UNIT I

INTRODUCTION

Systems analysis is the study of a problem domain of any organization which help to recommend improvements and specify the requirements for the solution where as the systems study is the specification of computer based solution for the requirements identified in a system analysis. To illustrate, let us imagine a traditional library without computers.

NEED OF SYSTEMS ANALYSIS

It is hard to imagine any industry of business that has not been affected by computer-based information systems and computer applications. Many businesses consider management of their information resource to be equal in importance to managing their information resource to be equal in importance to managing their other key resources like property, facilities, equipment, employees and capital.

Organizations consider information systems and information technology to be essential to their ability to compete or gain a competitive advantage. Most businesses realize that all workers need to participate in the development of information systems – not just the information technology specialists. This is true for the case of libraries also.

WHAT IS A SYSTEM

"**System":**- The term system is derived from the Greek word 'systema' which means- an organized relationship among functioning component or unit ".

Example: -

- The transportation system
- The telephone system

"Elias M. Awad defines system as an orderly grouping of interdependent components linked together according to a plan to achieve a specific objective."

THE SYSTEM CONCEPT

Herbert A. Simon, a political scientist, related the systems concept to the study of organization by viewing an ongoing system as a processor of information for making decisions. Systems analysis and design for information systems were founded in general systems theory, which emphasizes a close look at all parts of a system. Too often analysts focus on only one component and overlook other equally important components. General systems theory is concerned with "developing a systematic, theoretical framework upon which to make decisions." It discourages thinking in a vacuum and encourages consideration of all the activities of the organization and its external environment. Pioneering work in general systems theory emphasized that organizations be viewed as total systems. The idea of systems has become most practical and necessary in conceptualizing the interrelationships and integration of operations, especially when using computers. Thus, a *system* is a way of thinking about organizations and their problems; It also involves a set of techniques that helps in solving problem.

Characteristics of a System

All systems, be it a library system, billing system, accounting system, transportation system or medical system have certain common characteristics. These are: -

(a) Systems are organized:

Structure and order are an integral part of a system. The components in the system are arranged in a particular order and have a structure to achieve specific objectives.

For example, in a library system, the hierarchical relationships start with the Chief Librarian on the top and go downward to the level of library clerks. This arrangement portrays a system-subsystem relationship, defines the authority structure, specifies

the formal flow of communication, and formalizes the chain of command (Figure 1.1)

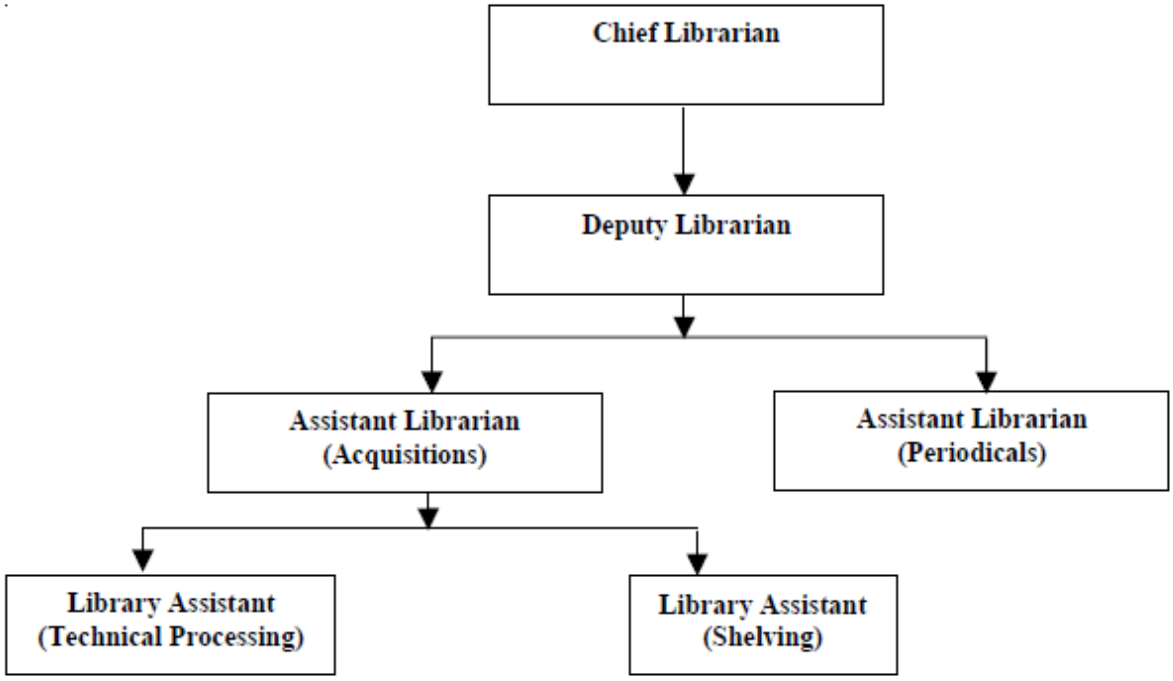


Figure 1.1: Library Structure (*abstract and illustrative*)

b) System components *interact* with each other:

Interaction is the manner in which each component in the system functions with other components. In a library, circulation component interacts with users; them acquisition component department interacts with the cataloguing and classification department.

c) System components are *interdependent:*

Interdependence means that parts of the organization or computer system depend on one another. They are co-ordinate or linked together according to a plan. One subsystem depends on the input of another system for proper functioning; that is, the output of one subsystem is the required input for another system. No subsystem can function in isolation because it is dependent on the inputs it receives from other sub-systems to perform its required tasks.

d) System components are *integrated:*

Integration is concerned with how a system is tied together. It is more than sharing a physical part or location. It means that parts of the system work together within the system even though each part performs a unique function. Successful integration will typically produce synergistic effect and greater total impact than if each component works separately.

e) System has a *central objective:*

A system works to achieve a central objective. The objective of a library system is to meet the information needs of the user.

Element of System: -

A system may be constructed by considering the following six elements.

(1) Output and Input: A major objective of a system is to produce an output that has value to its user.

Input are the element that enter in a system for processing, and to produce output in much the same way that a business bring in human, financial and other resource to produce good and services.

(2) Processors: - The processor is an element of a system that involves the actual transformation of input into output. The processor is the operational component of system.

(3) Control: - The control element guides the system. It is the decision-making subsystems that control the pattern of activities governing input, processing and input.

(4) Feedback: - In a dynamic system, control is archived by means of feedback, which measure output against a standard in some form of procedure that include communication and control.

(5) Environment: - The environment is the surrounding system within which an organization operates. It is the source of external elements that impose on the system.

(6) Boundaries and interface: -A system should be defined by its boundaries – the limit that identify its components, processes and interrelationship when it interfaces it interfaces with another system.

TYPES OF SYSTEM

The frame of reference within which one views a system is related to the use of the systems approach for analysis. Systems have been classified in different ways. Common classifications are;

(1) physical or abstract

(2) open or closed

(3) "man-made" information systems.

Physical or Abstract - *Physical* systems are tangible entities that may be static *or* dynamic in operation. For example, the physical parts of the computer center are the offices, desks, and chairs that facilitate operation of the computer. They can be seen and counted; they are static. In contrast, a programmed computer is a dynamic system. Data, programs, output, and applications change as the user's demands or the priority of the information requested changes. *Abstract* systems arc conceptual or nonphysical entities. They may be as straightforward as formulas of relationships among sets of variables or models—the abstract conceptualization of physical situations. A model is a representation of a real or a planned system. The use of models makes it easier for the analyst to visualize relationships in the system under study. The objective is to point out the significant elements and the key interrelationship of the complex system.

Open system

"A system is said to be open system if there exist an interaction between the system and environment".

It is permitted to have interaction across the boundary in an open system. In system analysis, organization, computer systems are open, dynamic system, which is influenced by their environment.

Five important characteristic of open system: -

1) **Steady state: -** Open system are self-adjusting and self-regulating. When functioning properly, an open system achieves a steady state.
2) **Entropy: -** All dynamic system tends to run down over time, resulting in entropy or loss of energy. Open system resists entropy by seeking new input or modifying process.
3) **Process, Output and Cycle:** - Open system operates in cycle and follows a defined process to produce useful output.
4) **Differentiation:** - Open system usually has increasing specialization of function and a greater differentiation of their component.
5) **Equifinality:** -This characteristic implies those goals are achieved through differing course of action and a variety of path.

Closed Systems

A closed system is one that is isolated from environmental influences. In reality, a closed system is rare. In system analysis, organizations, applications, and computers are invariably open, dynamic systems influenced by their environment. The concept of closed system is more relevant to scientific systems than to social systems.

Formal Information Systems

A formal information system is based on the organization represented by the organization chart. The chart is a map of position and their authority

relationship, Indicated by boxes and connected by straight lines it is concern with a pattern of authority,

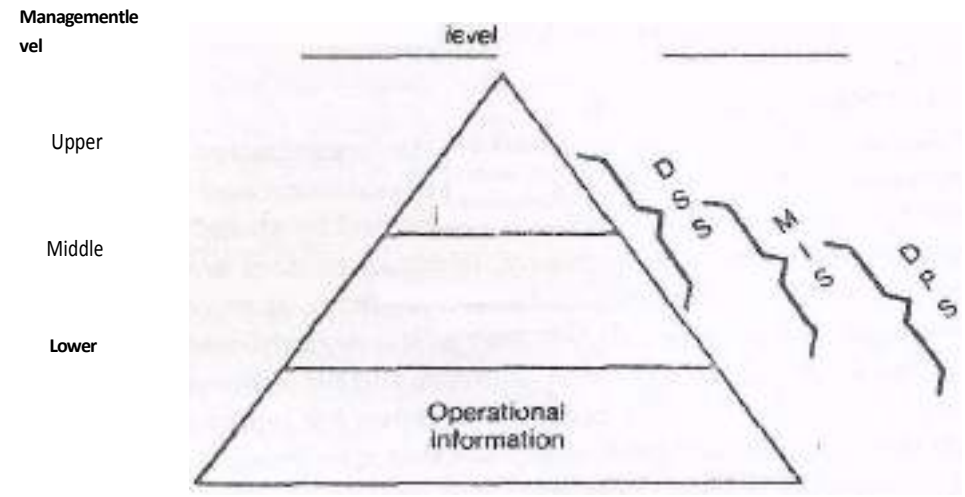
communication and work flow. Information is formally maintained in instructions, memos, and reports from top management to the intended user in the organization.

The rules of formal information system are based policies are translated into directives, rules and regulations and transmitted to lower level management for implementation. The output represents employee performance.

Categories of Information: There are three categories of information related to managerial levels and decisions managers make.

- The first level is strategic information, which relates to long range planning policies that are direct interacting to upper management.
- The second level information is managerial levels and department heads for implementation and control.

- The third information level is optional information, which is short term, daily information used to operate departments and enforce.



Informal information system

The formal information system is a power structure designed to achieve company goals. An organization emphasis on control to ensure performance tends to restrict the communication flow among the employee, however as a result an informal system develops. It is an employee based system designed to meet personnel and vocational needs and to help solve work related problems. In this respect, it is a useful system because it works within the framework of the business and its stated policies.

In doing a system study, the analyst should have a knowledge of the chain command, the power of authority influence network, and how decision are made to get a feel for how much support can be expected for a prospective installation.

Man-made Information Systems

An information system is the basis for interaction between the user and the analyst. It provides instructions, commands and feedback. It determines the nature of relationships among decision makers. In fact, information system

may be viewed as a decision center for personnel at all levels. From this basis, an information system may be defined as a set of devices, procedures, and operating systems designed around user-based criteria to produce information and communicate it to the user for planning, control, and performance .In systems analysis, it is important to keep in mind that considering an alternative system means improving one or more of these criteria.

Computer Based Information system

A third class of information system relies on the computer for handling business applications. The computer is now a required source of information. Systems analysis relies heavily on computers for problem solving. This suggests that the analyst must be familiar with computer technology and have experience in handling people in an organizational context.

Management Information System: -

Management functioning is based on accurate, relevant, complete, concise, and timely information. Management information system are concerned with such management function in a quickly and responsive manner.

"MIS can be described as information system that can be providing all levels of management with information essential to the running of smooth business."

The information must be as relevant, timely, accurate, complete, and concise and economically feasible.

- MIS should always operate in real time so that information must be available to affect a decision.
- The primary user of middle and top management, operational manager and support staff.

- Middle and top management use MIS to prepare forecasts, special request for analysis, long range plans and periodic reports.
- Operational manager use MIS primarily for short planning and periodic and exception report.
- Support staffs find MIS useful for the special analyst\is of information and reports to help management in planning and control.

Decision Support System: -

DSS advances capabilities of MIS in making decisions; it is actually a continually evolving model that relies heavily on operation research. The term quite simple as:

"Decision-emphasize decision making in problem situations".

DSS advances capabilities of MIS in making decisions; it is actually a continually evolving model that relies heavily on operation research.

Decision—emphasizes decision making in problem situations, not information processing, retrieval, or reporting.

Support—requires computer-aided decision situations wish enough "structure" to permit computer support.

System-accentuates the integrated nature of problem solving, suggesting a combined "man" machine, and decision environment.

MIS and DSS are closely related to each other since a DSS cannot offer the required results until supported by an MIS. MIS is having the database, i.e. a collection of data items that can be processed through application programs. It is the only repository of data, which provides input for a system like DSS. On the other hand, MIS can only provide the limited support to the top management for the decision-making. DSS advances the capabilities of

MIS.Decision support system assist manager, who must make decision that is not highly structured, often called unstructured or semi structured decision.

Herbert Simon described decision making as a three-phase continuous process model beginning with ***intelligence*** and moving toward ***design*** and ***choice.***

System Development Life Cycle Method

It is a step-by-step, structured process for the development of a system. This process draws upon the engineering approach to the analysis and solution of complex problems. It is based upon the concept of a sequence of necessary activities through which systems must pass during the period of their creation and existence, or life cycle. Within the lifecycle, significant series of related activities are combined into phases, called the lifecycle phases. The concept of lifecycle has lead to the adoption of a project-oriented approach to the design and development of systems. This method is referred to as the life-cycle methodology, commonly referred to as the systems development life cycle, or SDLC.

System development life cycle is sometimes also referred to as system study. To understand system development, it has to be recognized that a system has a life cycle, just like a living system. In general the activities that must be performed throughout the systems development life cycle and the sequence in which they are performed are well identified. The stages in system development life cycle are shown in figure 1.2

a) **Recognition of need**: One must know what the problem is before it can be

solved. It could be improving or redesigning an existing system or procedure.

For example, compiling a bibliography on a topic would involve scanning periodicals for collecting the references which is time consuming. An alternative could be to build a database that can solve the problem of time delay.

An analyst's first task is to prepare a statement specifying the scope and objective of the problem. Atthis stage, only a rough estimate of the development cost of the project may be reached.

Impetus for System Change

• The idea for change originates in the environment or from within the firm. environment-based idea originates from customers, vendors, government sources and the like.

• For example, new unemployment compensation regulations may make it necessary to change the reporting procedure, format, and content of various reports, as well as file structures.

• Customer complains about the delivery of orders may prompt an investigation of the delivery schedule, the experience of truck drivers, or the volume of orders to be delivered.

• When investigated, each of this idea may lead to a problem definition as a first step in that system life cycle process.

• Ideas for change may also come from within the organization top management, the user, the analyst).

• As an organization changes its operations or faces advances in computer technology, someone within the organization may fell the need to update existing application or improve procedures. Here are some examples

• An organization acquires another organization.

• A local bank branches into the suburbs.

• A department spends 80 percent of its budget in one month.

• A department is doing essentially the same work, and each department head insists the other department should be eliminated.

• A request for a new form discloses the use of bootleg (unauthorized) forms.

Serious problem in operations, a high rate of labor turnover, labor intensive activities, and high reject rats of finished goods, also prompt top management to initiate an investigation. Other examples are:

• A report reaches a senior vice president and she suspect the figures.

• The company comptroller read an IRS audit report and starts thinking.

• An executive read about decision support systems for sales forecasting and it gives him an idea.

• Many of these ideas lead to further studies by management request, often funneled downward and carried out by lower management.

• User-originated ideas also prompt initial investigations.

• For example, a bank's need teller has been noticing long customer line in the lobby.

• She wants to know whether they are due to the computer's slow responses to inquiries, the new tellers limited training, or just a sudden increase in bank business.

• To what extent and how quickly a user-originated idea is converted to a feasibility study depend on several factors:

• The risks and potential returns.

• Management's bias toward the user.

• Financial costs and the funds available for system work.

• Priorities of other projects in the firm.

• The persuasive ability of the user.

• All these factors are crucial for a prompt response to a user request for change.

• A system analyst is in a unique position to detect and even recommend change.

• Experience and previous involvement in the user area of operations make him/her a convenient resource for idea.

• The role and status of the analyst as a professional add credibility to the suggestions made.

b) **Feasibility study:**

• Depending on the results of the initial investigation, the survey is expanded to a more detailed feasibility study.

• A feasibility study is a test of system proposal according to it workability, feasibility study is a test of a system proposal according to it workability, impact on the organization, ability to meet user needs, and effective use of resources.

• It focuses on three major questions:

1. What are the user's demonstrable needs and how does a candidate system meet them?

2. What resources are available for give candidate system? Is the problem worth solving.?

3. What is the likely impact of the candidate system on the organization?

How well done it fit within the organization's master MIS plane?

• Each of these questions must be answered carefully. They revolve around investigation and evaluation of he problem identification and description of candidates system, specification of performance and the cost of each system, and final selection of the best system.

• The objective of a feasibility study is not to solve the problem but to acquire a sense of its scope. During the study the problem definition is crystallized and aspects of the problem to be included in the system are determined.

• Consequently, costs and benefits are estimated with greater accuracy at this stage. The result of the feasibility study is a formal proposal. This is implying a report-a formal document detailing the nature and scope of the proposed solution. The proposal summarizes what is known what is going to be done.

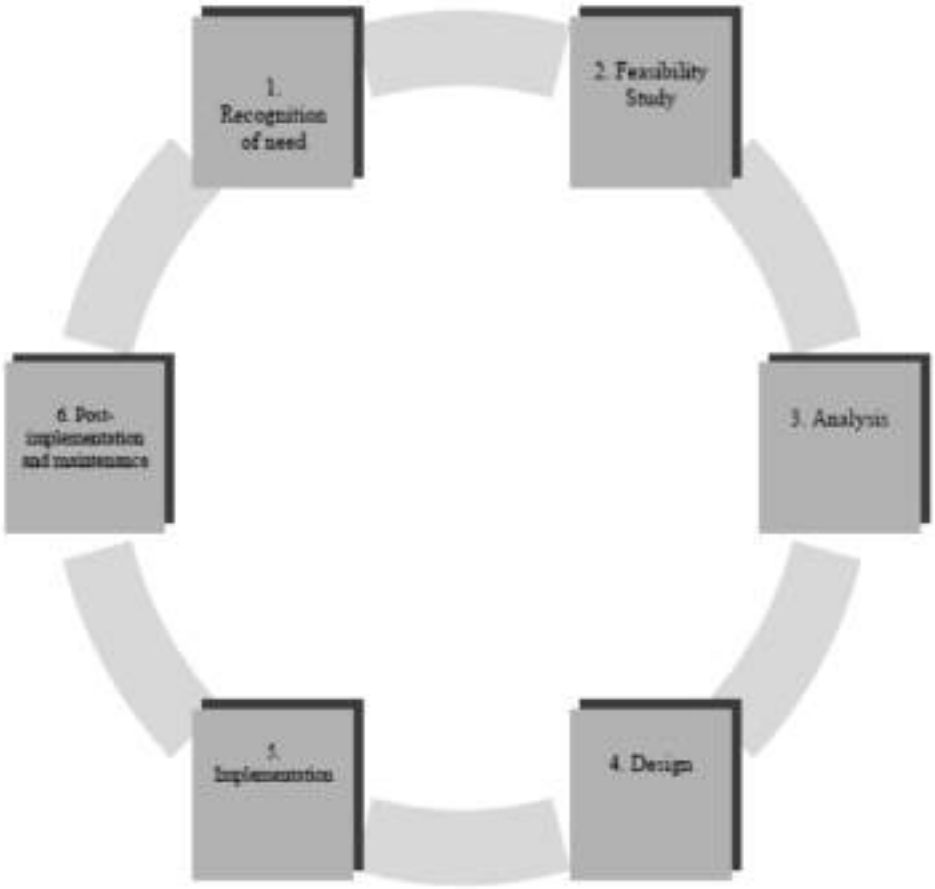


Figure 1.2: System Development Lifecycle

c) **Analysis:** Analysis is a detailed study of the various operations performed by a system and their relationships within and outside the system. During analysis, data are collected on the available files, decision points, and transactions handled by the present system. Data flow diagrams, interviews, on-site observations, and questionnaires are tools used collect data and information. Once analysis is completed, the analyst has a firm understanding of what is to be done. The next step is to decide how the problem might be solved.

• Gather the requirements for the system.

• This stage includes a detailed study of the business needs of the organization.

• Options for changing the business process may be considered

d) **Design:** The term design describes a final system and the process by which it is developed. It refers to technical specifications that will be applied to implementing the candidate system. It also includes the construction of programs and program testing.

The first step in system design is to determine who the output is to be produced and in what format. Samples of the output (and input) are also presented. Second, input data and master files (data base) have to be designed to meet the requirements of the output. The operational (processing) phase handled through program construction and testing, including a list of the programs needed to meet the system's objectives and complete documentation. Finally, details related to justification of the system and an estimate of the impact of the candidate system on the user and the organization are documented and evaluated by management as a step towards implementation.

This focuses on high level design (what programs are we going to need and how are they going to interact), low level design (how the individual programs are going to work), interface design (what are the interfaces going to look like) and data design (what data are we going to need).

e) **Implementation:** The implementation stage is concerned with user training, site preparation, and file conversion. During the final testing, user acceptance is tested, followed by training. Conversion usually takes place at about the same time the user is being trained or later. System testing checks the readiness and accuracy of the system to access, update, and retrieve data from new files. Once the programs become

available, test data are read into the computer and processed against the files provided for testing. If successful, the program(s) is then run with actual data. The new system runs simultaneously with the "old" system to enables the staff to gain experience and learn the new system.

• The designs are translated into code.

• Computer programs may be written using a conventional programming language to a *fourth* generation language (4GL) or an application generator.

f) **Post-implementation and maintenance**: For the successful and trouble free Working of the system periodic maintenance of hardware and software is carried Out. Table 1 summarizes the stages giving the key question that should be asked in each stage and the expected outcome from the stage.

• The system is tested. Normally programs are written as a series of individual modules - these should be subject to separate and detailed test.

• The *system* is then tested as a whole - the separate modules are brought together and tested as a complete system.

• The system needs to be tested to ensure that interfaces between modules work (integration testing), the system works on the intended platform and with the expected volume of data

(volume testing) and that the system does what the user requires(acceptance/beta testing).

• Inevitably the system will need maintenance - hopefully we haven't got anything wrong but people will want extra things added or existing things changed over time.

• This paradigm is the oldest and most widely used approach to system development, it was developed by Royce in 1970.

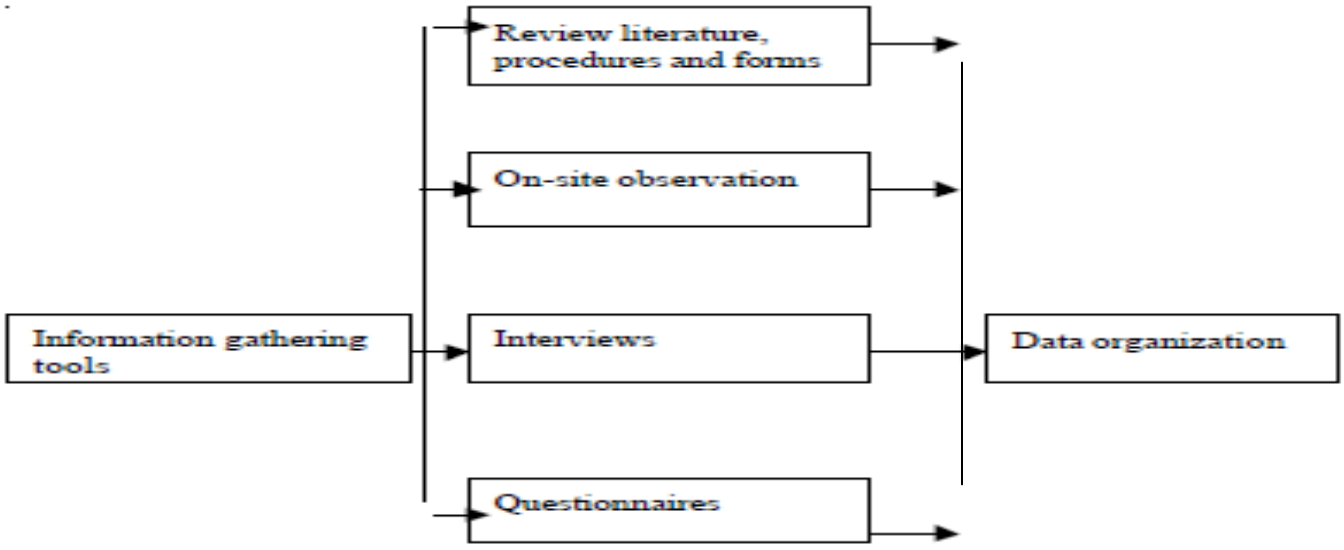
Table 1.1: System Development Life Cycle

Stage	Key Question	Result
1. Recognition of need Preliminary survey/initial investigation	What is the problem or opportunity?	Statement of scope and objectives. Performance criteria
2. Feasibility study Evaluation of existing system and procedures Analysis of alternative candidate systems Cost estimates	What are the user's demonstrable needs? Is the problem worth solving? How can the problem be redefined?	Technical/ behavioral feasibility. Cost/ benefit analysis. System scope and objectives. Statement of new scope and objectives
3. Analysis Detailed evaluation of present system Data collection	What must be done to solve problem? What are the facts?	Logical model of system- e.g. data dictionary, data flow diagram. Pertinent data
4. Design General design specifications Detailed design specifications Output Input Files Procedures Program construction Testing Unit testing Combined module testing User acceptance testing	In general how the problem must be solved? Specifically, how must the problem be solved? What is the system (processing) flow? Does the user approve the system? How well do individual programs/ modules test out? How ready are programs for acceptance test? and operating	Design of alternative solutions. Final cost/ benefit analysis. Hardware specifications. Cost estimates Implementation specifications Implementation schedule Approval of systems by user Programs Test plans. Security, audit, procedures. Actual hardware use. Formal system test
5. Implementation User training File/system conversion	What is the actual operation? Are user manuals ready? Are there delays in loading files?	Training program User-friendly documentation
6. Post-implementation and maintenance Evaluation Maintenance Enhancements	Is the key system running? Should the system be modified?	User requirements met. User standards met. Satisfied user.

Information-gathering Tools

No two projects are ever the same. This means that the analyst must decide on the information-gathering tool and how it must be used. Although there are no standard rules for specifying their use, an important rule is that information must be acquired accurately, completely, and concisely. For example, if the analyst need only information available in existing manuals, then interviewing is unnecessary except where the manual is not up to date. If additional information is needed, on-site observation or a questionnaire may be considered. Therefore, we need to be familiar with various information-gathering tools. Each tool has a special function, depending on the information needed.

Information-Gathering Methods



the information may be outdated due to a time lag in publication. Procedures manuals and forms are useful sources for the analyst. They describe the format and functions of the present system. Included in most manuals are

systems requirements that help determine however various objectives are met. Up-to-date manuals save hours of information-gathering time. Printed forms are widely used for capturing and providing information. The objective is to understand how forms are used.

The following questions may be useful.

1. Who uses the form(s)? How important are they to the user?

2. Do the forms include all the necessary information? What items should be added or deleted?

3. How many departments receive the existing form(s)? Why?

4. How readable and easy to follow are the form?

5. How does the information in the form help other users make better decisions? What other uses does the form offer the user area?

On-Site Observation

Another information-gathering tool used in system studies is on-site observation. It is the process of recognizing and noting people, objects, and occurrences to obtain information. The analyst's role is that of an information seeker who is expected to be detached from the system being observe. This role permits participation with the user staff openly and freely. The major objective of on-site observation is to get as close as possible to the "real" system being studied.

The following questions can serve as a guide for on-site observations.

1. What kind of system is it? What does it do?

2. Who runs the system? Who are the important people in it?

3. What is the history of the system? How did it get to its present stage of development?

4. A part from its formal function, what kind of system is it in comparison with other systems in the organization? Is it fast paced or is it a leisurely system that responds slowly to external crises?

As an observer, the analyst follows a set of rules. While making observations, he/she is more likely to listen than talk and to listen with a sympathetic and genuine interest when information is conveyed. The emphasis is not on giving advice or passing moral judgment on what is observed. Furthermore, care is taken no to argue with the persons being observed or to show undue friendliness toward another. When human observers are used, four alternative observations methods are considered:

Natural or contrived

A natural observation occurs in a setting such as the employee's place of work. A contrived observation is set up by the observer in his place.

Obtrusive or Unobtrusive.

An obtrusive observation takes place when the respondent knows he/she is being observed. An unobtrusive observation takes place in a contrived way such as being a one-way mirror.

Direct or indirect.

A direct observation takes place when the analyst actually observes the subject or the system at work. In an indirect observation, the analyst uses mechanical devices such as cameras and videotapes to capture information.

Structured or unstructured.

In a structured observation the observer looks for and records specific action such as the number of soup cans a shopper picks up before choosing one. Unstructured methods place the observer in a situation to observe whatever might be pertinent at the time.

Any of these methods may be used in information gathering. Natural, direct, obtrusive, and unstructured observations are frequently used to get an overview of an operation. The degree of structure is increased when observations has a specific purpose. The degree of obtrusiveness may decrease when one wants to observe the tasks that make up a given job. Contrived situations are used to test or debug a candidate system. They are also used in training program to help evaluate the progress of trainees. Electronic observation and monitoring methods are becoming widely used information-gathering tools because of their speed, efficiency, and low cost.

On-site observations are not without problems: -

1. Adequate preparation and training are important.

2. Attitudes and motivations of subjects cannot be readily observed.

3. Observation is subject to error due to the observer’s misinterpretation and subjective selection of what to observe.

4. Unproductive, long hours are often spent in an attempt to observe specific, one-time activities or events.

For on-site observation to be done properly in a complex situation it can be very time-consuming Proper sampling procedures must be used to ascertain the stability of the behavior being observed.

Without knowledge of stability, inferences drawn from small samples of behavior (small time slices) can be inaccurate.

In deciding to use an on-site Observation, Several Questions are considered

- What behaviour can be observed that cannot be described in other ways?
- What data can be obtained more easily or more reliably by observation than by other means?
- What assurances can be given that the observation process is not seriously, affecting the system or the behaviour being observed?
- How much skill is required and available for the actual observation?

As we have seen, on-site observation is directed primarily toward describing and understanding event as they occur. It has limitations when we need to learn about people's perceptions, feelings, or motivations. Therefore, other information-gathering tools are also use for analysis. Information-gathering tools can be categorized by their degree of directness. If we wish to know about something, we simply ask someone about it directly, but we may not get an answer. Most of the information-gathering tools used in system analysis are relatively direct. Yet properly handled, information can be successfully obtained with interviews or questionnaires.

Interviews and Questionnaires

The interview is a face-to-face interpersonal role situation in which a person called the interviewer asks a person being interviewed questions designed to gather information about a problem area. The interview is the oldest and most often used device for gathering information in system work. It has qualities that behavioural and on-site observations do not possess.

It can be used for two Main Purposes

(1) As an exploratory device to verify information.

(2) To capture information.

There are four primary advantages of the interview

1. Its flexibility makes the interview a superior technique for exploring areas where not much is known about what question to ask or how to formulate questions.

2. It offers a better opportunity than the questionnaire to evaluate the validity of the information gathered.

3. It is an effective technique for collecting information about complex subjects.

4.Many people enjoy being interviewed, regardless of the subject.

The major drawback of the interview is the long preparation time. Interviews also take a lot of time to conduct, which means time and money. So whenever a more economical alternative captures the same information, the interview is generally not used.

UNIT II

Feasibility Study

To determine what the candidate system is to do by defining its expected performance. Thus a feasibility study is carried out to select the best system that meets performance requirements

Types of Feasibility Study

- Economic Feasibility
- Technical Feasibility
- Behavioral Feasibility

Economic Feasibility

Also known as cost benefit analysis. To determine the benefits and savings that are expected from a candidate system and compare them with costs. If Benefits outweigh Costs, then the decision is made to Design and Implement the system.

Cost-Benefit Analysis

When new systems are proposed, the cost is a major consideration which impacts on the decision to accept or reject the proposed system.

The people funding the system want to know whether they will get a good return for the money they invest in the system. In large companies there is often competition to get access to financial resources for innovative projects.

So, to win support for a project, estimates of costs and benefits must be calculated. This is called a cost-benefit analysis.

There are two main areas for costs: the development costs and the operating costs once a system is introduced.

There are tangible and intangible costs and tangible and intangible benefits. Tangibles can be easily calculated; intangibles cannot.

Tangible costs

Tangible costs include the following.

1. **Personnel Costs**

- salaries of analysts, programmers, consultants, data entry personnel , etc
- costs for trainers and employees' time

2. **Costs for equipment for the new system**

- computer hardware
- space/rooms
- furniture

3. Supporting material

- stationery
- photocopying

4. **Converting to new system**

- designing new processes and procedures

- running new and old systems together for a period of time

5. **Miscellaneous**

- travel
- overheads
- telephone

Intangible costs

Intangible costs include

- loss of customer good will
- staff distress
- supplier confusion when processes change

Intangible benefits

Although the costs of intangible benefits cannot be easily calculated, it is very important to identify them. Often intangible benefits may make the difference between a project being funded or not being funded.

Some intangible benefits are

- improved work practices and employee morale
- customer access to account details over the telephone
- up to date product information on the web
- increased loyalty of customers by offering competitions and prizes

To determine whether a proposed system is cost-effective three techniques are often used. These involve formulas to calculate the time value of money. This is based on the assumption that a dollar today is worth more than a

dollar next year, because if you invest a dollar today you would have more than a dollar next year.

Technical Feasibility

It checks whether the existing computer system supports the candidate system or not or up to what extent it supports. It basically centres around Hardware, Software etc. For e.g. Current Computer is operating at 77 % capacity and running another application can Overload the system so need new system.

Behavioral Feasibility

An estimate should be made of how strong a reaction the user staff is likely to have towards the development of a computerized system. It is common knowledge that computer installations have something to do with Turnover, Transfers and changes in employee Job Status.

Steps in Feasibility Analysis

1. Form a project team and appoint a project leader

2. Prepare system flowcharts

3. Enumerate potential candidate systems

4. Describe and identify characteristics of candidate systems

5. Determine and evaluate performance and cost effectiveness of each candidate system

6. Weight system performance and cost data

7. Select the best candidate system

8. Prepare and report final project directive to Management

Joint application design (JAD)

Joint application design (JAD) is a process used in the prototyping life cycle area of the Dynamic Systems Development Method (DSDM) to collect business requirements while developing new information systems for a company. "The JAD process also includes approaches for enhancing user participation, expediting development, and improving the quality of specifications." It consists of a workshop where "knowledge workers and IT specialists meet, sometimes for several days, to define and review the business requirements for the system." The attendees include high level management officials who will ensure the product provides the needed reports and information at the end. This acts as "a management process which allows Corporate Information Services (IS) departments to work more effectively with users in a shorter time frame".

Through JAD workshops the knowledge workers and IT specialists are able to resolve any difficulties or differences between the two parties regarding the new information system. The workshop follows a detailed agenda in order to guarantee that all uncertainties between parties are covered and to help prevent any miscommunications. Miscommunications can carry far more serious repercussions if not addressed until later on in the process. (See below for Key Participants and Key Steps to an Effective JAD). In the end, this process will result in a new information system that is feasible and appealing to both the designers and end users.

"Although the JAD design is widely acclaimed, little is actually known about its effectiveness in practice." According to Journal of Systems and Software, a field study was done at three organizations using JAD practices to determine how JAD influenced system development outcomes. The results of the study suggest that organizations realized modest improvement in systems development outcomes by using the JAD method. JAD use was most effective in small, clearly focused projects and less effective in large complex projects.

Advantages

JAD decreases time and costs associated with requirements elicitation process. During 2-4 weeks information not only is collected, but requirements, agreed upon by various system users, are identified. Experience with JAD allows companies to customize their systems analysis process into even more dynamic ones like Double Helix, a methodology for mission-critical work.

- JAD sessions help bring experts together giving them a chance to share their views, understand views of others, and develop the sense of project ownership.
- The methods of JAD implementation are well-known, as it is "the first accelerated design technique available on the market and probably best known", and can easily be applied by any organization.
- Easy integration of CASE tools into JAD workshops improves session productivity and provides systems analysts with discussed and ready to use models.

Role and Tasks of System Analyst

The primary objective of any system analyst is to identify the need of the organization by acquiring information by various means and methods. Information acquired by the analyst can be either computer based or manual. Collection of information is the vital step as indirectly all the major decisions taken in the organizations are influenced. The system analyst has to coordinate with the system users, computer programmers, manager and number of people who are related with the use of system.

Following are the tasks performed by the system analyst:

- **Defining Requirement:** The basic step for any system analyst is to understand the requirements of the users. This is achieved by various fact finding techniques like interviewing, observation, questionnaire etc. The information should be collected in such a way that it will be useful to develop such a system which can provide additional features to the users apart from the desired.
- **Prioritizing Requirements**: Number of users use the system in the organization. Each one has a different requirement and retrieves different information. Due to certain limitations in computing capacity it may not be possible to satisfy the needs of all the users. Even if the computer capacity is good enough is it necessary to take some tasks and update the tasks as per the changing requirements. Hence it is important to create list of priorities according to users requirements. The best way to overcome the above limitations is to have a common formal or informal discussion with the users of the system. This helps the system analyst to arrive at a better conclusion.
- **Gathering Facts, data and opinions of Users**: After determining the necessary needs and collecting useful information the analyst starts the development of the system with active cooperation from the users of the system. Time to time, the users update the analyst with the necessary information for developing the system. The analyst while developing the system continuously consults the users and acquires their views and opinions.
- **Evaluation and Analysis:** As the analyst maintains continuous he constantly changes and modifies the system to make it better and more user friendly for the users.
- **Solving Problems**: The analyst must provide alternate solutions to the management and should a in dept study of the system to avoid future problems. The analyst should provide with some flexible alternatives to the

management which will help the manager to pick the system which provides the best solution.

- **Drawing Specifications**: The analyst must draw certain specifications which will be useful for the manager. The analyst should lay the specification which can be easily understood by the manager and they should be purely non-technical. The specifications must be in detailed and in well presented form.

ATTRIBUTES OF A SYSTEM ANALYST

1. Communication Skills – System analyst should have the ability to articulate and speak and knack of working with all levels of managerial positions of the organization.

2. Understanding – System analyst should have the ability to identify problems and assess their solution, grasping of company goals and objectives, show sensitivity to the impact of the system on people at work and understanding their problems,

3. Teaching & selling ideas – System analyst should have the skill to educate other people in the use of computer systems and selling ideas and promoting innovations in problem solving using computers.

The various ***TECHNICAL SKILLS*** that a system analyst should have are as follows –

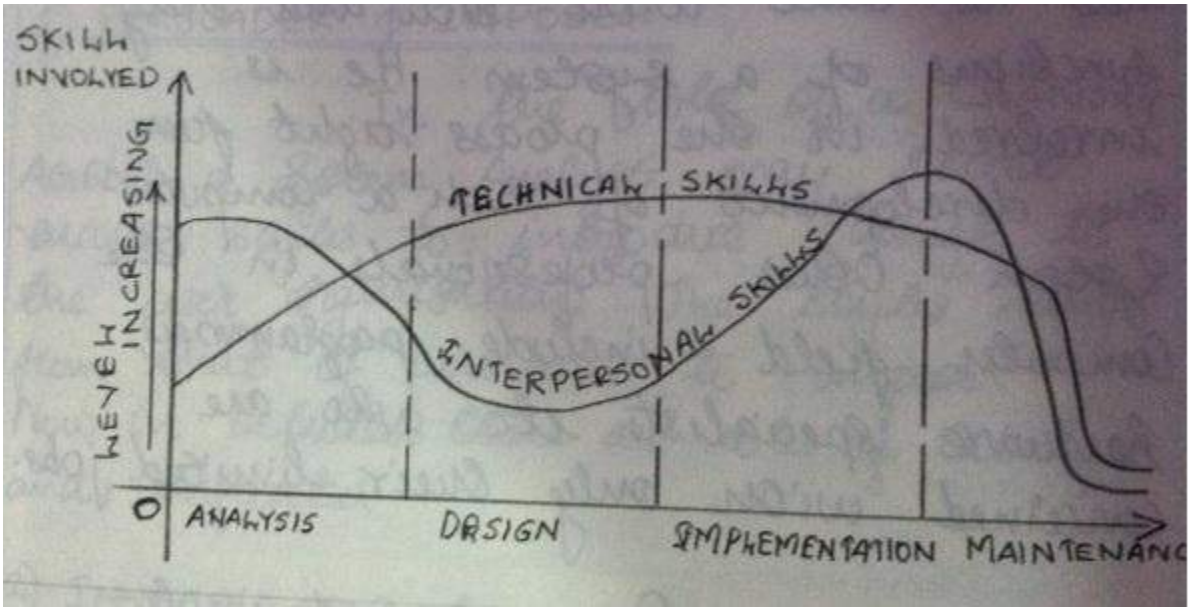
1. Creativity – The analyst should be creative to help the users to model ideas into real plans and developing candidate systems to match user requirements.

2. **Problem Solving & project management** – System analyst should have the skill of problem solving, developing alternative solutions, scheduling, overcome constraints, coordinating team efforts and managing costs and accounts.

3. Dynamic interface – System analyst should be a perfect blend of both technical and non-technical skills in functional specifications and general design. He should also have a questioning attitude and inquiring mind.

4. Thorough knowledge of computers – System analyst should have the knowledge of basics of computers and business functions.

A graph showing the involvement of interpersonal and technical skills during the system development phase for a good system analyst is shown below –



Now, in addition to these personal qualifications, the system analyst should have proper academic qualifications in system analysis and design or other computer oriented similar degrees.

Cost-Benefit Analysis

When new systems are proposed, the cost is a major consideration which impacts on the decision to accept or reject the proposed system.

The people funding the system want to know whether they will get a good return for the money they invest in the system. In large companies there is often competition to get access to financial resources for innovative projects. So, to win support for a project, estimates of costs and benefits must be calculated. This is called a cost-benefit analysis.

There are two main areas for costs: the development costs and the operating costs once a system is introduced. There are tangible and intangible costs and also tangible and intangible benefits. Tangibles can be easily calculated; intangibles cannot.

Tangible costs

Tangible costs include the following.

1. Personnel Costs

salaries of analysts, programmers, consultants, data entry personnel , etc

costs for trainers and employees' time
2. Costs for equipment for the new system

computer hardware
space/room
furniture

3. Supporting material

stationery
photocopying

4. Converting to new system

designing new processes and procedures
running new and old systems together for a period of time

5. Miscellaneous

travel
overheads
telephone

Intangible costs

Intangible costs include

loss of customer good will
staff distress
supplier confusion when processes change

Intangible benefits

Although the costs of intangible benefits cannot be easily calculated, it is very important to identify them. Often intangible benefits may make the difference between a project being funded or not being funded.

Some intangible benefits are

improved work practices and employee morale
customer access to account details over the telephone

up to date product information on the web
increased loyalty of customers by offering competitions and prizes

To determine whether a proposed system is cost-effective three techniques are often used. These involve formulas to calculate the time value of money. This is based on the assumption that a dollar today is worth more than a dollar next year, because if you invest a dollar today you would have more than a dollar next year.

Tools used by system analyst

Software analysis and design includes all activities, which help the transformation of requirement specification into implementation. Requirement specifications specify all functional and non-functional expectations from the software. These requirement specifications come in the shape of human readable and understandable documents, to which a computer has nothing to do.

Software analysis and design is the intermediate stage, which helps human-readable requirements to be transformed into actual code.

Let us discuss few analysis and design tools used by software designers:

Data Flow Diagram(DFD)

Data flow diagram is graphical representation of flow of data in an information system. It is capable of depicting incoming data flow, outgoing data flow and stored data. The DFD does not mention anything about how data flows through the system.There is a prominent difference between DFD and Flowchart. The flowchart depicts flow of control in program modules. DFDs depict flow of data in the system at various levels. DFD does not contain any control or branch elements.

Types of DFD

Data Flow Diagrams are either Logical or Physical.

Logical DFD - This type of DFD concentrates on the system process, and flow of data in the system.For example in a Banking software system, how data is moved between different entities.

Physical DFD - This type of DFD shows how the data flow is actually implemented in the system. It is more specific and close to the implementation.

DFD Components

DFD can represent Source, destination, storage and flow of data using the following set of components -

Entities - Entities are source and destination of information data. Entities are represented by a rectangles with their respective names.

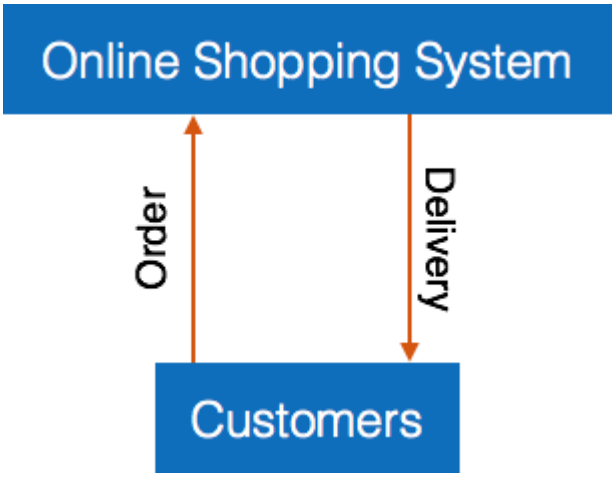
Process - Activities and action taken on the data are represented by Circle or Round-edged rectangles.

Data Storage - There are two variants of data storage - it can either be represented as a rectangle with absence of both smaller sides or as an open-sided rectangle with only one side missing.

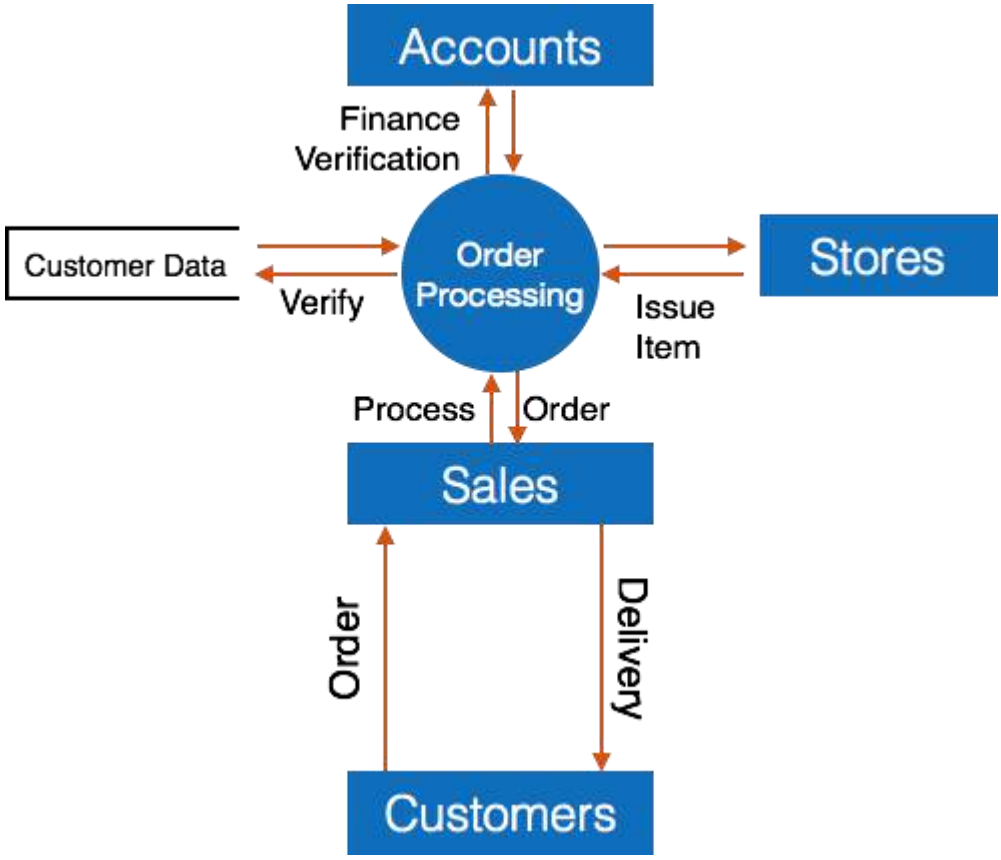
Data Flow - Movement of data is shown by pointed arrows. Data movement is shown from the base of arrow as its source towards head of the arrow as destination.

Levels of DFD

Level 0 - Highest abstraction level DFD is known as Level 0 DFD, which depicts the entire information system as one diagram concealing all the underlying details. Level 0 DFDs are also known as context level DFDs.



Level 1 - The Level 0 DFD is broken down into more specific, Level 1 DFD. Level 1 DFD depicts basic modules in the system and flow of data among various modules. Level 1 DFD also mentions basic processes and sources of information.



Level 2 - At this level, DFD shows how data flows inside the modules mentioned in Level 1.

Higher level DFDs can be transformed into more specific lower level DFDs with deeper level of understanding unless the desired level of specification is achieved.

Structure Charts

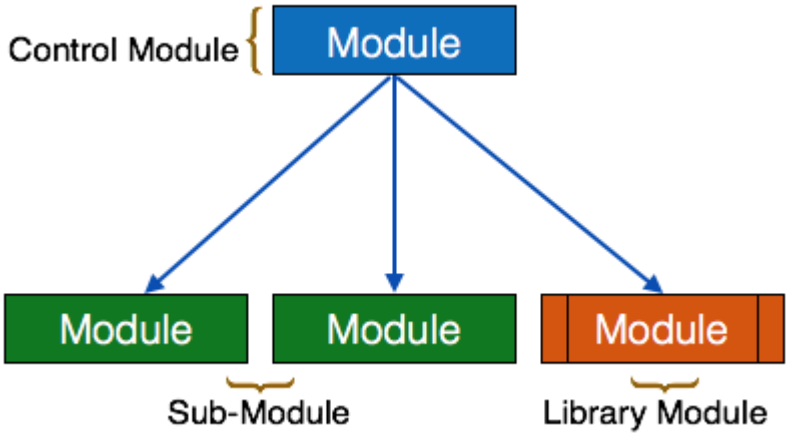
Structure chart is a chart derived from Data Flow Diagram. It represents the system in more detail than DFD. It breaks down the entire system into lowest functional modules, describes functions and sub-functions of each module of the system to a greater detail than DFD.

Structure chart represents hierarchical structure of modules. At each layer

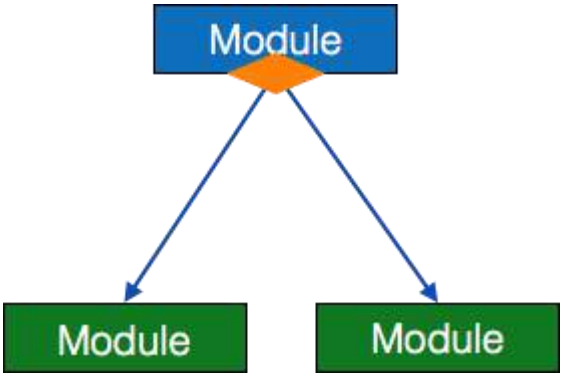
a specific task is performed.

Here are the symbols used in construction of structure charts -

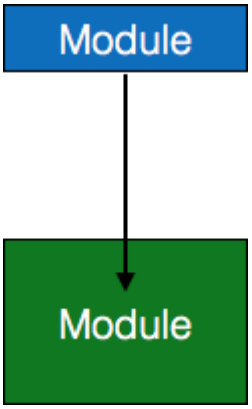
Module - It represents process or subroutine or task. A control module branches to more than one sub-module. Library Modules are re-usable and invokable from any module.



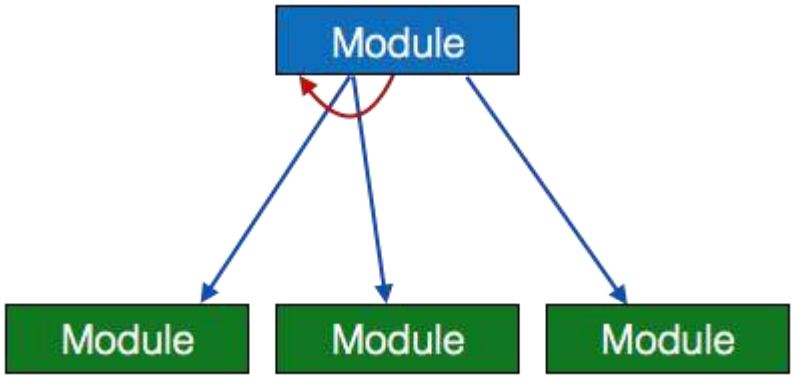
Condition - It is represented by small diamond at the base of module. It depicts that control module can select any of sub-routine based on some condition.



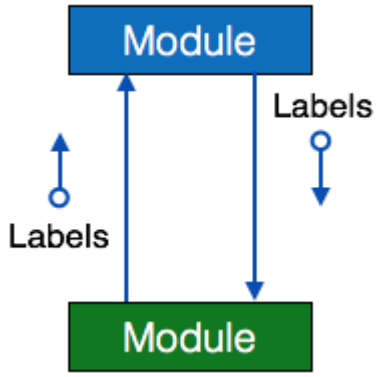
Jump - An arrow is shown pointing inside the module to depict that the control will jump in the middle of the sub-module.



Loop - A curved arrow represents loop in the module. All sub-modules covered by loop repeat execution of module.

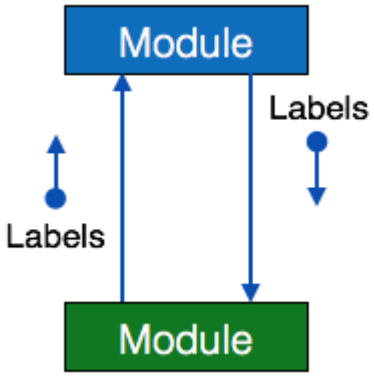


Data flow - A directed arrow with empty circle at the end represents data



flow.

Control flow - A directed arrow with filled circle at the end represents control

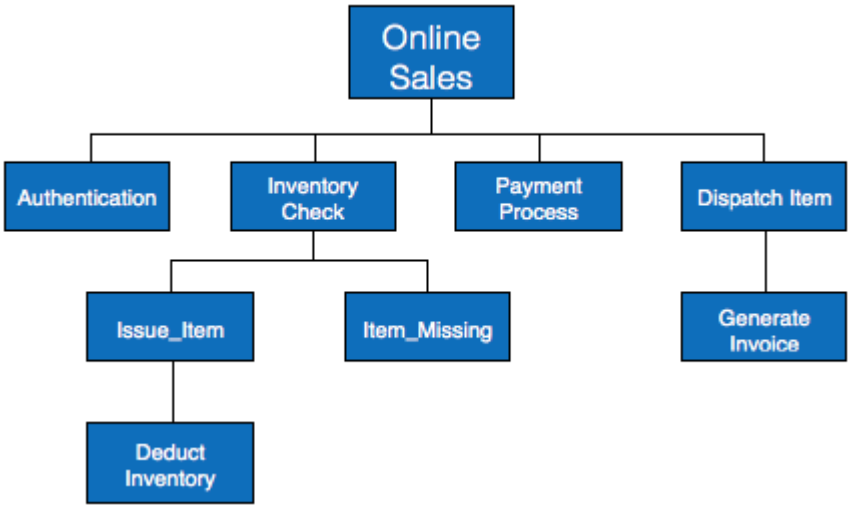


HIPO Diagram

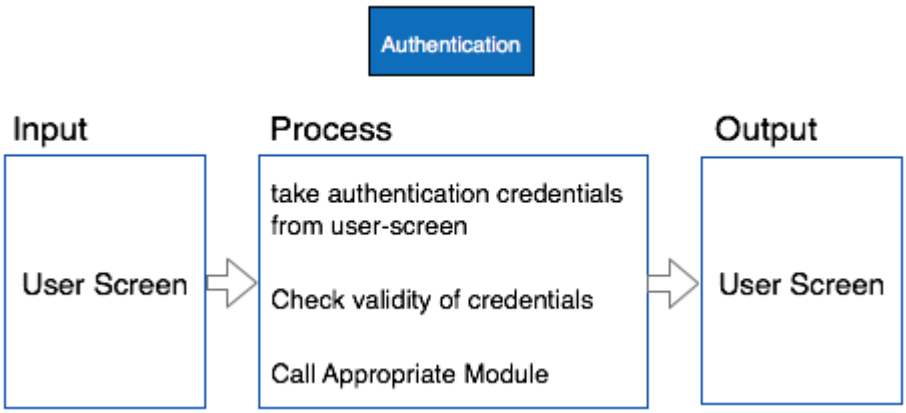
HIPO (Hierarchical Input Process Output) diagram is a combination of two organized method to analyze the system and provide the means of documentation. HIPO model was developed by IBM in year 1970.

HIPO diagram represents the hierarchy of modules in the software system. Analyst uses HIPO diagram in order to obtain high-level view of system functions. It decomposes functions into sub-functions in a hierarchical manner. It depicts the functions performed by system.

HIPO diagrams are good for documentation purpose. Their graphical representation makes it easier for designers and managers to get the pictorial idea of the system structure.



In contrast to IPO (Input Process Output) diagram, which depicts the flow of control and data in a module, HIPO does not provide any information about data flow or control flow.



Example

Both parts of HIPO diagram, Hierarchical presentation and IPO Chart are used for structure design of software program as well as documentation of the same.

Structured English

Most programmers are unaware of the large picture of software so they only rely on what their managers tell them to do. It is the responsibility of higher software management to provide accurate information to the programmers to develop accurate yet fast code.

Other forms of methods, which use graphs or diagrams, may are sometimes interpreted differently by different people.

Hence, analysts and designers of the software come up with tools such as Structured English. It is nothing but the description of what is required to code and how to code it. Structured English helps the programmer to write error-free code.

Other form of methods, which use graphs or diagrams, may are sometimes interpreted differently by different people. Here, both Structured English and Pseudo-Code tries to mitigate that understanding gap.

Structured English is the It uses plain English words in structured programming paradigm. It is not the ultimate code but a kind of description what is required to code and how to code it. The following are some tokens of structured programming.

```
IF-THEN-ELSE,

DO-WHILE-UNTIL
```

Analyst uses the same variable and data name, which are stored in Data Dictionary, making it much simpler to write and understand the code.

Example

We take the same example of Customer Authentication in the online shopping environment. This procedure to authenticate customer can be written in Structured English as:

```
EnterCustomer_Name

SEEK Customer_NameinCustomer_Name_DB file

IF Customer_Name found THEN

Call procedure USER_PASSWORD_AUTHENTICATE()

ELSE

   PRINT error message

Call procedure NEW_CUSTOMER_REQUEST()

ENDIF
```

The code written in Structured English is more like day-to-day spoken English. It cannot be implemented directly as a code of software.

Structured English is independent of programming language.

Pseudo-Code

Pseudo code is written more close to programming language. It may be considered as augmented programming language, full of comments and descriptions.

Pseudo code avoids variable declaration but they are written using some actual programming language's constructs, like C, Fortran, Pascal etc.

Pseudo code contains more programming details than Structured English. It provides a method to perform the task, as if a computer is executing the code.

Example

Program to print Fibonacci up to n numbers.

```
voidfunctionFibonacci
Get value of n;
Set value of a to 1;
Set value of b to 1;
Initialize I to 0
for(i=0;i< n;i++)
{
if a greater than b
{
Increase b by a;
Print b;
```

```
}
elseif b greater than a
{
    increase a by b;
print a;
}
}
```

Decision Tables

A Decision table represents conditions and the respective actions to be taken to address them, in a structured tabular format.

It is a powerful tool to debug and prevent errors. It helps group similar information into a single table and then by combining tables it delivers easy and convenient decision-making.

Creating Decision Table

To create the decision table, the developer must follow basic four steps:

Identify all possible conditions to be addressed

Determine actions for all identified conditions

Create Maximum possible rules

Define action for each rule

Decision Tables should be verified by end-users and can lately be simplified by eliminating duplicate rules and actions.

Example

Let us take a simple example of day-to-day problem with our Internet connectivity. We begin by identifying all problems that can arise while starting the internet and their respective possible solutions.

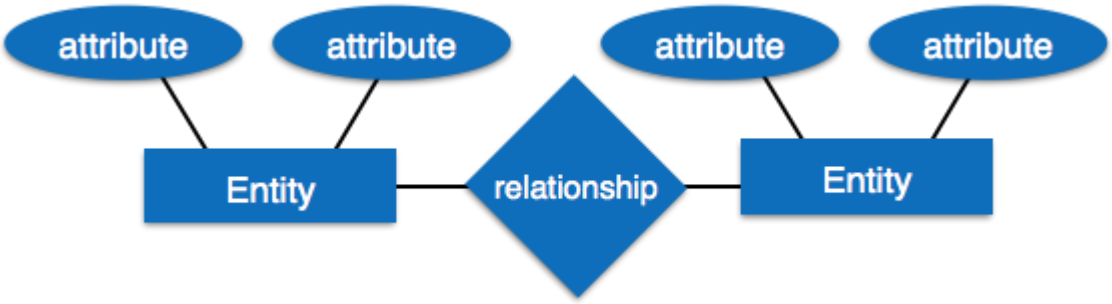
We list all possible problems under column conditions and the prospective actions under column Actions.

	Conditions/Actions	Rules							
Conditions	Shows Connected	N	N	N	N	Y	Y	Y	Y
	Ping is Working	N	N	Y	Y	N	N	Y	Y
	Opens Website	Y	N	Y	N	Y	N	Y	N
Actions	Check network cable	X							
	Check internet router	X				X	X	X	
	Restart Web Browser							X	
	Contact Service provider		X	X	X	X	X	X	
	Do no action								

Table: Decision Table – In-house Internet Troubleshooting.

Entity-Relationship Model

Entity-Relationship model is a type of database model based on the notion of real world entities and relationship among them. We can map real world scenario onto ER database model. ER Model creates a set of entities with their attributes, a set of constraints and relation among them. ER Model is best used for the conceptual design of database. ER Model can be represented as follows :



Entity - An entity in ER Model is a real world being, which has some properties called attributes. Every attribute is defined by its corresponding set of values, called domain.

For example, Consider a school database. Here, a student is an entity. Student has various attributes like name, id, age and class etc.

Relationship - The logical association among entities is called relationship. Relationships are mapped with entities in various ways. Mapping cardinalities define the number of associations between two entities.

Mapping cardinalities:

one to one

one to many

many to one

many to many

Data Dictionary

Data dictionary is the centralized collection of information about data. It stores meaning and origin of data, its relationship with other data, data format for usage etc. Data dictionary has rigorous definitions of all names in order to facilitate user and software designers.

Data dictionary is often referenced as meta-data (data about data) repository. It is created along with DFD (Data Flow Diagram) model of software program and is expected to be updated whenever DFD is changed or updated.

Requirement of Data Dictionary

The data is referenced via data dictionary while designing and implementing software. Data dictionary removes any chances of ambiguity. It helps keeping work of programmers and designers synchronized while using same object reference everywhere in the program.

Data dictionary provides a way of documentation for the complete database system in one place. Validation of DFD is carried out using data dictionary.

Contents

Data dictionary should contain information about the following

Data Flow

Data Structure

Data Elements

Data Stores

Data Processing

Data Flow is described by means of DFDs as studied earlier and represented in algebraic form as described.

=	Composed of
{}	Repetition
()	Optional
+	And
[/]	Or

Example

Address = House No + (Street / Area) + City + State

Course ID = Course Number + Course Name + Course Level + Course Grades

Data Elements

Data elements consist of Name and descriptions of Data and Control Items, Internal or External data stores etc. with the following details:

Primary Name

Secondary Name (Alias)

Use-case (How and where to use)

Content Description (Notation etc.)

Supplementary Information (preset values, constraints etc.)

Data Store

It stores the information from where the data enters into the system and exists out of the system. The Data Store may include -

Files

Internal to software.

External to software but on the same machine.

External to software and system, located on different machine.

Tables

Naming convention

Indexing property

Data Processing

There are two types of Data Processing:

Logical: As user sees it

Physical: As software sees it

PROTOTYPING AND DOCUMENTATION

Software prototyping is the activity of creating prototypes of software applications, i.e., incomplete versions of the software program being developed. It is an activity that can occur in software development and is comparable to prototyping as known from other fields, such as mechanical engineering or manufacturing. A prototype typically simulates only a few aspects and may be completely different from, the final product.

Prototyping has several benefits such as the software designer and implementer can get valuable feedback from the users early in the project. The client and the contractor can compare if the software made matches the software specification, according to which the software program is built. It also allows the software engineer some insight into the accuracy of initial project estimates and whether the deadlines and milestones proposed can be successfully met. The degree of completeness and the techniques used in the prototyping have been in development and debate since its proposal in the early 1970s.

Advantages of prototyping

There are many advantages to using prototyping in software development – some tangible, some abstract.

- Reduced time and costs: Prototyping can improve the quality of requirements and specifications provided to developers. Because changes cost exponentially more to implement as they are detected later in development, the early determination of what the user really wants can result in faster and less expensive software.
- Improved and increased user involvement: Prototyping requires user involvement and allows them to see and interact with a prototype allowing them to provide better and more complete feedback and specifications. The presence of the prototype being examined by the user prevents many misunderstandings and miscommunications that occur when each side believe the other understands what they said. Since users know the problem domain better than anyone on the development team does, increased interaction can result in a final product that has greater tangible and intangible quality. The final product is more likely to satisfy the user's desire for look, feel and performance.

Disadvantages of prototyping

- Insufficient analysis: The focus on a limited prototype can distract developers from properly analyzing the complete project. This can lead to overlooking better solutions, preparation of incomplete specifications or the conversion of limited prototypes into poorly

engineered final projects that are hard to maintain. Further, since a prototype is limited in functionality it may not scale well if the prototype is used as the basis of a final deliverable, which may not be noticed if developers are too focused on building a prototype as a model.

- User confusion of prototype and finished system: Users can begin to think that a prototype, intended to be thrown away, is actually a final system that merely needs to be finished or polished. (They are, for example, often unaware of the effort needed to add error-checking and security features which a prototype may not have.) This can lead them to expect the prototype to accurately model the performance of the final system when this is not the intent of the developers. Users can also become attached to features that were included in a prototype for consideration and then removed from the specification for a final system. If users are able to require all proposed features be included in the final system this can lead to conflict.
- Developer misunderstanding of user objectives: Developers may assume that users share their objectives (e.g. to deliver core functionality on time and within budget), without understanding wider commercial issues. For example, user representatives attending Enterprise software events may have seen demonstrations of "transaction auditing" (where changes are logged and displayed in a difference grid view) without being told that this feature demands additional coding and often requires more hardware to handle extra database accesses. Users might believe they can demand auditing on every field, whereas developers might think this is feature creep because they have made assumptions about the extent of user requirements. If the developer has committed delivery before the user requirements were reviewed, developers are between a rock and a hard place, particularly if user management derives some advantage from their failure to implement requirements.
- Developer attachment to prototype: Developers can also become

attached to prototypes they have spent a great deal of effort producing; this can lead to problems like attempting to convert a limited prototype into a final system when it does not have an appropriate underlying architecture.

- Excessive development time of the prototype: A key property to prototyping is the fact that it is supposed to be done quickly. If the developers lose sight of this fact, they very well may try to develop a prototype that is too complex. When the prototype is thrown away the precisely developed requirements that it provides may not yield a sufficient increase in productivity to make up for the time spent developing the prototype. Users can become stuck in debates over details of the prototype, holding up the development team and delaying the final product.
- Expense of implementing prototyping: the start up costs for building a development team focused on prototyping may be high. Many companies have development methodologies in place, and changing them can mean retraining, retooling, or both. Many companies tend to just jump into the prototyping without bothering to retrain their workers as much as they should.

A common problem with adopting prototyping technology is high expectations for productivity with insufficient effort behind the learning curve. In addition to training for the use of a prototyping technique, there is an often overlooked need for developing corporate and project specific underlying structure to support the technology. When this underlying structure is omitted, lower productivity can often result.

Documentation

Documentation is one of the system which is used to communicate, instruct and record the information for any reference or operational purpose. They are very useful for representing the formal flow of the present system. With the help of documentation it is very easy to track the flow of the system's progress and they working of the system can be explained very

easily.

It helps to provide the clear description of the work done so far. It is essential that the documents prepared must be updated on regular basis this will help to trace the progress of work easily. With appropriate and good documentation it is very easy to understand the how aspects of the system will work for the company where the system is to installed. It is also help to understand the type of data which will be inputted in the system and how the output can be produced.

After the system is installed, and if in case the system is not working properly it will be very easy for the administrator to understand the flow of data in the system with documentation which will help him/ her to correct the flaws and get the system working in no time.

Uses of Documentation

- It facilitates effective communication regarding the system between the technical and the non technical users.
- It is very useful in training new users. With a Good documentation new users can easily get acquainted with the flow of the systems.
- Documentation also helps the users to solve problems like trouble shooting even a non technical user can fix the problems.
- It plays a significant role in evaluation process.
- It not only helps to exercise a better control over the internal working of the firm, but it also external as well especially during audit.
- Documentations can help the manager to take better financial decisions of the organization.

UNIT III

CASE Tools

CASE stands for" Computer Aided Software Engineering".It means development and maintenance of software projects with help of various automated software tools.

CASE tools are set of software application programs, which are used to automate SDLC activities. CASE tools are used by software project managers, analysts and engineers to develop software system.

There are number of CASE tools available to simplify various stages of Software Development Life Cycle such as Analysis tools, Design tools, Project management tools, Database Management tools, Documentation tools are to name a few.

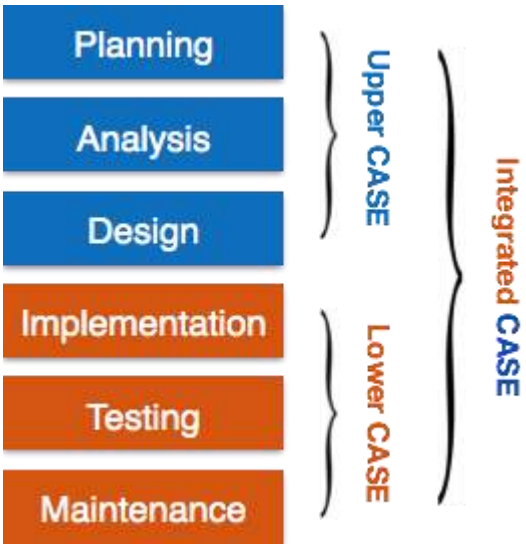
Use of CASE tools accelerates the development of project to produce desired result and helps to uncover flaws before moving ahead with next stage in software development.

Components of CASE Tools

CASE tools can be broadly divided into the following parts based on their use at a particular SDLC stage:

Central Repository - CASE tools require a central repository, which can serve as a source of common, integrated and consistent information. Central repository is a central place of storage where product specifications, requirement documents, related reports and diagrams, other useful information regarding management are stored. Central repository

also serves as data dictionary.



Upper Case Tools - Upper CASE tools are used in planning, analysis and design stages of SDLC.

Lower Case Tools - Lower CASE tools are used in implementation, testing and maintenance.

Integrated Case Tools - Integrated CASE tools are helpful in all the stages of SDLC, from Requirement gathering to Testing and documentation.

CASE tools can be grouped together if they have similar functionality, process activities and capability of getting integrated with other tools.

Case Tools Types

Now we briefly go through various CASE tools

Diagram tools

These tools are used to represent system components, data and control flow among various software components and system structure in a graphical form. For example, Flow Chart Maker tool for creating state-of-

the-art flowcharts.

Process Modeling Tools

Process modeling is method to create software process model, which is used to develop the software. Process modeling tools help the managers to choose a process model or modify it as per the requirement of software product. For example, EPF Composer

Project Management Tools

These tools are used for project planning, cost and effort estimation, project scheduling and resource planning. Managers have to strictly comply project execution with every mentioned step in software project management. Project management tools help in storing and sharing project information in real-time throughout the organization. For example, Creative Pro Office, Trac Project, Base camp.

Documentation Tools

Documentation in a software project starts prior to the software process, goes throughout all phases of SDLC and after the completion of the project.

Documentation tools generate documents for technical users and end users. Technical users are mostly in-house professionals of the development team who refer to system manual, reference manual, training manual, installation manuals etc. The end user documents describe the functioning and how-to of the system such as user manual.

Analysis Tools

These tools help to gather requirements, automatically check for any inconsistency, inaccuracy in the diagrams, data redundancies or erroneous omissions.

Design Tools

These tools help software designers to design the block structure of the

software, which may further be broken down in smaller modules using refinement techniques. These tools provides detailing of each module and interconnections among modules. For example, Animated Software Design

Configuration Management Tools

An instance of software is released under one version. Configuration Management tools deal with –

Version and revision management

Baseline configuration management

Change control management

Change Control Tools

These tools are considered as a part of configuration management tools. They deal with changes made to the software after its baseline is fixed or when the software is first released. CASE tools automate change tracking, file management, code management and more. It also helps in enforcing change policy of the organization.

Programming Tools

These tools consist of programming environments like IDE (Integrated Development Environment), in-built modules library and simulation tools. These tools provide comprehensive aid in building software product and include features for simulation and testing.

Prototyping Tools

Software prototype is simulated version of the intended software product. Prototype provides initial look and feel of the product and simulates few aspect of actual product.

Prototyping CASE tools essentially come with graphical libraries. They can create hardware independent user interfaces and design. These tools help us to build rapid prototypes based on existing information. In addition,

they provide simulation of software prototype.

Web Development Tools

These tools assist in designing web pages with all allied elements like forms, text, script, graphic and so on. Web tools also provide live preview of what is being developed and how will it look after completion.

Quality Assurance Tools

Quality assurance in a software organization is monitoring the engineering process and methods adopted to develop the software product in order to ensure conformance of quality as per organization standards. QA tools consist of configuration and change control tools and software testing tools.

Maintenance Tools

Software maintenance includes modifications in the software product after it is delivered. Automatic logging and error reporting techniques, automatic error ticket generation and root cause Analysis are few CASE tools, which help software organization in maintenance phase of SDLC.

Object Modeling

Object modeling develops the static structure of the software system in terms of objects. It identifies the objects, the classes into which the objects can be grouped into and the relationships between the objects. It also identifies the main attributes and operations that characterize each class.

The process of object modeling can be visualized in the following steps:

Identify objects and group into classes

Identify the relationships among classes

Create user object model diagram

Define user object attributes

Define the operations that should be performed on the classes

Dynamic Modeling

After the static behavior of the system is analyzed, its behavior with respect to time and external changes needs to be examined. This is the purpose of dynamic modeling.

Dynamic Modeling can be defined as "a way of describing how an individual object responds to events, either internal events triggered by other objects, or external events triggered by the outside world".

The process of dynamic modeling can be visualized in the following steps:

Identify states of each object

Identify events and analyze the applicability of actions

Construct dynamic model diagram, comprising of state transition diagrams

Express each state in terms of object attributes

Validate the state–transition diagrams drawn

Functional Modeling

Functional Modeling is the final component of object-oriented analysis. The functional model shows the processes that are performed within an object and how the data changes as it moves between methods. It specifies the meaning of the operations of object modeling and the actions of dynamic modeling. The functional model corresponds to the data flow diagram of traditional structured analysis.

The process of functional modeling can be visualized in the following steps:

- Identify all the inputs and outputs
- Construct data flow diagrams showing functional dependencies
- State the purpose of each function
- Identify constraints
- Specify optimization criteria

Advantages/Disadvantages of Object Oriented Analysis

Advantages	Disadvantages
Focuses on data rather than the procedures as in Structured Analysis.	Functionality is restricted within objects. This may pose a problem for systems which are intrinsically procedural or computational in nature.
The principles of encapsulation and data hiding help the developer to develop systems that cannot be tampered by other parts of the system.	It cannot identify which objects would generate an optimal system design.
The principles of encapsulation and data hiding help the developer to develop systems that cannot be tampered by other	The object-oriented models do not easily show the communications between the objects in the system.

parts of the system.	
It allows effective management of software complexity by the virtue of modularity.	All the interfaces between the objects cannot be represented in a single diagram.

Object Oriented Design for implementation

After the analysis phase, the conceptual model is developed further into an object-oriented model using object-oriented design (OOD). In OOD, the technology-independent concepts in the analysis model are mapped onto implementing classes, constraints are identified, and interfaces are designed, resulting in a model for the solution domain. In a nutshell, a detailed description is constructed specifying how the system is to be built on concrete technologies

The stages for object–oriented design can be identified as:

Definition of the context of the system

Designing system architecture

Identification of the objects in the system

Construction of design models

Specification of object interfaces

Object-Oriented System Design

System Design

Object-oriented system design involves defining the context of a system followed by designing the architecture of the system.

Context : The context of a system has a static and a dynamic part. The static context of the system is designed using a simple block diagram of the whole system which is expanded into a hierarchy of subsystems. The subsystem model is represented by UML packages. The dynamic context describes how the system interacts with its environment. It is modelled using use case diagrams.

System Architecture : The system architecture is designed on the basis of the context of the system in accordance with the principles of architectural design as well as domain knowledge. Typically, a system is partitioned into layers and each layer is decomposed to form the subsystems.

Object-Oriented Decomposition

Decomposition means dividing a large complex system into a hierarchy of smaller components with lesser complexities, on the principles of divide–and–conquer. Each major component of the system is called a subsystem. Object-oriented decomposition identifies individual autonomous objects in a system and the communication among these objects.

The advantages of decomposition are:

The individual components are of lesser complexity, and so more understandable and manageable.

It enables division of workforce having specialized skills.

It allows subsystems to be replaced or modified without affecting other

subsystems.

Identifying Concurrency

Concurrency allows more than one objects to receive events at the same time and more than one activity to be executed simultaneously. Concurrency is identified and represented in the dynamic model.

To enable concurrency, each concurrent element is assigned a separate thread of control. If the concurrency is at object level, then two concurrent objects are assigned two different threads of control. If two operations of a single object are concurrent in nature, then that object is split among different threads.

Concurrency is associated with the problems of data integrity, deadlock, and starvation. So a clear strategy needs to be made whenever concurrency is required. Besides, concurrency requires to be identified at the design stage itself, and cannot be left for implementation stage.

Identifying Patterns

While designing applications, some commonly accepted solutions are adopted for some categories of problems. These are the patterns of design. A pattern can be defined as a documented set of building blocks that can be used in certain types of application development problems.

Some commonly used design patterns are:

- Model view separation pattern
- Observer pattern
- Model view controller pattern
- Publish subscribe pattern
- Proxy pattern
- Controlling Events

During system design, the events that may occur in the objects of the system need to be identified and appropriately dealt with.

An event is a specification of a significant occurrence that has a location in time and space.

There are four types of events that can be modelled, namely:

Signal Event : A named object thrown by one object and caught by another object.

Call Event : A synchronous event representing dispatch of an operation.

Time Event : An event representing passage of time.

Change Event : An event representing change in state.

Object Design

After the hierarchy of subsystems has been developed, the objects in the system are identified and their details are designed. Here, the designer details out the strategy chosen during the system design. The emphasis shifts from application domain concepts toward computer concepts. The objects identified during analysis are etched out for implementation with an aim to minimize execution time, memory consumption, and overall cost.

Object design includes the following phases:

Object identification

Object representation, i.e., construction of design models

Classification of operations

Algorithm design

Design of relationships

Implementation of control for external interactions

Package classes and associations into modules

Object Identification:

The first step of object design is object identification. The objects identified in the object–oriented analysis phases are grouped into classes and refined so that they are suitable for actual implementation.

The functions of this stage are:

- Identifying and refining the classes in each subsystem or package
- Defining the links and associations between the classes
- Designing the hierarchical associations among the classes, i.e., the generalization/specialization and inheritances
- Designing aggregations

Object Representation

Once the classes are identified, they need to be represented using object modeling techniques. This stage essentially involves constructing UML diagrams.

There are two types of design models that need to be produced:

Static Models : To describe the static structure of a system using class diagrams and object diagrams.

Dynamic Models : To describe the dynamic structure of a system and show the interaction between classes using interaction diagrams and state–chart diagrams.

Classification of Operations

In this step, the operation to be performed on objects are defined by

combining the three models developed in the OOA phase, namely, object model, dynamic model, and functional model. An operation specifies what is to be done and not how it should be done.

The following tasks are performed regarding operations:

The state transition diagram of each object in the system is developed.

Operations are defined for the events received by the objects.

Cases in which one event triggers other events in same or different objects are identified.

The sub–operations within the actions are identified.

The main actions are expanded to data flow diagrams.

Algorithm Design

The operations in the objects are defined using algorithms. An algorithm is a stepwise procedure that solves the problem laid down in an operation. Algorithms focus on how it is to be done.

There may be more than one algorithm corresponding to a given operation. Once the alternative algorithms are identified, the optimal algorithm is selected for the given problem domain. The metrics for choosing the optimal algorithm are:

Computational Complexity : Complexity determines the efficiency of an algorithm in terms of computation time and memory requirements.

Flexibility : Flexibility determines whether the chosen algorithm can be implemented suitably, without loss of appropriateness in various environments.

Understandability : This determines whether the chosen algorithm is easy

to understand and implement.

Design of Relationships

The strategy to implement the relationships needs to be chalked out during the object design phase. The main relationships that are addressed comprise of associations, aggregations, and inheritances.

The designer should do the following regarding associations:

Identify whether an association is unidirectional or bidirectional.

Analyze the path of associations and update them if necessary.

Implement the associations as a distinct object, in case of many–to-many relationships; or as a link to other object in case of one–to-one or one–to-many relationships.

Regarding inheritances, the designer should do the following:

Adjust the classes and their associations.

Identify abstract classes.

Make provisions so that behaviors are shared when needed.

Implementation of Control

The object designer may incorporate refinements in the strategy of the state–chart model. In system design, a basic strategy for realizing the dynamic model is made. During object design, this strategy is aptly embellished for appropriate implementation.

The approaches for implementation of the dynamic model are:

Represent State as a Location within a Program : This is the traditional procedure-driven approach whereby the location of control defines the program state. A finite state machine can be implemented as a program. A transition forms an input statement, the main control path forms the

sequence of instructions, the branches form the conditions, and the backward paths form the loops or iterations.

Control as Concurrent Tasks : In this approach, an object is implemented as a task in the programming language or the operating system. Here, an event is implemented as an inter-task call. It preserves inherent concurrency of real objects.

Design Optimization

The analysis model captures the logical information about the system, while the design model adds details to support efficient information access. Before a design is implemented, it should be optimized so as to make the implementation more efficient. The aim of optimization is to minimize the cost in terms of time, space, and other metrics.

However, design optimization should not be excess, as ease of implementation, maintainability, and extensibility are also important concerns. It is often seen that a perfectly optimized design is more efficient but less readable and reusable. So the designer must strike a balance between the two.

The various things that may be done for design optimization are:

Add redundant associations

Omit non-usable associations

Optimization of algorithms

Save derived attributes to avoid re-computation of complex expressions

UNIT IV

Introduction to software projects

A project is well-defined task, which is a collection of several operations done in order to achieve a goal (for example, software development and delivery). A Project can be characterized as:

Every project may have a unique and distinct goal.

Project is not routine activity or day-to-day operations.

Project comes with a start time and end time.

Project ends when its goal is achieved hence it is a temporary phase in the lifetime of an organization.

Project needs adequate resources in terms of time, manpower, finance, material and knowledge-bank.

Software Project

A Software Project is the complete procedure of software development from requirement gathering to testing and maintenance, carried out according to the execution methodologies, in a specified period of time to achieve intended software product.

Need of software project management

Software is said to be an intangible product. Software development is a kind of all new stream in world business and there's very little experience in building software products. Most software products are tailor made to fit client's requirements. The most important is that the underlying technology changes and advances so frequently and rapidly that experience of one product may not be applied to the other one. All such business and

environmental constraints bring risk in software development hence it is essential to manage software projects efficiently.

The image above shows triple constraints for software projects. It is an essential part of software organization to deliver quality product, keeping the cost within client's budget constrain and deliver the project as per scheduled. There are several factors, both internal and external, which may impact this triple constrain triangle. Any of three factor can severely impact the other two.

Therefore, software project management is essential to incorporate user requirements along with budget and time constraints.

Software Project Manager

A software project manager is a person who undertakes the responsibility of executing the software project. Software project manager is thoroughly aware of all the phases of SDLC that the software would go through. Project manager may never directly involve in producing the end product but he controls and manages the activities involved in production.

A project manager closely monitors the development process, prepares and

executes various plans, arranges necessary and adequate resources, maintains communication among all team members in order to address issues of cost, budget, resources, time, quality and customer satisfaction.

Let us see few responsibilities that a project manager shoulders -

Managing People

- Act as project leader
- Liaison with stakeholders
- Managing human resources
- Setting up reporting hierarchy etc.

Managing Project

- Defining and setting up project scope
- Managing project management activities
- Monitoring progress and performance
- Risk analysis at every phase
- Take necessary step to avoid or come out of problems
- Act as project spokesperson

Software Management Activities

Software project management comprises of a number of activities, which contains planning of project, deciding scope of software product, estimation of cost in various terms, scheduling of tasks and events, and resource management. Project management activities may include:

Project Planning

Scope Management

Project Estimation

Project Planning

Software project planning is task, which is performed before the production of software actually starts. It is there for the software production but involves no concrete activity that has any direction connection with software production; rather it is a set of multiple processes, which facilitates software production. Project planning may include the following:

Scope Management

It defines the scope of project; this includes all the activities, process need to be done in order to make a deliverable software product. Scope management is essential because it creates boundaries of the project by clearly defining what would be done in the project and what would not be done. This makes project to contain limited and quantifiable tasks, which can easily be documented and in turn avoids cost and time overrun.

During Project Scope management, it is necessary to -

- Define the scope
- Decide its verification and control
- Divide the project into various smaller parts for ease of management.
- Verify the scope
- Control the scope by incorporating changes to the scope

Project Estimation

For an effective management accurate estimation of various measures is a must. With correct estimation managers can manage and control the project more efficiently and effectively.

Project estimation may involve the following:

Software size estimation

Software size may be estimated either in terms of KLOC (Kilo Line of Code)

or by calculating number of function points in the software. Lines of code depend upon coding practices and Function points vary according to the user or software requirement.

Effort estimation:

The managers estimate efforts in terms of personnel requirement and man-hour required to produce the software. For effort estimation software size should be known. This can either be derived by managers' experience, organization's historical data or software size can be converted into efforts by using some standard formulae.

Time estimation

Once size and efforts are estimated, the time required to produce the software can be estimated. Efforts required is segregated into sub categories as per the requirement specifications and interdependency of various components of software. Software tasks are divided into smaller tasks, activities or events by Work Breakthrough Structure (WBS). The tasks are scheduled on day-to-day basis or in calendar months.

The sum of time required to complete all tasks in hours or days is the total time invested to complete the project.

Cost estimation

- This might be considered as the most difficult of all because it depends on more elements than any of the previous ones. For estimating project cost, it is required to consider -
    - Size of software
    - Software quality
    - Hardware
    - Additional software or tools, licenses etc.
    - Skilled personnel with task-specific skills

- Travel involved
- Communication
- Training and support

Project Estimation Techniques

We discussed various parameters involving project estimation such as size, effort, time and cost.

Project manager can estimate the listed factors using two broadly recognized techniques –

Decomposition Technique

This technique assumes the software as a product of various compositions.

There are two main models -

Line of Code Estimation is done on behalf of number of line of codes in the software product.

Function Points Estimation is done on behalf of number of function points in the software product.

Empirical Estimation Technique

This technique uses empirically derived formulae to make estimation.These formulae are based on LOC or FPs.

Putnam Model

This model is made by Lawrence H. Putnam, which is based on Norden's frequency distribution (Rayleigh curve). Putnam model maps time and efforts required with software size.

COCOMO

COCOMO stands for Constructive Cost MOdel, developed by Barry W.

Boehm. It divides the software product into three categories of software: organic, semi-detached and embedded.

Project Scheduling

Project Scheduling in a project refers to roadmap of all activities to be done with specified order and within time slot allotted to each activity. Project managers tend to define various tasks, and project milestones and them arrange them keeping various factors in mind. They look for tasks lie in critical path in the schedule, which are necessary to complete in specific manner (because of task interdependency) and strictly within the time allocated. Arrangement of tasks which lies out of critical path are less likely to impact over all schedule of the project.

For scheduling a project, it is necessary to -

Break down the project tasks into smaller, manageable form

Find out various tasks and correlate them

Estimate time frame required for each task

Divide time into work-units

Assign adequate number of work-units for each task

Calculate total time required for the project from start to finish

Resource management

All elements used to develop a software product may be assumed as resource for that project. This may include human resource, productive tools and software libraries.

The resources are available in limited quantity and stay in the organization as a pool of assets. The shortage of resources hampers the development of project and it can lag behind the schedule. Allocating extra resources increases development cost in the end. It is therefore necessary to estimate and allocate adequate resources for the project.

Resource management includes -

Defining proper organization project by creating a project team and allocating responsibilities to each team member

Determining resources required at a particular stage and their availability

Manage Resources by generating resource request when they are required and de-allocating them when they are no more needed.

Project Risk Management

Risk management involves all activities pertaining to identification, analyzing and making provision for predictable and non-predictable risks in the project. Risk may include the following:

- Experienced staff leaving the project and new staff coming in.
- Change in organizational management.
- Requirement change or misinterpreting requirement.
- Under-estimation of required time and resources.
- Technological changes, environmental changes, business competition.
- Risk Management Process

There are following activities involved in risk management process:

Identification - Make note of all possible risks, which may occur in the project.

Categorize - Categorize known risks into high, medium and low risk intensity as per their possible impact on the project.

Manage - Analyze the probability of occurrence of risks at various phases. Make plan to avoid or face risks. Attempt to minimize their side-effects.

Monitor - Closely monitor the potential risks and their early symptoms. Also monitor the effects of steps taken to mitigate or avoid them.

Project Execution & Monitoring

In this phase, the tasks described in project plans are executed according to their schedules.

Execution needs monitoring in order to check whether everything is going according to the plan. Monitoring is observing to check the probability of risk and taking measures to address the risk or report the status of various tasks.

These measures include -

Activity Monitoring - All activities scheduled within some task can be monitored on day-to-day basis. When all activities in a task are completed, it is considered as complete.

Status Reports - The reports contain status of activities and tasks completed within a given time frame, generally a week. Status can be marked as finished, pending or work-in-progress etc.

Milestones Checklist - Every project is divided into multiple phases where major tasks are performed (milestones) based on the phases of SDLC. This milestone checklist is prepared once every few weeks and reports the status of milestones.

Project Communication Management

Effective communication plays vital role in the success of a project. It bridges gaps between client and the organization, among the team members as well as other stake holders in the project such as hardware suppliers.

Communication can be oral or written. Communication management process may have the following steps:

Planning - This step includes the identifications of all the stakeholders in the project and the mode of communication among them. It also considers if any additional communication facilities are required.

Sharing - After determining various aspects of planning, manager focuses on sharing correct information with the correct person on correct time. This keeps every one involved the project up to date with project progress and its status.

Feedback - Project managers use various measures and feedback mechanism and create status and performance reports. This mechanism ensures that input from various stakeholders is coming to the project manager as their feedback.

Closure - At the end of each major event, end of a phase of SDLC or end of the project itself, administrative closure is formally announced to update every stakeholder by sending email, by distributing a hardcopy of document or by other mean of effective communication.

After closure, the team moves to next phase or project.

Configuration Management

Configuration management is a process of tracking and controlling the changes in software in terms of the requirements, design, functions and development of the product.

IEEE defines it as "the process of identifying and defining the items in the system, controlling the change of these items throughout their life cycle, recording and reporting the status of items and change requests, and verifying the completeness and correctness of items".

Generally, once the SRS is finalized there is less chance of requirement of changes from user. If they occur, the changes are addressed only with

prior approval of higher management, as there is a possibility of cost and time overrun.

Baseline

A phase of SDLC is assumed over if it baselined, i.e. baseline is a measurement that defines completeness of a phase. A phase is baselined when all activities pertaining to it are finished and well documented. If it was not the final phase, its output would be used in next immediate phase.

Configuration management is a discipline of organization administration, which takes care of occurrence of any change (process, requirement, technological, strategical etc.) after a phase is baselined. CM keeps check on any changes done in software.

Change Control

Change control is function of configuration management, which ensures that all changes made to software system are consistent and made as per organizational rules and regulations.

A change in the configuration of product goes through following steps -

Identification - A change request arrives from either internal or external source. When change request is identified formally, it is properly documented.

Validation - Validity of the change request is checked and its handling procedure is confirmed.

Analysis - The impact of change request is analyzed in terms of schedule, cost and required efforts. Overall impact of the prospective change on system is analyzed.

Control - If the prospective change either impacts too many entities in the system or it is unavoidable, it is mandatory to take approval of high

authorities before change is incorporated into the system. It is decided if the change is worth incorporation or not. If it is not, change request is refused formally.

Execution - If the previous phase determines to execute the change request, this phase take appropriate actions to execute the change, does a thorough revision if necessary.

Close request - The change is verified for correct implementation and merging with the rest of the system. This newly incorporated change in the software is documented properly and the request is formally is closed.

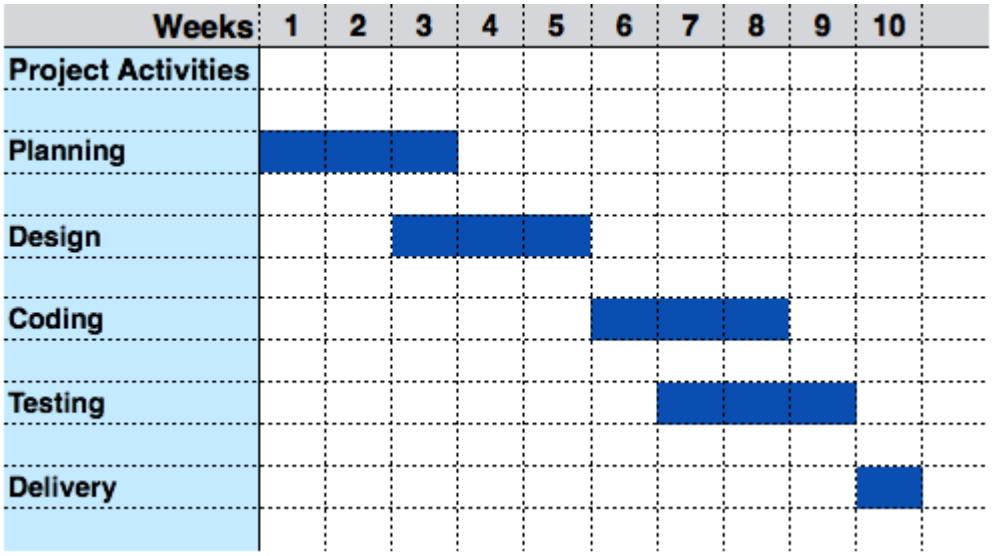
Project Management Tools

The risk and uncertainty rises multifold with respect to the size of the project, even when the project is developed according to set methodologies.

There are tools available, which aid for effective project management. A few are described -

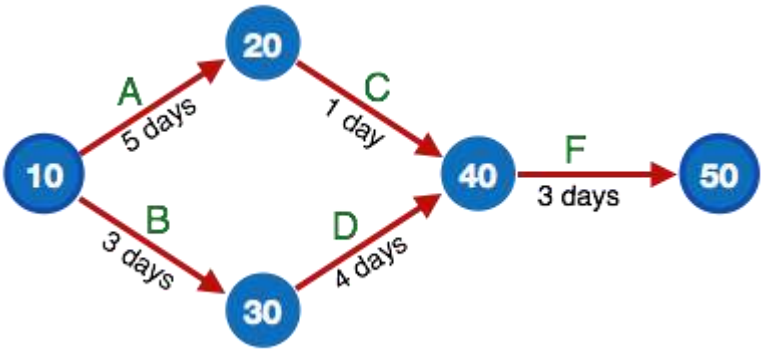
Gantt Chart

Gantt charts was devised by Henry Gantt (1917). It represents project schedule with respect to time periods. It is a horizontal bar chart with bars representing activities and time scheduled for the project activities.



PERT Chart

PERT (Program Evaluation & Review Technique) chart is a tool that depicts project as network diagram. It is capable of graphically representing main events of project in both parallel and consecutive way. Events, which occur one after another, show dependency of the later event over the previous one.

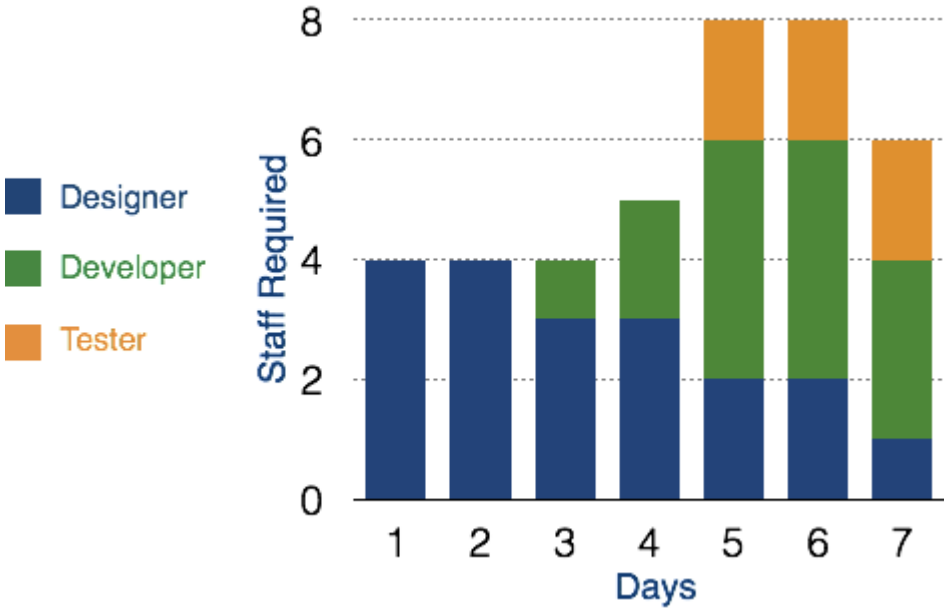


Events are shown as numbered nodes. They are connected by labeled arrows depicting sequence of tasks in the project.

Resource Histogram

This is a graphical tool that contains bar or chart representing number of resources (usually skilled staff) required over time for a project event (or phase). Resource Histogram is an effective tool for staff planning and coordination.

Staff	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7
Designer	4	4	3	3	2	2	1
Developer	0	0	1	2	4	4	3
Tester	0	0	0	0	2	2	2
Total	4	4	4	5	8	8	6



Critical Path Analysis

This tools is useful in recognizing interdependent tasks in the project. It also helps to find out the shortest path or critical path to complete the

project successfully. Like PERT diagram, each event is allotted a specific time frame. This tool shows dependency of event assuming an event can proceed to next only if the previous one is completed.

The events are arranged according to their earliest possible start time. Path between start and end node is critical path which cannot be further reduced and all events require to be executed in same order.

Measurement of software quality and productivity

Software quality is the degree of conformance to explicit or implicit requirements and expectations.

Definition by IEEE The degree to which a system, component, or process meets specified requirements.

Software quality measurement is about quantifying to what extent a system or software possesses desirable characteristics. This can be performed through qualitative or quantitative means or a mix of both. In both cases, for each desirable characteristic, there are a set of measurable attributes the existence of which in a piece of software or system tend to be correlated and associated with this characteristic. For example, an attribute associated with portability is the number of target-dependent statements in a program. More precisely, using the Quality Function Deployment approach, these measurable attributes are the "how's" that need to be enforced to enable the "what's" in the Software Quality definition above.
The measurement of critical application characteristics involves measuring

structural attributes of the application's architecture, coding, and in-line documentation, as displayed in the picture above. Thus, each characteristic is affected by attributes at numerous levels of abstraction in the application and all of which must be included calculating the characteristic's measure if it is to be a valuable predictor of quality outcomes that affect the business. The layered approach to calculating characteristic measures displayed in the figure above was first proposed by Boehm and his colleagues at TRW (Boehm, 1978 and is the approach taken in the ISO 9126 and 25000 series standards. These attributes can be measured from the parsed results of a static analysis of the application source code. Even dynamic characteristics of applications such as reliability and performance efficiency have their causal roots in the static structure of the application. Structural quality analysis and measurement is performed through the analysis of the source code, the architecture, software framework, database schema in relationship to principles and standards that together define the conceptual and logical architecture of a system. This is distinct from the basic, local, component-level code analysis typically performed by development tools which are mostly concerned with implementation considerations and are crucial during debugging and testing activities.

Reliability

The root causes of poor reliability are found in a combination of non-compliance with good architectural and coding practices. This non-compliance can be detected by measuring the static quality attributes of an application. Assessing the static attributes underlying an application's reliability provides an estimate of the level of business risk and the likelihood of potential application failures and defects the application will experience when placed in operation.

Assessing reliability requires checks of at least the following software engineering best practices and technical attributes:

- Application Architecture Practices
- Coding Practices
- Complexity of algorithms

- Complexity of programming practices
- Compliance with Object-Oriented and Structured Programming best practices (when applicable)
- Component or pattern re-use ratio
- Dirty programming
- Error & Exception handling (for all layers - GUI, Logic & Data)
- Multi-layer design compliance
- Resource bounds management
- Software avoids patterns that will lead to unexpected behaviors
- Software manages data integrity and consistency
- Transaction complexity level

Depending on the application architecture and the third-party components used (such as external libraries or frameworks), custom checks should be defined along the lines drawn by the above list of best practices to ensure a better assessment of the reliability of the delivered software.

Efficiency

As with Reliability, the causes of performance inefficiency are often found in violations of good architectural and coding practice which can be detected by measuring the static quality attributes of an application. These static attributes predict potential operational performance bottlenecks and future scalability problems, especially for applications requiring high execution speed for handling complex algorithms or huge volumes of data.

Assessing performance efficiency requires checking at least the following software engineering best practices and technical attributes:

- Application Architecture Practices
- Appropriate interactions with expensive and/or remote resources
- Data access performance and data management
- Memory, network and disk space management
- Coding Practices

- Compliance with Object-Oriented and Structured Programming best practices (as appropriate)
- Compliance with SQL programming best practices

Security

Most security vulnerabilities result from poor coding and architectural practices such as SQL injection or cross-site scripting. These are well documented in lists maintained by CWE,[32] and the SEI/Computer Emergency Center (CERT) at Carnegie Mellon University.

Assessing security requires at least checking the following software engineering best practices and technical attributes:

Application Architecture Practices

- Multi-layer design compliance
- Security best practices (Input Validation, SQL Injection, Cross-Site Scripting, etc.
- Programming Practices (code level)
- Error & Exception handling
- Security best practices (system functions access, access control to programs)

Maintainability

Maintainability includes concepts of modularity, understandability, changeability, testability, reusability, and transferability from one development team to another. These do not take the form of critical issues at the code level. Rather, poor maintainability is typically the result of thousands of minor violations with best practices in documentation, complexity avoidance strategy, and basic programming practices that make

the difference between clean and easy-to-read code vs. unorganized and difficult-to-read code

Assessing maintainability requires checking the following software engineering best practices and technical attributes:

- Application Architecture Practices
- Architecture, Programs and Code documentation embedded in source code
- Code readability
- Complexity level of transactions
- Complexity of algorithms
- Complexity of programming practices
- Compliance with Object-Oriented and Structured Programming best practices (when applicable)
- Component or pattern re-use ratio
- Controlled level of dynamic coding
- Coupling ratio
- Dirty programming
- Documentation
- Hardware, OS, middleware, software components and database independence
- Multi-layer design compliance
- Portability
- Programming Practices (code level)
- Reduced duplicate code and functions
- Source code file organization cleanliness

Maintainability is closely related to Ward Cunningham's concept of technical debt, which is an expression of the costs resulting of a lack of maintainability. Reasons for why maintainability is low can be classified as reckless vs. prudent and deliberate vs. inadvertent,[35] and often have

their origin in developers' inability, lack of time and goals, their carelessness and discrepancies in the creation cost of and benefits from documentation and, in particular, maintainable source code.[36]

Size

Measuring software size requires that the whole source code be correctly gathered, including database structure scripts, data manipulation source code, component headers, configuration files etc. There are essentially two types of software sizes to be measured, the technical size (footprint) and the functional size:

There are several software technical sizing methods that have been widely described. The most common technical sizing method is number of Lines of Code (#LOC) per technology, number of files, functions, classes, tables, etc., from which backfiring Function Points can be computed;

The most common for measuring functional size is function point analysis. Function point analysis measures the size of the software deliverable from a user's perspective. Function point sizing is done based on user requirements and provides an accurate representation of both size for the developer/estimator and value (functionality to be delivered) and reflects the business functionality being delivered to the customer. The method includes the identification and weighting of user recognizable inputs, outputs and data stores. The size value is then available for use in conjunction with numerous measures to quantify and to evaluate software delivery and performance (development cost per function point; delivered defects per function point; function points per staff month.).

The function point analysis sizing standard is supported by the International Function Point Users Group (IFPUG). It can be applied early in the software development life-cycle and it is not dependent on lines of code like the somewhat inaccurate Backfiring method. The method is technology agnostic and can be used for comparative analysis across organizations and across industries.

Since the inception of Function Point Analysis, several variations have evolved and the family of functional sizing techniques has broadened to include such sizing measures as COSMIC, NESMA, Use Case Points, FP Lite, Early and Quick FPs, and most recently Story Points. However, Function Points has a history of statistical accuracy, and has been used as a common unit of work measurement in numerous application development management (ADM) or outsourcing engagements, serving as the "currency" by which services are delivered and performance is measured.

One common limitation to the Function Point methodology is that it is a manual process and therefore it can be labor-intensive and costly in large scale initiatives such as application development or outsourcing engagements. This negative aspect of applying the methodology may be what motivated industry IT leaders to form the Consortium for IT Software Quality focused on introducing a computable metrics standard for automating the measuring of software size while the IFPUG keep promoting a manual approach as most of its activity rely on FP counters certifications.

CISQ announced the availability of its first metric standard, Automated Function Points,to the CISQ membership, in CISQ Technical. These recommendations have been developed in OMG's Request for Comment format and submitted to OMG's process for standardization.[citation needed]

Identifying critical programming errors

Critical Programming Errors are specific architectural and/or coding bad practices that result in the highest, immediate or long term, business disruption risk.

These are quite often technology-related and depend heavily on the context, business objectives and risks. Some may consider respect for naming conventions while others – those preparing the ground for a knowledge transfer for example – will consider it as absolutely critical.

Critical Programming Errors can also be classified per CISQ Characteristics.

Basic example below:

Reliability

Avoid software patterns that will lead to unexpected behavior (Uninitialized variable, null pointers, etc.)

Methods, procedures and functions doing Insert, Update, Delete, Create Table or Select must include error management

Multi-thread functions should be made thread safe, for instance servlets or struts action classes must not have instance/non-final static fields

Efficiency

Ensure centralization of client requests (incoming and data) to reduce network traffic

Avoid SQL queries that don't use an index against large tables in a loop

Security

Avoid fields in servlet classes that are not final static

Avoid data access without including error management

Check control return codes and implement error handling mechanisms

Ensure input validation to avoid cross-site scripting flaws or SQL injections flaws

Maintainability

Deep inheritance trees and nesting should be avoided to improve comprehensibility

Modules should be loosely coupled (fanout, intermediaries,) to avoid propagation of modifications

Enforce homogeneous naming conventions

SOFTWARE PRODUCTIVITY

Software productivity is a deceptively simple concept, but a matter of some debate. Although its earliest measurement was in lines of code per man-hours worked, a better definition is the ratio between the functional

value of software produced to the labor and expense of producing it. There are several ways to measure software productivity, including Function Point Analysis, Cost Component Modeling, Cyclomatic Complexity, and program performance metrics that take into account the costs of running and maintaining the software.

Using these tools, the software development process can be managed and productivity enhanced by reusing code to leverage existing programs, minimizing rework through reliability initiatives, and adopting sound development practices and standards. However, even when sound practices are adhered to, software productivity may not increase because of circumstances outside the control of the development team. This includes rapidly changing technologies and the fixed-cost behavior of significant parts of the software development process.

The concept of software productivity is not a theoretical abstract. It is a critical part of the software engineering process. Understanding software productivity becomes important in systems analysis when you consider that good systems analysis enhances software productivity and software productivity is a success measure of systems analysis.

In order to define software productivity, we must first establish a definition of software. At its most fundamental level, software is a computer program comprised of lines of code. However, lines of code, in and of themselves, are not the primary deliverables of a software project and customers often do not know how many lines of code are in the software they are buying.

A broader definition of software encompasses not only the computer program, but also the related procedures and documentation associated with the program. This often includes documentation of requirements, specifications, software design, and end-user procedures. The complete set of documentation provides a more tangible deliverable of a software project than does the program itself.

However, even though program code and documentation are the primary outputs of software production, they are not of direct interest to

the software consumer. Software is bought based on what it can do, not on how it was coded or documented. This means that the economic value of goods and services consumed is not measured in the same units as the natural units of production. Subsequently, a different measure of software needs to be used in order to get a meaningful definition of software productivity. This measure needs to reflect the utility value of the software, to wit the function that the software is intended to perform.

Basing our measurements on the utility value of software, we can revise our original assumption and define software productivity as the ratio between the functional value of software produced to the labor and expense of producing it. This definition allows us to measure productivity based on the value of results to the software consumer, which is more realistic than basing results on lines of code.

How is Software Productivity Measured?

With a working definition of software productivity established, we are next faced with the question of what to measure. Unlike lines of code and pages of documentation that are easy to count, program functionality does not have a natural unit of measure and is thus harder to quantify. Software metrics that we can use as quantifiable measure of various characteristics of a software system or software development process need to be established. These metrics need to capture both the effort required to produce the software and the functionality provided to the software consumer.

There are various methods by which software productivity is measured, but whichever method is employed the goal should be uniform: to give software managers and professionals a set of useful, tangible data points for sizing, estimating, managing, and controlling software projects with rigor and precision .Some of the more common methods of measuring software productivity are Function Point Analysis, Constructive Cost Modeling, and Cyclomatic Complexity.

Constructive Cost Model (CoCoMo)

Another approach to measuring the effort required to produce software is the Constructive Cost Model approach developed by Barry Boehm to predict the work effort and development time required for the management and technical staffs in a software development project. The CoCoMo model is designed to provide predictions at three levels (basic, intermediate, or detailed) depending on the information known about the product being developed.

The basic level is used to obtain a quick-and-dirty estimate of the overall project's effort and development time early on, either right after analysis ends or as soon as a reasonable estimate of the lines of source code is available. The intermediate level is used later on in the system design phase to refine and to update the estimates calculated using basic CoCoMo. Detailed CoCoMo is used to further refine estimates to the module, subsystem, and systems levels, but the increased complexity of the calculations often does not significantly improve estimate accuracy. Detailed CoCoMo is not commonly used in general practice.

CoCoMo estimates factor in the number of source code lines (in thousands), project type (organic, embedded, or semi-detached) and a series of 15 cost drivers to determine the effort and development time of the project. Organic projects are ones involving small, highly experienced teams familiar with the technology and embedded projects involve large teams with very little experience with the technology, with semi-detached projects falling somewhere in between. The cost factors relate to product attributes, computer attributes, personnel attributes, and project attributes.

It is important to note that CoCoMo is size oriented because its estimates are based on the number of lines of source code delivered. As such, this method does not cover the full software life cycle. It is best used as an estimator of the effort required for system design (after system requirements have been analyzed) through integration and testing.

Factors Improve Software Productivity

In keeping with our definition of software productivity as the ratio between the functional value of software produced to the labor and expense of producing it, our next step is to determine ways to improve software productivity. Whereas computer hardware has a reputation for performance and cost improvements unprecedented in the history of technology, software productivity has lagged behind (Shaw 5). This is due in part to shortcomings in software productivity measurements as noted above, and in part to the fact that faster, more powerful computers can provide performance gains without software productivity gains.

Even so, software development companies are constantly looking for ways to increase both developer productivity and code quality. The first step toward improving either is to establish productivity and quality metrics and benchmarks as described above. After benchmarks have been established, areas for improvement can be determined and action plans put in place to improve performance. This could be as simple as rearranging developer work areas. Studies have shown that the design of the developers' workspace can have a large effect on their productivity.

Leveraging, or code reuse, is perhaps the best technique for increasing software productivity. Leveraging is reusing or porting application software across multiple business sites.

Sometimes the highest real productivity is the result of finding how to reuse programs already written – possibly for a quite different looking purpose – or in finding how to solve problems with past existing programs, revising them as subprograms.

Minimizing rework is another excellent way to increase software productivity. This means catching mistakes and problems as early in the software life cycle as possible. A mistake caught in one phase can reduce the work required in subsequent phases by a factor of three. That is, a good requirements analysis can reduce the design job by a factor of three, a good design can reduce the implementation job by a factor of three, and

a good implementation can reduce the maintenance job by a factor of three .Software reliability can be enhanced by applying various different analysis methods, which include software verification and testing.

Of course, a substantial part of software productivity involves the skill and personal behavior of the software developers themselves. It has been noted that there is a 10 to 1 difference in productivity among programmers, brought about in large part by their differing levels of problem-solving skills and programming knowledge. Individual developer productivity can be enhanced by providing software developers with adequate training in, and insisting they adhere to, disciplined processes, such as structured analysis, top-down design, modular design, design reviews, code inspections, and quality assurance programs.

System testing

System testing of software or hardware is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. System testing falls within the scope of black-box testing, and as such, should require no knowledge of the inner design of the code or logic.

As a rule, system testing takes, as its input, all of the "integrated" software components that have passed integration testing and also the software system itself integrated with any applicable hardware system(s). The purpose of integration testing is to detect any inconsistencies between the software units that are integrated together (called assemblages) or between any of the assemblages and the hardware. System testing is a more limited type of testing; it seeks to detect defects both within the "inter-assemblages" and also within the system as a whole.

Test Plan

Test planning, the most important activity to ensure that there is initially a list of tasks and milestones in a baseline plan to track the progress of the project. It also defines the size of the test effort.

It is the main document often called as master test plan or a project test plan and usually developed during the early phase of the project.

Test Plan Identifiers:

S.No.	Parameter	Description
1.	Test plan identifier	Unique identifying reference.
2.	Introduction	A brief introduction about the project and to the document.
3.	Test items	A test item is a software item that is the application under test.

4.	Features to be tested	A feature that needs to tested on the test ware.
5.	Features not to be tested	Identify the features and the reasons for not including as part of testing.
6.	Approach	Details about the overall approach to testing.
7.	Item pass/fail criteria	Documented whether a software item has passed or failed its test.
8.	Test deliverables	The deliverables that are delivered as part of the testing process, such as test plans, test specifications and test summary reports.
9.	Testing tasks	All tasks for planning and executing the testing.
10.	Environmental needs	Defining the environmental requirements such as hardware, software, OS, network configurations, tools required.
11.	Responsibilities	Lists the roles and responsibilities of the team members.

12.	Staffing and training needs	Captures the actual staffing requirements and any specific skills and training requirements.
13.	Schedule	States the important project delivery dates and key milestones.
14.	Risks and Mitigation	High-level project risks and assumptions and a mitigating plan for each identified risk.
15.	Approvals	Captures all approvers of the document, their titles and the sign off date.

Test Planning Activities:

To determine the scope and the risks that need to be tested and that are NOT to be tested.

Documenting Test Strategy.

Making sure that the testing activities have been included.

Deciding Entry and Exit criteria.

Evaluating the test estimate.

Planning when and how to test and deciding how the test results will be evaluated, and defining test exit criterion.

The Test artifacts delivered as part of test execution.

Defining the management information, including the metrics required and defect resolution and risk issues.

Ensuring that the test documentation generates repeatable test assets.

Types of testing

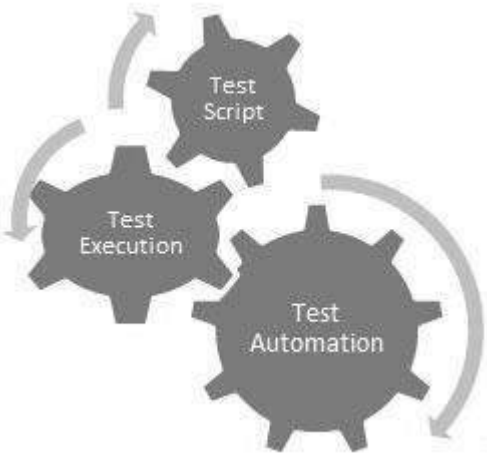
Manual Testing

Manual testing includes testing a software manually, i.e., without using any automated tool or any script. In this type, the tester takes over the role of an end-user and tests the software to identify any unexpected behavior or bug. There are different stages for manual testing such as unit testing, integration testing, system testing, and user acceptance testing.

Testers use test plans, test cases, or test scenarios to test a software to ensure the completeness of testing. Manual testing also includes exploratory testing, as testers explore the software to identify errors in it.

Automation Testing

Automation testing, which is also known as Test Automation, is when the tester writes scripts and uses another software to test the product. This process involves automation of a manual process. Automation Testing is used to re-run the test scenarios that were performed manually, quickly, and repeatedly.



Black-Box Testing

The technique of testing without having any knowledge of the interior workings of the application is called black-box testing. The tester is oblivious to the system architecture and does not have access to the source code. Typically, while performing a black-box test, a tester will interact with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon.

The following table lists the advantages and disadvantages of black-box testing.

Advantages	Disadvantages
Well suited and efficient for large code segments. Code access is not required. Clearly separates user's perspective from the developer's perspective through visibly defined roles. Large numbers of moderately skilled testers can test the application with no knowledge of implementation, programming language, or operating systems.	Limited coverage, since only a selected number of test scenarios is actually performed. Inefficient testing, due to the fact that the tester only has limited knowledge about an application. Blind coverage, since the tester cannot target specific code segments or error-prone areas. The test cases are difficult to design.

White-Box Testing

White-box testing is the detailed investigation of internal logic and structure of the code. White-box testing is also called glass testing or open-box testing. In order to perform white-box testing on an application, a tester needs to know the internal workings of the code.

The tester needs to have a look inside the source code and find out which unit/chunk of the code is behaving inappropriately.

The following table lists the advantages and disadvantages of white-box testing.

Advantages	Disadvantages
As the tester has knowledge of the source code, it becomes very easy to find out which type of data can help in testing the application effectively. It helps in optimizing the code. Extra lines of code can be removed which can bring in hidden defects. Due to the tester's knowledge about the code, maximum coverage is attained during test scenario writing.	Due to the fact that a skilled tester is needed to perform white-box testing, the costs are increased. Sometimes it is impossible to look into every nook and corner to find out hidden errors that may create problems, as many paths will go untested. It is difficult to maintain white-box testing, as it requires specialized tools like code analyzers and debugging tools.

Grey-Box Testing

Grey-box testing is a technique to test the application with having a limited knowledge of the internal workings of an application. In software testing, the phrase the more you know, the better carries a lot of weight while testing an application.

Mastering the domain of a system always gives the tester an edge over someone with limited domain knowledge. Unlike black-box testing, where the tester only tests the application's user interface; in grey-box testing, the tester has access to design documents and the database. Having this knowledge, a tester can prepare better test data and test scenarios while making a test plan.

Advantages	Disadvantages
Offers combined benefits of black-box and white-box testing wherever possible. Grey box testers don't rely on the source code; instead they rely on interface definition and functional specifications. Based on the limited information available, a grey-box tester can design excellent test scenarios especially around communication protocols and data type handling. The test is done from the point of view of the user and not the designer.	Since the access to source code is not available, the ability to go over the code and test coverage is limited. The tests can be redundant if the software designer has already run a test case. Testing every possible input stream is unrealistic because it would take an unreasonable amount of time; therefore, many program paths will go untested.

A Comparison of Testing Methods

The following table lists the points that differentiate black-box testing, grey-box testing, and white-box testing.

Black-Box Testing	Grey-Box Testing	White-Box Testing
The internal workings of an application need not be known.	The tester has limited knowledge of the internal workings of the	Tester has full knowledge of the internal workings of

	application.	the application.
Also known as closed-box testing, data-driven testing, or functional testing.	Also known as translucent testing, as the tester has limited knowledge of the insides of the application.	Also known as clear-box testing, structural testing, or code-based testing.
Performed by end-users and also by testers and developers.	Performed by end-users and also by testers and developers.	Normally done by testers and developers.
Testing is based on external expectations - Internal behavior of the application is unknown.	Testing is done on the basis of high-level database diagrams and data flow diagrams.	Internal workings are fully known and the tester can design test data accordingly.
It is exhaustive and the least time-consuming.	Partly time-consuming and exhaustive.	The most exhaustive and time-consuming type of testing.
Not suited for algorithm testing.	Not suited for algorithm testing.	Suited for algorithm testing.

This can only be done by trial-and-error method.	Data domains and internal boundaries can be tested, if known.	Data domains and internal boundaries can be better tested.

Unit Testing

This type of testing is performed by developers before the setup is handed over to the testing team to formally execute the test cases. Unit testing is performed by the respective developers on the individual units of source code assigned areas. The developers use test data that is different from the test data of the quality assurance team.

The goal of unit testing is to isolate each part of the program and show that individual parts are correct in terms of requirements and functionality.

Limitations of Unit Testing

Testing cannot catch each and every bug in an application. It is impossible to evaluate every execution path in every software application. The same is the case with unit testing.

There is a limit to the number of scenarios and test data that a developer can use to verify a source code. After having exhausted all the options, there is no choice but to stop unit testing and merge the code segment with other units.

Integration Testing

Integration testing is defined as the testing of combined parts of an application to determine if they function correctly. Integration testing can be done in two ways: Bottom-up integration testing and Top-down integration testing.

S.N.	Integration Testing Method
1	Bottom-up integration This testing begins with unit testing, followed by tests of progressively higher-level combinations of units called modules or builds.
2	Top-down integration In this testing, the highest-level modules are tested first and progressively, lower-level modules are tested thereafter.

In a comprehensive software development environment, bottom-up testing is usually done first, followed by top-down testing. The process concludes with multiple tests of the complete application, preferably in scenarios designed to mimic actual situations.

System Testing

System testing tests the system as a whole. Once all the components are integrated, the application as a whole is tested rigorously to see that it meets the specified Quality Standards. This type of testing is performed by a specialized testing team.

System testing is important because of the following reasons:

System testing is the first step in the Software Development Life Cycle, where the application is tested as a whole.

The application is tested thoroughly to verify that it meets the functional

and technical specifications.

The application is tested in an environment that is very close to the production environment where the application will be deployed.

System testing enables us to test, verify, and validate both the business requirements as well as the application architecture.

Regression Testing

Whenever a change in a software application is made, it is quite possible that other areas within the application have been affected by this change. Regression testing is performed to verify that a fixed bug hasn't resulted in another functionality or business rule violation. The intent of regression testing is to ensure that a change, such as a bug fix should not result in another fault being uncovered in the application.

Regression testing is important because of the following reasons:

- Minimize the gaps in testing when an application with changes made has to be tested.
- Testing the new changes to verify that the changes made did not affect any other area of the application.
- Mitigates risks when regression testing is performed on the application.
- Test coverage is increased without compromising timelines.
- Increase speed to market the product.

Acceptance Testing

This is arguably the most important type of testing, as it is conducted by the Quality Assurance Team who will gauge whether the application meets the intended specifications and satisfies the client’s requirement. The QA

team will have a set of pre-written scenarios and test cases that will be used to test the application.

More ideas will be shared about the application and more tests can be performed on it to gauge its accuracy and the reasons why the project was initiated. Acceptance tests are not only intended to point out simple spelling mistakes, cosmetic errors, or interface gaps, but also to point out any bugs in the application that will result in system crashes or major errors in the application.

By performing acceptance tests on an application, the testing team will deduce how the application will perform in production. There are also legal and contractual requirements for acceptance of the system.

Alpha Testing

This test is the first stage of testing and will be performed amongst the teams (developer and QA teams). Unit testing, integration testing and system testing when combined together is known as alpha testing. During this phase, the following aspects will be tested in the application:

- Spelling Mistakes
- Broken Links
- Cloudy Directions

The Application will be tested on machines with the lowest specification to test loading times and any latency problems.

Beta Testing

This test is performed after alpha testing has been successfully performed. In beta testing, a sample of the intended audience tests the application. Beta testing is also known as pre-release testing. Beta test versions of software are ideally distributed to a wide audience on the Web, partly to

give the program a "real-world" test and partly to provide a preview of the next release. In this phase, the audience will be testing the following:

Users will install, run the application and send their feedback to the project team.

Typographical errors, confusing application flow, and even crashes.

Getting the feedback, the project team can fix the problems before releasing the software to the actual users.

The more issues you fix that solve real user problems, the higher the quality of your application will be.

Having a higher-quality application when you release it to the general public will increase customer satisfaction.

Non-Functional Testing

This section is based upon testing an application from its non-functional attributes. Non-functional testing involves testing a software from the requirements which are nonfunctional in nature but important such as performance, security, user interface, etc.

Some of the important and commonly used non-functional testing types are discussed below.

Performance Testing

It is mostly used to identify any bottlenecks or performance issues rather than finding bugs in a software. There are different causes that contribute in lowering the performance of a software:

- Network delay
- Client-side processing
- Database transaction processing

- Load balancing between servers
- Data rendering

Performance testing is considered as one of the important and mandatory testing type in terms of the following aspects:

Speed (i.e. Response Time, data rendering and accessing)

Capacity

Stability

Scalability

Performance testing can be either qualitative or quantitative and can be divided into different sub-types such as Load testing and Stress testing.

Load Testing

It is a process of testing the behavior of a software by applying maximum load in terms of software accessing and manipulating large input data. It can be done at both normal and peak load conditions. This type of testing identifies the maximum capacity of software and its behavior at peak time.

Most of the time, load testing is performed with the help of automated tools such as Load Runner, AppLoader, IBM Rational Performance Tester, Apache JMeter, Silk Performer, Visual Studio Load Test, etc.

Virtual users (VUsers) are defined in the automated testing tool and the script is executed to verify the load testing for the software. The number of users can be increased or decreased concurrently or incrementally based upon the requirements.

Stress Testing

Stress testing includes testing the behavior of a software under abnormal conditions. For example, it may include taking away some resources or applying a load beyond the actual load limit.

The aim of stress testing is to test the software by applying the load to the system and taking over the resources used by the software to identify the breaking point. This testing can be performed by testing different scenarios such as:

Shutdown or restart of network ports randomly

Turning the database on or off

Running different processes that consume resources such as CPU, memory, server, etc.

Usability Testing

Usability testing is a black-box technique and is used to identify any error(s) and improvements in the software by observing the users through their usage and operation.

According to Nielsen, usability can be defined in terms of five factors, i.e. efficiency of use, learn-ability, memory-ability, errors/safety, and satisfaction. According to him, the usability of a product will be good and the system is usable if it possesses the above factors.

Nigel Bevan and Macleod considered that usability is the quality requirement that can be measured as the outcome of interactions with a computer system. This requirement can be fulfilled and the end-user will be satisfied if the intended goals are achieved effectively with the use of proper resources.

Security Testing

Security testing involves testing a software in order to identify any flaws and gaps from security and vulnerability point of view. Listed below are the main aspects that security testing should ensure:

- Confidentiality
- Integrity

- Authentication
- Availability
- Authorization
- Non-repudiation
- Software is secure against known and unknown vulnerabilities
- Software data is secure
- Software is according to all security regulations
- Input checking and validation

- SQL insertion attacks
- Injection flaws
- Session management issues
- Cross-site scripting attacks
- Buffer overflows vulnerabilities
- Directory traversal attacks