user classes that you anticipate will use this product. User classes may be differentiated based on frequency of use, subset of product functions used, technical expertise, security or privilege levels, educational level, or experience. Describe the pertinent characteristics of each user class. Certain requirements may pertain only to certain user classes. Distinguish the favored user classes from those who are less important to satisfy.

OPERATING ENVIRONMENT

Describe the environment in which the software will operate, including the hardware platform, operating system and versions, and any other software components or applications with which it must peacefully coexist.

DESIGN AND IMPLEMENTATION CONSTRAINTS

Describe any items or issues that will limit the options available to the developers. These might include: corporate or regulatory policies; hardware limitations (timing requirements,

memory requirements); interfaces to other applications; specific technologies, tools, and databases to be used; parallel operations; language requirements; communications protocols; security considerations; design conventions or programming standards (for example, if the customer's organization will be responsible for maintaining the delivered software).

SYSTEM FEATURES

This template illustrates organizing the functional requirements for the product by system features, the major services provided by the product. You may prefer to organize this section by use case, mode of operation, user class, object class, functional hierarchy, or combinations of these, whatever makes the most logical sense for your product.

System Feature 1

Description and Priority

Provide a short description of the feature and indicate whether it is of High, Medium, or Low priority. You could also include specific priority component ratings, such as benefit, penalty, cost, and risk (each rated on a relative scale from a low of 1 to a high of 9).

Stimulus/Response Sequences

List the sequences of user actions and system responses that stimulate the behavior defined for this feature. These will correspond to the dialog elements associated with use cases.

Functional Requirements

Itemize the detailed functional requirements associated with this feature. These are the software capabilities that must be present in order for the user to carry out the services provided by the feature, or to execute the use case. Include how the product should respond to anticipated error conditions or invalid inputs. Requirements should be concise, complete, unambiguous, verifiable, and necessary.

EXTERNAL INTERFACE REQUIREMENTS

USER INTERFACES

Describe the logical characteristics of each interface between the software product and the users. This may include sample screen images, any GUI standards or product family style guides that are to be followed, screen layout constraints, standard buttons and functions (e.g.,

help) that will appear on every screen, keyboard shortcuts, error message display standards, and so on. Define the software components for which a user interface is needed. Details of the user interface design should be documented in a separate user interface specification.

HARDWARE INTERFACES

Describe the logical and physical characteristics of each interface between the software product and the hardware components of the system. This may include the supported device types, the nature of the data and control interactions between the software and the hardware, and communication protocols to be used.

SOFTWARE INTERFACES

Describe the connections between this product and other specific software components (name and version), including databases, operating systems, tools, libraries, and integrated commercial components. Identify the data items or messages coming into the system and going out and describe the purpose of each. Describe the services needed and the nature of communications. Refer to documents that describe detailed application programming interface protocols. Identify data that will be shared across software components. If the data sharing mechanism must be implemented in a specific way (for example, use of a global data area in a multitasking operating system), specify this as an implementation constraint.

COMMUNICATIONS INTERFACES

Describe the requirements associated with any communications functions required by this product, including e-mail, web browser, network server communications protocols, electronic forms, and so on. Define any pertinent message formatting. Identify any communication standards that will be used, such as FTP or HTTP. Specify any communication security or encryption issues, data transfer rates, and synchronization mechanisms.

NONFUNCTIONAL REQUIREMENTS

PERFORMANCE REQUIREMENTS

If there are performance requirements for the product under various circumstances, state them here and explain their rationale, to help the developers understand the intent and make suitable design choices. Specify the timing relationships for real time systems. Make such requirements as specific as possible. You may need to state performance requirements for individual functional requirements or features.

SAFETY REQUIREMENTS

Specify those requirements that are concerned with possible loss, damage, or harm that could result from the use of the product. Define any safeguards or actions that must be taken, as well as actions that must be prevented. Refer to any external policies or regulations that state safety issues that affect the product's design or use. Define any safety certifications that must be satisfied.

SECURITY REQUIREMENTS

Specify any requirements regarding security or privacy issues surrounding use of the product or protection of the data used or created by the product. Define any user identity authentication requirements. Refer to any external policies or regulations containing security issues that affect the product. Define any security or privacy certifications that must be satisfied.

SOFTWARE QUALITY ATTRIBUTES

Specify any additional quality characteristics for the product that will be important to either the customers or the developers. Some to consider are: adaptability, availability, correctness, flexibility, interoperability, maintainability, portability, reliability, reusability, robustness, testability, and usability. Write these to be specific, quantitative, and verifiable when possible. At the least, clarify the relative preferences for various attributes, such as ease of use over ease of learning.

OTHER REQUIREMENTS

Define any other requirements not covered elsewhere in the SRS. This might include database requirements, internationalization requirements, legal requirements, reuse objectives for the project, and so on. Add any new sections that are pertinent to the project.

PREPARATION OF SOFTWARE REQUIREMENT SPECIFICATION DOCUMENT FOR BOOK BANK:

Users Characteristics:

Student: They are the people who desire to obtain the books and submit the information to the database.

Librarian: He has the certain privileges to add the books and to approval of the reservation of books.

System Modules:

Log in: Secure registration of student and librarian by filling online registration form.

Book bank: Book bank contains all the books. New book added to the book bank with bookno, title name, author, edition, publisher name details to the database. Any book is deleted if damaged. Update of the book information also done.

Operations: student and administrator perform their operations like add book, delete book, update information, view book details are implemented in log in WebPages.

Non-functional requirements:

Privacy: privacy maintained for each and every user by providing user credentials username and password. **Portability:** installation on multiple platforms and execution of software.

Preparation of Software Configuration Management**Software Requirements:**

Operating system: windows 7/10

Rational Rose

Front end : J2EE

Back end : My SQL Server

IDE used : Netbeans

Hardware Requirements:

Processor:i3 or higher

RAM : 4 GB

Hard Disk drive: 500 GB

CONCLUSION:

Thus the broad SRS (Software Requirement Specification) for the book bank are prepared

PREPARE USE CASES AND DRAW USE CASE DIAGRAM

AIM:

To prepare use cases and draw use case diagram using modelling tool

USE CASE DIAGRAM

In the Unified Modeling Language (UML), a use case diagram can summarize the details of your system's users (also known as actors) and their interactions with the system. To build one, you'll use a set of specialized symbols and connectors. An effective use case diagram can help your team discuss and represent:

- Scenarios in which your system or application interacts with people, organizations, or external systems
- Goals that your system or application helps those entities (known as actors) achieve
- The scope of your system

WHEN TO APPLY USE CASE DIAGRAMS

A use case diagram doesn't go into a lot of detail—for example, don't expect it to model the order in which steps are performed. Instead, a proper use case diagram depicts a high-level overview of the relationship between use cases, actors, and systems. Experts recommend that use case diagrams be used to supplement a more descriptive textual use case.

UML is the modeling toolkit that you can use to build your diagrams. Use cases are represented with a labeled oval shape. Stick figures represent actors in the process, and the actor's participation in the system is modeled with a line between the actor and use case. To depict the system boundary, draw a box around the use case itself.

UML use case diagrams are ideal for:

- Representing the goals of system-user interactions
- Defining and organizing functional requirements in a system
- Specifying the context and requirements of a system
- Modeling the basic flow of events in a use case

Use case diagram components

To answer the question, "What is a use case diagram?" you need to first understand its building blocks. Common components include:

- **Actors:** The users that interact with a system. An actor can be a person, an organization, or an outside system that interacts with your application or system. They must be external objects that produce or consume data.

- **System:** A specific sequence of actions and interactions between actors and the system. A system may also be referred to as a scenario.
- **Goals:** The end result of most use cases. A successful diagram should describe the activities and variants used to reach the goal.

Use case diagram symbols and notation

The notation for a use case diagram is pretty straightforward and doesn't involve as many types of symbols as other UML diagrams.

Use cases: Horizontally shaped ovals that represent the different uses that a user might have.

- **Actors:** Stick figures that represent the people actually employing the use cases.
- **Associations:** A line between actors and use cases. In complex diagrams, it is important to know which actors are associated with which use cases.
- **System boundary boxes:** A box that sets a system scope to use cases. All use cases outside the box would be considered outside the scope of that system. For example, Psycho Killer is outside the scope of occupations in the chainsaw example found below.
- **Packages:** A UML shape that allows you to put different elements into groups. Just as with component diagrams, these groupings are represented as file folders.

Use_Case Diagram:

The book bank use cases are:

- book_issue
- book_return
- book_order
- book_entry
- search book_details

Actors Involved:

- Student
- Librarian
- Vendor

Usecase Name : Search Book_Details

The librarian initiates this use case when any member returns or request the book and checking if the book is available.

Precondition: The librarian should enter all Book details.

Normal Flow: Build message for librarian who search the book.

Post Condition: Send message to respective member who reserved the book.

Usecase Name : Book_ Issue

Initiated by librarian when any member wants to borrow the desired book. If the book is available, the book is issued.

Precondition: Member should be valid member of library.

Normal Flow: Selected book will be issued to the member.

Alternative Flow: If book is not available then reserved book use case should be initiate.

Post Condition: Update the catalogue.

Usecase Name : Book_Order

Initiated by librarian when the requested book is not available in the library at that moment. The book is reserved for the future and issued to the person when it is available.

Precondition: Initiated only when book is not available.

Normal Flow: It reserved the book if requested.

Post Condition : Mention the entry in catalogue for reservation.

Usecase Name : Book_Return

Invoked by the librarian when a member returns the book.

Precondition: Member should be valid member of library.

Normal Flow: Librarian enters bookid and system checks for return date of the book.

Alternative Flow: System checks for return date and if it returned late fine message will be displayed.

Post Condition: Check the status of reservation.

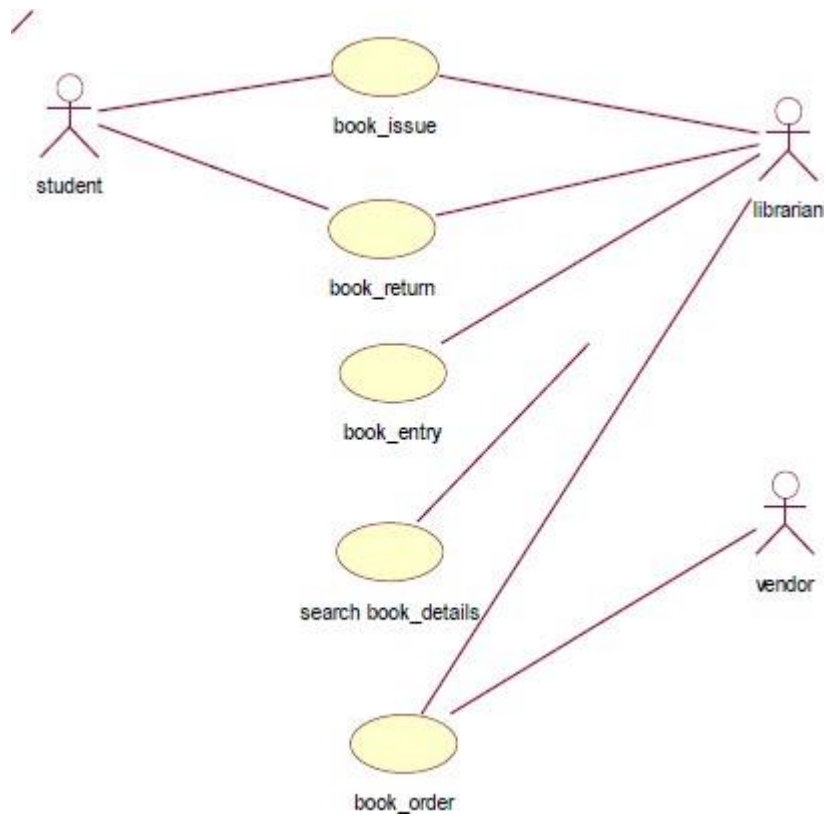
Usecase Name : Book_Entry

The purchase book use-case when new books invoke it or magazines are added to the library.

Precondition: Not available or more copies are required.

Normal Flow: Enter bookid,author information, publication information, purchased date, prize and number of copies.

Post Condition: Update the information in catalogue.



Use case diagram for Book Bank System

CONCLUSION:

Thus, use case diagram is drawn by a using modelling tool.

DEVELOP THE ACTIVITY DIAGRAM

AIM:

To develop the activity diagram to represent flow from one activity to another for software development

ACTIVITY DIAGRAM

The Unified Modeling Language includes several subsets of diagrams, including structure diagrams, interaction diagrams, and behavior diagrams. Activity diagrams, along with use case and state machine diagrams, are considered behavior diagrams because they describe what must happen in the system being modeled.

Stakeholders have many issues to manage, so it's important to communicate with clarity and brevity. Activity diagrams help people on the business and development sides of an organization come together to understand the same process and behavior. You'll use a set of specialized symbols—including those used for starting, ending, merging, or receiving steps in the flow—to make an activity diagram, which we'll cover in more depth within this activity diagram guide.

Benefits of activity diagrams

Activity diagrams present a number of benefits to users. Consider creating an activity diagram to:

- Demonstrate the logic of an algorithm.
- Describe the steps performed in a UML use case.
- Illustrate a business process or workflow between users and the system.
- Simplify and improve any process by clarifying complicated use cases.
- Model software architecture elements, such as method, function, and operation.

BASIC COMPONENTS OF AN ACTIVITY DIAGRAM


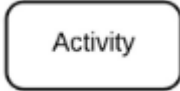

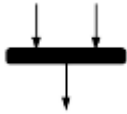

Before you begin making an activity diagram, you should first understand its makeup. Some of the most common components of an activity diagram include:








- **Action:** A step in the activity wherein the users or software perform a given task.
- **Decision node:** A conditional branch in the flow that is represented by a diamond. It includes a single input and two or more outputs.
- **Control flows:** Another name for the connectors that show the flow between steps in the diagram.


- Start node: Symbolizes the beginning of the activity. The start node is represented by a black circle.
- End node: Represents the final step in the activity. The end node is represented by an outlined black circle.

Activity diagram symbols

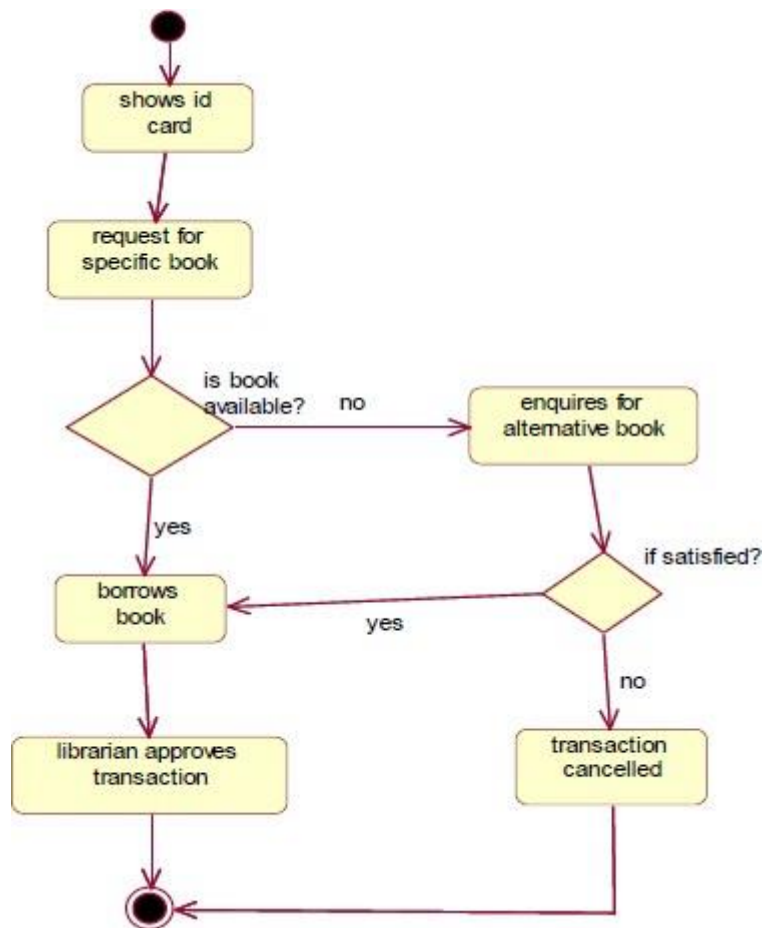
These activity diagram shapes and symbols are some of the most common types you'll find in UML diagrams.

Symbol	Name	Description
	Start symbol	Represents the beginning of a process or workflow in an activity diagram. It can be used by itself or with a note symbol that explains the starting point.
	Activity symbol	Indicates the activities that make up a modeled process. These symbols, which include short descriptions within the shape, are the main building blocks of an activity diagram.
	Connector symbol	Shows the directional flow, or control flow, of the activity. An incoming arrow starts a step of an activity; once the step is completed, the flow continues with the outgoing arrow.
	Joint symbol/ Synchronization bar	Combines two concurrent activities and re-introduces them to a flow where only one activity occurs at a time. Represented with a thick vertical or horizontal line.
	Fork symbol	Splits a single activity flow into two concurrent activities. Symbolized with multiple arrowed lines from a join.

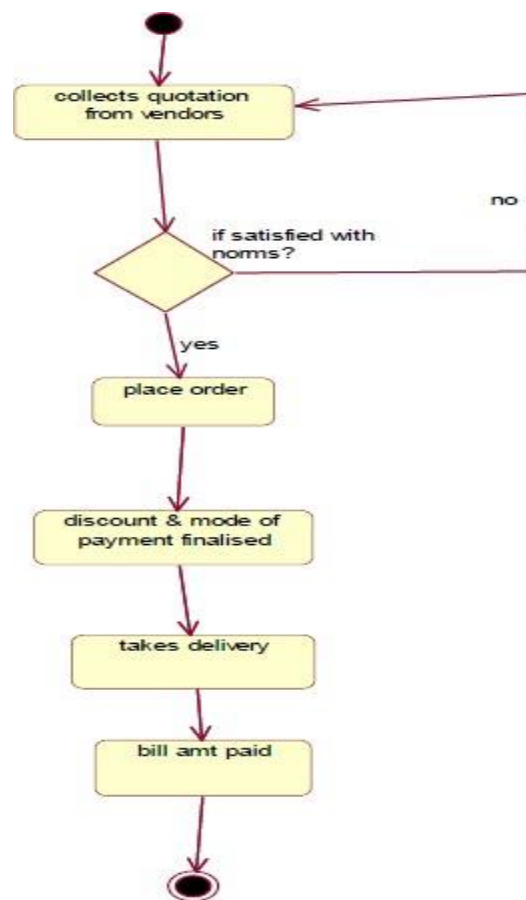
Symbol	Name	Description
	Decision symbol	Represents a decision and always has at least two paths branching out with condition text to allow users to view options. This symbol represents the branching or merging of various flows with the symbol acting as a frame or container.
	Note symbol	Allows the diagram creators or collaborators to communicate additional messages that don't fit within the diagram itself. Leave notes for added clarity and specification.
	Send signal symbol	Indicates that a signal is being sent to a receiving activity.
	Receive signal symbol	Demonstrates the acceptance of an event. After the event is received, the flow that comes from this action is completed.
	Shallow history pseudostate symbol	Represents a transition that invokes the last active state.
	Option loop symbol	Allows the creator to model a repetitive sequence within the option loop symbol.
	Flow final symbol	Represents the end of a specific process flow. This symbol shouldn't represent the end of all flows in an activity; in that instance, you would use the end symbol. The flow final symbol should be placed at the end of a process in a single activity flow.

Symbol	Name	Description
[Condition]	Condition text	Placed next to a decision marker to let you know under what condition an activity flow should split off in that direction.
	End symbol	Marks the end state of an activity and represents the completion of all flows of a process.

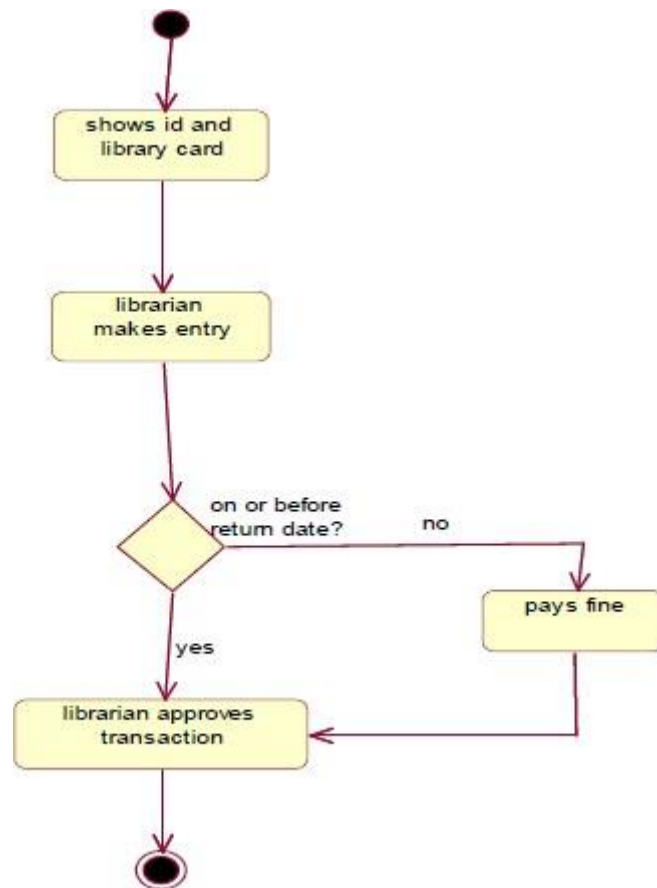
Activity Diagrams For Book Bank System



Activity Diagram for Book Bank System [borrow book]



Activity Diagram for Book Bank System [order book]



Activity Diagram for Book Bank System [Return book]

CONCLUSION:

Thus, the activity diagram to represent flow from one activity to another for software development is developed.

DEVELOP DATA DESIGNS USING DFD DECISION TABLE & ER DIAGRAM

AIM:

To develop data Designs using DFD Decision Table & ER Diagram

DATA FLOW DIAGRAMS (DFDS)

Data flow diagrams (DFDs) visually map your process or system, so you can uncover opportunities to improve efficiency and performance. Whether you are improving an existing process or implementing a new one, a data flow diagram will make the task easier. However, if you've never created a DFD before, getting started can be intimidating. There is a lot to take in: different levels of diagrams, symbols and notation, not to mention actually creating the diagram—navigating it all will take more than looking at a few examples. If you're new to data flow diagrams, this guide will help get you started.

WHAT IS A DATA FLOW DIAGRAM?

A data flow diagram shows the way information flows through a process or system. It includes data inputs and outputs, data stores, and the various subprocesses the data moves through. DFDs are built using standardized symbols and notation to describe various entities and their relationships.

Data flow diagrams visually represent systems and processes that would be hard to describe in just words. You can use these diagrams to map out an existing system and make it better or to plan out a new system for implementation. Visualizing each element makes it easy to identify inefficiencies and produce the best possible system.

Physical and logical data flow diagrams

Before actually creating your data flow diagram, you'll need to determine whether a physical or logical DFD best suits your needs. If you're new to data flow diagrams, don't worry—the distinction is pretty straightforward.

Logical data flow diagrams focus on *what* happens in a particular information flow: what information is being transmitted, what entities are receiving that info, what general processes occur, etc. The processes described in a logical DFD are business activities—a logical DFD doesn't delve into the technical aspects of a process or system, such as how the process is constructed and implemented. So you don't need to include details like configuration or data storage technology. Non-technical employees should be able to understand these diagrams, making logical DFDs an excellent tool for communicating with project stakeholders.

Physical data flow diagrams focus on *how* things happen in an information flow. These diagrams specify the software, hardware, files, and people involved in an information flow. A detailed physical data flow diagram can facilitate the development of the code needed to implement a data system.

How to create a data flow diagram

Now that you have some background knowledge on data flow diagrams and how they are categorized, you're ready to build your own DFD. The process can be broken down into 5 steps:

1. Identify major inputs and outputs in your system

Nearly every process or system begins with input from an external entity and ends with the output of data to another entity or database. Identifying such inputs and outputs gives a macro view of your system—it shows the broadest tasks the system should achieve. The rest of your DFD will be built on these elements, so it is crucial to know them early on.

2. Build a context diagram

Once you've identified the major inputs and outputs, building a context diagram is simple. Draw a single process node and connect it to related external entities. This node represents the most general process that information follows to go from input to output.

The data diagram flow example below shows how information flows between various entities via an online community. Data flows to and from the external entities, representing both input and output. The center node, "online community," is the general process.

3. Expand the context diagram into a level 1 DFD

The single process node of your context diagram doesn't provide much information—you need to break it down into sub-processes. In your level 1 data flow diagram, you should include several process nodes, major databases, and all external entities. Walk through the flow of information: where does the information start and what needs to happen to it before each data store?

4. Expand to a level 2+ DFD

To enhance the detail of your data flow diagram, follow the same process as in step 3. The processes in your level 1 DFD can be broken down into more specific sub-processes. Once again, ensure you add any necessary data stores and flows—at this point, you should have a fairly detailed breakdown of your system. To progress beyond a level 2 data flow diagram, simply repeat this process. Stop once you've reached a satisfactory level of detail.

5. Confirm the accuracy of your final diagram

When your diagram is completely drawn, walk through it. Pay close attention to the flow of information: does it make sense? Are all necessary data stores included? By looking at your final diagram, other parties should be able to understand the way your system functions. Before presenting your final diagram, check with co-workers to ensure your diagram is comprehensible.

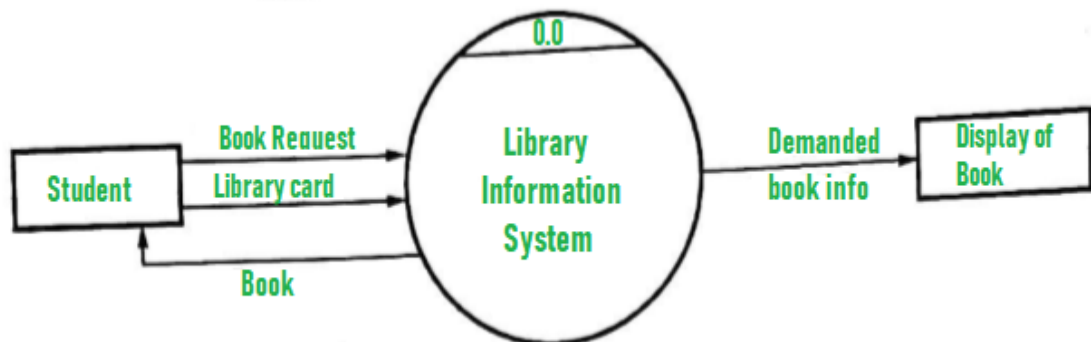
Data Flow Diagram (DFD) depicts the flow of information and the transformation applied when a data moves in and out from a system. The overall system is represented and described using input, processing and output in the DFD. The inputs can be:

- **Book request** when a student requests for a book.
- **Library card** when the student has to show or submit his/her identity as a proof.

The overall processing unit will contain the following output that a system will produce or generate:

- Book will be the output as the book demanded by the student will be given to them.
- Information of demanded book should be displayed by the library information system that can be used by the student while selecting the book which makes it easier for the student.

1. Level 0 DFD –



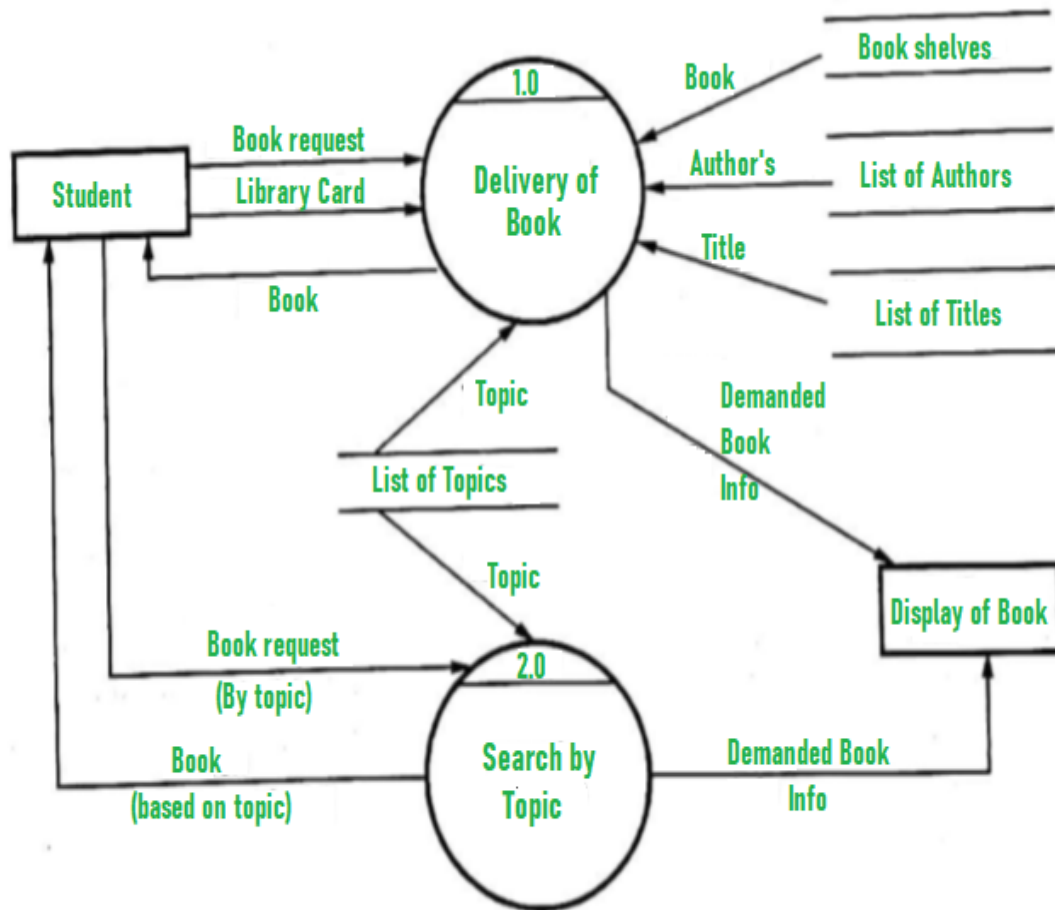
DFD

At this level, the system has to show or exposed with more details of processing.

The processes that are important to be carried out are:

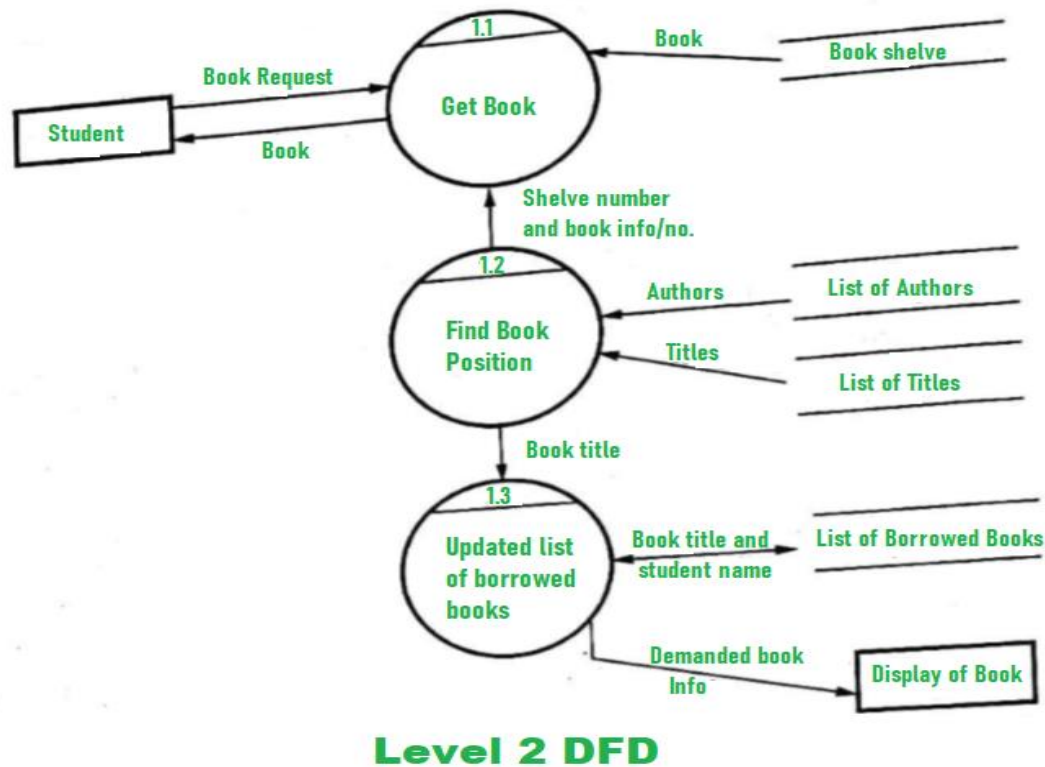
- Book delivery
- Search by topic

List of authors, List of Titles, List of Topics, the bookshelves from which books can be located are some information that is required for these processes. **Data store** is used to represent this type of information.



Level 1 DFD

2. Level 2 DFD –

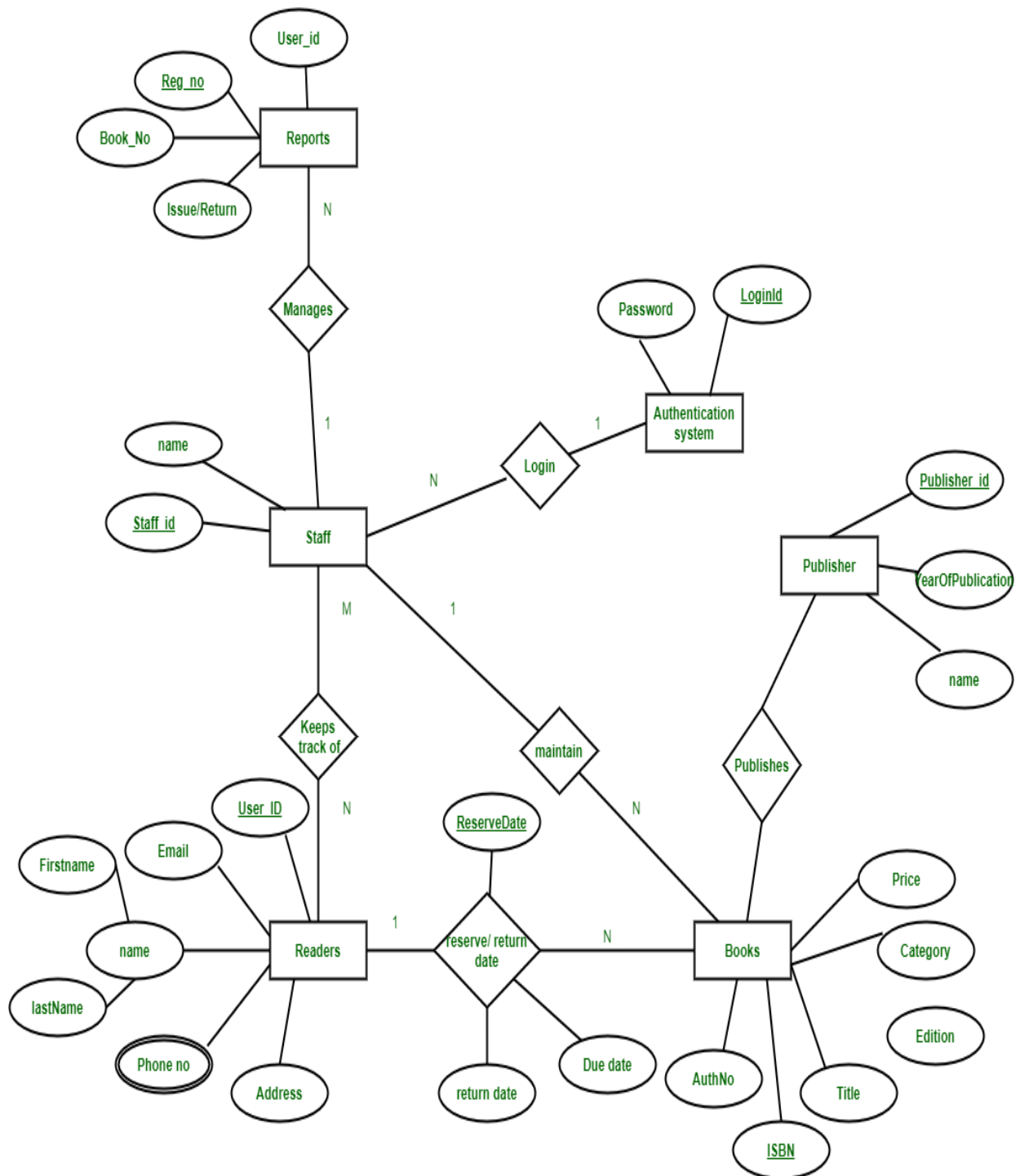


ER DIAGRAM is known as Entity-Relationship Diagram, it is used to analyze the structure of the Database. It shows relationships between entities and their attributes. An ER Model provides a means of communication.

The Library Management System database keeps track of readers with the following considerations –

- The system keeps track of the staff with a single point authentication system comprising login Id and password.
- Staff maintains the book catalog with its ISBN, Book title, price(in INR), category(novel, general, story), edition, author Number and details.
- A publisher has publisher Id, Year when the book was published, and name of the book.
- Readers are registered with their user_id, email, name (first name, last name), Phone no (multiple entries allowed), communication address. The staff keeps track of readers.
- Readers can return/reserve books that stamps with issue date and return date. If not returned within the prescribed time period, it may have a due date too.

- Staff also generate reports that has readers id, registration no of report, book no and return/issue info.



ER Diagram of Library Management System

This Library ER diagram illustrates key information about the Library, including entities such as staff, readers, books, publishers, reports, and authentication system. It allows for understanding the relationships between entities.

Entities and their Attributes –

- **Book Entity** : It has authno, isbn number, title, edition, category, price. ISBN is the Primary Key for Book Entity.
- **Reader Entity** : It has UserId, Email, address, phone no, name. Name is composite attribute of firstname and lastname. Phone no is multi valued attribute. UserId is the Primary Key for Readers entity.
- **Publisher Entity** : It has PublisherId, Year of publication, name. PublisherID is the Primary Key.
- **Authentication System Entity** : It has LoginId and password with LoginID as Primary Key.
- **Reports Entity** : It has UserId, Reg_no, Book_no, Issue/Return date. Reg_no is the Primary Key of reports entity.
- **Staff Entity** : It has name and staff_id with staff_id as Primary Key.
- **Reserve/Return Relationship Set** : It has three attributes: Reserve date, Due date, Return date.

Relationships between Entities –

- A reader can reserve N books but one book can be reserved by only one reader. The relationship 1:N.
- A publisher can publish many books but a book is published by only one publisher. The relationship 1:N.
- Staff keeps track of readers. The relationship is M:N.
- Staff maintains multiple reports. The relationship 1:N.
- Staff maintains multiple Books. The relationship 1:N.
- Authentication system provides login to multiple staffs. The relation is 1:N

CONCLUSION:

Thus, the data Designs using DFD Decision Table & ER Diagram is developed.

DRAW CLASS DIAGRAM, SEQUENCE DIAGRAM, COLLABORATION DIAGRAM, STATE TRANSITION DIAGRAM

AIM:

To draw class diagram, sequence diagram, collaboration diagram, state transition diagram for the book bank

CLASS DIAGRAM:

Class diagram is a static diagram. It represents the static view of an application. Class diagram is not only used for visualizing, describing, and documenting different aspects of a system but also for constructing executable code of the software application.

Class diagram describes the attributes and operations of a class and also the constraints imposed on the system. The class diagrams are widely used in the modeling of object-oriented systems because they are the only UML diagrams, which can be mapped directly with object-oriented languages.

Class diagram shows a collection of classes, interfaces, associations, collaborations, and constraints. It is also known as a structural diagram.

Purpose of Class Diagrams

The purpose of class diagram is to model the static view of an application. Class diagrams are the only diagrams which can be directly mapped with object-oriented languages and thus widely used at the time of construction.

UML diagrams like activity diagram, sequence diagram can only give the sequence flow of the application, however class diagram is a bit different. It is the most popular UML diagram in the coder community.

The purpose of the class diagram can be summarized as –

- Analysis and design of the static view of an application.
- Describe responsibilities of a system.
- Base for component and deployment diagrams.
- Forward and reverse engineering.

How to Draw a Class Diagram?

Class diagrams are the most popular UML diagrams used for construction of software applications. It is very important to learn the drawing procedure of class diagram. Class diagrams have a lot of properties to consider while drawing but here the diagram will be considered from a top level view. Class diagram is basically a graphical representation of the static view of the system and represents different aspects of the application. A collection of class diagrams represents the whole system.

The following points should be remembered while drawing a class diagram –

- The name of the class diagram should be meaningful to describe the aspect of the system.
- Each element and their relationships should be identified in advance.
- Responsibility (attributes and methods) of each class should be clearly identified
- For each class, minimum number of properties should be specified, as unnecessary properties will make the diagram complicated.
- Use notes whenever required to describe some aspect of the diagram. At the end of the drawing it should be understandable to the developer/coder.
- Finally, before making the final version, the diagram should be drawn on plain paper and reworked as many times as possible to make it correct.

Where to Use Class Diagrams?

Class diagram is a static diagram and it is used to model the static view of a system. The static view describes the vocabulary of the system. Class diagram is also considered as the foundation for component and deployment diagrams. Class diagrams are not only used to visualize the static view of the system but they are also used to construct the executable code for forward and reverse engineering of any system. Generally, UML diagrams are not directly mapped with any object-oriented programming languages but the class diagram is an exception.

Class diagram clearly shows the mapping with object-oriented languages such as Java, C++, etc. From practical experience, class diagram is generally used for construction purpose.

In a nutshell it can be said, class diagrams are used for –

- Describing the static view of the system.
- Showing the collaboration among the elements of the static view.

- Describing the functionalities performed by the system.
- Construction of software applications using object oriented languages.

The book bank system class diagram consists of four classes:

1. Student
2. Book
3. Issue
4. Return
5. Vendor
6. Details

1) Student:

It consists of twelve attributes and three operations. The attributes are enrollno, name, DOB, fathurname, address, dept name, batch and book limits. The operations of this class are addStInfo(), deleteStInfo(), modifyStInfo().

2) Book:

It consists of ten attributes and four operations. This class is used to keep book information such as author, title, vendor, price, etc

3) Issue:

It consists of eight attributes and two operations to maintain issue details such as, issue date, accno of issued book, name of the student who borrowed book.

4) Return:

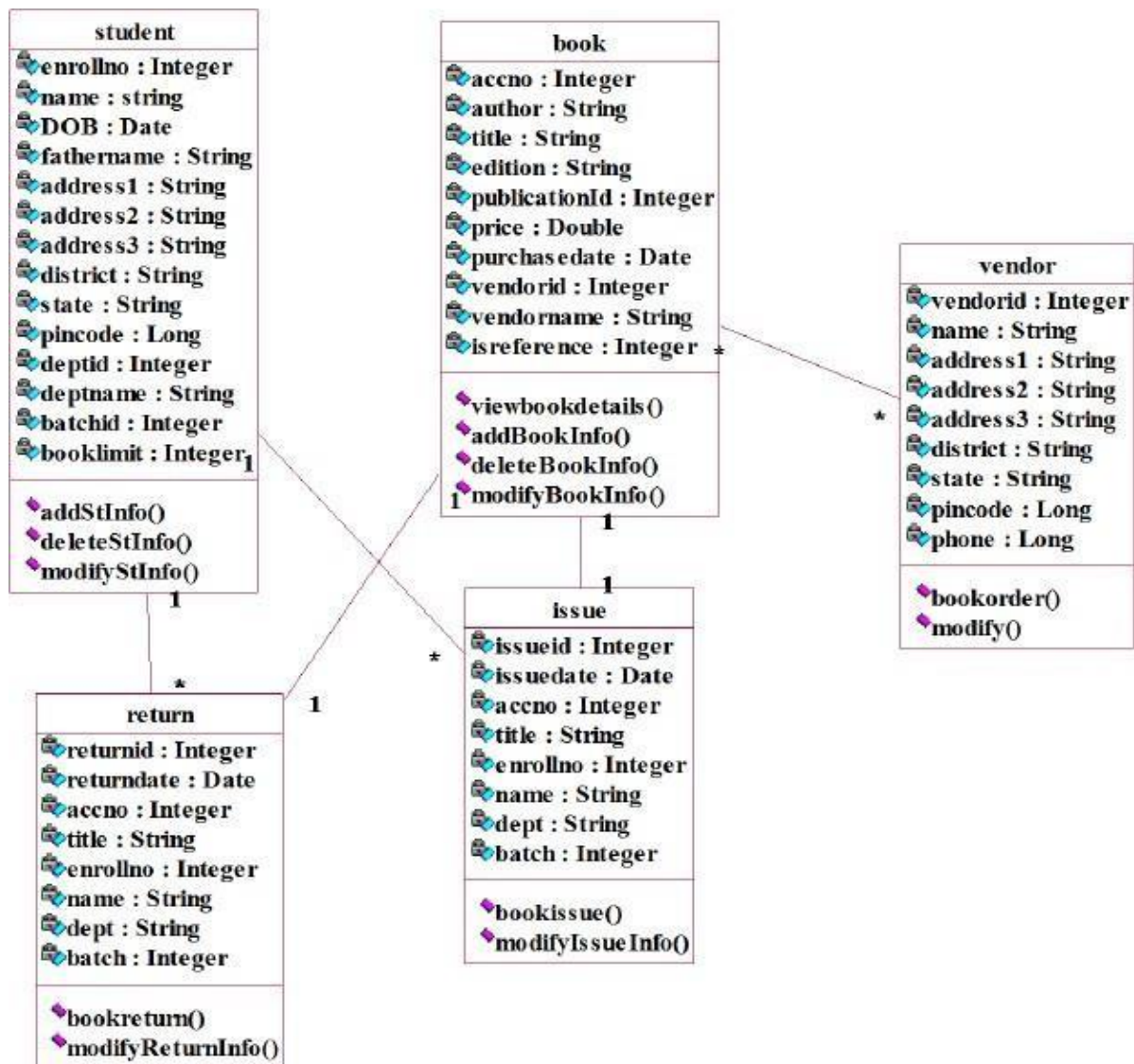
It consists of eight attributes and two operations to maintain issue details such as, issue date, accno of issued book, name of the student who borrowed book.

5) Students:

The attributes of this class are name, dept ,year ,bcode no The operation is display students().

6) Detail:

The attributes of this class are book name, author, bcode no The operations are delete details().



Class Diagram For Book Bank System

STATE CHART DIAGRAM

Used to describe a sequence of state transitions in an object in response to events

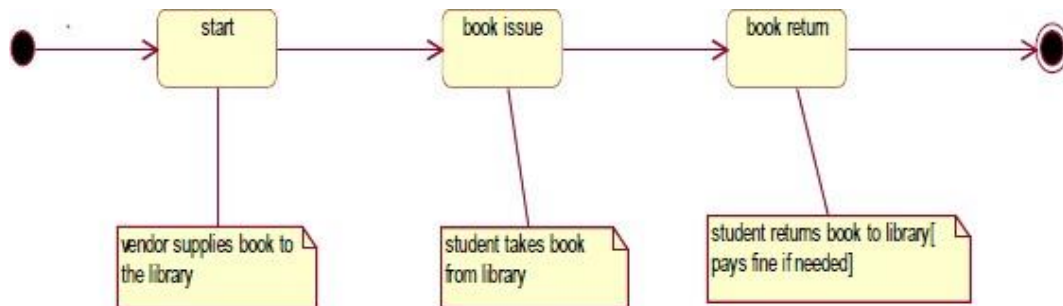
- Can help identify important object attributes
- Can help refine the behavior description of an object
- Should focus on the important attributes that affect the object's behavior
- Should be used to represent non-trivial behavior of a given object type (or can apply to a subsystem)

Notation

- Similar to a finite state machine diagram (computer theory)
- A *state*

- is a condition satisfied by the attributes of the object (i.e. determined by the current values in the attributes)
- is represented by a rounded rectangle
- *A transition*
 - is a change of state triggered by an external event, a condition, or perhaps time elapsed
 - is represented by a directed edge (arrow) between states
- Transitions should be labelled to indicate the triggering event

State Chart Diagram for BookBank System



INTERACTION DIAGRAMS

Interactive behavior is represented in UML by two diagrams known as **Sequence diagram** and **Collaboration diagram**. The basic purpose of both the diagrams are similar.

Sequence diagram emphasizes on time sequence of messages and collaboration diagram emphasizes on the structural organization of the objects that send and receive messages.

Purpose of Interaction Diagrams

The purpose of interaction diagrams is to visualize the interactive behavior of the system. Visualizing the interaction is a difficult task. Hence, the solution is to use different types of models to capture the different aspects of the interaction.

Sequence and collaboration diagrams are used to capture the dynamic nature but from a different angle.

The purpose of interaction diagram is –

- To capture the dynamic behaviour of a system.
- To describe the message flow in the system.
- To describe the structural organization of the objects.

- To describe the interaction among objects.

SEQUENCE DIAGRAM:

A sequence diagram represents the sequence and interactions of a given USE-CASE or scenario. Sequence diagrams can capture most of the information about the system. Most object to object interactions and operations are considered events and events include signals, inputs, decisions, interrupts, transitions and actions to or from users or external devices.

An event also is considered to be any action by an object that sends information. The event line represents a message sent from one object to another, in which the “from” object is requesting an operation be performed by the “to” object. The “to” object performs the operation using a method that the class contains.

It is also represented by the order in which things occur and how the objects in the system send message to one another.

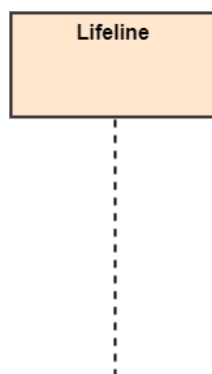
Purpose of a Sequence Diagram

1. To model high-level interaction among active objects within a system.
2. To model interaction among objects inside a collaboration realizing a use case.
3. It either models generic interactions or some certain instances of interaction.

Notations of a Sequence Diagram

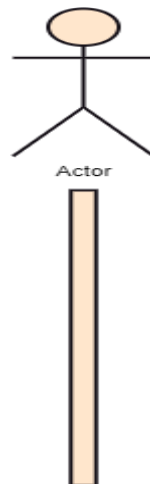
Lifeline

An individual participant in the sequence diagram is represented by a lifeline. It is positioned at the top of the diagram.



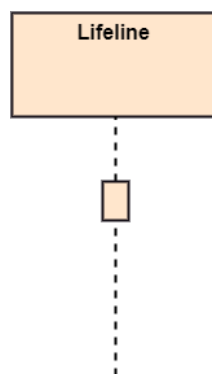
Actor

A role played by an entity that interacts with the subject is called as an actor. It is out of the scope of the system. It represents the role, which involves human users and external hardware or subjects. An actor may or may not represent a physical entity, but it purely depicts the role of an entity. Several distinct roles can be played by an actor or vice versa.



Activation

It is represented by a thin rectangle on the lifeline. It describes that time period in which an operation is performed by an element, such that the top and the bottom of the rectangle is associated with the initiation and the completion time, each respectively.

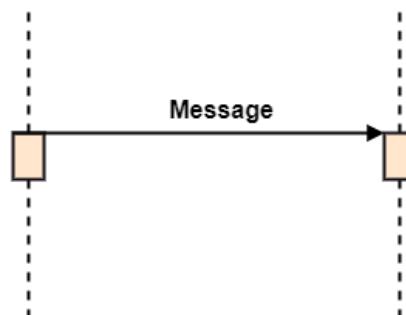


Messages

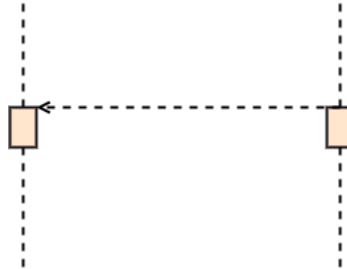
The messages depict the interaction between the objects and are represented by arrows. They are in the sequential order on the lifeline. The core of the sequence diagram is formed by messages and lifelines.

Following are types of messages enlisted below:

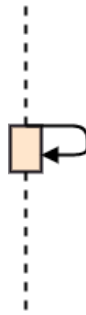
- **Call Message:** It defines a particular communication between the lifelines of an interaction, which represents that the target lifeline has invoked an operation.



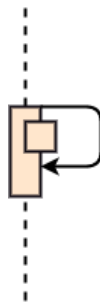
- **Return Message:** It defines a particular communication between the lifelines of interaction that represent the flow of information from the receiver of the corresponding caller message.



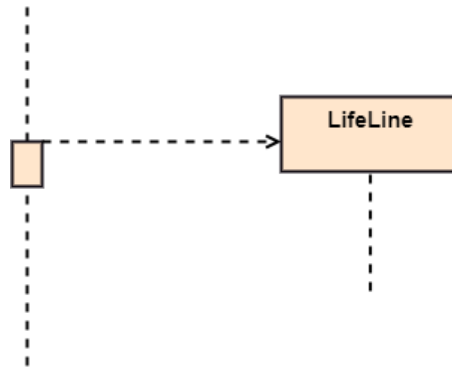
- **Self Message:** It describes a communication, particularly between the lifelines of an interaction that represents a message of the same lifeline, has been invoked.



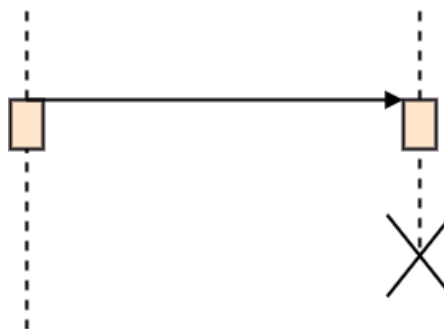
- **Recursive Message:** A self message sent for recursive purpose is called a recursive message. In other words, it can be said that the recursive message is a special case of the self message as it represents the recursive calls.



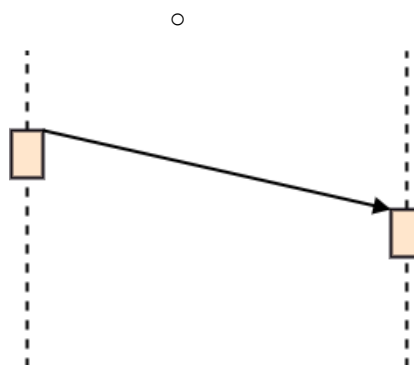
- **Create Message:** It describes a communication, particularly between the lifelines of an interaction describing that the target (lifeline) has been instantiated.



- **Destroy Message:** It describes a communication, particularly between the lifelines of an interaction that depicts a request to destroy the lifecycle of the target.



- **Duration Message:** It describes a communication particularly between the lifelines of an interaction, which portrays the time passage of the message while modeling a system.



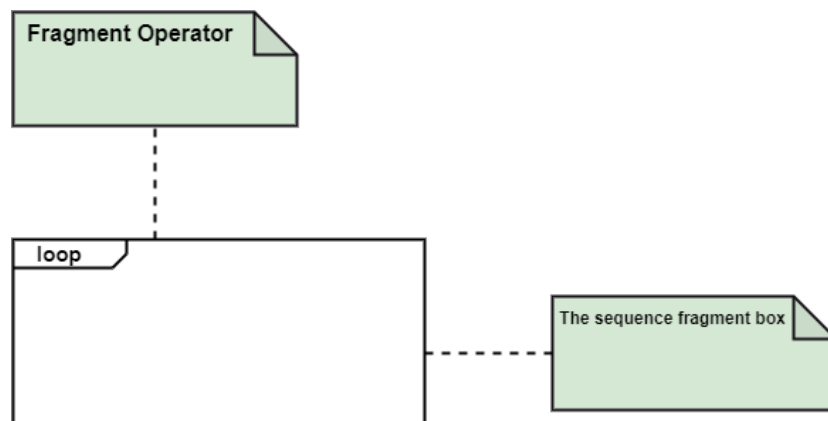
Note

A note is the capability of attaching several remarks to the element. It basically carries useful information for the modelers.

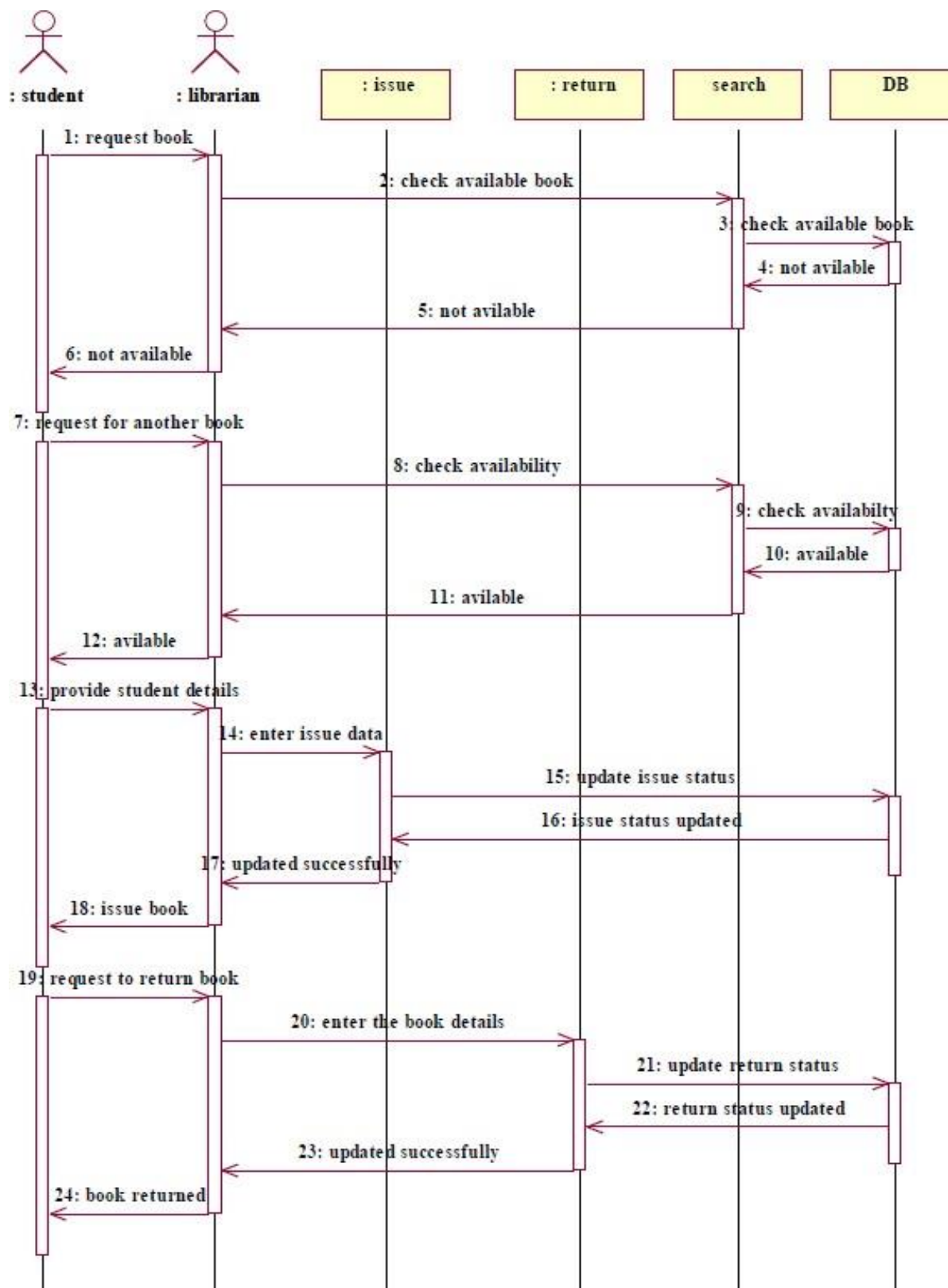


Sequence Fragments

1. Sequence fragments have been introduced by UML 2.0, which makes it quite easy for the creation and maintenance of an accurate sequence diagram.
2. It is represented by a box called a combined fragment, encloses a part of interaction inside a sequence diagram.
3. The type of fragment is shown by a fragment operator.



Sequence Diagram For Book Issue & Return



COLLABORATION DIAGRAM

The collaboration diagram is used to show the relationship between the objects in a system. Both the sequence and the collaboration diagrams represent the same information but differently. Instead of showing the flow of messages, it depicts the architecture of the object residing in the system as it is based on object-oriented programming. An object consists of several features. Multiple objects present in the system are connected to each other. The collaboration diagram, which is also known as a communication diagram, is used to portray the object's architecture in the system.

Notations of a Collaboration Diagram

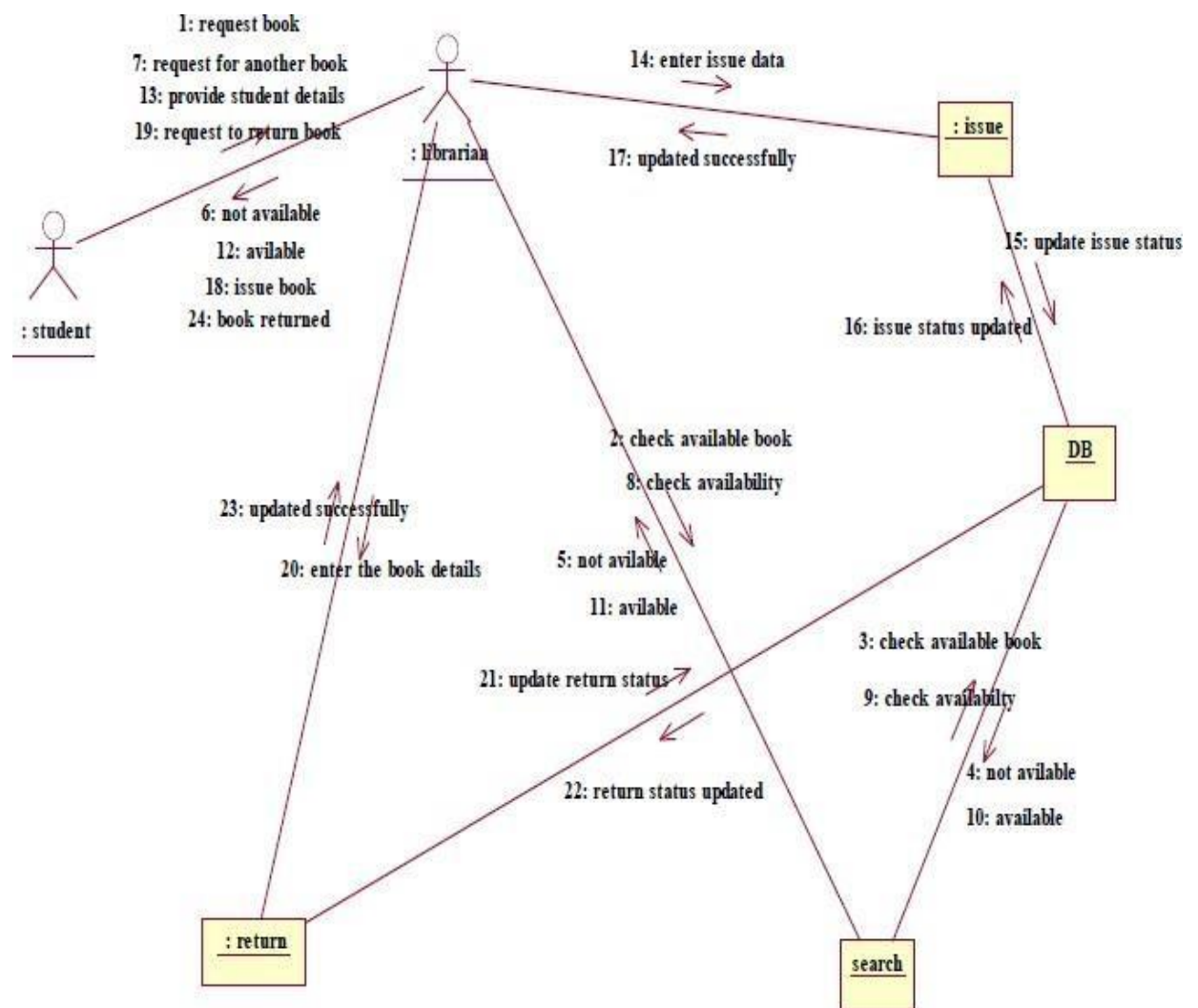
Following are the components of a component diagram that are enlisted below:

1. **Objects:** The representation of an object is done by an object symbol with its name and class underlined, separated by a colon.

In the collaboration diagram, objects are utilized in the following ways:

- The object is represented by specifying their name and class.
 - It is not mandatory for every class to appear.
 - A class may constitute more than one object.
 - In the collaboration diagram, firstly, the object is created, and then its class is specified.
 - To differentiate one object from another object, it is necessary to name them.
2. **Actors:** In the collaboration diagram, the actor plays the main role as it invokes the interaction. Each actor has its respective role and name. In this, one actor initiates the use case.
 3. **Links:** The link is an instance of association, which associates the objects and actors. It portrays a relationship between the objects through which the messages are sent. It is represented by a solid line. The link helps an object to connect with or navigate to another object, such that the message flows are attached to links.
 4. **Messages:** It is a communication between objects which carries information and includes a sequence number, so that the activity may take place. It is represented by a labelled arrow, which is placed near a link. The messages are sent from the sender to the receiver, and the direction must be navigable in that particular direction. The receiver must understand the message.

Collaboration Diagram For Book Issue & Return



DEPLOYMENT DIAGRAM AND COMPONENT DIAGRAM

Component diagrams are different in terms of nature and behavior. Component diagrams are used to model the physical aspects of a system. Now the question is, what are these physical aspects? Physical aspects are the elements such as executables, libraries, files, documents, etc. which reside in a node.

Component diagrams are used to visualize the organization and relationships among components in a system. These diagrams are also used to make executable systems.

Purpose of Component Diagrams

Component diagram is a special kind of diagram in UML. The purpose is also different from all other diagrams discussed so far. It does not describe the functionality of the system but it describes the components used to make those functionalities.

Thus from that point of view, component diagrams are used to visualize the physical components in a system. These components are libraries, packages, files, etc.

Component diagrams can also be described as a static implementation view of a system. Static implementation represents the organization of the components at a particular moment.

A single component diagram cannot represent the entire system but a collection of diagrams is used to represent the whole.

The purpose of the component diagram can be summarized as –

- Visualize the components of a system.
- Construct executables by using forward and reverse engineering.
- Describe the organization and relationships of the components.

DEPLOYMENT DIAGRAMS are used to visualize the topology of the physical components of a system, where the software components are deployed.

Deployment diagrams are used to describe the static deployment view of a system. Deployment diagrams consist of nodes and their relationships.

Purpose of Deployment Diagrams

The term Deployment itself describes the purpose of the diagram. Deployment diagrams are used for describing the hardware components, where software components are deployed. Component diagrams and deployment diagrams are closely related.

Component diagrams are used to describe the components and deployment diagrams shows how they are deployed in hardware.

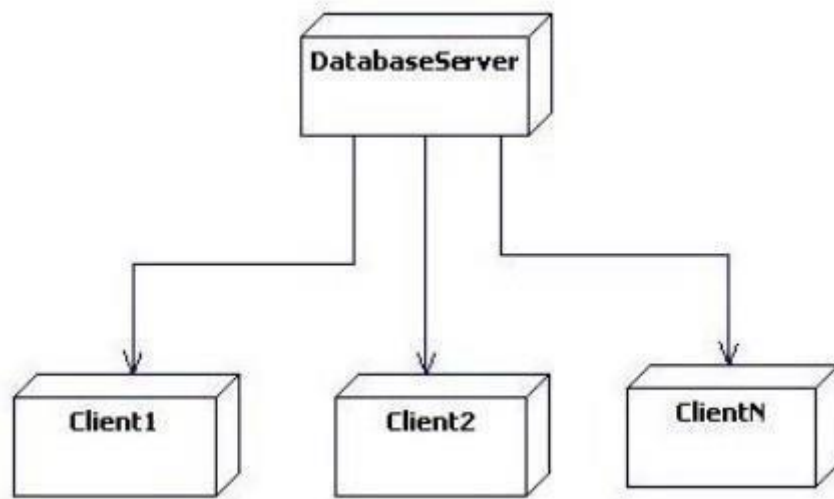
UML is mainly designed to focus on the software artifacts of a system. However, these two diagrams are special diagrams used to focus on software and hardware components.

Most of the UML diagrams are used to handle logical components but deployment diagrams are made to focus on the hardware topology of a system. Deployment diagrams are used by the system engineers.

The purpose of deployment diagrams can be described as –

- Visualize the hardware topology of a system.
- Describe the hardware components used to deploy software components.
- Describe the runtime processing nodes.

Deployment Diagram for Book Bank System



CONCLUSION:

Thus, the class diagram, sequence diagram, collaboration diagram, state transition diagram for the book bank is drawn.

WRITE TEST CASES TO VALIDATE REQUIREMENTS OF BOOK BANK FROM SRS DOCUMENT

AIM:

To Write Test Cases to Validate requirements of book bank from SRS Document

WHAT IS A TEST CASE?

A test case has components that describe input, action, and an expected response, in order to determine if a feature of an application works correctly.

A test case is a set of instructions on “HOW” to validate a particular test objective/target, which, when followed will tell us if the expected behavior of the system is satisfied or not.

The basic objective of writing cases is to validate the test coverage of an application.

In the library management system the test case are

- 1)verify all the books are in ascending order
- 2)verify the functionality of login field
- 3)verify how max no customer create the account
- 4)verify the acceptability when customer chose same books.
- 5)verify the extra charge executed if customer late to submit the book
- 6)verify the finding button pf books its working properly it accepted input like numeric, alphanumeric, special character, character,
- 7)verify the name field accepted max character in name field
- 8)verify the status of book field its define the book status with their availability
- 9)verify the customer info field that define the all info when customer issue the book and when submit the book
- 10)verify the price field of books if customer want to purchase the book

LOGIN FORM:

SL.No	Test Case	Excepted Result	Test Result
1	Enter valid name and password & click on login button	Software should display main window	Successful
2	Enter invalid	Software should not display main window	successful

BOOK ENTRY FORM:

SL.No	Test Case	Excepted Result	Test Result
1	On the click of ADD button	At first user have to fill all fields with proper data , if any Error like entering text data instead of number or entering number instead of text..is found then it gives proper message otherwise Adds Record To the Database	successful
2.	On the Click of DELETE Button	This deletes the details of book by using Accession no.	Successful
3.	On the Click of UPDATE Button	Modified records are Updated in database by clicking UPDATE button.	Successful
4.	On the Click of SEARCH Button	Displays the Details of book for entered Accession no. Otherwise gives proper Error message.	Successful
5.	On the Click of CLEAR Button	Clears all fields	Successful
6.	On the Click of EXIT button	Exit the current book details form	successful
7.	On the Click of NEXT button	Display the next form	successful

USER ACCOUNT FORM:

SL.No	Test Case	Excepted Result	Test Result
1	On the click of ADD button	At first user have to fill all fields with proper data , if any Error like entering text data instead of number or entering number instead of text..is found then it gives proper message otherwise Adds Record To the Database	successful
2.	On the Click of DELETE Button	This deletes the details of student by using Register no.	Successful
3.	On the Click of UPDATE Button	Modified records are Updated in database by clicking UPDATE button.	Successful

4.	On the Click of SEARCH Button	Displays the Details of book for entered Register no. Otherwise gives proper Error message.	Successful
5.	On the Click of CLEAR Button	Clears all fields	Successful
6.	On the Click of EXIT button	Exit the current book details form	successful
7.	On the Click of NEXT button	Display the next form	successful

BOOK ISSUE FORM:

SL.No	Test Case	Excepted Result	Test Result
1	On the click of ADD button	At first user have to fill all fields with proper data ,if the accession number book is already issued then it will giving proper msg.	successful
2.	On the Click of DELETE Button	This deletes the details of book by using Register no.	Successful
3.	On the Click of UPDATE Button	Modified records are Updated in database by clicking UPDATE button.	Successful
4.	On the Click of SEARCH Button	Displays the Details of issued book..Otherwise gives proper Error message.	Successful
5.	On the Click of CLEAR Button	Clears all fields	Successful
6.	On the Click of EXIT button	Exit the current book details form	successful
7.	On the Click of NEXT button	Display the next form	successful

BOOK RETURN FORM:

SL.No	Test Case	Excepted Result	Test Result
1	On the click of ADD button	At first user have to fill all fields with proper data , if any Error like entering text data instead of number or entering number instead of text..is found then it gives proper message otherwise Adds Record To the Database	successful
2.	On the Click of DELETE Button	Which deletes the details of book by using Register no.	Successful
3.	On the Click of UPDATE Button	Modified records are Updated in database by clicking UPDATE button.	Successful

4.	On the Click of SEARCH Button	Displays the Details of returned book ... Otherwise gives proper Error message.	Successful
5.	On the Click of CLEAR Button	Clears all fields	Successful
6.	On the Click of EXIT button	Exit the current book details form	successful
7.	On the Click of NEXT button	Display the next form	successful

CONCLUSION:

Thus, the test cases to validate requirements of assigned project from SRS document for the book bank is written.

EVALUATE SIZE OF THE PROJECT USING FUNCTION POINT METRIC FOR THE BOOK BANK

AIM:

To Evaluate Size of the project using function point metric for the book bank from SRS Document

ESTIMATION OF THE SIZE OF THE SOFTWARE is an essential part of Software Project Management. It helps the project manager to further predict the effort and time which will be needed to build the project. Various measures are used in project size estimation. Some of these are:

- Lines of Code
- Number of entities in ER diagram
- Total number of processes in detailed data flow diagram
- Function points

FUNCTION POINT ANALYSIS: In this method, the number and type of functions supported by the software are utilized to find FPC(function point count). The steps in function point analysis are:

Count the number of functions of each proposed type.

- Compute the Unadjusted Function Points(UFP).
- Find Total Degree of Influence(TDI).
- Compute Value Adjustment Factor(VAF).
- Find the Function Point Count(FPC).

The explanation of the above points is given below:

- **Count the number of functions of each proposed type:** Find the number of functions belonging to the following types:
 - External Inputs: Functions related to data entering the system.
 - External outputs: Functions related to data exiting the system.
 - External Inquiries: They lead to data retrieval from the system but don't change the system.
 - Internal Files: Logical files maintained within the system. Log files are not included here.

- **External interface Files:** These are logical files for other applications which are used by our system.
- **Compute the Unadjusted Function Points(UFP):** Categorise each of the five function types like simple, average, or complex based on their complexity. Multiply the count of each function type with its weighting factor and find the weighted sum. The weighting factors for each type based on their complexity are as follows:

Function type	Simple	Average	Complex
External Inputs	3	4	6
External Output	4	5	7
External Inquiries	3	4	6
Internal Logical Files	7	10	15
External Interface Files	5	7	10

- **Find Total Degree of Influence:** Use the '14 general characteristics' of a system to find the degree of influence of each of them. The sum of all 14 degrees of influence will give the TDI. The range of TDI is 0 to 70. The 14 general characteristics are: Data Communications, Distributed Data Processing, Performance, Heavily Used Configuration, Transaction Rate, On-Line Data Entry, End-user Efficiency, Online Update, Complex Processing Reusability, Installation Ease, Operational Ease, Multiple Sites and Facilitate Change. Each of the above characteristics is evaluated on a scale of 0-5.

Compute Value Adjustment Factor(VAF): Use the following formula to calculate VAF

$$VAF = (TDI * 0.01) + 0.65$$

- **Find the Function Point Count:** Use the following formula to calculate FPC

$$FPC = UFP * VAF$$

Advantages:

- It can be easily used in the early stages of project planning.
- It is independent of the programming language.
- It can be used to compare different projects even if they use different technologies(database, language, etc).

Disadvantages:

- It is not good for real-time systems and embedded systems.
- Many cost estimation models like COCOMO uses LOC and hence FPC must be converted to LOC.

CONCLUSION:

Thus, the test cases to validate requirements of the book bank is done.

PROJECT MANAGEMENT USING GANTTPROJECT

AIM-

Project management using GanttProject

Objective:

To show how Project management is carried out in GanttProject.

Software Required:-

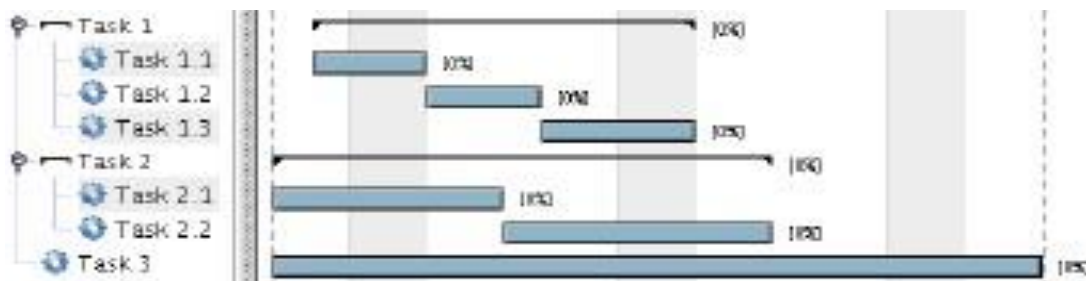
GanttProject

OVERALL DESCRIPTION:

Gantt Project is an open source framework used to perform planning, scheduling and resource allocation activities.

TASK CREATION

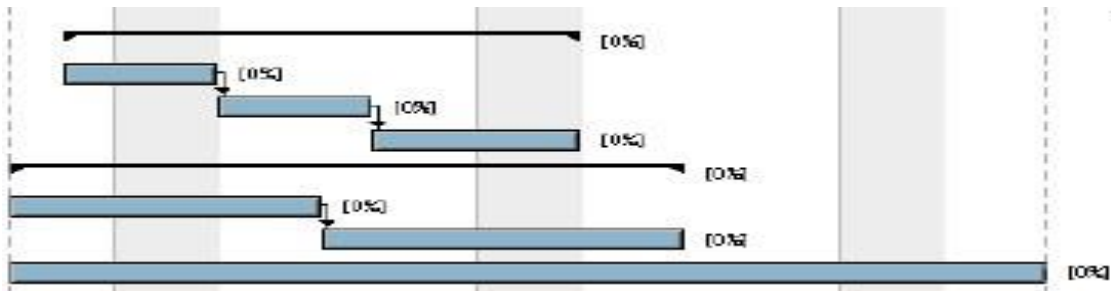
First, you create some tasks by using the New Task button or directly from the Tasks menu choose New Task. The tasks appear on the tree on the left pane; you can directly change their name here. Next, you can organize tasks by indenting them forming groups or categories. So you could have a hierarchy like this:



Tasks can also be re-organized by using the up and down functions. These functions move the selected task up or down in its hierarchy reordering it.

RELATIONSHIPS

Gantt project allows you to specify a relationship between two tasks. You can set them by dragging directly on the chart. Click and hold on the first task and moving the cursor to the second task. An arrow will appear, following the mouse. Drag the arrowhead to the second task and release the mouse. The second task will be dependent on the first one. You will have a chart like this:



EDITING PROPERTIES

For each task you can edit the properties in a dialog box, by using the Properties menu, or by double-clicking on either the task's name, or it's Gantt bar. The properties box allows you to edit the name of the task, the duration, the percent complete, the start and end dates, the color on the chart, the priority, and the explanatory notes. You can also define the relationship between tasks by choosing different predecessors. You do this by selecting the second panel of the box and choosing the name of the predecessor task, and the type of relationship.

CREATING RESOURCES

A project is composed of tasks and people (or resources) who are assigned to each task. You can create resources on the Resources panel by specifying the name, the function and contact information (mail or phone by example).

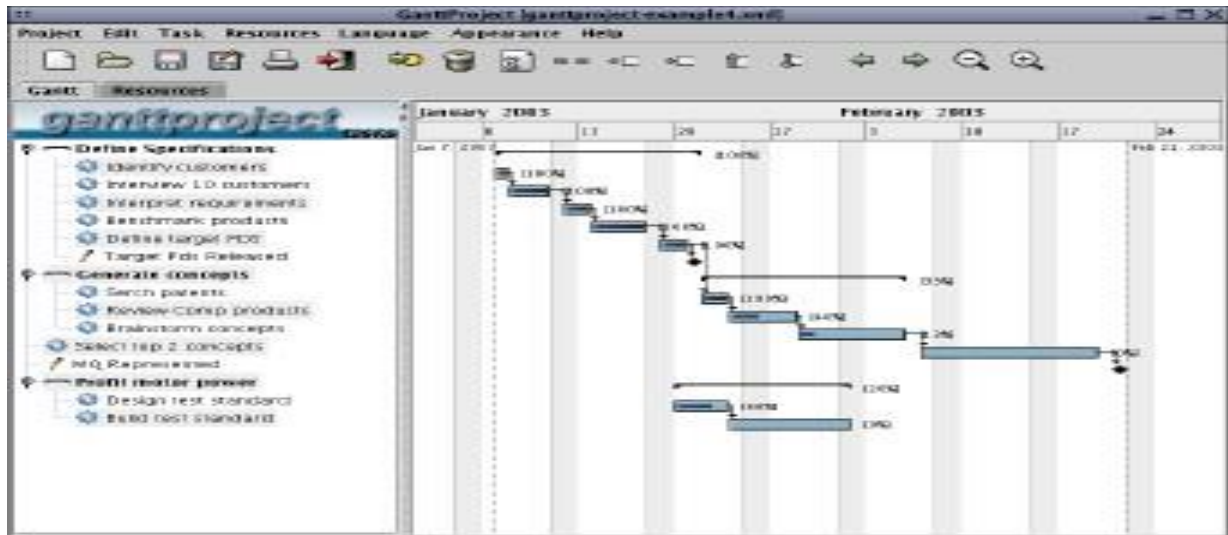
ASSIGN TO TASKS

A resource can be assigned to a task directly on the properties dialog box of the task. Select the third tabbed panel and choose the name of the resource you want to assign. Then, specify a unit for the resources

RESOURCES CHART

A special chart is available for all resources on the panel. It shows the resource time allocation and is similar to the Gantt Chart.

Here are some snapshots of Gantt Project:



CONCLUSION:

The Gantt chart was made successfully by following the steps described above.