SYSTEM ANALYSIS AND DESIGN

Unit I

Overview of System Analysis And Design

Introduction to system analysis and design

Information systems analysis and design is a method used by companies ranging from IBM to PepsiCo to Sony to create and maintain information systems that perform basic business functions such as keeping track of customer names and addresses, processing orders, and paying employees. The main goal of systems analysis and design is to improve organizational systems, typically through applying software that can help employees accomplish key business tasks more easily and efficiently. As a systems analyst, you will be at the center of developing this software. The analysis and design of information systems are based on:

Understanding of the organization's objectives, structure, and processes

Your knowledge of how to exploit information technology for advantage

Definition of a System and Its Parts

A **system** is an interrelated set of business procedures (or components) used within one business unit, working together for some purpose. For example, a system in the payroll department keeps track of checks, whereas an inventory system keeps track of supplies. The two systems are separate. A system has nine characteristics, seven of which are shown in Figure 1-4. A detailed explanation of each characteristic follows, but from the figure you can see that a system exists within a larger world, an environment. A boundary separates the system from its environment. The system takes input from outside, processes it, and sends the resulting output back to its environment. The arrows in the figure show this interaction between the system and the world outside of it.

1. Components An irreducible part or aggregation of parts that makes up a system; also called a subsystem.

2. Interrelated Components: Dependence of one part of the system on one or more other system parts

3. Boundary- The line that marks the inside and outside of a system and that sets off the system from its environment

4. Purpose- The overall goal or function of a system

5. Environment -Everything external to a system that interacts with the system

6. Interfaces-Point of contact where a system meets its environment or where subsystems meet each other.

7.Constraints -A limit to what a system can accomplish.

8. Input 9. Output

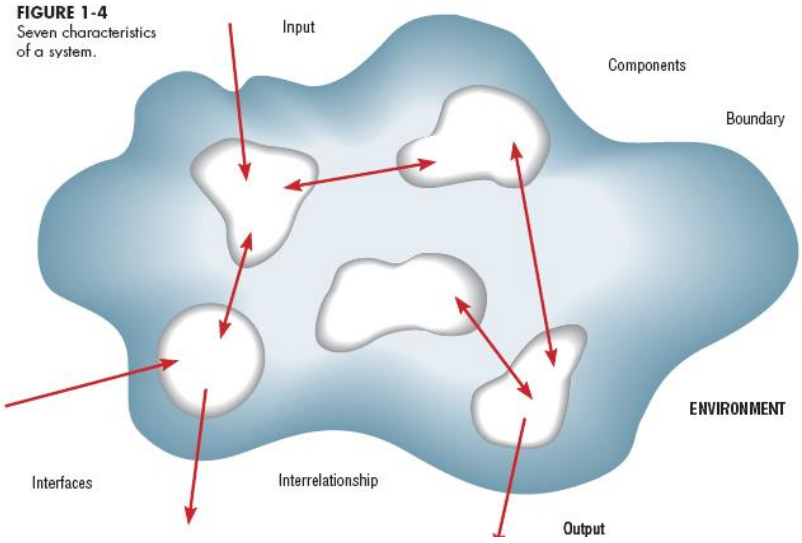
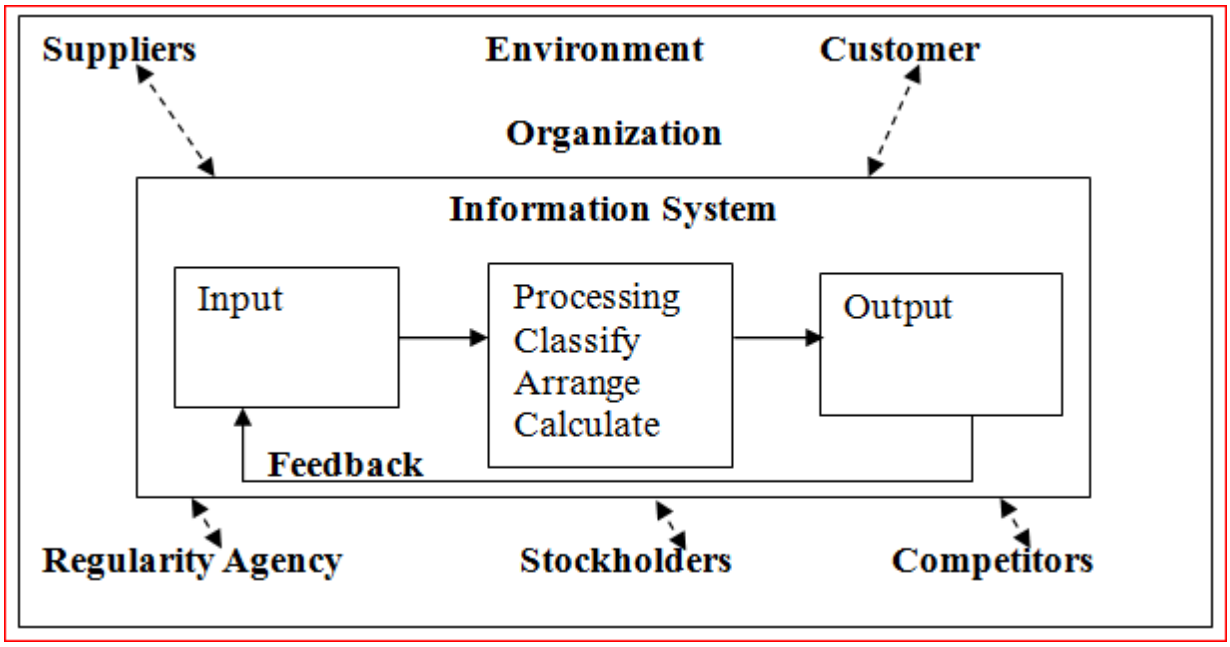


FIGURE 1-4
Seven characteristics of a system.

INFORMATION SYSTEM AND ITS TYPES:

Information system(IS):

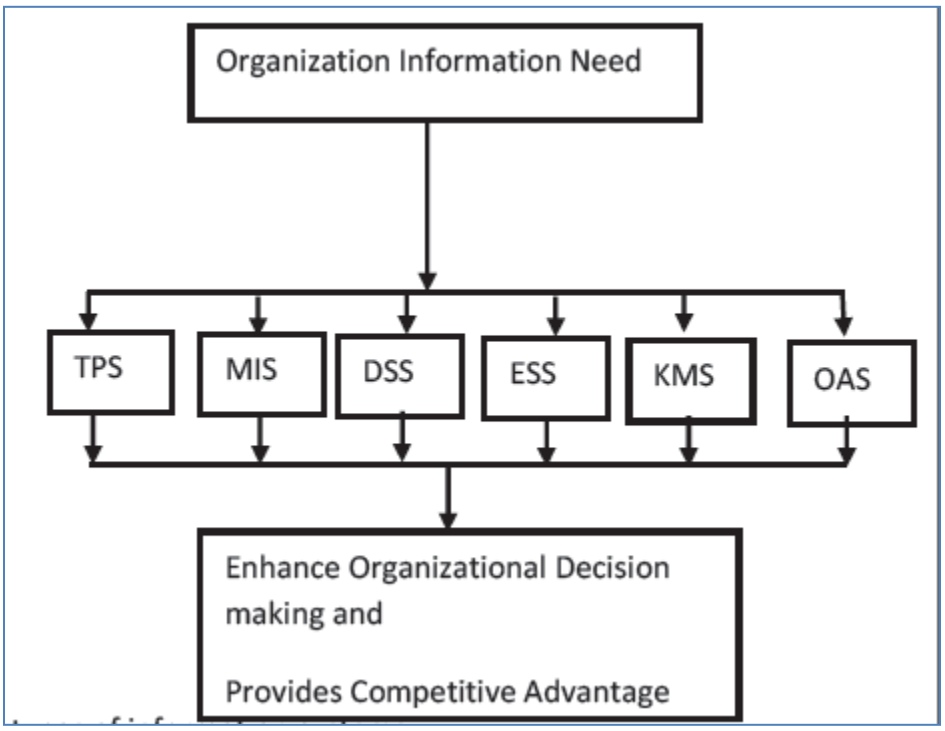
An information system can be defined technically as a set of interrelated components that collect or retrieve, process, store and distribute information to support decision making and control in an organization. In addition to supporting decision making, coordination and control, information system may also help managers and workers analyze problem visualization complex subjects, and create new products



An information system contains information about organizations and its surrounding environment. Three basic activities; input, processing, output produce the information organization need. Feedback is output returned appropriate people or activities in the organization to evaluate and refine the input environmental factors such as customers, suppliers, competitors, stockholders, and regulatory agencies interact with the organization and its information.

o **Input:** It is used to capture or collects raw data from within the organization of from its external environments for processing in an information system.
o **Processing:** The conversion, manipulation and analysis of raw input into a form that is more meaningful to humans.
o **Output:** The distribution of processed information to the people who will use it or to the activities for which it will be used.
o **Feedback:** Output that is returned to appropriate members of the organization to help them evaluate or correct input

Major types of Information System:
- Executive Support System(ESS)
- Management Information System(MIS)
- Knowledge Management Systems
- Decision-Support System(DSS)
- Office automation System(OAS)
- Transaction processing system(TPS).



1.Transaction processing System(TPS):
A transaction process system is information system that record internal and external transaction .A TPS meets the need of operation managers ; the output of the TPS become the input to an MIS.TPS normally records company transaction at operation level. A transaction processing systems is a computerized system performs and records the daily routine transactions necessary to conduct
business. For examples are sales order entry, hotel reservation systems, payroll, employee record keeping, and shipping. At the operational level, tasks, resources, and goals are predefined and highly structured. The decision to grant credit to a customer for instance is made by a lower level supervisor according to predefined criteria. All that must be determined is whether the customer meet the criteria.
Characteristic of transaction processing system:

 A TPS records internal and external transaction for a company. It is repository of data that is frequently accessed by other system.
 A TPS performs routine, repetitive tasks. It is mostly used by lower level manager to make operational decisions.
 Transaction can be recorded in batch mode or online. In batch mode, the files are updated automatically; in online , transaction is recorded at it occurs.

2. Management Information System:

Management information system is the study of information and impact on the individual, the organization, and society also, systems that create, process, store, and retrieve information. A system is a collection of parts that work together to achieve a common goal. The primary goal of MIS is to support organizational decision making. It is well-integrated system that meet tactical information needs of middle managers. These system generate summary and exception report. MIS serve the management level of the organization that serve the function of planning, controlling and decision making by providing routine summary and exception report.

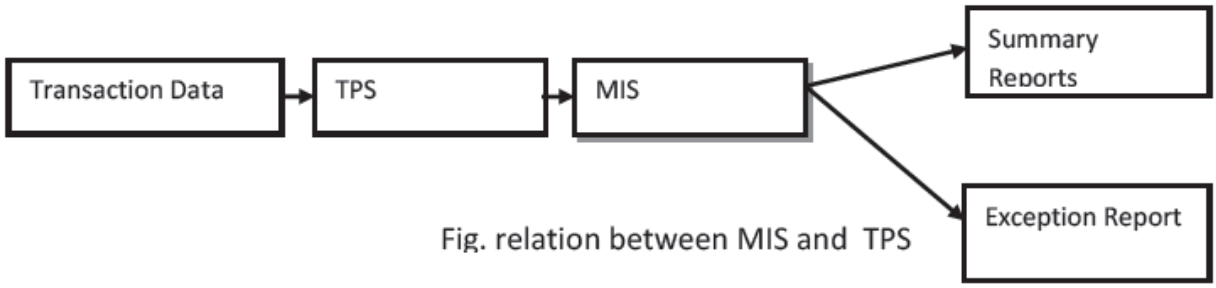
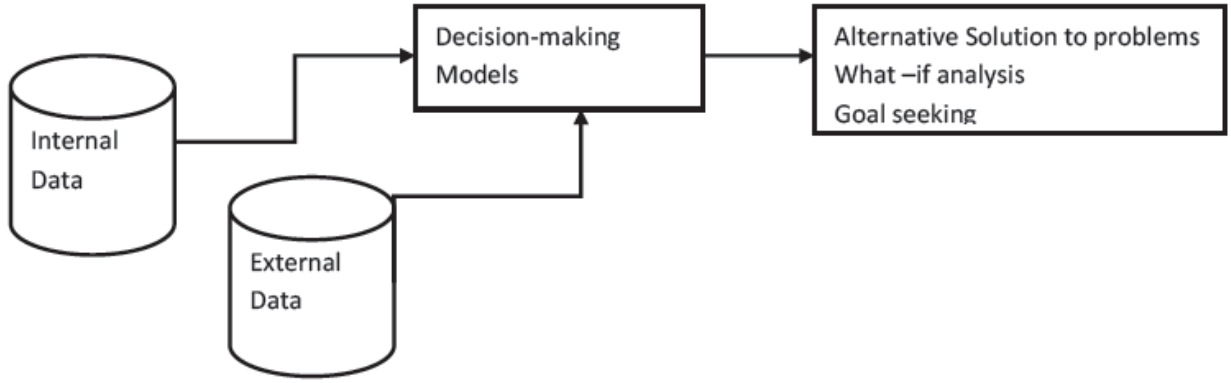


Fig. relation between MIS and TPS

3. Decision support system (DSS):

A set of well integrated, user friendly, computer-based tools that combine internal and external data with various decisions making models to solve semi-structured and unstructured problems. Among the functions of a DSS are "What if" analysis, model building, goal seeking, and graphical analysis. DSS is a set of well-integrated, user-friendly, computer-based tools that combine data with various decision-making models (both quantitative and qualitative) to solve semi-structured and unstructured problems. Such systems solve a specific problem or a class of problems, such as scheduling, planning, resource allocation and forecasting. They allow managers to ask ad-hoc (non routine) queries and receive customized responses. Although DSS were initially targeted for top managers and middle managers, today these systems are being deployed at all levels of the organization



4. Executive information System (EIS):

It is primarily used by top level management, is user friendly, interactive system, designed to meet information needs of top management engaged in long-range planning, crisis management, and other strategic decision(unique, non-repetitive and future oriented), which address ling-term issues such as emerging markets, merger and acquisition strategies, new-product development and investment strategies. Such system assist in the making of decision that require an in-depth understanding of the firm and of the industry in which the firm operates.

5.Office automation system(OAS):

Computer system, such as word processing, electronic mail systems, and scheduling systems, that are designed to increase the productivity of data workers in office the automation of everyday office tasks is one of the key results of the electronic revolution. Word processors, spreadsheets, databases, accounting packages, networks, and e-mail are but some of the innovations that have

transformed the way we work in the late twentieth century.

6. Knowledge Management Systems

Knowledge Management Systems ("KMS") exist to help businesses create and share information. These are typically used in a business where employees create new knowledge and expertise which can then be shared by other people in the organization to create further commercial opportunities. Good examples include firms of lawyers, accountants and management consultants. KMS are built around systems which allow efficient categorization and distribution of knowledge. For example, the knowledge itself might be contained in word processing documents, spreadsheets, PowerPoint presentations. internet pages or whatever. To share the knowledge, a KMS would use group collaboration systems such as an intranet

Stakeholders, The Players of an Information System

Stakeholder is a person or organization who directly or indirectly take part or involve in information system . here are different classes of actors who have a stake in the success of an information system. These are the system stakeholders. Since one person can play several roles, it is possible for one person to be a member of several of the stakeholder classes.

External Stakeholders

The external stakeholders are the actors in the environment of the organization, and therefore in the environment of the system, that interact with the system. They include:

Customers (Clients)

System customers are actors that stimulate our system to provide a service to the customer. Customers are also external system users.

Vendors (Servers)

System vendors are actors that are stimulated by our system to provide a service to our system.

Internal Stakeholders

The internal stakeholders are actors inside the organization boundary.

System Sponsors

The system sponsors pay for the system to be built and maintained. They provide that justification for the system and determine policies for its use. System sponsors may also be system users. Systems sponsors generally come from the ranks of executives or managers.

System Users

The system users are the people who actually use the system on a regular basis to support the operation and management of the organization. System users are also internal system customers. Systems users come from all levels of the organization.

System Analysts

Systems analysts are people who determine the requirements that must be met by the system to meets the needs of the customers, sponsors, and users.

Systems Designers

Systems designers are technical specialists who select appropriate technologies andmay the essential requirements into practical requirements.

System Builders

Systems builders are technical specialists who build, test, and deliver the system.

Duties Of A System Analyst:

An individual who analyses a system by the use of scientific techniques in order to determine where and how improvements can be made with a view to meet objectives in a more efficient, effective and economical manner.

Duties of systems analyst are:

a) Investigating the existing information usage. Systems and procedures of the organization with a view of discovering inefficiencies and problems.

b) After investigating the problems . the system analyst states the objectives of the system to make requirements clear and confirm.

c) To meet the objectives, the systems analyst gathers data, facts and opinions of users. For it he may use various fact finding methods.

d) Analyzing the findings of the investigation so that they can be used effectively in designing new systems.

e) Systems analysts coordinate the process of developing solutions. Since, many problems have many solutions; the systems analyst must evaluate the merit of each proposed solution before recommending one of the user sufficiencies. Minimizes problems and achieves the objectives set for it.

g) Testing and implementing the new system, its documentation and continuing maintenance

SYTEM DEVELOPMENT LIFE CYCLE AND LIFE CYCLE MODEL

A software life cycle model (also called process model) is a descriptive and diagrammatic representation of the software life cycle. A life cycle model represents all the activities required to make a software product transit through its life cycle phases. It also captures the order in which these activities are to be undertaken. In other words, a life cycle model maps the different activities performed on a software product from its inception to retirement. Different life cycle models may map the basic development activities to phases in different ways. Thus, no matter which life cycle model is followed, the basic activities are included in all life cycle models though the activities may be carried out in different orders in different life cycle models. During any life cycle phase, more than one activity may also be carried out. For example, the design phase might consist of the structured analysis activity followed by the structured design activity.

The need for a software life cycle model

The development team must identify a suitable life cycle model for the particular project and then adhere to it. Without using of a particular life cycle model the development of a software product would not be in a systematic and disciplined manner. When a software product is being developed by a team there must be a clear understanding among team members about when and what to do. Otherwise it would lead to chaos and project failure. This problem can be illustrated by using an example. Suppose a software development problem is divided into several parts and the parts are assigned to the team members. From then on, suppose the team members are allowed the freedom to develop the parts assigned to them in whatever way they like. It is possible that one member might start writing the code for his part, another might decide to prepare the test documents first, and some other engineer might begin with the design phase of the parts assigned to him. This would be one of the perfect recipes for project failure.

A software life cycle model defines entry and exit criteria for every phase. A phase can start only if its phase-entry criteria have been satisfied. So without software life cycle model the entry and exit criteria for a phase cannot be recognized. Without software life cycle models (such as classical waterfall model, iterative waterfall model, prototyping model, evolutionary model, spiral model etc.) it becomes difficult for software project managers to monitor the progress of the project.

Different software life cycle models

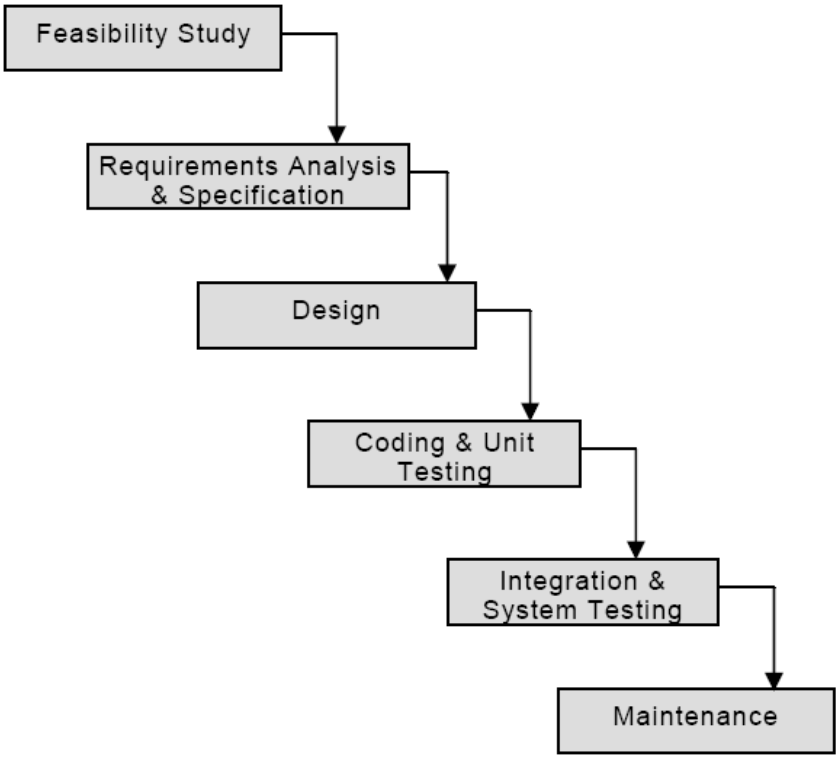
Many life cycle models have been proposed so far. Each of them has some advantages as well as some disadvantages. A few important and commonly used life cycle models are as follows:

- Classical Waterfall Model
- Iterative Waterfall Model
- Prototyping Model
- Evolutionary Model
- Spiral Model

Different phases of the classical waterfall model

The classical waterfall model is intuitively the most obvious way to develop software. Though the classical waterfall model is elegant and intuitively obvious, it is not a practical model in the sense that it can not be used in actual software development projects. Thus, this model can be considered to be a *theoretical way of developing software*. But all other life cycle models are essentially derived from the classical waterfall model.

So, in order to be able to appreciate other life cycle models it is necessary to learn the classical waterfall model.
Classical waterfall model divides the life cycle into the following phases as
shown in fig.
 Feasibility Study
 Requirements Analysis and Specification
 Design
 Coding and Unit Testing
 Integration and System Testing
 Maintenance



Activities in each phase of the life cycle

• Activities undertaken during feasibility study: -

The main aim of feasibility study is to determine whether it would be financially and technically feasible to develop the product. At first project managers or team leaders try to have a rough understanding of what is required to be done by visiting the client side. They study different input data to the system and output data to be produced by the system. They study what kind of processing is needed to be done on these data and they look at the various constraints on the behavior of the system.

 After they have an overall understanding of the problem they investigate the different solutions that are possible. Then they examine each of the solutions in terms of what kind of resources required, what would be the cost of development and what would be the development time for each solution.

 Based on this analysis they pick the best solution and determine whether the solution is feasible financially and technically. They check whether the customer budget would meet the cost of the product and whether they have sufficient technical expertise in the area of development.

Activities undertaken during requirements analysis and specification: -

The aim of the requirements analysis and specification phase is to understand the exact requirements of the customer and to document them properly. This phase consists of two distinct activities, namely

 Requirements gathering and analysis, and
 Requirements specification

The goal of the requirements gathering activity is to collect all relevant information from the customer regarding the product to be developed. This is done to clearly understand the customer requirements so that incompleteness and inconsistencies are removed.
The requirements analysis activity is begun by collecting all relevant data regarding the product to be developed from the users of the product and from the customer through interviews and discussions. For example, to perform the requirements analysis of a business accounting software required by an organization, the analyst might
interview all the accountants of the organization to ascertain their requirements. The data collected from such a group of users usually contain several contradictions and ambiguities, since each user typically has only a partial and incomplete view of the system. Therefore it is necessary to identify all ambiguities and contradictions in the requirements and resolve them through further discussions with the customer. After all ambiguities, inconsistencies, and incompleteness have been resolved and all the requirements properly understood, the requirements specification activity can start. During
this activity, the user requirements are systematically organized into a Software Requirements Specification (SRS) document. The customer requirements identified during the requirements gathering and analysis activity are organized into a SRS document.
The important components of this document are functional requirements, the nonfunctional requirements, and the goals of implementation.

Activities undertaken during design: -
The goal of the design phase is to transform the requirements specified in the SRS document into a structure that is suitable for implementation in some programming language. In technical terms, during the design phase the software architecture is derived from the SRS document. Two distinctly different approaches are available: the traditional
design approach and the object-oriented design approach.
 Traditional design approach
Traditional design consists of two different activities; first a structured analysis of the requirements specification is carried out where the detailed structure of the problem is examined. This is followed by a structured design activity. During structured design, the results of structured analysis are transformed into the software design.
 Object-oriented design approach
In this technique, various objects that occur in the problem domain and the solution domain are first identified, and the different relationships that exist among these objects are identified. The object structure is further refined to obtain the detailed design.
• Activities undertaken during coding and unit testing:-
The purpose of the coding and unit testing phase (sometimes called the implementation phase) of software development is to translate the software design into source code. Each component of the design is implemented as a program module. The end-product of this phase is a set of program modules that have been individually tested. During this phase, each module is unit tested to determine the correct working of all the individual modules. It involves testing each module in isolation as this is the most efficient way to debug the errors identified at this stage.
• Activities undertaken during integration and system testing: -
Integration of different modules is undertaken once they have been coded and unit tested. During the integration and system testing phase, the modules are integrated in a planned manner. The different modules making up a software product are almost never integrated in one shot. Integration is normally carried out incrementally over a number of steps. During each integration step, the partially integrated system is tested and a set of previously planned modules are added to it. Finally, when all the modules have been successfully integrated and tested, system testing is carried out. The goal of system testing is to ensure that the developed system conforms to its requirements laid out in the SRS document.
System testing usually consists of three different kinds of testing activities:
 α – testing: It is the system testing performed by the development team.
 β – testing: It is the system testing performed by a friendly set of customers.
 acceptance testing: It is the system testing performed by the customer himself after the product delivery to determine whether to accept or reject the delivered product. System testing is normally carried out in a

planned manner according to the system test plan document. The system test plan identifies all testing related activities that must be performed, specifies the schedule of testing, and allocates resources. It also lists all the test cases and the expected outputs for each test case.

• **Activities undertaken during maintenance: -**

Maintenance of a typical software product requires much more than the effort necessary to develop the product itself. Many studies carried out in the past confirm this and indicate that the relative effort of development of a typical software product to its maintenance effort is roughly in the 40:60 ratio. Maintenance involves performing any one or more of the following three kinds of activities:

 Correcting errors that were not discovered during the product development phase. This is called corrective maintenance. Improving the implementation of the system, and enhancing the functionalities of the system according to the customer's requirements. This is called perfective maintenance.

 Porting the software to work in a new environment. For example, porting may be required to get the software to work on a new computer platform or with a new operating system. This is called adaptive maintenance.

Shortcomings of the classical waterfall model

The classical waterfall model is an idealistic one since it assumes that no development error is ever committed by the engineers during any of the life cycle phases. However, in practical development environments, engineers do commit a large number of errors in almost every phase of the life cycle. The source of the defects can be many: oversight, wrong assumptions, use of inappropriate technology, communication gap among the project engineers, etc. These defects usually get detected much later in the life cycle. For example, a design defect might go unnoticed till we reach the coding or testing phase. Once a defect is detected, the engineers need to go back to the phase where the defect had occurred and redo some of the work done during that phase and the subsequent phases to correct the defect and its effect on the later phases. Therefore, in any practical software development work, it is not possible to strictly follow the classical waterfall model.

Phase-entry and phase-exit criteria of each phase

At the starting of the feasibility study, project managers or team leaders try to understand what is the actual problem by visiting the client side. At the end of that phase they pick the best solution and determine whether the solution is feasible financially and technically. At the starting of requirements analysis and specification phase the required data is collected. After that requirement specification is carried out. Finally, SRS document is produced. At the starting of design phase, context diagram and different levels of DFDs are produced according to the SRS document. At the end of this phase module structure (structure chart) is produced. During the coding phase each module (independently compilation unit) of the design is coded. Then each module is tested independently as a stand-alone unit and debugged separately. After this each module is documented individually. The end product of the implementation phase is a set of program modules that have been tested individually but not tested together. After the implementation phase, different modules which have been tested individually are integrated in a planned manner. After all the modules have been successfully integrated and tested, system testing is carried out. Software maintenance denotes any changes made to a software product after it has been delivered to the customer. Maintenance is inevitable for almost any kind of product. However, most products need maintenance due to the wear and tear caused by use.

Prototype

A prototype is a toy implementation of the system. A prototype usually exhibits limited functional capabilities, low reliability, and inefficient performance compared to the actual software. A prototype is usually built using several shortcuts. The shortcuts might involve using inefficient, inaccurate, or dummy functions. The shortcut implementation of a function, for example, may produce the desired results by using a table look-up instead of performing the actual computations. A prototype usually turns out to be a very crude version of the actual system.

Need for a prototype in software development

There are several uses of a prototype. An important purpose is to illustrate the input data formats, messages, reports, and the interactive dialogues to the customer. This is a valuable mechanism for gaining better understanding of the customer's needs:

• how the screens might look like

• how the user interface would behave
• how the system would produce outputs

This is something similar to what the architectural designers of a building do; they show a prototype of the building to their customer. The customer can evaluate whether he likes it or not and the changes that he would need in the actual product. A similar thing happens in the case of a software product and its prototyping model. Another reason for developing a prototype is that it is impossible to get the perfect product in the first attempt. Many researchers and engineers advocate that if you want to develop a good product you must plan to throw away the first version. The experience gained in developing the prototype can be used to develop the final product. A prototyping model can be used when technical solutions are unclear to the development team. A developed prototype can help engineers to critically examine the technical issues associated with the product development. Often, major design decisions depend on issues like the response time of a hardware controller, or the efficiency of a sorting algorithm, etc. In such circumstances, a prototype may be the best or the only way to resolve the technical issues.

Examples for prototype model

A prototype of the actual product is preferred in situations such as:
• user requirements are not complete
• technical issues are not clear

Let's see an example for each of the above category.

Example 1: User requirements are not complete

In any application software like billing in a retail shop, accounting in a firm, etc the users of the software are not clear about the different functionalities required. Once they are provided with the prototype implementation, they can try to use it and find out the missing functionalities.

Example 2: Technical issues are not clear

Suppose a project involves writing a compiler and the development team has never written a compiler. In such a case, the team can consider a simple language, try to build a compiler in order to check the issues that arise in the process and resolve them. After successfully building a small compiler (prototype), they would extend it to one that supports a complete language.

Spiral model

The Spiral model of software development is shown in fig. 2.2. The diagrammatic representation of this model appears like a spiral with many loops. The exact number of loops in the spiral is not fixed. Each loop of the spiral represents a phase of the software process. For example, the innermost loop might be concerned with feasibility study. The next loop with requirements specification, the next one with design, and so on. Each phase in this model is split into four sectors (or quadrants) as shown in fig. 2.2. The following activities are carried out during each phase of a spiral model

First quadrant (Objective Setting)
• During the first quadrant, it is needed to identify the objectives of the phase.
• Examine the risks associated with these objectives.

- **Second Quadrant (Risk Assessment and Reduction)**

• A detailed analysis is carried out for each identified project risk.
• Steps are taken to reduce the risks. For example, if there is a risk that the requirements are inappropriate, a prototype system may be developed.

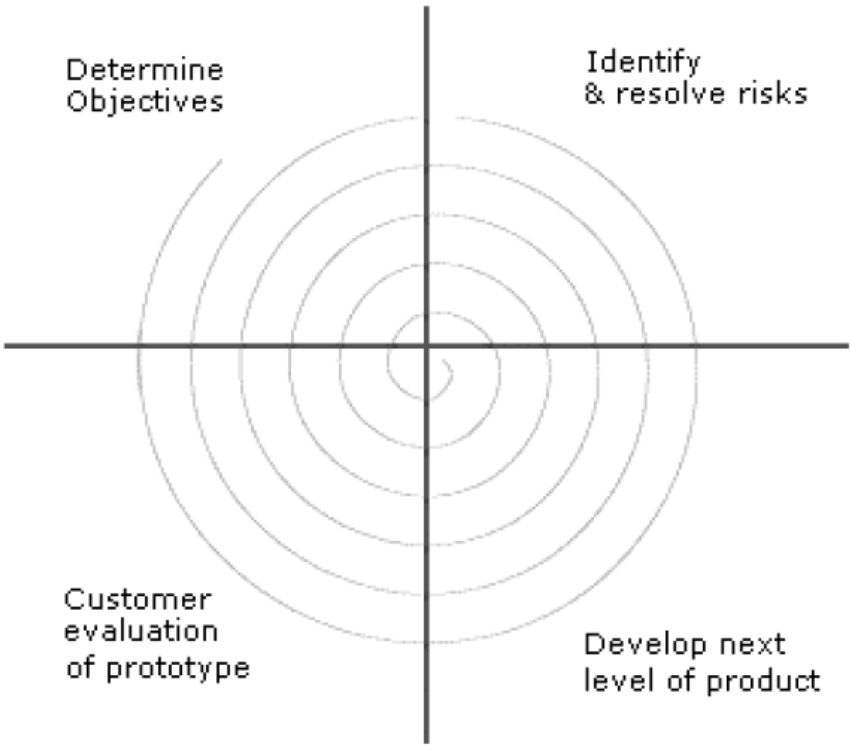


Fig. 2.2: Spiral Model

Third Quadrant (Development and Validation)

• Develop and validate the next level of the product after resolving the identified risks.

- **Fourth Quadrant (Review and Planning)**

• Review the results achieved so far with the customer and plan the next iteration around the spiral.

• Progressively more complete version of the software gets built with each iteration around the spiral

Circumstances to use spiral model

The spiral model is called a meta model since it encompasses all other life cycle models. Risk handling is inherently built into this model. The spiral model is suitable for development of technically challenging software products that are prone to several kinds of risks. However, this model is much more complex than the other models – this is probably a factor deterring its use in ordinary projects.

Comparison of different life-cycle models

The classical waterfall model can be considered as the basic model and all other life cycle models as embellishments of this model. However, the classical waterfall model can not be used in practical development projects, since this model supports no mechanism to handle the errors committed during any of the phases.

This problem is overcome in the iterative waterfall model. The iterative waterfall model is probably the most widely used software development model evolved so far. This model is simple to understand and use. However, this model is suitable only for well-understood problems; it is not suitable for very large projects and for projects that are subject to many risks. The prototyping model is suitable for projects for which either the user requirements or the underlying technical aspects are not well understood. This model is especially popular for development of the user-interface part of the projects.

The evolutionary approach is suitable for large problems which can be decomposed into a set of modules for incremental development and delivery. This model is also widely used for object-oriented development projects. Of course, this model can only be used if the incremental delivery of the system is acceptable to the customer.

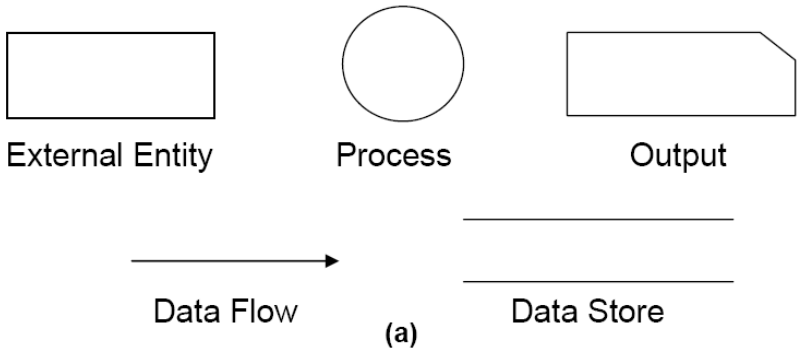
The spiral model is called a meta model since it encompasses all other life cycle models. Risk handling is inherently built into this model. The spiral model is suitable for development of technically challenging software products that are prone to several kinds of risks. However, this model is much more complex than other models – this is probably a factor deterring its use in ordinary projects. The different software life cycle models can be compared from the viewpoint of the customer. Initially, customer confidence in the development team is usually high irrespective of the development model followed. During the lengthy development process, customer confidence normally drops off, as no working product is immediately visible. Developers answer customer queries using technical slang, and delays are announced. This gives rise to customer resentment.

On the other hand, an evolutionary approach lets the customer experiment with a working product much earlier than the monolithic approaches.
Another important advantage of the incremental model is that it reduces the customer's trauma of getting used to an entirely new system. The gradual introduction of the product via incremental phases provides time to the customer to adjust to the new product. Also, from the customer's financial viewpoint, incremental development does not require a large upfront capital outlay. The customer can order the incremental versions as and when he can afford them.

2. PROCESS AND CONCEPTUAL MODELING

2.1 INTRODUCTION TO DATA FLOW DIAGRAM

The DFD (also known as a bubble chart) is a hierarchical graphical model of a system that shows the different processing activities or functions that the system performs and the data interchange among these functions. Each function is considered as a processing station (or process) that consumes some input data and produces some output data. The system is represented in terms of the input data to the system, various processing carried out on these data, and the output data generated by the system. A DFD model uses a very limited number of primitive symbols [as shown in fig.] to represent the functions performed by a system and the data flow among these functions



(a)

Importance of DFDs in a good system design

The main reason why the DFD technique is so popular is probably because of the fact that DFD is a very simple formalism – it is simple to understand and use. Starting with a set of high-level functions that a system performs, a DFD model hierarchically represents various sub-functions. In fact, any hierarchical model is simple to understand. Human mind is such that it can easily understand any hierarchical model of a system – because in a hierarchical model, starting with a very simple and abstract model of a system, different details of the system are slowly introduced through different hierarchies. The data flow diagramming technique also follows a very simple set of intuitive concepts and rules. DFD is an elegant modeling technique that turns out to be useful not only to represent the results of structured analysis of a software problem, but also for several other applications such as showing the flow of documents or items in an organization.

2.2 Concepts used in drawing DFD's

Context diagram

The context diagram is the most abstract data flow representation of a system. It represents the entire system as a single bubble. This bubble is labeled according to the main function of the system. The various external entities with which the system interacts and the data flow occurring between the system and the external entities are also represented. The data input to the system and the data output from the system are represented as incoming and outgoing arrows. These data flow arrows should be annotated with the corresponding data names. The name 'context diagram' is well justified because it represents the context in which the system is to exist, i.e. the external entities who would interact with the system and the specific data items they would be supplying

the system and the data items they would be receiving from the system. The context diagram is also called as the level 0 DFD. To develop the context diagram of the system, it is required to analyze the SRS document to identify the different types of users who would be using the system and the kinds of data they would be inputting to the system and the data they would be receiving the system. Here, the term “users of the system” also includes the external systems which supply data to or receive data from the system.

The bubble in the context diagram is annotated with the name of the system being developed (usually a noun). This is in contrast with the bubbles in all other levels which are annotated with verbs. This is expected since the purpose of the context diagram is to capture the context of the system rather than its functionality.

Level 1 DFD:-

To develop the level 1 DFD, examine the high-level functional requirements. If there are between 3 to 7 high-level functional requirements, then these can be directly represented as bubbles in the level 1 DFD. We can then examine the input data to these functions and the data output by these functions and represent them appropriately in the diagram.

If a system has more than 7 high-level functional requirements, then some of the related requirements have to be combined and represented in the form of a bubble in the level 1 DFD. Such a bubble can be split in the lower DFD levels. If a system has less than three high-level functional requirements, then some of them need to be split into their sub-functions so that we have roughly about 5 to 7 bubbles on the diagram.

Decomposition:-

Each bubble in the DFD represents a function performed by the system. The bubbles are decomposed into sub-functions at the successive levels of the DFD. Decomposition of a bubble is also known as factoring or exploding a bubble. Each bubble at any level of DFD is usually decomposed to anything between 3 to 7 bubbles. Too few bubbles at any level make that level superfluous. For example, if a bubble is decomposed to just one bubble or two bubbles, then this decomposition becomes redundant. Also, too many bubbles, i.e. more than 7 bubbles at any level of a DFD makes the DFD model hard to understand. Decomposition of a bubble should be carried on until a level is reached at which the function of the bubble can be described using a simple algorithm.

Numbering of Bubbles:-

It is necessary to number the different bubbles occurring in the DFD. These numbers help in uniquely identifying any bubble in the DFD by its bubble number. The bubble at the context level is usually assigned the number 0 to indicate that it is the 0 level DFD. Bubbles at level 1 are numbered, 0.1, 0.2, 0.3, etc, etc. When a bubble numbered x is decomposed, its children bubble are numbered x.1, x.2, x.3, etc. In this numbering scheme, by looking at the number of a bubble we can unambiguously determine its level, its ancestors, and its successors.

2.3 DFD design(upto level 2)

Example:-

A supermarket needs to develop the following system to encourage regular customers. For this, the customer needs to supply his/her residence address, telephone number, and the driving license number. Each customer who registers for this scheme is assigned a unique customer number (CN) by the computer. A customer can present his CN to the check out staff when he makes any purchase. In this case, the value of his purchase is credited against his CN. At the end of each year, the supermarket intends to award surprise gifts to 10 customers who make the highest total purchase over the year. Also, it intends to award a 22 caret gold coin to every customer whose purchase exceeded Rs.10,000. The entries against the CN are the reset on the day of every year after the prize winners’ lists are generated. The context diagram for this problem is shown in fig. 5.5, the level 1 DFD in fig. 5.6, and the level 2 DFD in fig. 5.7.

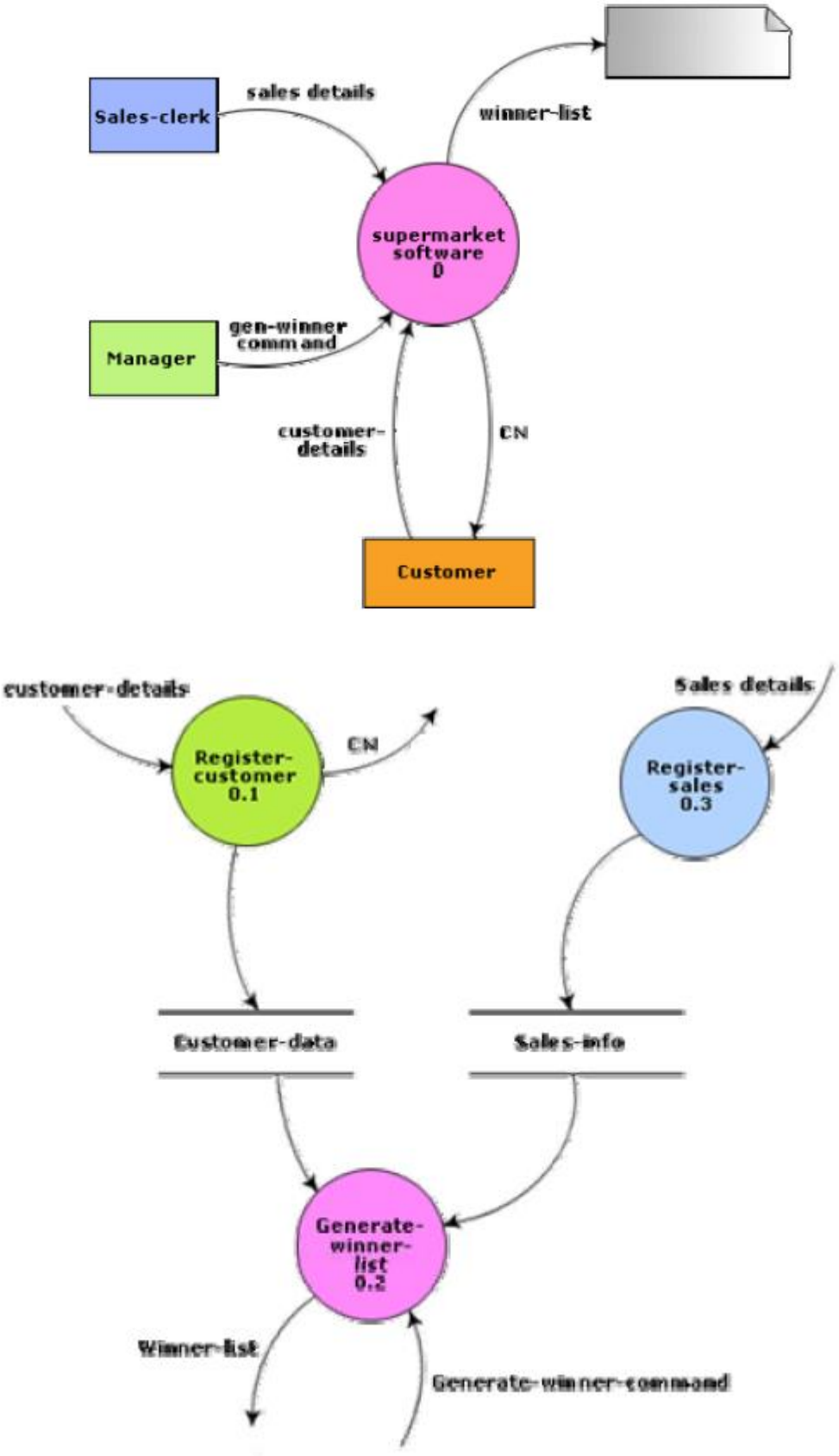


Fig. 5.6: Level 1 diagram for supermarket problem

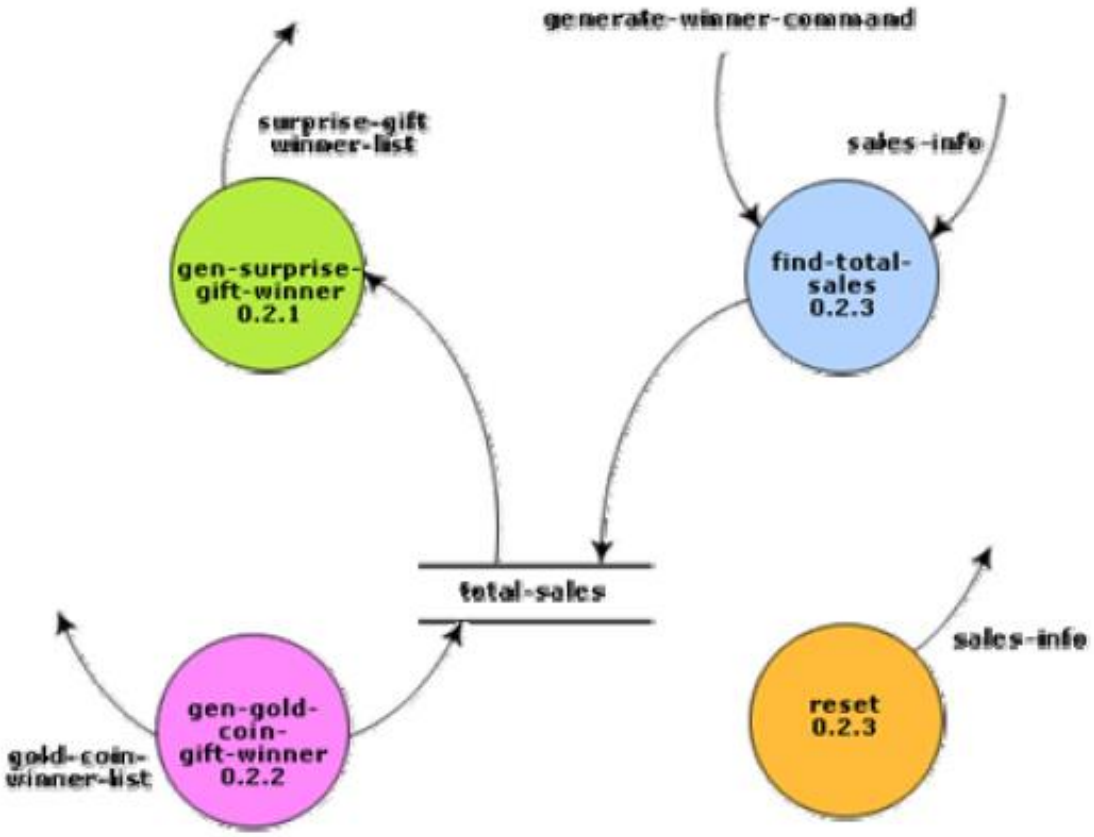


Fig. 5.7: Level 2 diagram for supermarket problem

Example 2. Library Management System

Scan paper here

2.4 Conceptual Modeling

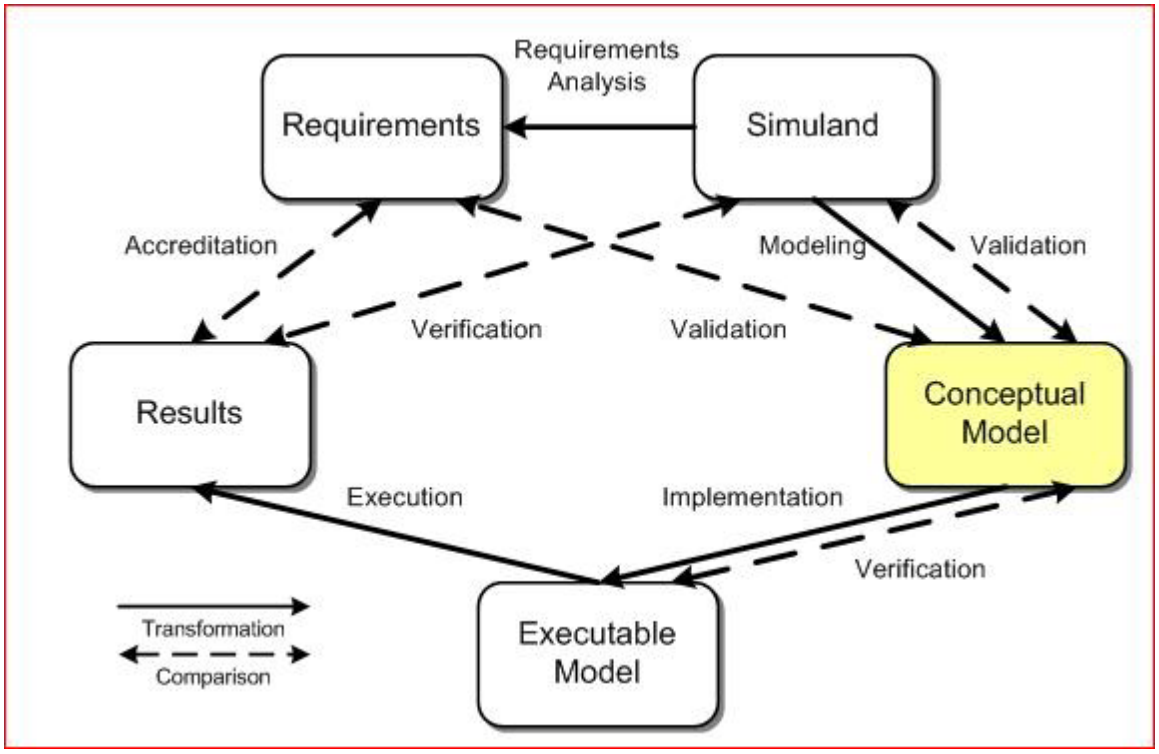
Conceptual modeling is the activity of formally describing some aspects of the physical and social world around us for the purposes of understanding and communication." A conceptual models primary objective is to convey the fundamental principles and basic functionality of the system in which it represents. Also, a conceptual model must be developed in such a way as to provide an easily understood system interpretation for the models users. A conceptual model, when implemented properly, should satisfy four fundamental objectives.

1. Enhance an individual's understanding of the representative system
2. Facilitate efficient conveyance of system details between stakeholders
3. Provide a point of reference for system designers to extract system specifications
4. Document the system for future reference and provide a means for collaboration

The conceptual model plays an important role in the overall system development life cycle. Figure below, depicts the role of the conceptual model in a typical system development scheme. It is clear that if the conceptual model is not fully developed, the execution of fundamental system The conceptual model plays an important role in the overall system development life cycle. Figure below, depicts the role of the conceptual model in a typical system development scheme. It is clear that if the conceptual model is not fully developed, the execution of fundamental system properties may not be implemented properly, giving way to future problems or system shortfalls. These failures do occur in the industry and

have been linked to; lack of user input, incomplete or unclear requirements, and changing requirements. Those weak links in the system design and development process can be traced to improper execution of the fundamental objectives of conceptual modeling. The importance of conceptual modeling is evident when such systemic failures are mitigated by thorough system development and adherence to proven development objectives/techniques may not be implemented properly, giving way to future problems or system shortfalls.

These failures do occur in the industry and have been linked to; lack of user input, incomplete or unclear requirements, and changing requirements. Those weak links in the system design and development process can be traced to improper execution of the fundamental objectives of conceptual modeling. The importance of conceptual modeling is evident when such systemic failures are mitigated by thorough system development and adherence to proven development objectives/techniques



Data Flow Modeling

Data flow modeling (DFM) is a basic conceptual modeling technique that graphically represents elements of a system. DFM is a fairly simple technique, however, like many conceptual modeling techniques, it is possible to construct higher and lower level representative diagrams. The data flow diagram usually does not convey complex system details such as parallel development considerations or timing information, but rather works to bring the major system functions into context.

- ***Entity Relationship Modeling***

Entity-relationship modeling (ERM) is a conceptual modeling technique used primarily for software system representation. Entity-relationship diagrams, which are a product of executing the ERM technique, are normally used to represent database models and information systems. The main components of the diagram are the entities and relationships. The entities can represent independent functions, objects, or events. The relationships are responsible for relating the entities to one another. To form a system process, the relationships are combined with the entities and any attributes needed to further describe the process.

UNIT 3
LOGIC MODELING

3.1 Decision Table:

A decision table shows some condition and the action that is to be taken under the different conditions. The rules for drawing a decision table can be understood with example.

Example 1.

Suppose in a store there is following scheme.

1. A customer can be a member or not. If he/she is a member and purchase the product above Rs 12,000 he will be provided a discount of 20%.
2. If he/she is a member and purchase the product less than Rs. 12,000, he/she will be provided with the discount of 12%
3. If a customer is not a member and purchase the product of price >12,000 he/she will be provided with the discount of 6%
4. In all other condition there will be no discount.

We can design the decision table for the above condition as follows:

1. Place the name of the process in a heading at the top left of the table.
2. Enter the conditions under the heading with one condition per line to represent the status of the customer for buying the product.
3. Enter the potential condition of Yes or No on the R.H.S of the table as a rule. Each column represents a rule
4. In the action area, place the tick mark for each rule when the discount is allowed.

Discount Scheme Rule

Condition	Member	y	y	N	N
	Purchase above 12,000	Y	N	Y	N
Actions	20%	✓			
	12%		✓		
	6%			✓	
	0%				✓

Example 2. Business Order is verified under the following conditions.

1.An order will be accepted only if the product is in stock and the customer's credit status is OK

2. All other orders will be rejected.

The decision table for the above situation is as follows:

Verify Business Order		**Rule**			
Condition	Customer Credit Status	y	y	N	N
	Product Available	Y	N	Y	N
Actions	Accept Order	✓			
	Reject Order		✓	✓	✓

Example 3. Table with three conditions:

In the example 2, if one more condition is added and set of new conditions are as follows:

1. An order will be accepted only if the product is available and the customer credit status is Ok.
2. If the customer credit status is not valid, manager can waive the credit status requirement
3. All other orders will be rejected.

The decision table for the above requirement is as follows:

Verify Business Order		**RULE**							
Condition	CUSTOMER CREDIT STATUS	Y	Y	Y	Y	N	N	N	N
	PRODUCT IN STOCK	Y	Y	N	N	Y	Y	N	N
	WAIVER FROM MANAGER	Y	N	Y	N	Y	N	Y	N
Actions	ACCEPT ORDER	✓	✓			✓			
	REJECT ORDER			✓	✓		✓	✓	✓

Simplification of the table

When we have rules with three conditions, only one or two of them may control the outcome, and the other conditions simply do not matter.

If we identify conditions that do not affect the outcome, we need to mark them with dashes

Condition	CUSTOMER CREDIT STATUS	Y	Y	-	-	N	N	-	-
	PRODUCT IN STOCK	Y	Y	N	N	Y	Y	N	N
	WAIVER FROM MANAGER	-	-	-	-	Y	N	-	-
Actions	ACCEPT ORDER	✓	✓			✓			
	REJECT ORDER			✓	✓		✓	✓	✓

After marking the non required rules, we can combine the similar conditions as follows..

Condition	CUSTOMER CREDIT STATUS	Y	N	N	-
	PRODUCT IN STOCK	Y	Y	Y	N
	WAIVER FROM MANAGER	-	Y	N	-

Actions	ACCEPT ORDER	✓	✓		
	REJECT ORDER			✓	✓

3.2 Decision Tree

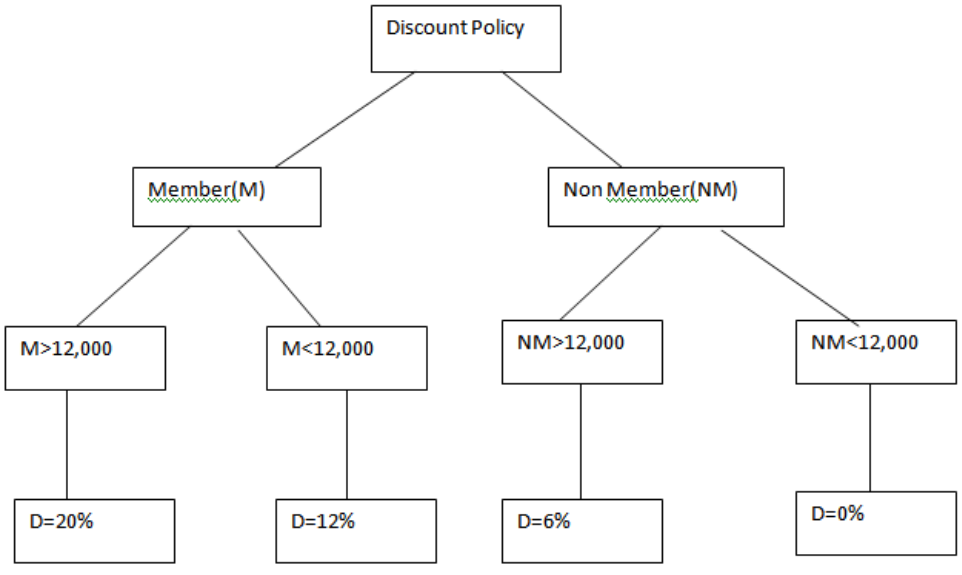
A decision tree is a graphical representation of a condition and actions and rules found in a decision table. Decision tree shows the logical structure in a vertical form that resembles a tree with the roots at the top and branches at the down. Like flowcharts decision trees are useful ways to represent the system management. Decision trees and decision table provides the same result but in different forms. It helps a system analyst to study the relationship of conditions and actions. Though the simple decision tree can be easily understood by a non-technical user, it doesn't work for complex system analysis.

Example 1.

Suppose in a store there is following scheme.

1. A customer can be a member or not. If he/she is a member and purchase the product above Rs 12,000 he will be provided a discount of 20%.
2. If he/she is a member and purchase the product less than Rs. 12,000, he/she will be provided with the discount of 12%
3. If a customer is not a member and purchase the product of price >12,000 he/she will be provided with the discount of 6%
4. In all other condition there will be no discount.

The decision tree for the above case is as follows

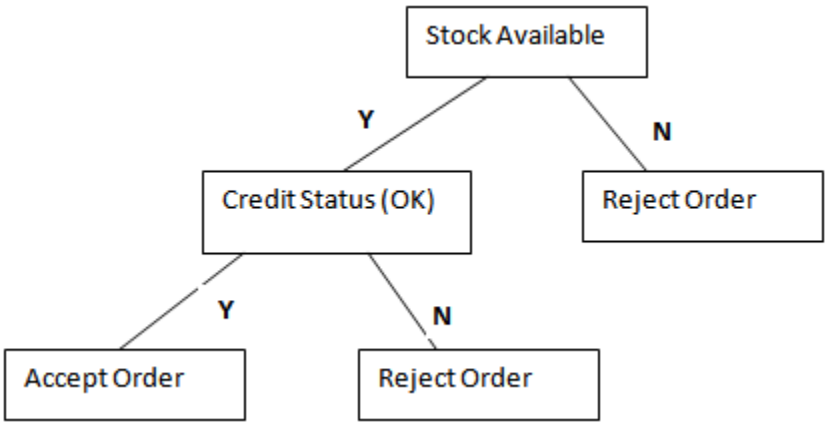


Example 2. Business Order is verified under the following conditions.

1.An order will be accepted only if the product is in stock and the customer's credit status is OK

2. All other orders will be rejected.

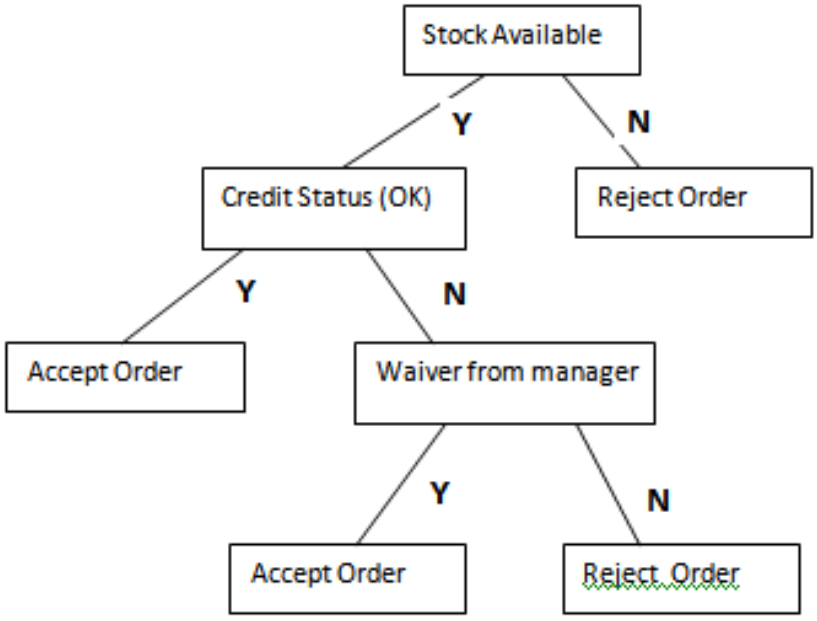
The decision tree for the above situation is as follows



Example 3. In the example 2, if one more condition is added and set of new conditions are as follows:

1. An order will be accepted only if the product is available and the customer credit status is Ok.
2. If the customer credit status is not valid, manager can waive the credit status requirement
3. All other orders will be rejected.

The decision tree for the above requirement is as follows:



Structured English:

Structured English is derived from structured programming and its use of logical construction and imperative statement. Statement with some verbs and actions is called imperative statement. Structured English is designed to carry out instructions for actions by creating decision statements that use

structure programming term such as if, else, then etc. Structured English consists of following statements.

1. Sequence structure: This is a single step or action included in the sequence process. It doesn't depend on the existence of other conditions.
2. Decision structure: This occurs when two or more actions rely on the value of a specific condition. The condition is expanded and necessary decisions are made
3. Iteration Structure: Certain conditions will only occur after specific conditions are executed. Iterative instructions help an analyst to describe these specific cases.

Example 1.

Suppose in a store there is following scheme.

1. A customer can be a member or not. If he/she is a member and purchase the product above Rs 12,000 he will be provided a discount of 20%.
2. If he/she is a member and purchase the product less than Rs. 12,000, he/she will be provided with the discount of 12%
3. If a customer is not a member and purchase the product of price >12,000 he/she will be provided with the discount of 6%

In all other condition there will be no discount.

Discount policy:

```
Input data flow: PURCHASE
Output data flow: DISCOUNT
BEGIN
For each PURCHASE
   If  Customer is a member
         If  PURCHASE>12,000
               Output: Discount=20%
         Else
               Output: Discount=12%
   Else
         if Purchase>12,000
               Output: Discount=6%
         Else
               Output: Discount=0%
   END
```

Example 2:

. Business Order is verified under the following conditions.

1.An order will be accepted only if the product is in stock and the customer's credit status is OK

2. All other orders will be rejected.

Verify Order Process

Input data flow: ORDER, CREDIT STATUS, PRODUCT DETAIL
Output data flow: ACCEPTED ORDER, REJECTED ORDER
BEGIN
For each ORDER
If CREDIT STATUS=Y and PRODUCT DETAIL=OK
Output: ACCEPTED ORDER
Else
Output: REJECTED ORDER
END

Example 3:

In the example 2, if one more condition is added and set of new conditions are as follows:

1. An order will be accepted only if the product is available and the customer credit status is Ok.
2. If the customer credit status is not valid, manager can waive the credit status requirement
3. All other orders will be rejected.

Verify Order Process

Input data flow: ORDER, CREDIT STATUS, PRODUCT DETAIL, MANAGER WAIVER
Output data flow: ACCEPTED ORDER, REJECTED ORDER
BEGIN
For each ORDER
If CREDIT STATUS=Y and PRODUCT DETAIL=OK
Output: ACCEPTED ORDER
Else if
CREDIT STATUS=N and PRODUCT DETAIL=OK
If MANAGER WAIVER=Y
Output: ACCEPTED ORDER
Else
Output: REJECTED ODER

UNIT 4

Passed year questions:

1. **What is feasibility study? Describe briefly the various types of feasibility study?**

2. **Assume that you have a project with seven activities labeled A-G as shown in the given table. Determine the early finish(EF),latest finish(LF), and slack for each task. What is the earliest project completion time?**

Task	Preceeding task	Expected duration(week)
A	-	5
B	A	3
C	A	4
D	C	4
E	B,C	6
F	D	4
G	D,E,F	5

3. **What are the various requirement gathering methods? Describe them in detail.**
4. **A project has been define to contain the following list of activities along with their required time for completion:**

Activity	Time for completion	Predecessors
1	2	-
2	3	1
3	3	2
4	7	2
5	6	2
6	1	3,4
7	6	4,5
8	4	6,7
9	8	7
10	2	8,9

Draw the network diagram. Calculate the earliest finish(EF) and latest finish(LF) for each activity. Also find out the earliest project completion time.

5. **What is cost-benefit analysis? Discuss about the NPV method and payback method in brief.**
6. **Why understanding of different requirements is essential in system development task? Explain different type of requirement gathering techniques.**
7. **What is the significance of feasibility study? Explain different types of feasibility methods.**
8. **Why it is important to do cost benefit analysis? Discuss any two types of cost-benefit analysis method.**
9. **What is feasibility analysis? Explain.**
10. **Discuss different methods of data collection in system analysis.**
11. **Compare the advantages of interviewing and questionnaires.**
12. **What are the different types of data collecting methods? What are the considerations made for designing a good questionnaire? Prepare a sample**

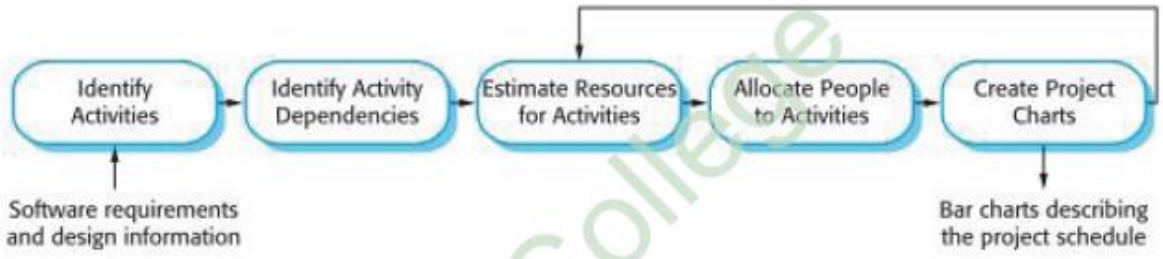
questionnaire for collecting information about the new trends in technical colleges.

13. **How is cost benefit analysis economically feasible? Describe any one cost-benefit analysis method.**

System Analysis

Project scheduling

Project scheduling is the process of deciding how the work in a project will be organized as separate tasks, and when and how these tasks will be executed. **Estimation of the the calendar time needed to complete each task**, **the effort required, and who will work on the tasks that have been identified. You also have to estimate the resources needed to complete each task, such as the disk space required on a server, the time required on specialized hardware, such as a simulator, and what the travel budget will be.** An initial project schedule is usually created during the project startup phase. This schedule is then refined and modified during development planning. Both plan based and agile processes need an initial project schedule, although the level of detail may be less in an agile project plan. This initial schedule is used to plan how people will be allocated to projects and to check the progress of the project against its contractual commitments.



- System project scheduling is an activity that distributes estimated effort across the planned project duration by allocating the effort to specific system engineering tasks.
- Split project into tasks and estimate time and resources required to complete each task
- Organize tasks concurrently to make optimal use of workforce
- Minimize task dependencies to avoid delays caused by one task waiting for another to complete

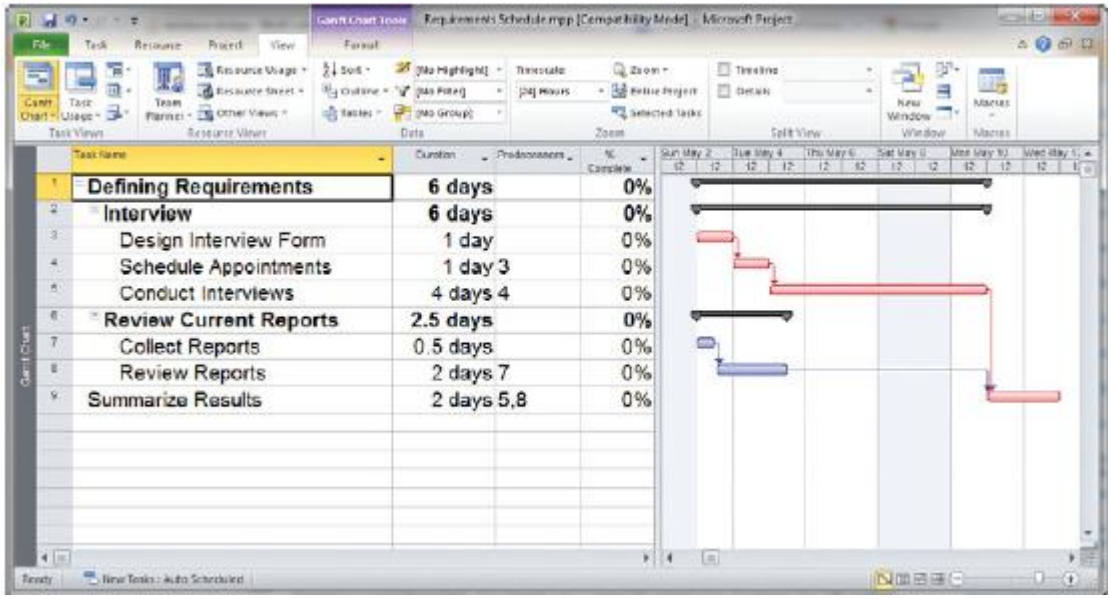
Tools for project scheduling:

Gantt Chart or Bar Chart: To allocate working hour or to schedule the working hour Bar chart/Gantt chart are used in project scheduling. Gantt chart are project planning tool that can be used to represent the timing of task required to complete a project because Gantt chart are simple to understand and easy to construct . They are used by most project manager to complete project.

In Gantt chart each task takes up on rows. Data run along the top in increment of days, weeks or month depending on the total length of the projects, the expected time for each task is represented by a horizontal bar whose left end marks the expected beginning of the task and whose right end marks the expected completion time. Tasks may run sequentially in parallel or over lapping. As the project progress, the chart is updated by

filling in the bar to a length proportional to the fraction of work that has been accomplished on the task.

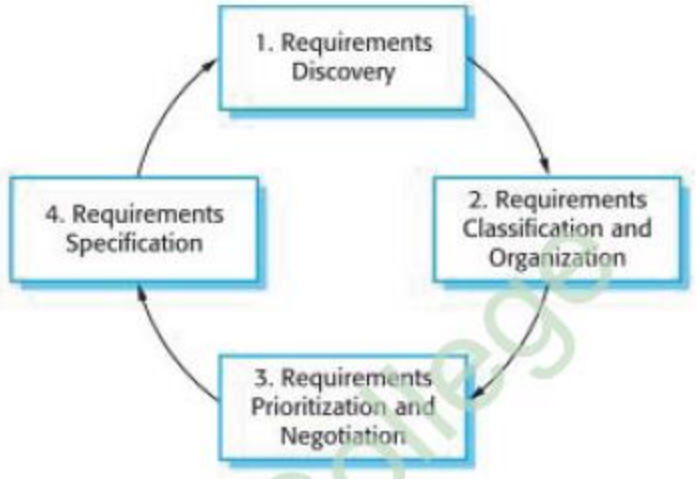
Gantt chart is a graphical representation of a project that shows each task as a horizontal bar whose length is proportional to its time for completion. Different colors, shades, or shapes can be used to highlight each kind of task.. Gantt charts do not show how tasks must be ordered (precedence) but simply show when an activity should begin



Requirement Analysis:

After an initial feasibility study, the next stage of the requirements engineering process is requirements analysis. In this activity, software engineers work with customers and system end users to find out about the application domain, what services the system should provide, the required performance of the system, hardware constraints, and so on. Requirements analysis may involve a variety of different kinds of people in an organization. A system stakeholder is anyone who should have some direct or indirect influence on the system requirements. Stakeholders include end-users who will interact with the system and anyone else in an organization who will be affected by it. Other system stakeholders might be engineers who are developing or maintaining other related systems, business managers, domain experts, and trade union representatives.

fig: Requirement analysis process



Requirement Analysis Process

1. ***Requirements discovery*** This is the process of interacting with stakeholders of the system to discover their requirements. Domain requirements from stakeholders and documentation are also discovered during this activity. There are several complementary techniques that can be used for requirements discovery.

2. ***Requirements classification and organization*** This activity takes the unstructured collection of requirements, groups related requirements, and organizes them into coherent clusters. The most common way of grouping requirements is to use a model of the system architecture to identify sub-systems and to associate requirements with each sub-system. In practice, requirements engineering and architectural design cannot be completely separate activities.

3. ***Requirements prioritization and negotiation*** Inevitably, when multiple stakeholders are involved, requirements will conflict. This activity is concerned with prioritizing requirements and finding and resolving requirements conflicts through negotiation. Usually, stakeholders have to meet to resolve differences and agree on compromise requirements.

4. ***Requirements specification*** The requirements are documented and input into the next round of the spiral. Formal or informal requirements documents may be produced

Types of requirements:

1. **User requirements** are statements, in a natural language plus diagrams, of what services the system is expected to provide to system users and the constraints under which it must operate.

2. **System requirement**s are more detailed descriptions of the system's functions, services, and operational constraints. The system requirements document (sometimes called a functional specification) should define exactly what is to be implemented. It may be part of the contract between the system buyer and the software developers.

3.System specification: A detailed system description which can serve as a basis for a design or implementation. Written for developers.

Functional and non-functional requirements

1. ***Functional requirements*** these are statements of services the system should provide, how the system should react to particular inputs, and how the system should behave in particular situations.
2. In some cases, the functional requirements may also explicitly state what the system should not do.

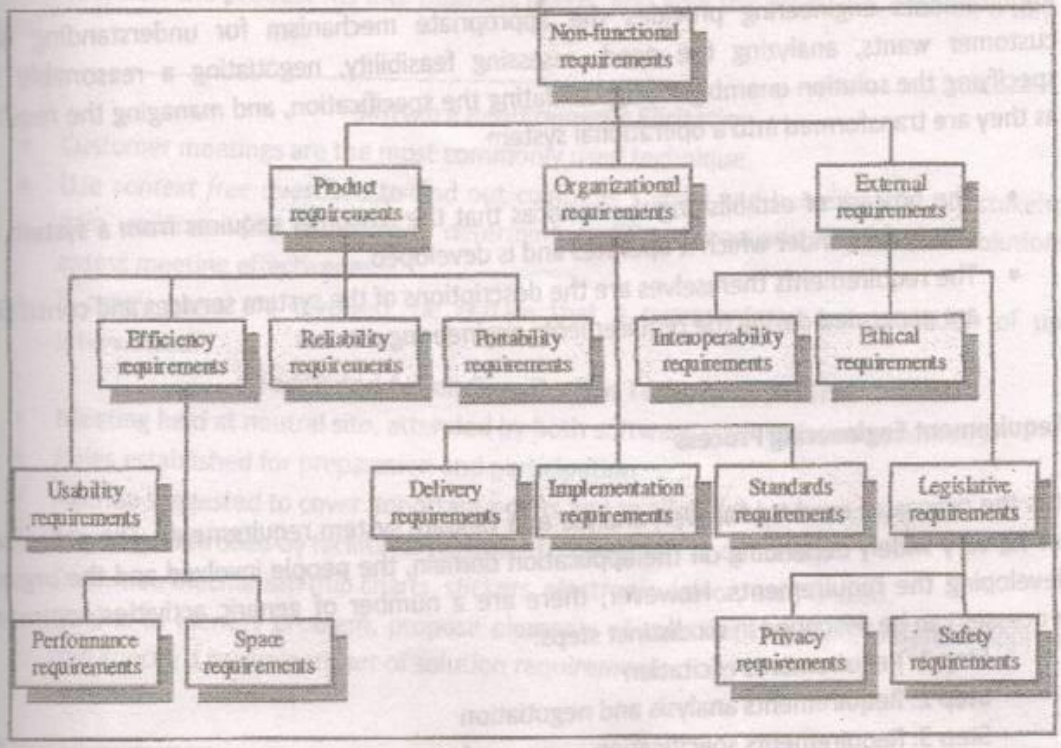
3. Describe the functionality or system services
4. Functional user requirements may be high-level statements of what the system should do but functional system requirements should describe the system services in detail.

1. ***Non-functional requirements*** these are constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.
2. Define system properties and constraints e.g reliability, response time and storage requirements. Constraints are I/O device capability, system representation, etc.
3. Process requirements may also be specified mandating a particular CASE system, programming language or development method.
4. Non-functional requirements may be more critical than functional requirements. If these are not met, the system is useless.

NON functional requirements classification:
Product Requirement: Requirements which specify that the delivered product must behave in a particular way e.g execution speed, reliability, etc.

Organizational Requirements: Requirements which are a consequence of organizational policies and procedures e.g. process standards used, implementation requirements, etc.
External Requirements: Requirements which arise from factors which are external to the system and its development process e.g interoperability requirements, legislative requirements, etc.

Non-functional requirement types



Requirements measures

Property	Measure
Speed	Processed transactions/second User/Event response time Screen refresh time
Size	K bytes Numbers of RAM chips
Ease of Use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target-dependent statements Number of target systems

Requirement Gathering Methods:

Brainstorming
Document Analysis

Focus Group
Interface Analysis
Interview
Observation
Prototyping
Requirements Workshop
Reverse Engineering
Survey
Questionnaires

1. Brainstorming Brainstorming is used in requirements elicitation to get as many ideas as possible from a group of people. Generally used to identify possible solutions to problems, and clarify details of opportunities. Brainstorming casts a wide net, identifying many different possibilities.Prioritization of those possibilities is important to finding the needles in the haystack.

2. Document Analysis Reviewing the documentation of an existing system can help when creating AS-IS process documents, as well as driving gap analysis for scoping of migration projects. In an ideal world, we would even be reviewing the *requirements* that drove creation of the existing system – a starting point for documenting current requirements. Nuggets of information are often buried in existing documents that help us ask questions as part of validating requirement completeness.

3. Focus Group A focus group is a gathering of people who are representative of the users or customers of a product to get feedback. The feedback can be gathered about needs / opportunities / problems to identify requirements, or can be gathered to validate and refine already elicited requirements. This form of market research is distinct from brainstorming in that it is a managed process with specific participants. There is danger in "following the crowd", and some people believe focus groups are at best ineffective. One risk is that we end up with the lowest common denominator features.

4. Interface Analysis Interfaces for a software product can be human or machine. Integration with external systems and devices is just another interface. User centric design approaches are very effective at making sure that we create usable software. Interface analysis – reviewing the touch points with other external systems – is important to make sure we don't overlook requirements that *aren't* immediately visible to users.

5. Interview Interviews of stakeholders and users are critical to creating the great software. Without understanding the goals and expectations of the users and stakeholders, we are very unlikely to satisfy them. We also have to recognize the perspective of each interviewee, so that we can properly weigh and address their inputs. Like a great reporter, listening is the skill that helps a great analyst to get more value from an interview than an average analyst.

6. Observation The study of users in their *natural habitats* is what observation is about. By observing users, an analyst can identify a process flow, awkward steps, pain points and opportunities for improvement. Observation can be passive or active (asking questions while observing). Passive observation is better for getting feedback on a prototype (to refine requirements), where active observation is more effective at getting an understanding of an existing business process. Either approach can be used to uncover *implicit* requirements that otherwise might go overlooked.

7. Prototyping Prototypes can be very effective at gathering feedback. Low fidelity prototypes can be used as an active listening tool. Often, when people cannot articulate a particular need in the abstract, they can quickly assess if a design approach would address the need. Prototypes are most efficiently done with quick sketches of interfaces and storyboards. Prototypes are even being used as the "official requirements" in some situations.

8. Requirements Workshop More commonly known as a joint application design (JAD) session, workshops can be very effective for gathering requirements. More structured than a brainstorming session, involved parties collaborate to document requirements. One way to capture the collaboration is with creation of domain-model artifacts (like static diagrams, activity diagrams). A workshop will be more effective with two analysts than with one, where a facilitator and a scribe work together.

9. Reverse Engineering Is this a starting point or a last resort? When a migration project does not have access to sufficient documentation of the existing system, reverse engineering will identify what the system does. It will not identify what the system should do, and will not identify when the system does the wrong thing.

10. Survey When collecting information from many people – too many to interview with budget and time constraints – a survey or questionnaire can be used. The survey can force users to select from choices, rate something ("Agree Strongly, Agree…"), or have open ended questions allowing freeform responses. Survey design is hard – questions can bias the respondents. Don't assume that you can create a survey on your own, and get meaningful insight from the results. I would expect that a well designed survey would provide qualitative guidance for characterizing the market. It should not be used for prioritization of features or requirements.

11. Questionnaires If face-to-face meetings are possible, they are always preferable, because they provide a better means of uncovering the problem behind the problem. Sometimes, though, face-to-face meetings with stakeholders are not feasible (when developing products for the consumer market, for example). In those situations, consider using questionnaires. Send a set of questions, possibly with multiple choice responses, to the relevant stakeholders, and ask them to complete it and return it to you. Success tips: Keep it short and given them a deadline.

Feasibility study and its types

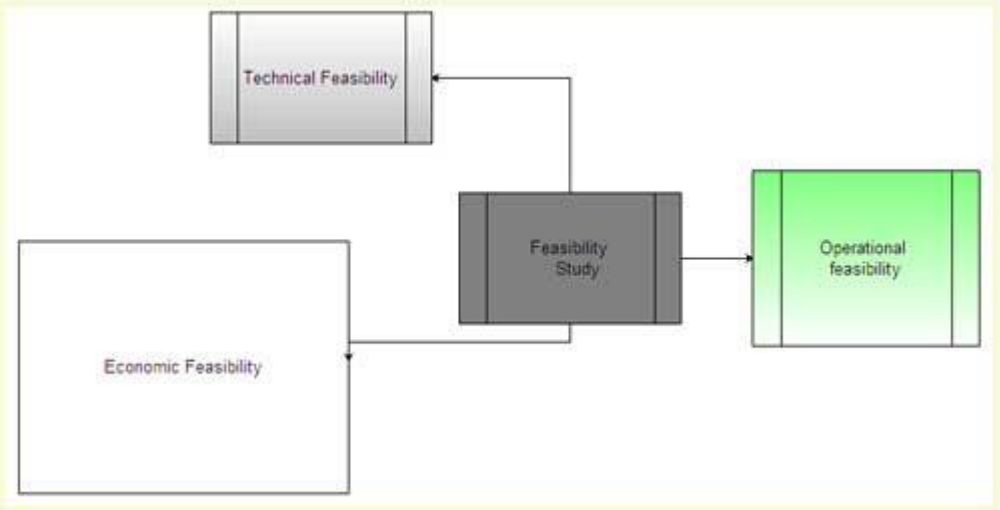
Feasibility is defined as the practical extent to which a project can be performed successfully. To evaluate feasibility, a feasibility study is performed, which determines whether the solution considered to accomplish the requirements is practical and workable in the software. Information such as resource availability, cost estimation for software development, benefits of the software to the organization after it is developed and cost to be incurred on its maintenance are considered during the feasibility study. The objective of the feasibility study is to establish the reasons for developing the software that is acceptable to users, adaptable to change and conformable to established standards. Various other objectives of feasibility study are listed below.

- To analyze whether the software will meet organizational requirements

- To determine whether the software can be implemented using the current technology and within the specified budget and schedule
- To determine whether the software can be integrated with other existing software.

Types of Feasibility

Various types of feasibility that are commonly considered include technical feasibility, operational feasibility, and economic feasibility.



Technical feasibility assesses the current resources (such as hardware and software) and technology, which are required to accomplish user requirements in the software within the allocated time and budget. For this, the software development team ascertains whether the current resources and technology can be upgraded or added in the software to accomplish specified user requirements. Technical feasibility also performs the following tasks.

- Analyzes the technical skills and capabilities of the software development team members
- Determines whether the relevant technology is stable and established
- Ascertains that the technology chosen for software development has a large number of users so that they can be consulted when problems arise or improvements are required.

Operational feasibility assesses the extent to which the required software performs a series of steps to solve business problems and user requirements. This feasibility is dependent on human resources (software development team) and involves visualizing whether the software will operate after it is developed and be operative once it is installed. Operational feasibility also performs the following tasks.

- Determines whether the problems anticipated in user requirements are of high priority
- Determines whether the solution suggested by the software development team is acceptable
- Analyzes whether users will adapt to a new software

- Determines whether the organization is satisfied by the alternative solutions proposed by the software development team.

Economic feasibility determines whether the required software is capable of generating financial gains for an organization. It involves the cost incurred on the software development team, estimated cost of hardware and software, cost of performing feasibility study, and so on. For this, it is essential to consider expenses made on purchases (such as hardware purchase) and activities required to carry out software development. In addition, it is necessary to consider the benefits that can be achieved by developing the software. Software is said to be economically feasible if it focuses on the issues listed below.

- Cost incurred on software development to produce long-term gains for an organization
- Cost required to conduct full software investigation (such as requirements elicitation and requirements analysis)
- Cost of hardware, software, development team, and training.

Steps of Feasibility study

Feasibility study comprises the following steps.

1. **Information assessment:** Identifies information about whether the system helps in achieving the objectives of the organization. It also verifies that the system can be implemented using new technology and within the budget and whether the system can be integrated with the existing system.
2. **Information collection:** Specifies the sources from where information about software can be obtained. Generally, these sources include users (who will operate the software), organization (where the software will be used), and the software development team (which understands user requirements and knows how to fulfill them in software).
3. **Report writing:** Uses a feasibility report, which is the conclusion of the feasibility study by the software development team. It includes the recommendations whether the software development should continue. This report may also include information about changes in the software scope, budget, and schedule and suggestions of any requirements in the system.
4. **General information:** Describes the purpose and scope of feasibility study. It also describes system overview, project references, acronyms and abbreviations, and points of contact to be used. **System overview** provides description about the name of the organization responsible for the software development, system name or title, system category, operational status, and so on. **Project references** provide a list of the references used to prepare this document such as documents relating to the project or previously developed documents that are related to the project.**Acronyms and abbreviations** provide a list of the terms that are used in this document along with their meanings.**Points of contact** provide a list of points of organizational contact with users for information and

coordination. For example, users require assistance to solve problems (such as troubleshooting) and collect information such as contact number, e-mail address, and so on.

5. **Management summary:** Provides the following information.
6. **Environment:** Identifies the individuals responsible for software development. It provides information about input and output requirements, processing requirements of the software and the interaction of the software with other software. It also identifies system security requirements and the system's processing requirements
7. **Current functional procedures:** Describes the current functional procedures of the existing system, whether automated or manual. It also includes the data-flow of the current system and the number of team members required to operate and maintain the software.
8. **Functional objective:** Provides information about functions of the system such as new services, increased capacity, and so on.
9. **Performance objective:** Provides information about performance objectives such as reduced staff and equipment costs, increased processing speeds of software, and improved controls.
10. **Assumptions and constraints:** Provides information about assumptions and constraints such as operational life of the proposed software, financial constraints, changing hardware, software and operating environment, and availability of information and sources.
11. **Methodology:** Describes the methods that are applied to evaluate the proposed software in order to reach a feasible alternative. These methods include survey, modeling, benchmarking, etc.
12. **Evaluation criteria:** Identifies criteria such as cost, priority, development time, and ease of system use, which are applicable for the development process to determine the most suitable system option.
13. **Recommendation:** Describes a recommendation for the proposed system. This includes the delays and acceptable risks.
14. **Proposed software:** Describes the overall concept of the system as well as the procedure to be used to meet user requirements. **In** addition, it provides information about improvements, time and resource costs, and impacts. Improvements are performed to enhance the functionality and performance of the existing software. Time and resource costs include the costs associated with software development from its requirements to its maintenance and staff training. Impacts describe the possibility of future happenings and include various types of impacts as listed below.
15. **Equipment impacts:** Determine new equipment requirements and changes to be made in the currently available equipment requirements.
16. **Software impacts:** Specify any additions or modifications required in the existing software and supporting software to adapt to the proposed software.
17. **Organizational impacts:** Describe any changes in organization, staff and skills requirement.
18. **Operational impacts:** Describe effects on operations such as user-operating procedures, data processing, data entry procedures, and so on.

19. **Developmental impacts:** Specify developmental impacts such as resources required to develop databases, resources required to develop and test the software, and specific activities to be performed by users during software development.
20. **Security impacts:** Describe security factors that may influence the development, design, and continued operation of the proposed software.
21. **Alternative systems:** Provide description of alternative systems, which are considered in a feasibility study. This also describes the reasons for choosing a particular alternative system to develop the proposed software and the reason for rejecting alternative systems.

Cost Benefit Analysis:

A study of economic feasibility is required for the baseline project plan. The purpose for assessing **economic feasibility** is to identify the financial benefits and costs associated with the development project. Economic feasibility is often referred to as *cost-benefit analysis.*

UNIT V

System Design

Based on the user requirements and the detailed analysis of the existing system, the new system must be designed. This is the phase of system designing. It is the most crucial phase in the developments of a system. The logical system design arrived at as a result of systems analysis is converted into physical system design. Normally, the design proceeds in two stages
: l Preliminary or General Design
l Structured or Detailed Design

Preliminary or General Design: In the preliminary or general design, the features of the new system are specified. The costs of implementing these features and the benefits to be derived are estimated. If the project is still considered to be feasible, we move to the detailed design stage.

Structured or Detailed Design: In the detailed design stage, computer oriented work begins in earnest. At this stage, the design of the system becomes more structured. Structure design is a blue print of a computer system solution to a given problem having the same components and inter-relationships among the same components as the original problem. Input, output, databases, forms, codification schemes and processing specifications are drawn up in detail. In the design stage, the programming language and the hardware and software platform in which the new system will run are also decided.

There are several tools and techniques used for describing the system design of the system. These tools and techniques are: l Flowchart l Data flow diagram (DFD) l Data dictionary l Structured English l Decision table l Decision tree The system design involves: i. Defining precisely the required system output ii. Determining the data requirement for producing the output iii. Determining the medium and format of files and databases iv. Devising processing methods and use of software to produce output v. Determine the methods of data capture and data input vi. Designing Input forms vii.

Designing Codification Schemes viii. Detailed manual procedures ix. Documenting the Design

Process and Stages of System Design

The *design phase* decides *how* the system will operate in terms of the hardware, software, and network infrastructure that will be in place; the user interface, forms, and reports that will be used; and the specific programs, databases, and files that will be needed. Although most of the strategic decisions about the system are made in the development of the system concept during the analysis phase, the steps in the design phase determine exactly how the system will operate. The design phase has four steps:

1. The *design strategy* must be determined. This clarifies whether the system will be developed by the company's own programmers, whether its development will be outsourced to another firm (usually a consulting firm), or whether the company will buy an existing software package.
2. This leads to the development of the basic *architecture design* for the system that describes the hardware, software, and network infrastructure that will be used. In most cases, the system will add to or change the infrastructure that already exists in the organization. The *interface design* specifies how the users will move through the system (e.g., by navigation methods such as menus and on-screen buttons) and the forms and reports that the system will use.
3. The *database and file specifications* are developed. These define exactly what data will be stored and where they will be stored.
4. The analyst team develops the *program design*, which defines the programs that need to be written and exactly what each program will do. This collection of deliverables (architecture design, interface design, database and file specifications, and program design) is the *system specification* that is used by the programming team for implementation. At the end of the design phase, the feasibility analysis and project plan are reexamined and revised, and another decision is made by the project sponsor and approval committee about whether to terminate the project or continue.

Design Focus: How will this system work? Primary output: — System specification	7	Design physical system	Design strategy	Alternative matrix System specification
	8	Design architecture	Architecture design Hardware & software selection	— Architecture report — Hardware & software specification
	9	Design interface	Use scenario Interface structure Interface standards Interface prototype Interface evaluation	— Interface design
	10	Design programs	Data flow diagramming Program structure chart Program specification	— Physical process model — Program design
	11	Design databases and files	Data format selection Entity relationship modeling Denormalization Performance tuning Size estimation	— Database & file specification — Physical data model

Logical and Physical Design

System design can be of two types

1. Logical design

2. Physical design

   Logical design concerns with the specifications of major features of the system that would meet the objectives. The delivered product of a logical design may be called as a blueprint of the new system. A logical design defines *what* must take place, not *how* it will be accomplished. Logical designs do not address the actual methods of implementation. Logical design of a system includes content requirement and some or all of the following components:

   a. Outputs(reports, displays)
   b. Inputs(forms)
   c. Procedures(structure of procedures to collect, transform and output data)
   d. Storage(requirements for data to be stored in the database)
   e. Control(requirements for data integrity, security and procedures for recovery).

   In contrast, a physical design of a system takes the logical design blueprint and produces the program specifications, physical file or database definition with the help of this blueprint. Physical design shows how a task is completed. Typically, a physical design describes the actual processes of entering, verifying, and storing data; the physical layout of data files and sorting procedures, the format of reports, and so on.

   Because logical and physical designs are related so closely, good systems design is impossible without careful, accurate systems analysis.

Introduction to System Implementation:

System Implementation is the phase of the system development life cycle, in which the information system is programmed, tested, installed and supported. The purpose of System Implementation can be summarized as: making the new system available to a prepared set of users (the deployment), and positioning on-going support and maintenance of the system within the performing organization (the transition). At a finer level of detail, deploying the system consists of executing all steps necessary to educate the consumers on the use of the new system, placing the newly developed system into production, confirming that all data required at the start of operations is available and accurate and validating that business functions that interact with the system are functioning properly.

Transitioning the system support responsibilities involves changing from a system development to a system support and maintenance mode of operation, with ownership of new system moving from the project team to the performing organization. This phase consists of the following processes:

 Prepare for system implementation, where all steps needed in advance of actually deploying the application are performed, including preparation of both the production environment and the consumer communities.

 Deploy System, where the full deployment plan, initially developed during System Design and evolved throughout subsequent lifecycle phases, is executed and validated.

 Transition to Performing Organization, where responsibility for and ownership of the application are transitioned from the Project Team to the unit in the Performing Organization what will provide system support and maintenance. Implementation is the stage of a project during which theory is turned into practice. The major steps involved in this phase are:

 Acquisition and Installation of Hardware and Software

 Conversion

 User Training

 Documentation

The hardware and the relevant software required for running the system must be made fully operational before implementation. The conversion is also one of the most critical and expensive activities in the system development life cycle. The data from the old system needs to be converted to operate in the new format of the new system. The database needs to be setup with security and recovery procedures fully defined. During this phase, all the programs of the system are loaded onto the user's computer. After loading the system, training of the user starts. Main topics of such type of training are:

 How to execute the package

 How to enter the data

 How to process the data (processing details)

 How to take out the reports After the users are trained about the computerized system, working has to shift from manual to computerized working. The process is called 'Changeover'. The following strategies are followed for changeover of the system.

1. **Direct Changeover:** This is the complete replacement of the old system by the new system. It is a risky approach and requires comprehensive system testing and training.

2. **Parallel run:** In parallel run both the systems, i.e., computerized and manual, are executed simultaneously for certain defined period. The same data is processed by both the systems. This strategy is less risky but more expensive because of the following:

 Manual results can be compared with the results of the computerized system.

 The operational work is doubled.

 Failure of the computerized system at the early stage does not affect the working of the organization, because the manual system continues to work, as it used to do.

3. **Pilot run:** In this type of run, the new system is run with the data from one or more of the previous periods for the whole or part of the system. The results are compared with the old system results. It is less expensive and risky than parallel run approach. This strategy builds the confidence and the errors are traced easily without affecting the operations. The purpose of System Implementation is to take all possible steps to ensure that the upcoming system deployment and transition occurs smoothly, efficiently and flawlessly.

 System Installation:

System installation includes the installation of appropriate hardware and software. After development and testing, system must be put into operation.

Installation of System Hardware and Software Installation:

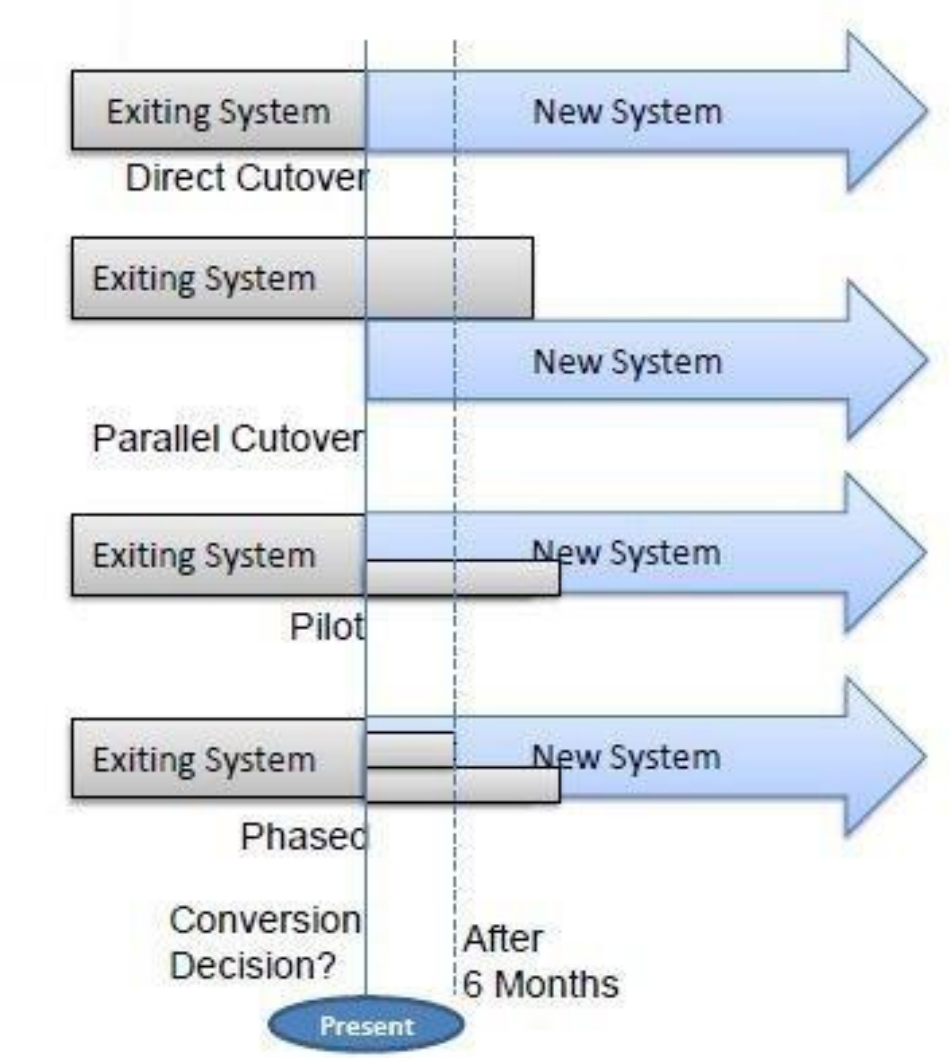
 Site preparation -- air-conditioning installation, cable trays, cable conduits, cable laying, installation of satellite relay station, power increase, installation of clean power, ergonomic furniture installation, negotiation for new space, building false floors; all this applies for multi-user equipment, not for personal machines and/or workstations.

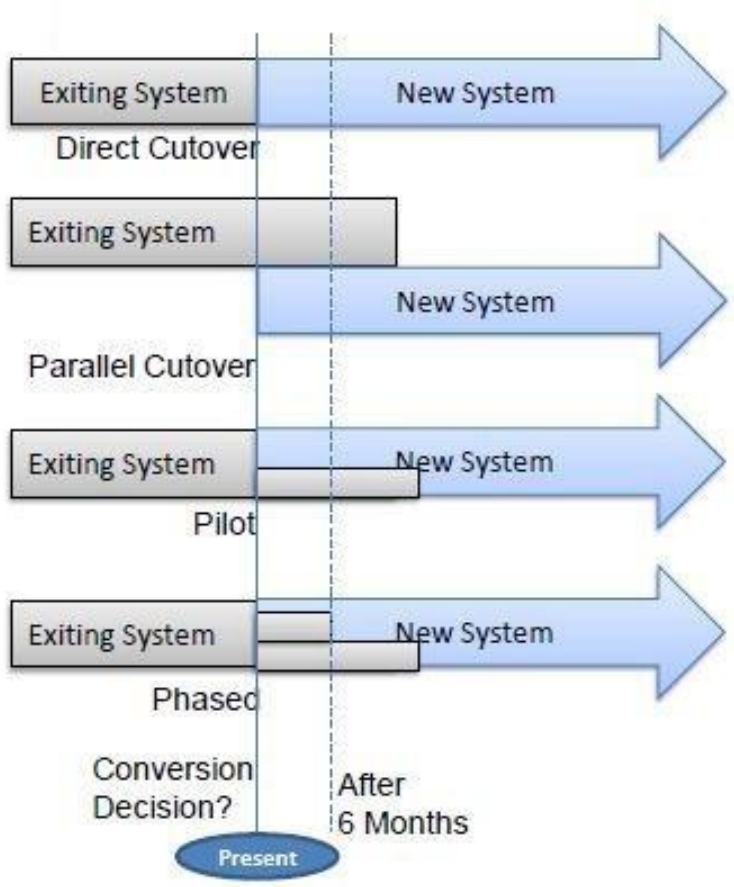
 Machine setup -- system loading and system testing, testing software for performance,

arrangement of furniture, training of programming personnel. System installation consists of conversions process which includes the activities such as data and system conversion.

 Data Conversion:

o Acquire **required** data from the existing systems
 Use data export tools
 Develop a data extraction program
o Consolidate & clean data from multiple sources
o Transform data as required by the new system
o Feed the data into the new system
 System Conversion:
o Choose a type of conversion
 Direct cutover
 Parallel
 Pilot
 Phased





Types of System Installation:

 Direct Installation:

o New system installed and quickly made operational

o Overlapping systems turned off

o Both systems concurrent for brief time

o Advantage – simplicity and fewer logistics issues to manage

o Disadvantage – risk due to no backup

 Parallel Installation:

o Old and new systems operated together for extended period of time

o **Advantages** – low risk of system failure and continual backup

o **Disadvantage** – cost to operate both systems

 Hiring temporary personnel

 Acquiring extra space

 Increasing managerial and logistical complexity

 Phased Installation:

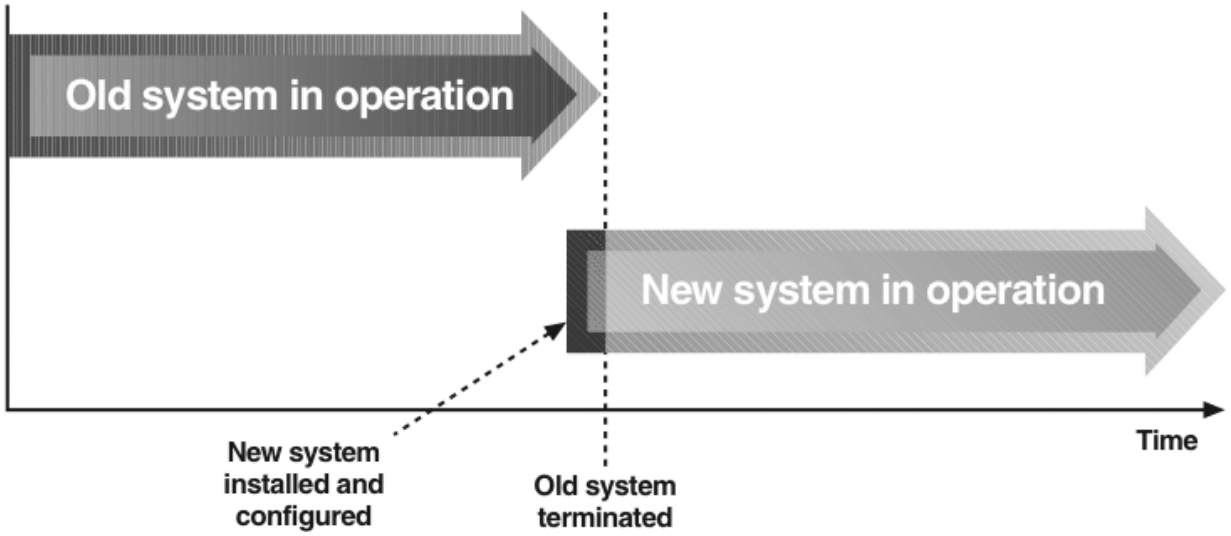
o New system installed in series of steps or phases

o Each phase adds components to existing system

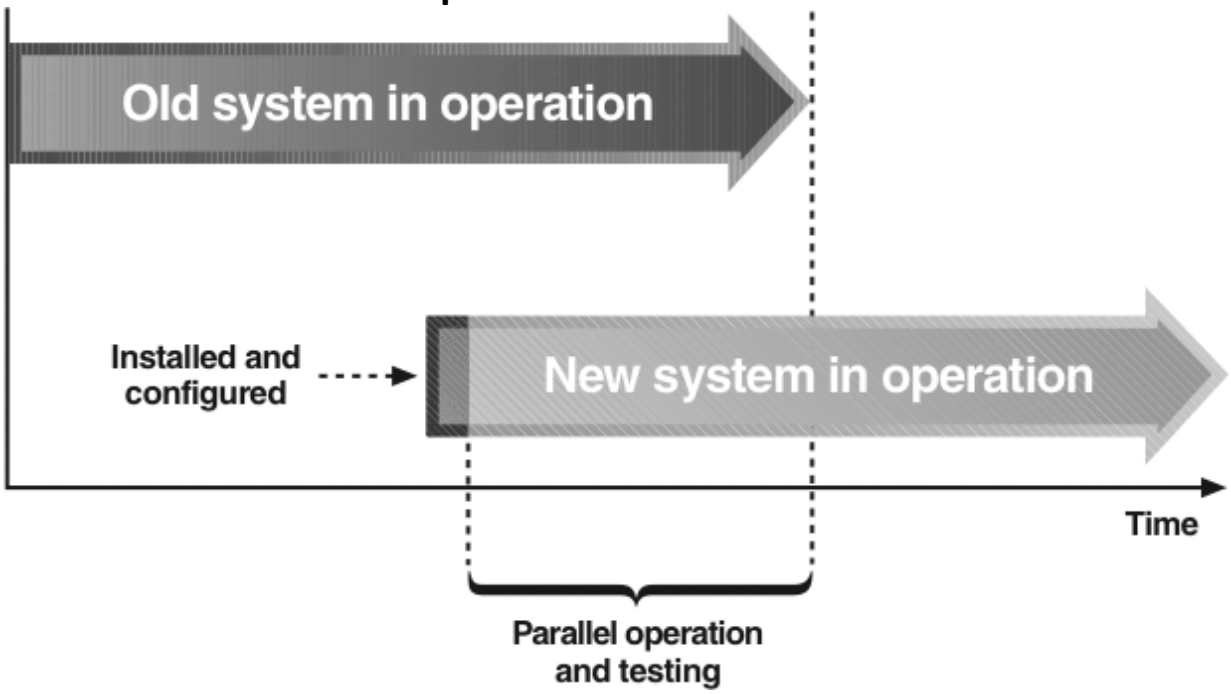
o **Advantage** – reduces risk because phase failure is less serious than system failure

o **Disadvantage** – multiple phases cause more activities, milestones, and management complexity for entire effort

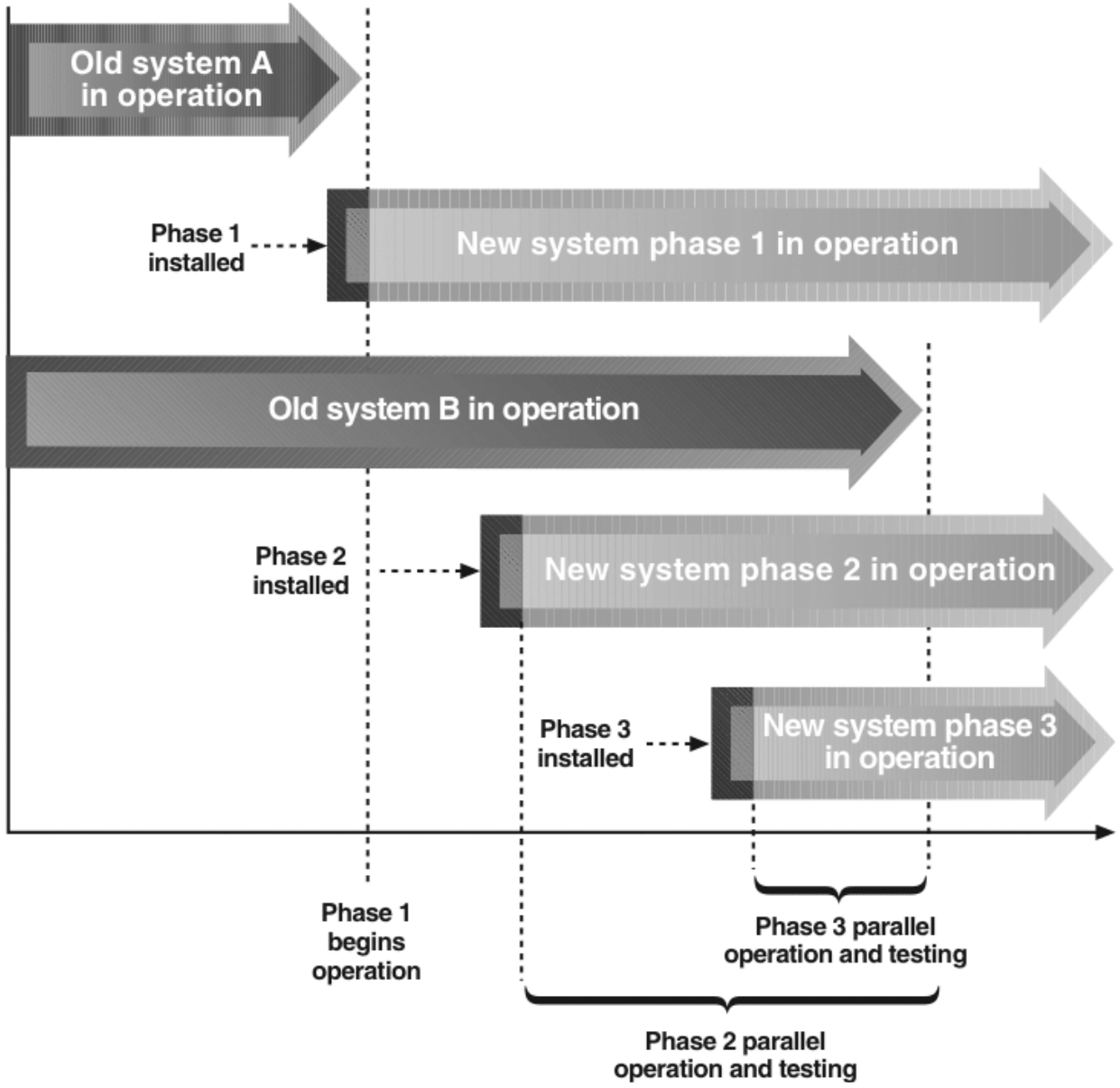
Direct Installation and Cutover:



Parallel Installation and Operation:



Phased Installation with Direct Cutover and Parallel Operation:



System Quality:

Software/system quality, by definition, is the degree to which software possesses a desired combination of attributes [IEEE 1992]. Therefore, to improve system quality, we must focus our attention on software-quality attributes. Ultimately, there are only a few system-quality attribute primitives to which all system qualities can map.

System-quality attributes:

Agility The ability of a system to both be flexible and undergo change rapidly.

Flexibility The ease with which a system or component can be modified for use in applications or environments, other than those for which it was specifically designed.

Interoperability The ability of two or more systems or components to exchange information and use the information that has been exchanged.

Maintainability The aptitude of a system to undergo repair and evolution.

(1) The ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment. (2) The ease with which a hardware system or component can be retained in, or restored to, a state in which it can perform its required functions.

Performance The responsiveness of the system—that is, the time required to respond to stimuli (events) or the number of events processed in some interval of time. Performance qualities are often expressed by the number of transactions per unit time, or by the amount of time that it takes to complete a transaction
with the system.

Reliability The ability of the system to keep operating over time. Reliability is usually measured by mean time to failure.

Reusability The degree to which a software module or other work product can be used in more than one computing program or software system. This is typically in the form of reusing software that is an encapsulated unit of functionality.

Scalability The ability to maintain or improve performance while system demand increases.

Security A measure of the system's ability to resist unauthorized attempts at usage and denial of service, while still providing its services to legitimate users. Security is categorized in terms of the types of threats that might be made to the system.

Supportability The ease with which a software system can be operationally maintained.

Testability The degree to which a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met.

Usability The measure of a user's ability to utilize a system effectively.The ease with which a user can learn to operate, prepares inputs for, and interprets outputs of a system or component. A measure of how well

users can take advantage of some system functionality. Usability is different from utility, which is a measure of whether that functionality does what is needed.

Software Quality Assurance:

Quality assurance is an integral part of information system development. It is important to keep in mind that quality assurance is not something that goes on separate to the development process. On the contrary, the development process must include checks throughout the process to ensure that the final product meets the original user requirements. Quality assurance thus becomes an important component of the development cycle through validation and verification performed at crucial system development steps. The goal of the management process is to institute and monitor a quality assurance program within the development process. A quality assurance program includes:

- validation of the system against requirements;
- checking for errors in design documents and in the system itself;
- checking for qualitative features such as portability and flexibility;
- checking for usability.

Each of these objectives may be met by different kind of quality assurance activity such as Technical Review, Walkthrough and Inspections. Reviews are usually made to check whether project management goals have been achieved, walkthroughs ate usually made to detect errors in the system, and inspections ate made to evaluate its qualitative features. It is also usual to distinguish between reviews made about project resource use, and checks as to whether a system model or technical proposal is correct.

Implementing Quality Assurance:

The approval is usually part of the management process and is included to ensure that all the necessary reviews have been completed. Reviews can take a number of forms-the most common are inspections and walkthroughs. Two things are often considered important in achieving quality products. One is a precise set of user requirements, against which general quality is measured. The second is documentation. In a large project, documentation is the central source of all the information needed by the team members. If it is incorrect or out of date, then team

members are working toward the wrong goals or using wrong inputs to their work. ○

Inspections:

An examination of a product to assure quality which consists of simplest checks is inspection. It is usual to allocate role to people involved in an inspection and outline a procedure for them to follow. Some common roles are the **procedure**, whose product is under review, the **inspector**, who evaluates the product, and the **moderator** who controls the review process. There is also a **reader**, who may guide inspectors through the product. It is important in such reviews that the people doing the inspection are not those that produced the product. Some software engineers suggest that there be a phased program of inspections running parallel with system development, with an objective for each phase and an evaluation made against the objective for that phase. Inspections themselves would be formal, with reports made at defined life-cycle phases. Each of the roles in an inspection team would have well-defied responsibilities within the inspection team. The inspections are formal and have a report that must be acted on. It is also important that any recommendations made during inspections be acted upon and followed up to ensure that any deficiencies are corrected.

Walkthroughs:

Walkthrough is a quality assurance activity to detect errors. The walkthrough is a procedure that is commonly used to check the correctness of models produced by structured systems analysis, although its techniques are applicable to other design methodologies. Such checking has always been necessary in systems analysis and design. Walkthroughs differ from earlier methods in that they recommend a specific checking procedure and walkthrough team structure. Furthermore, they allocate specific tasks to various members of the walkthrough team and require documentation to be produced during and after the walkthrough. The team must check that the model: ▪ meets system objectives;

- is a correct representation of the system;
- has no omissions and ambiguities;
- will do the job it is supposed to do; and
- is easy to understand.

Walkthrough can take place throughout system development. In structured systems analysis, they begin when the physical and logical

models of the existing system have been completed. The first walkthrough checks the existing system models to detect omissions and inaccuracies in them. Walkthrough should also be carried out on the new logical design to detect flaws, weaknesses, errors and omissions in the proposed design. The walkthroughs should be made first on the new logical model and later the new physical model.

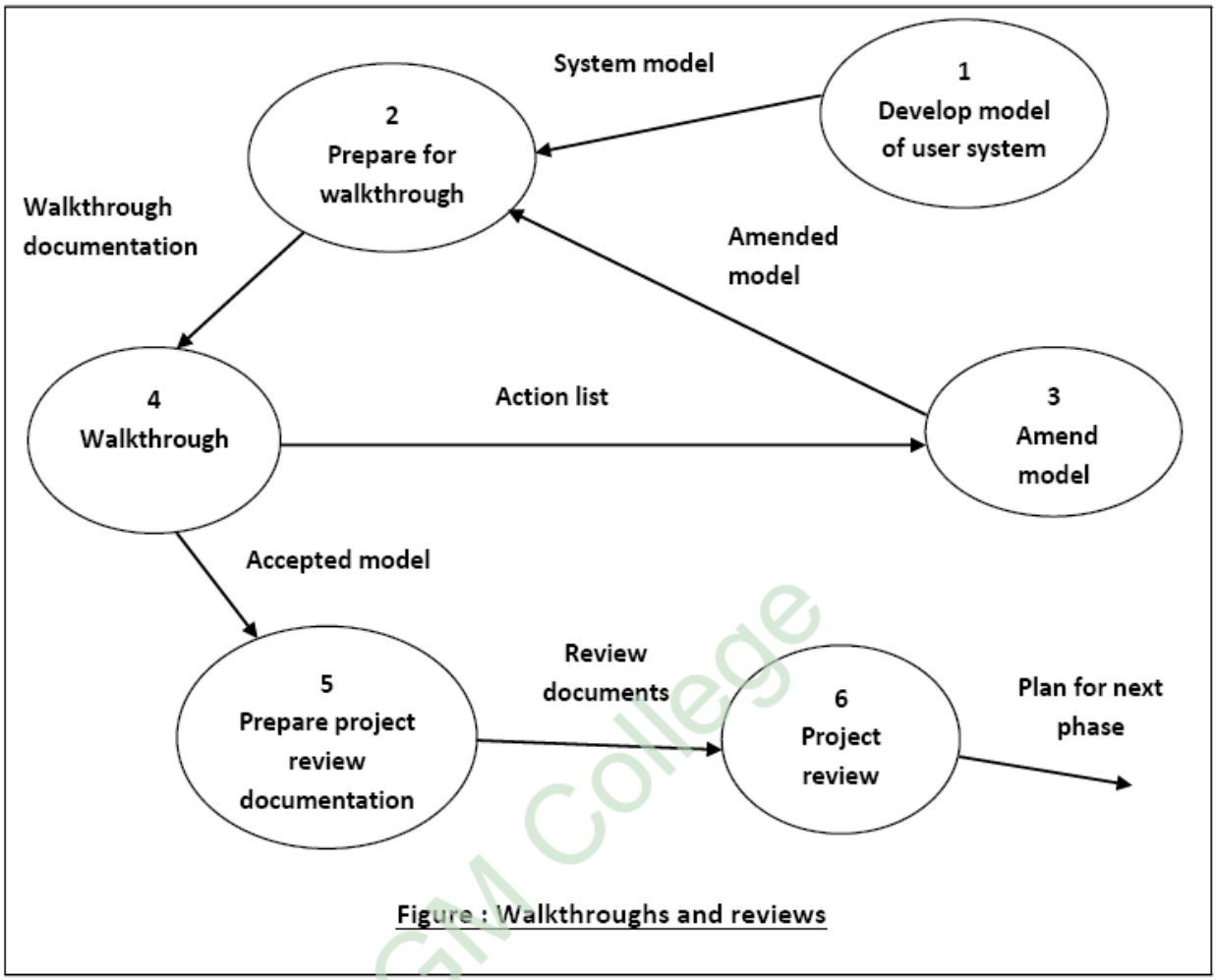


Figure : Walkthroughs and reviews

System Maintenance:

A common perception of maintenance is that if merely involves fixing defects. System maintenance is a very broad activity that includes error corrections, enhancements of capabilities, deletion of obsolete capabilities and optimization. Because change is inevitable, mechanism must be developed for evaluation, controlling and making modifications. The results obtained from the evaluation process help the organization to determine whether its information systems are effective and efficient or otherwise.

System maintenance is an ongoing activity, which covers a wide variety of activities, including removing program and design errors, updating

documentation and test data and updating user support. For the purpose of convenience, maintenance may be categorized
into three classes, namely:

1. Corrective:

This type of maintenance implies removing errors in a program, which might have crept in the system due to faulty design or wrong assumption. Thus, in corrective maintenance, processing or performance failures are repaired.

2. Adaptive:

In adaptive maintenance, program functions are changed to enable the information system to satisfy the information needs of the user. This type of maintenance may become necessary because of organizational changes which may include:

- Change in the organizational procedures
- Change in organizational objectives, goals, polices, etc
- Change in forms
- Change in information needs of managers
- Change in system controls and security needs, etc

3. Perfective:

Perfective maintenance means adding new programs or modifying the existing programs to enhance the performance of the information system. This type of maintenance is undertaken to respond to user's additional needs which may be due to the changes within or outside of the organization. Outside changes are primarily environmental changes, which may in the absence of system maintenance render the information system ineffective and inefficient. These environmental changes include:

- Changes in governmental policies, laws, etc
- Economic and competitive conditions
- New technology

System maintenance process:

This section describes the six software maintenance process as:

1. The implementation process contains software preparation and transition activities, such as the conception and creation of the maintenance plan; the preparation for handling problem identified during development; and the follow-up on product configuration management.
2. The problem and modification analysis process, which is executed once the application has become the responsibility of the maintenance group. The maintenance programmer must analyze each request, confirm

it(reproducing the situation) and check its validity, investigate it and propose a solution, document the request and the solution proposal, and finally, obtain all the required authorizations to apply the modifications.
3. The process considering the implementation of the modification itself.
4. The process acceptance of the modification, by confirming the modified work withthe individual who submitted the request in order to made sure the modification provide a solution.
5. The migration process is exceptional, and is not part of daily maintenance tasks. If the software must be ported to another platform without any change in functionality, this process will be used and a maintenance project team is likely to assign to this task.
6. Finally, the last maintenance process, also an event which does not occur on a daily basis, is the retirement of a piece of software (execution of the system).

System testing:

The importance of system testing to system quality cannot be overemphasized. System testing is a critical element of software quality assurance and represents the ultimate review of specification, design and code generation. Once source code has been generated, software must be tested to allow errors to be identified and removed before delivery to customer. While it is not possible to remove every error in a large software package, the software engineer's goal is to remove as many as possible early in the software development life cycle. It is important to remember that testing can only find error; it cannot prove that a program is bug free. The greater visibility of software system and the cost associated with software failure are motivating factors for planning through testing. It is not uncommon for a software organization to spent 40% of its effort on testing. Two basic test techniques involve testing module input/output (black-box) and exercising internal logic of software components (white-box). Testing must be planned and designed.

Testing Objectives:

- Testing is the process of executing a program with the intent of finding errors.
- A good test case is one with a high probability of finding an as-yet undiscovered error.
- A successful test is one that discovers an as-yet undiscovered error.

Testing process:

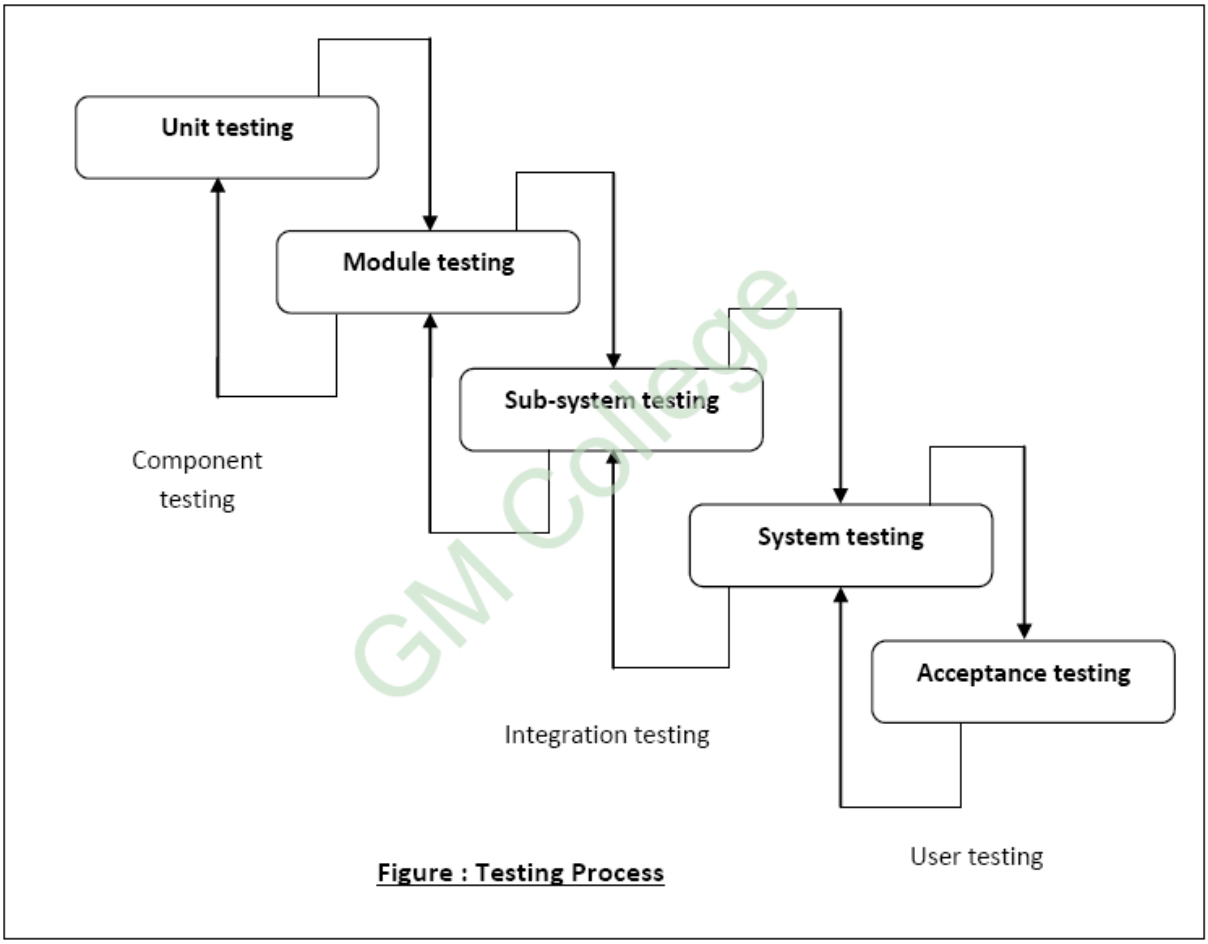


Figure : Testing Process

- **Unit testing: -** Individual components are tested independently, without other system components.
- **Module testing:-** Related collection of dependent components (class, ADT, procedures & functions) are tested, without other system module.
- **Sub-system testing:-** Modules are integrated into sub-systems are tested. The focus here should be on interface testing to detect module interface errors or mismatches.
- **System testing:**- Testing of the system as a whole. Validating functional and nonfunctional requirements & testing of emergent system properties.
- **Acceptance testing:**- Testing with customer data to check that it is acceptable. Also called Alpha testing.

Component testing:

Each individual program component is tested. Usually it is the responsibility of the component

developer to perform component testing. Such tests are derived from the developer's experience.

Integration testing:

Groups of components integrated to create a system or sub-system is tested. It is the responsibility of an independent testing team. Such tests are based on a system specification.

.

Test Case Design Strategies

Black-box testing	White-box testing
Also referred as Function testing as it tests the functionality of the system.	Also referred as Structural testing as it checks the working of all independent logic paths.
Also known as behavioral testing as system behavior is determined by studying its input and related the outputs.	Also known as glass-box testing as its internal coding part should be tested and verified.
Called "testing in the large"	Called "testing in the small" as applied to small program components.
Knowledge of internal program logic is not important.	Knowledge of internal program logic is an essential.
Objective is to check it meets the user requirements as per the specification has not been fulfilled.	Objective is to check all program statement as per the design specification.
Discover **faults of omission**, indicating the part of the specification has not been fulfilled.	Discover **faults of commission**, indicating that part of the implementation is faulty.
Mostly performed by the user's side.	Mostly performed by the programmer's side.
Black-box testing is done at the later stage.	White-box testing is done at the earlier stage.
Performed as per the acceptance test plan.	Performed as per the component and integration test plan.
Errors include incorrect or missing functions, interface error, errors in data structures or external database access, behavior or performance errors, and initiation and termination errors.	Errors include logical errors in the coding statement and typing error.

OBJECT ORIENTED ANALYSIS AND DESIGN

Object-oriented Development Life Cycle:

In the object-oriented design approach, the system is viewed as collection of objects (i.e. entities). The state is decentralized among the objects and each object manages its own state information. For example, in a Library Automation Software, each library member may be a separate object with its own data and functions to operate on these data. In fact, the functions defined for one object cannot refer or change data of other objects. Objects have their own internal data which define their state. Similar objects constitute a class. In other words, each object is a member of some class. Classes may inherit features from super class. Conceptually, objects communicate by message passing.

Function-oriented vs. object-oriented design approach

The following are some of the important differences between function-oriented and object oriented design.

o Unlike function-oriented design methods, in OOD, the basic abstraction are not real world functions such as sort, display, track, etc, but real world entities such as employee, picture, machine, radar system, etc. For example in OOD, an employee pay-roll software is not developed by designing functions such as update-employee-record, get employee address, etc. but by designing objects such as employees, departments, etc. Grady Booch sums up this difference as "identify verbs if you are after procedural design and nouns if you are after object-oriented design".

o In OOD, state information is not represented in a centralized shared memory but is distributed among the objects of the system. For example, while developing an employee pay-roll system, the employee data such as the names of the employees, their code numbers, basic salaries, etc. are usually implemented as global data in a traditional programming system; whereas in an object-oriented system these data are distributed among different employee objects of the system. Objects communicate by message passing. Therefore, one object may discover the state information of another object by interrogating it. Of course, somewhere or other the real-world functions must be implemented. In OOD, the functions are usually associated with specific real-world entities (objects); they directly access only part of the system state information.

o Function-oriented techniques such as SA/SD group functions together if, as a group, they constitute a higher-level function. On the other hand, object-oriented techniques group functions together on the basis of the data they operate on.

Object-Oriented Systems Development:

Object-oriented system development model is the conceptual frame work for all things of object oriented. There are four major elements of object model. They are:

1. Abstraction
2. Encapsulation
3. Modularity
4. Hierarchy

There are three minor elements which are useful but not essential part of object model.

Minor elements of object model are:

1. Typing
2. Concurrency
3. Persistence

Abstraction:

Abstraction is defined as a simplified description or specification of a system that emphasizes some of the system details or properties while suppressing others. A good abstraction is one that emphasizes details that are significant to the reader or user and suppressed details that are not so significant, immaterial. An abstraction denotes the essential characteristics of an object that distinguishes it from all other kinds of objects. An abstraction focuses on the outside view of an object.

Abstraction focuses up on the essential characteristics of some object, relative to the perspective of the viewer.

Encapsulation:

It is the act of grouping data and operations into a single object. Example:

```
Class heater {
Public:
heater(location);
void turnon();
void turnoff();
Boolean ison() const;
}
```

Modularity:

The act of partitioning a program into individual components is called modularity. It is reusable component which reduces complexity to some degree. Although portioning a program is helpful for this reason, a more powerful justification for partitioning a program is that it creates a number of well-defined, documented boundaries within the program. These boundaries, or interfaces, are invaluable in the comprehension of the program. Modularity is the property of a system that has been decomposed into a set of cohesive and
loosely coupled modules.

- Modules can be complied separately.
- Modules serve as physical containers in which classes and objects are declared like gate in IC of computer.

 Group logically related classes and objects in the same module.

Hierarchy:

Hierarchy is a ranking or ordering of abstractions. Encapsulation hides Company inside new of abstraction and modularity logically related abstraction & thus a set of abstractions form hierarchy. Hierarchies in complex system are its class structure and its object structure.

Benefits of the object oriented development life cycle:

Object model introduces several new elements which are advantageous over traditional method of structural programming. The significant benefits are:

 Use of object model helps us to exploit the expressive power of object based and object oriented programming languages. Without the application of elements of object model, more powerful feature of languages such as C++, object pascal, ada are either ignored or greatly missed.

 Use of object model encourages the reuse of software and entire designs, which results in the creation of reusable application framework.

 Use of the object model produces systems that are built upon stable intermediate

forms, which are more resilient to change.

 Object model appears to the working of human cognition, many people who have no idea how a computer works find the idea of object oriented systems quite natural.

- **The Unified Modeling Language:**

UML, as the name implies, is a modeling language. It may be used to visualize, specify, construct, and document the artifacts of a software system. It provides a set of notations (e.g. rectangles, lines, ellipses, etc.) to create a visual model of the system. Like any other language, UML has its own syntax (symbols and sentence formation rules) and semantics (meanings of symbols and sentences). Also, we should clearly understand that UML is not a system design or development methodology, but can be used to document object-oriented and analysis results obtained using some methodology

UML Diagram:

UML can be used to construct nine different types of diagrams to capture five different views of a system. Just as a building can be modeled from several views (or perspectives) such as ventilation perspective, electrical perspective, lighting perspective, heating perspective, etc.; the different UML diagrams provide different perspectives of the software system to be developed and facilitate a comprehensive understanding of the system. Such models can be refined to get the actual implementation of the system. The UML diagrams can capture thefollowing five views of a system:

- User's view
- Structural view
- Behavioral view
- Implementation view
- Environmental view

User's View:

This view defines the functionalities (facilities) made available by the system to its users. The users' view captures the external users' view of the system in terms of the functionalities offered by the system. The users' view is a black-box view of the system where the internal structure, the dynamic behavior of different system components, the implementation etc. are not visible. The users' view is very different from all other views in the sense that it is a functional model compared to the object model of all other views. The users' view can be considered as the central view and all other views are expected to conform to this view. This thinking is in fact the crux of any user centric development style.

Structural view:

The structural view defines the kinds of objects (classes) important to the understanding of the working of a system and to its implementation. It also captures the relationships among the classes (objects). The structural model is also called the static model, since the structure of a system does not change with time.

Behavioral view:

The behavioral view captures how objects interact with each other to realize the system behavior. The system behavior captures the time-dependent (dynamic) behavior of the system.

Implementation view:

This view captures the important components of the system and their dependencies.

Environmental view:

This view models how the different components are implemented on different pieces of hardware.

• Use case Modeling:

The use case model for any system consists of a set of “use cases”. Intuitively, use cases represent the different ways in which a system can be used by the users. A simple way to find all the use cases of a system is to ask the question: “What the users can do using the system?” Thus for the Library Information System (LIS), the use cases could be:

- issue-book
- query-book
- return-book
- create-member
- add-book, etc

Use cases correspond to the high-level functional requirements. The use cases partition the system behavior into transactions, such that each transaction performs some useful action from the user’s point of view. To complete each transaction may involve either a single message or multiple message exchanges between the user and the system to complete.

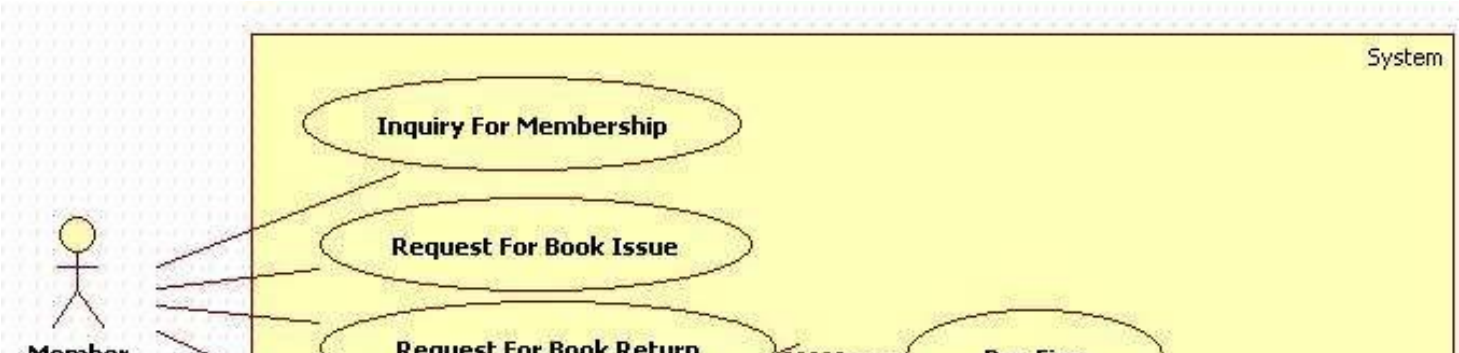
Purpose of use cases

The purpose of a use case is to define a piece of coherent behavior without revealing the internal structure of the system. The use cases do not mention any specific algorithm to be used or the internal data representation, internal structure of the software, etc. A use case typically represents a sequence of interactions between the user and the system. These interactions consist of one mainline sequence. The mainline sequence represents the normal interaction between a user and the system. The mainline sequence is the most occurring sequence of interaction. For example, the mainline sequence of the withdraw cash use case supported by a bank ATM drawn, complete the transaction, and get the amount. Several variations to the main line sequence may also exist. Typically, a variation from the mainline sequence occurs when some specific conditions hold. For the bank ATM example, variations or

alternate scenarios may occur, if the password is invalid or the amount to be withdrawn exceeds the amount balance. The variations are also called alternative paths. A use case can be viewed as a set of related scenarios tied together by a common goal. The mainline sequenceand each of the variations are called scenarios or instances of the use case. Each scenario is a single path of user events and system activity through the use case.

Representation of use cases

Use cases can be represented by drawing a use case diagram and writing an accompanying text elaborating the drawing. In the use case diagram, each use case is represented by an ellipse with the name of the use case written inside the ellipse. All the ellipses (i.e. use cases) of a system are enclosed within a rectangle which represents the system boundary. The name of the system being modeled (such as Library Information System) appears inside the rectangle. The different users of the system are represented by using the stick person icon. Each stick person icon is normally referred to as an actor. An actor is a role played by a user with respect to the system use. It is possible that the same user may play the role of multiple actors. Each actor can participate in one or more use cases. The line connecting the actor and the use case is called the communication relationship. It indicates that the actor makes use of the functionality provided by the use case. Both the human users and the external systems can be represented by stick person icons. When a stick person icon represents an external system, it is annotated by the stereotype <<external system>>.



I

- **Object Modeling: Class Diagrams:**

Class diagrams:

A class diagram describes the static structure of a system. It shows how a system is structured rather than how it behaves. The static structure of a system comprises of a number of class diagrams and their dependencies. The main constituents of a class diagram are classes and their relationships: generalization, aggregation, association, and various kinds of dependencies.

Classes:

The classes represent entities with common features, i.e. attributes and operations. Classes are represented as solid outline rectangles with compartments. Classes have a mandatory name compartment where the name is written centered in boldface. The class name is usually written using mixed case convention and begins with an uppercase. The class names are usually chosen to be singular nouns.

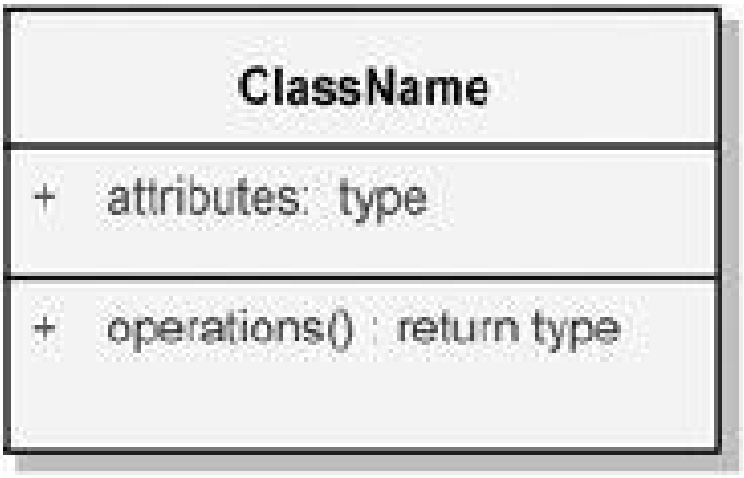
Classes have optional attributes and operations compartments. A class may appear on several diagrams. Its attributes and operations are suppressed on all but one diagram.

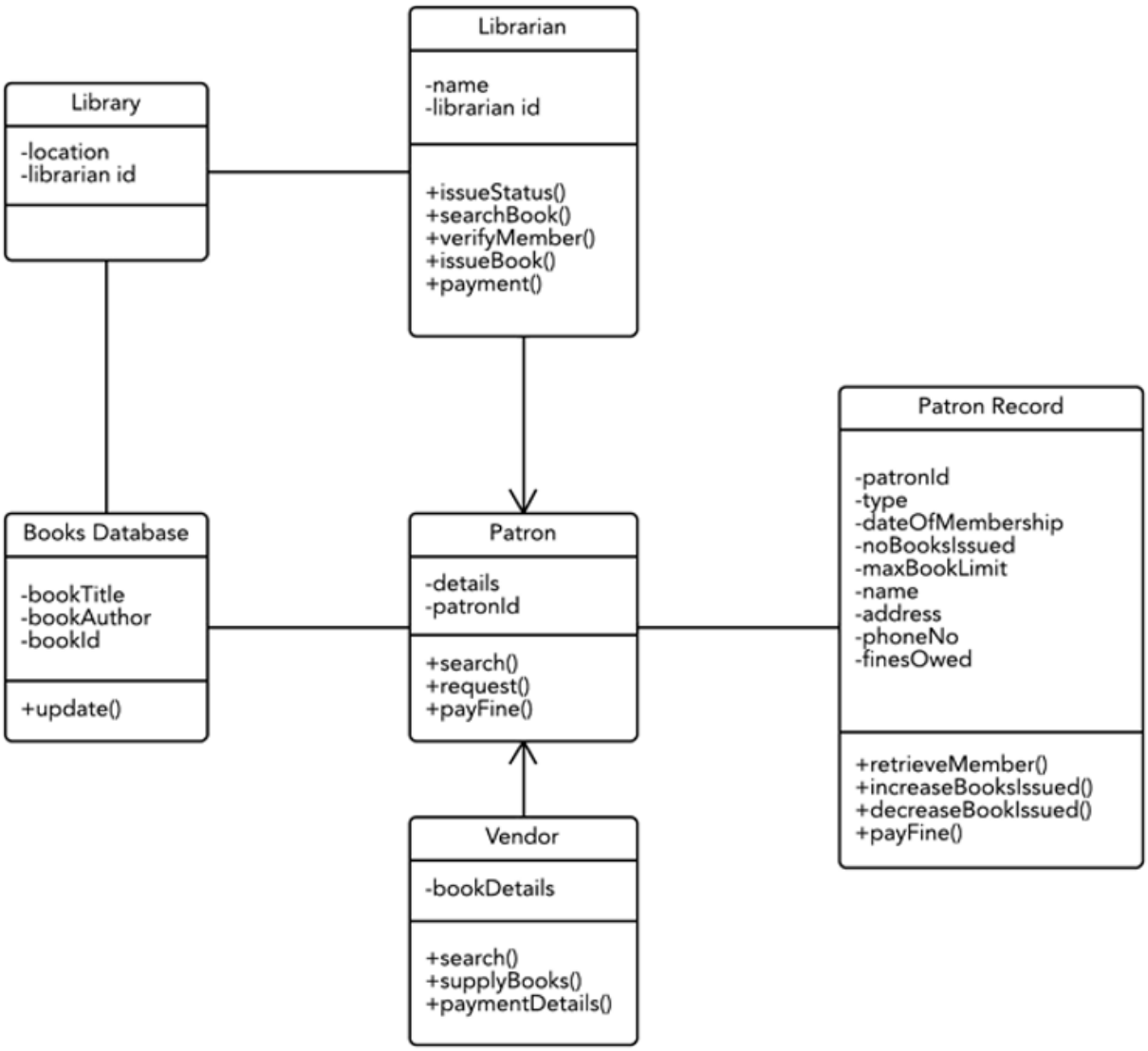
Attributes:

An attribute is a named property of a class. It represents the kind of data that an object might contain. Attributes are listed with their names, and may optionally contain specification of their type, an initial value, and constraints. The type of the attribute is written by appending a colon and the type name after the attribute name. Typically, the first letter of a class name is a small letter. An example for an attribute is given. **bookName: String**

Operation:

Operation is the implementation of a service that can be requested from any object of the class to affect behavior. An object's data or state can be changed by invoking an operation of the object. A class may have any number of operations or no operation at all. Typically, the first letter of an operation name is a small letter. Abstract operations are written in italics. The parameters of an operation (if any), may have a kind specified, which may be 'in', 'out' or 'inout'. An operation may have a return type consisting of a single return type expression. An example for an operation is given. **issueBook(in bookName):Boolean**





- **State chart diagram**

A state chart diagram is normally used to model how the state of an object changes in its lifetime. State chart diagrams are good at describing how the behavior of an object changes across several use case executions. However, if we are interested in modeling some behavior that involves several objects collaborating with each other, state chart diagram is not appropriate. State chart diagrams are based on the finite state machine (FSM) formalism. An FSM consists of a finite number of states corresponding to those of the object being
modeled. The object undergoes state changes when specific events occur. The FSM formalism existed long before the object-oriented technology and has been used for a wide variety of applications. Apart from modeling, it has even been used in theoretical computer science as a generator for regular languages.

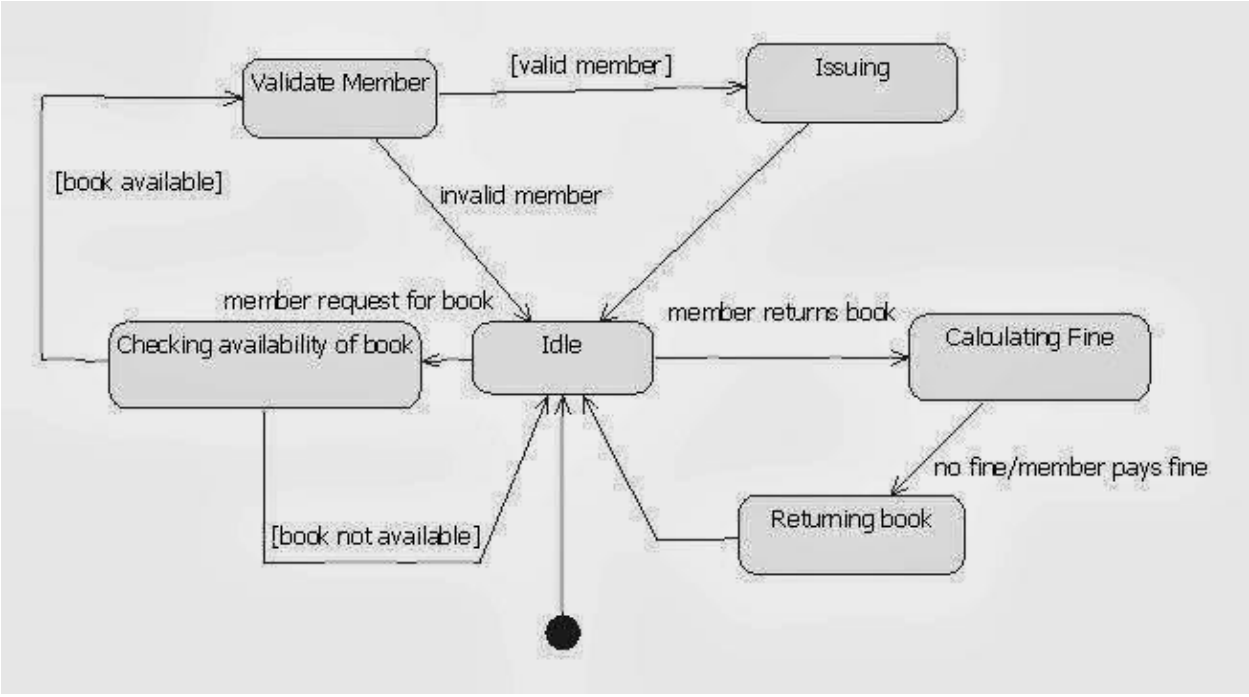
A major disadvantage of the FSM formalism is the state explosion problem. The number of states becomes too many and the model too complex when used to model practical systems.This problem is overcome in UML by using state charts. The state chart formalism was proposed by David Harel [1990]. A state chart is a hierarchical model of a system and introduces the concept of a composite state (also called nested state).
Actions are associated with transitions and are considered to be processes that occur quickly and are not interruptible. Activities are associated with states and can take longer. An activity can be interrupted by an event.

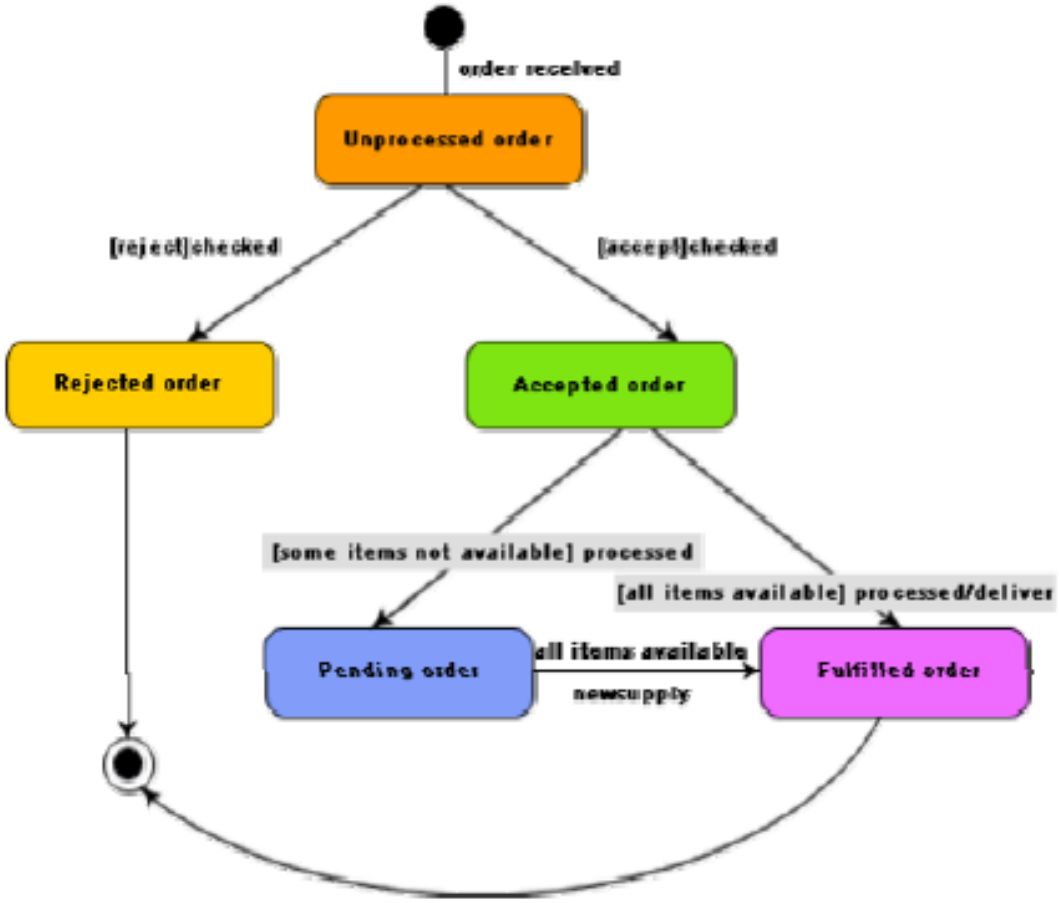
The basic elements of the state chart diagram are as follows:

- Initial state. This is represented as a filled circle.
- Final state. This is represented by a filled circle inside a larger circle.
- State. These are represented by rectangles with rounded corners.
- Transition. A transition is shown as an arrow between two states.

Normally, the name of the event which causes the transition is places alongside the arrow. A guard to the transition can also be assigned. A guard is a Boolean logic condition.

State diagram of library





- **Sequence Diagram:**

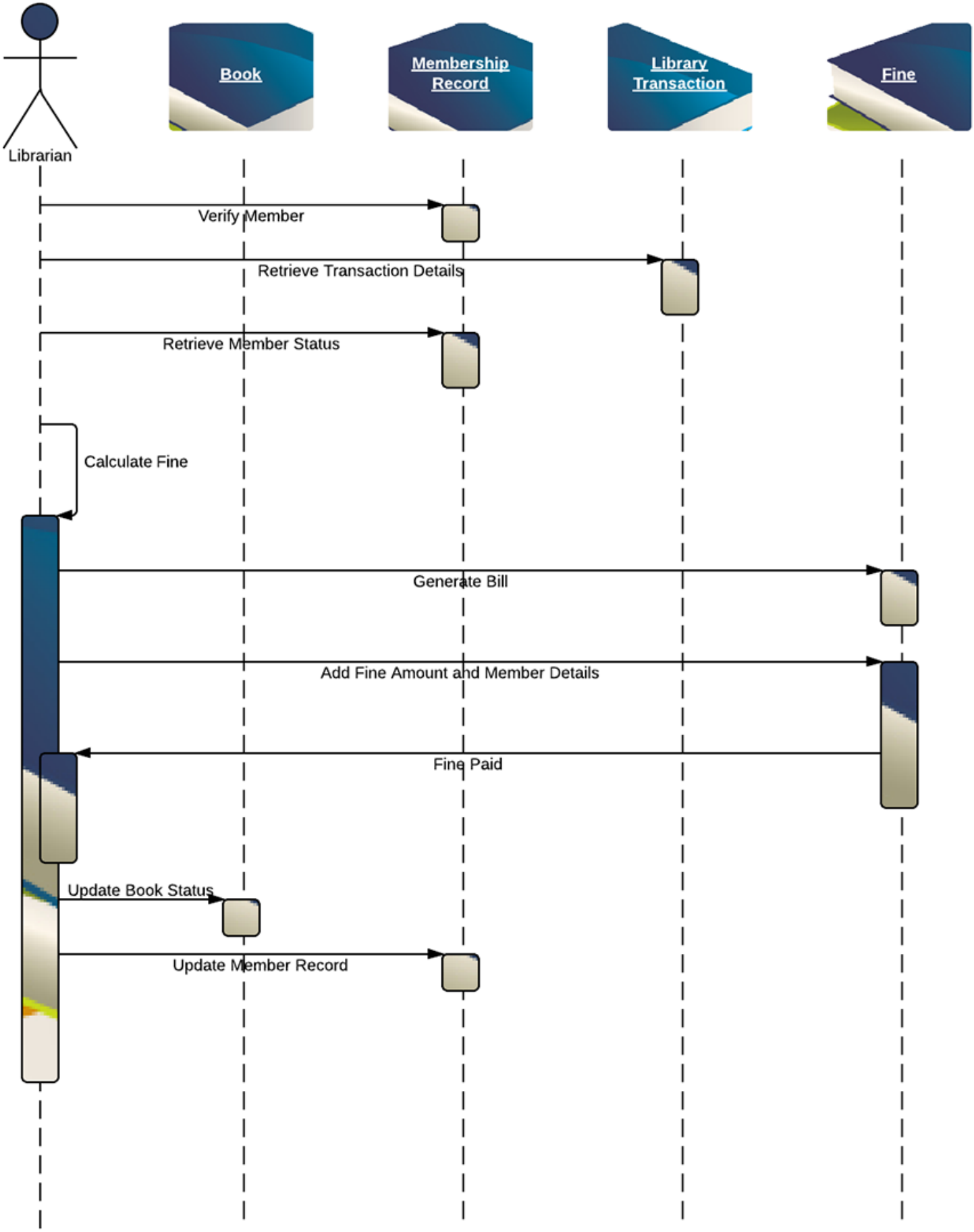
Sequence Diagram is a graphical view of a scenario that shows object interactions in a time based sequence – what happens first, what happens next. Sequence diagrams establish the roles of objects and help provide essential information to determine class responsibilities and interfaces. This type of diagram is best used during early analysis phases

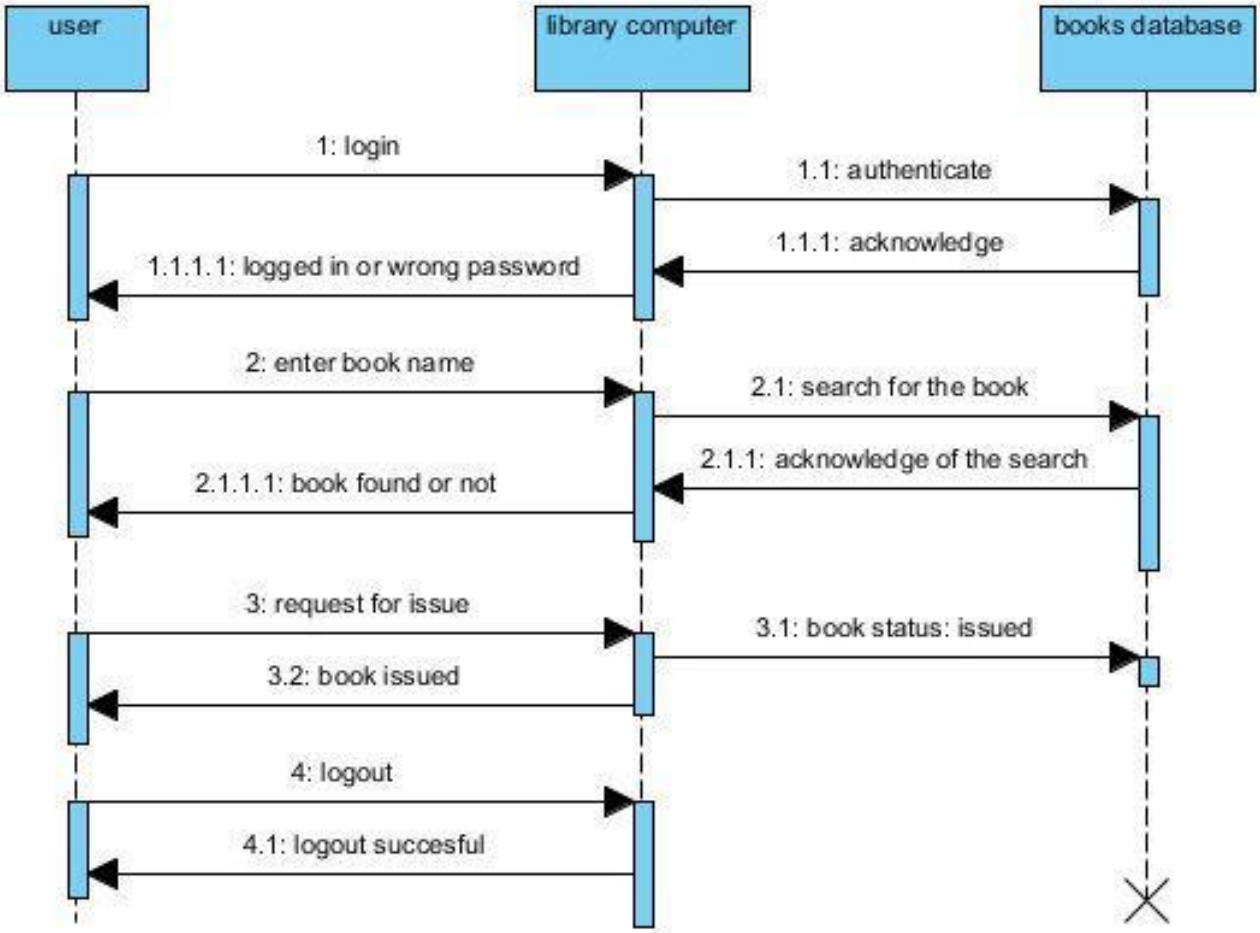
in design because they are simple and easy to comprehend. Sequence diagrams are normally associated with use cases.

A sequence diagram has two dimensions: typically, vertical placement represents time and horizontal placement represents different objects. The objects in a sequence diagrams are generally:

- User
- Boundary
- Control
- Entity

A sequence diagram represents the interactions between these objects and also shows the time involved in the interactions.


Librarian
Book
Membership Record
Library Transaction
Fine
Verify Member
Retrieve Transaction Details
Retrieve Member Status
Calculate Fine
Generate Bill
Add Fine Amount and Member Details
Fine Paid
Update Book Status
Update Member Record


user
library computer
books database
1: login
1.1: authenticate
1.1.1: acknowledge
1.1.1.1: logged in or wrong password
2: enter book name
2.1: search for the book
2.1.1: acknowledge of the search
2.1.1.1: book found or not
3: request for issue
3.1: book status: issued
3.2: book issued
4: logout
4.1: logout succesful