Unit - I

Introduction To Software Engineering

1

Q1. What do you mean by software? What are different types of Software? Also discuss the various components of software.

Ans. Software :

Software is a set of instructions, programs that enable the computer to perform specified task.

The term '**Software**' refers to the set of computer programs, procedure and associated documents (flow charts, manuals etc.) which describes the programs and how they are to be used.

More formally, software can be defined as :

- **Instructions (computer programs) :** That when executed, provide desired function and performance.
- **Data structures :** That enable the programs to adequately manipulate information.
- **Document :** That describe the operation and use of the programs.

Types of Software :

There are many different types of software. Today, the range of the software available today is vast and varied. ***Most softwares can be divided into different categories based on the purpose of the user's requirement as :***

1. System Software :

A system software is a set of one or more programs designed to control the operation and extend the processing capability of a computer system. It makes the operation of a computer system more effective and efficient. It helps the hardware components to work together and provides support for the development and execution of application software (programs).

Example : Compilers, editors, operating system, programming language translators, communications software and utility programs.

2. Application Software :

It is a set of one or more programs, designed to solve a specific problem or do a specific task.

Application Software are the programs which on the other hand, performs specific tasks for computer user, as if an application software is designed to design images, then it will only work for image designs.

Example : Word processing software, spreadsheet software, database software, graphics software, personal assistance software, education software and entertainment software.

Components of Software :

Computer software is information that exists in two basic forms :

1. Non-machine executable components.
2. Machine executable components.

Software components are created through a series of translations that map customer requirements to machine-executable code. It defines two principal components of software :

1. **Logicware :** The logical sequence of active instruction controlling the execution sequence (sequence of processing of the data) done by the hardware.
2. **Dataware :** The physical form in which all (passive) information, including logicware, appears to the hardware and which is processed as a result of the logic of the logicware.

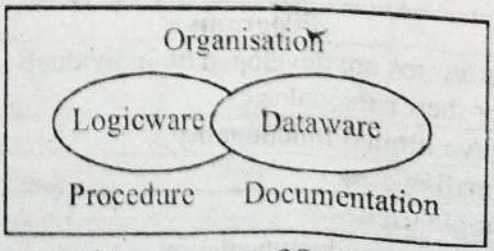


Figure : Components of Software System

Q2. What do you mean by Software Engineering ? Differentiate between program and software.

Ans. Software Engineering (SE) :

A number of definitions of software engineering can be found in literature.

According to **Pfleeger 87** as, *"Software engineering is a strategy for producing quality software."*

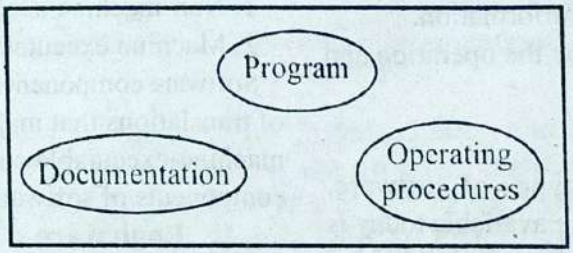
Definition of software engineering given at **NATO conference** is, *"It is the establishment and use of sound engineering principles in order to obtain economically, the software that is reliable and work efficiently on real machines."*

According to the **IEEE** SE is defined, *"Software engineering discipline, which is concerned with all aspect i.e. systematic, quantifiable approach to the development, operation and maintenance of software i.e., the application of engineering to software."*

According to **Bohem**, *"SE is the application of science and mathematics by which the capabilities of computer equipment are made useful to man via computer programs, procedures and associated documentation."*

Program versus Software :

Many people equate the term software with computer program. In fact, this is too restrictive to a view. Software is not just the program, but also all associated documentation and configuration data which is needed to make these programs operate correctly. A system usually consists of a number of separate programs, configuration files which are used to set up these programs, system documentation that describes the structure of the system and user documentation that explains how to use the system and for software products, websites for user to download recent product information.



Software = Program + Documentation + Operating procedures.

Software consists of following :

1. Source code
2. Executables
3. User manuals
4. Requirement analysis and design documents
5. Installation manuals

Programs	Software
Programs are developed by individuals for their personal use.	Software is usually developed by a group of software engineers working in a team.
Have limited functionality.	More functionality.
Small in size.	Large in size.
Single user.	Large number of users.
Lack proper documentation.	Good documentation support
Adhoc development.	Systematic development.
Lack of user interface.	Good user interface.

Q3. Describe the characteristics of the Software.

OR

State the characteristics of software that make it different from hardware.

OR

Write down the characteristics of a software. Illustrate with the diagram that the software doesn't wear out.

Ans. Software is a logical rather than a physical system element. Therefore, software has characteristics that are considerably different from those of hardware. Some of the major differences are the following:

1. Software is developed or engineered, it is not manufactured :

- The concept of 'raw material' is non-existent here. It is better visualized as a process, rather than a product.
- The 'human element' is extremely high in software development as compared to manufacturing.
- The development productivity is highly uncertain even with standard products, varying greatly with skill of the developers.
- The development tools, techniques, standards and procedures vary widely across and within an organisation.
- Quality problems in software development are very different from those in manufacturing whereas the manufacturing quality characteristics can be objectively specified and easily measured, those in the software engineering environment as rather elusive.

2. Software development presents a job-shop environment :

- Here, each product is custom-built and hence unique.
- It can't be assembled from existing components.
- Human skills, the most important element in a job shop, is also the most important element in software development.

3. Time and effort for software development are hard to estimate :

- Interesting work gets done at the expense of dull work and documentation, being a dull work, gets the least priority.
- Programmers tend to be optimistic, not realistic and their time estimates for task completion reflect their tendency.
- Programmers have trouble in communicating.

4. Software doesn't wear out :

There is a well known "bath tub curve" in reliability studies for hardware products. The curve is given in Figure (a). The shape of the curve is like "bath tub" and is known as bathtub curve.

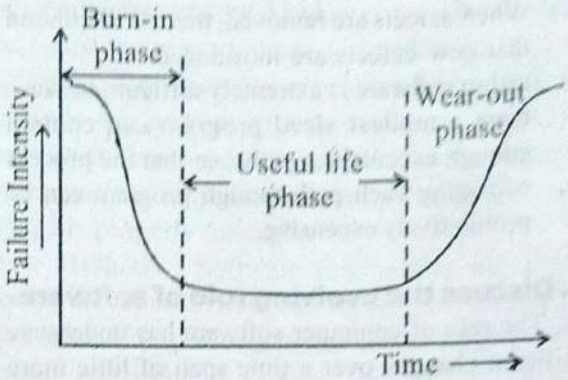


Figure (a). Bathtub Curve

Instead, the software becomes obsolete due to new operating environment, new user requirements, etc. Hence, it can only retire but not wear out. So, it should follow the curve shown in Figure (b).

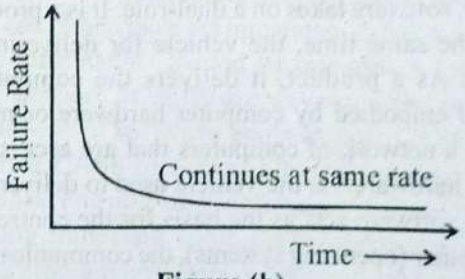


Figure (b)

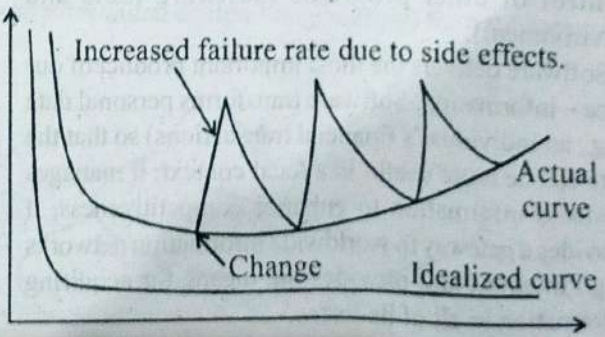


Figure (c). Idealized and actual failure curves for software.

(i) Software normally does not lose its functionality with use.

(ii) It may lose its functionality in time, however, as the user requirements changes.

(iii) When defects are encountered, they are removed by rewriting the relevant code, not by replacing it with available code. That means that the concept of replacing the defective code by spare code is very unusual in software development.

(iv) When defects are removed, there is likelihood that new defects are introduced.

(v). Testing software is extremely difficult, because even a modest sized program can contain enough executables paths, so that the process of testing each path though program can be prohibitively expensive.

Q4. Discuss the evolving role of software.

Ans. The role of computer software has undergone significant changes over a time span of little more than 50 years. Dramatic improvements in a hardware performance, profound changes in computing architecture, vast increase in memory and storage capacity and a wide variety of unusual input and output options have precipitated, more sophisticated and complex computer-based systems.

Today, software takes on a dual-role. It is a product and at the same time, the vehicle for delivering a product. As a product, it delivers the computing potential embodied by computer hardware or more broadly, a network of computers that are accessible by local hardware. As the vehicle used to deliver the product, software acts as the basis for the control of the computer (operating systems), the communication of information (networks) and the creation and control of other programs (software tools and environment).

Software delivers the most important product of our time – information. Software transforms personal data (e.g., an individual's financial transactions) so that the data can be more useful in a local context; it manages business information to enhance competitiveness; it provides a gateway to worldwide information networks (e.g., Internet) and provides the means for acquiring information in all of its forms.

Computer software has become a driving force. It is the engine that drives business decision-making. It serves as the basis for modern scientific investigation and engineering problem-solving. It is a key factor that differentiates modern products and services. It is embedded in systems of all kinds : transportation, medical, telecommunications, military, industrial process, entertainment, office products, etc.

Q5. Explain various elements of software engineering. Define its importance.

Ans. Software Engineering :

According to **Fritz Baurer**, ***"SE is the establishment and use of sound engineering principles in order to obtain economically the software, that is reliable and works efficiently on real machines."***

The basic objective of SE is to develop methods and procedures for software development that can scale up for large systems and that can be used to consistently produce high quality software at low cost and with a small cycle time.

The goal of SE is to produce good software quality product by improving development process, controlling cost of software development, producing software that satisfies specification, producing software that is reliable and efficient.

Need of Software Engineering :

- People developing systems were consistently wrong in their estimates of time, effort and costs.
- Reliability and maintainability were difficult to achieve.
- Delivered systems frequently did not work.
- Fixing bugs in delivered software produced more bugs.
- Increase in size of software systems.
- Changes in the ratio of hardware to software costs.
- Increased cost of software maintenance.
- Demand for larger and more complex software systems.

Elements of Software Engineering :

1. Methods : A method is a procedure for producing some result. It is also referred to as a technique. Methods generally involve some formal notation and processes.

2. Procedures : A procedure then combines tools and/or methods to produce a particular product. Algorithms are examples of procedures also.

3. Tools : Tools are automated systems that increase accuracy, efficiency, productivity, or quality of the end product. A tool that automates the preparation of a data-flow diagram can increase the accuracy, efficiency and productivity.

4. Paradigms : Paradigms refer to particular approaches for building software such as object-oriented paradigms. Different paradigms have their own pros and cons that make them suitable over one another in a given situation.

Importance of Software Engineering :

Now-a-days, software engineering is playing a very important role for software development. Today, software developing companies have realised the difference between software and a small program. The importance of software engineering is as follows:

- It improves the productivity of programming/ development process.
- Improve the comprehension of developed software systems.
- Controlling and predicting the cost of software developments.
- Producing what the customer wants.
- Producing software that satisfies specifications.
- Producing reliable software systems.
- Producing efficient software systems.
- Producing Extensible (that can be expanded) software systems.
- Producing the software within time limits.
- Improve the quality of software product at all levels.
- Producing the various products in addition to programs, **for example,** Documentation, Requirement documents, Specifications, Development plans, etc.

Q6. "Software Engineering is a layered technology". Comment.

Ans. "Software Engineering is a layered technology". Any engineering approach (including software engineering) must rest on an organisational commitment to quality. Total Quality Management and similar philosophies foster a continuous process improvement culture and this culture ultimately leads to the development of increasingly more mature approaches to software engineering. The software engineering layers are :

(a). Quality Focus : The bedrock that supports software engineering is quality focus. The foundation for software engineering is the process layer.

(b). Process : Software Engineering process is a glue that holds the technology layers together and enables rational and timely development of computer process. Process defines a framework for a set of key process areas (KPA's) that must be established for effective delivery of software engineering technology. The key process areas form the basis for management control of software projects and establish the context in which technical methods are applied, work products (models, documents, data, reports, forms, etc.) are produced, milestones are established, quality is assured and change is properly managed.

(c). Methods : Software Engineering methods provide the technical **how-to's** for building software. Methods encompasses a broad array of tasks that include requirement analysis, design, program, construction, testing and support. Software engineering methods rely on a set of basic principles that govern each area of the technology and include modeling activities and other descriptive techniques.

(d). Tools : Software Engineering tools provide automated or semi-automated support for the process or methods. When tools are integrated, so that another can use information created by one tool, a system for the support of software development called **computer-aided software engineering (CASE)**, is established. CASE combines software, hardware and a software engineering database (a repository containing important information about analysis, design, a program construction and testing) to create a software engineering analogous to CAD/CAE (computer-aided design / engineering) for hardware.

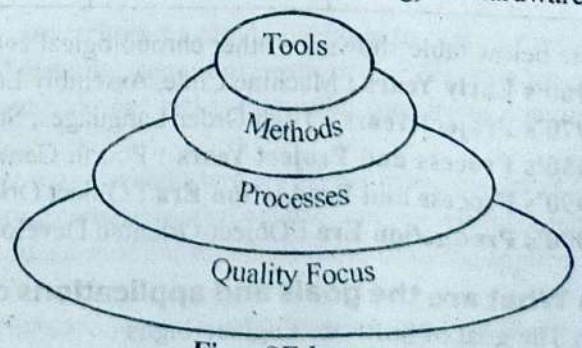


Fig. : SE Layers

Q7. Discuss evolution of Software Engineering.

Ans. Evolution of Software Engineering :

In 1967, a NATO study group coined the term 'Software Engineering'. The premise was building software or engineering software similar to other engineering tasks. This premise was endorsed in 1968 by the NATO software engineering conference which concluded that software engineering should use the philosophies and paradigms of established engineering disciplines to solve the existing software crisis.

During the Early Years, Software development was virtually unmanaged, until scheduler slipped or costs began to escalate. During the early years, general-purpose hardware became commonplace. Software, on the other hand, was custom – designed for each application and had a relatively limited distribution.

In Second Era, multiprogramming, multi-users introduced new concepts of human-machine interaction. Interactive techniques opened a new world of applications and new levels of hardware and software sophistication. Real-time systems could collect, analyse and transform data from multiple sources, thereby controlling processes and producing output in milliseconds. Advances in on-line storage led to the first generation of DBMS. This era was also characterised by the use of product software and the advent of "**software houses**".

In the Third Era, the distributed system consisting of multiple computers each performing functions concurrently and communicating with one another greatly increased the complexity of computer-based system. Global and local area networks, high-bandwidth communications and increasing demands for "instantaneous" data access put heavy demand on software developers.

The Fourth Era, moves us away from individual computers and computer programs and towards the collective impact of computers and softwares. Power desktop machines controlled by sophisticated operating systems, networked locally and globally and coupled with advanced software applications became the norm.

In between 80's to 90's, software companies evolved, where standards were set for developing software in a highly disciplined environment. Object-oriented methods emerged during this period. A number of CASE tools were also developed during this phase.

The trend in the development of software engineering has been an increased emphasis on technique to achieve higher quality. The latest trend use software as a product. Software companies are trying to develop good quality software as a product.

The summary of evolution over the years is shown in a tabular format given below :

The Early Years (1950 – 1960)	The Second Era (1960 – 1980)	The Third Era (1980 – 1990)	The Fourth Era (1990 – till now)
• Batch Orientation • Limited Distribution • Customer Software	• Multi-User • Real-Time • Database • Product Software	• Distributed System • Embedded "Intelligence" • Low Cost Hardware • Customer Impact	• Powerful Desktop System • Object-Oriented Technologies • Expert Systems • Artificial Neural Network • Parallel Computing • Network Computers

The below table shows the other chronological evolution of software engineering :

1960's Early Years : Machine Code, Assembly Language.

1970's Project Years : High Order Languages, Structural Programming.

1980's Process and Project Years : Fourth Generation Language, Process Improvement.

1990's Process and Production Era : Object Oriented Development, Measurement Paradigm.

2000's Production Era : Object-Oriented Development and Domain Engineering, Software Reuse.

Q8. What are the goals and applications of Software Engineering?

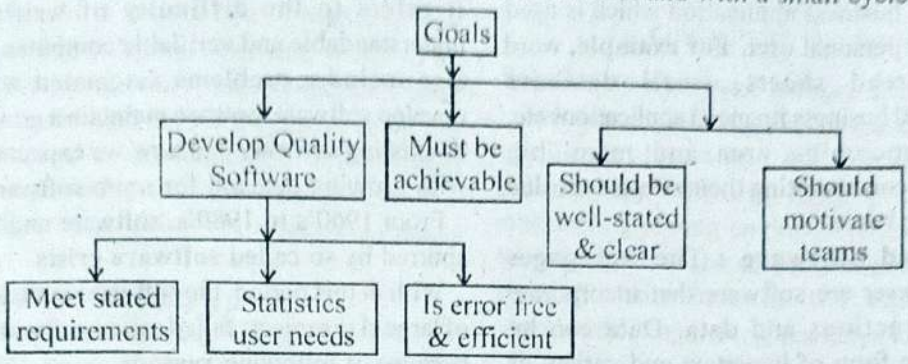
Ans. The goal of Software Engineering is :

1. To produce software quality product by -

(a). Improving development process.

(b). Controlling cost of software development.
(c). Producing software that satisfies specification.
(d). Producing software that is reliable and efficient.

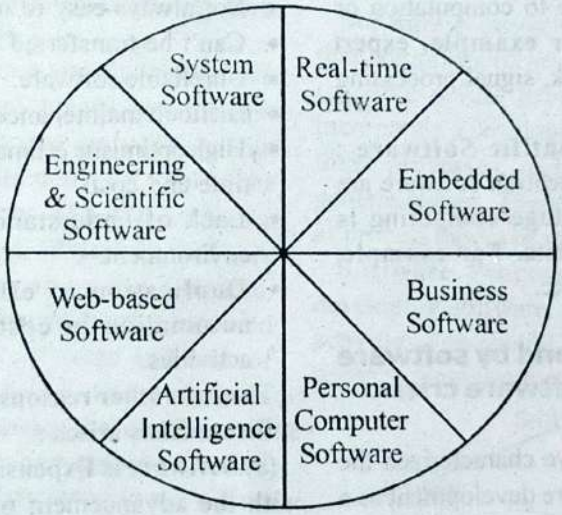
2. To develop methods and procedures for software development that can be
(a). Scaled up for large systems.
(b). Used to consistently produce high quality software at low cost and with a small cycle time.



The goals must be identified for every phase in the projects. These can be goals and sub-goals in keeping with activity distribution for every phase.

Software Applications :

Software has become integral part of most of the fields of human life. We name a field and we find a usage of software in that field. ***There are various categories of software applications are :***



(i). System Software : System software / programs are written to provide service to other programs. System software is a collection of various programs. **Example,** compilers, editors and file management utilities play important role. System software also performs various tasks. **Example,** concurrent operation, multiple external interfaces, process management.

(ii). Real-Time Software : These are the software that monitor/analyze/control real world events as they occur. Elements of real-time software include a data gathering component that collects and formats information from an external environment, an analysis component that transforms information as required by the application. **For example.** Weather forecasting.

(iii). Embedded Software : Intelligent consumer products are becoming very popular. It can perform very limited functions. **Example,** microwave oven, washing machine, digital functions in an automobile such as fuel control, etc. It handles hardware components and is termed as "Intelligent Software".

(iv). Business Software : Business information processing is the largest single software application area. It has a very broad area such as payroll, accounts receivable, accounts payable, inventory, purchase, marketing, etc.

(v). Personal Computer Software : It may be said to be a small business application which is used for single user or personal user. **For example,** word processing, spread sheets, small database management, small business financial applications etc. This is very upcoming area and many big organisations are concentrating their efforts here due to large customer base.

(vi). Web-based Software : The web-pages retrieved by browser are software that incorporate executable instructions and data. Data can be represented in the form of hypertext and variety of visual and audio formats. Web-based software can be used for E-Commerce and Internet Banking. **For example,** CGI, HTML, JAVA etc.

(vii). Artificial Intelligence Software : It makes use of non-numerical algorithms to solve complex problems that are not amenable to computation or straight-forward analysis. **For example,** expert systems, artificial neural network, signal processing software etc.

(viii). Engineering & Scientific Software : Scientific and engineering application software are categorised into this group. Huge computing is normally required to process data. **For example,** CAD, CAM, SPSS, MATLAB, etc.

Q9. What do you understand by software crisis? Give reasons of software crisis.

Ans. **Software Crisis :**

Many industries observers have characterized the problems associated with software development as a "Crisis". The word "Crisis" is defined as "a turning point in the course of anything; decisive or crucial time, stage or event." Yet in terms of overall software quality and the speed with which computer based systems and products are developed. There has been no "turning point", no "decisive time", only show evolutionary change, punctuated by explosive technological changes in disciplines associated with software.

It refers to a set of problems encountered in the development of computer software during 1960's. Within this period, software industry unsuccessfully attempted to build larger and larger software systems by simply scaling up existing development techniques. It covers all the ills of software and computing systems industry. It is characterized by an inability to develop software on time, on budget, and within requirements. It refers to the difficulty of writing correct, understandable and verifiable computer programs. It also includes problems associated with how we develop software, how we maintain a growing volume of existing software and how we expect to keep pace with growing demand for more software.

From 1960's to 1980's, software engineering was spurred by so called **software crisis**.

With in this period, the software industry, a number of large size projects failed called **software runaways** because of following **reasons** :

- Development teams exceeding the budget.
- Late delivery of software.
- Poor quality.
- User requirements not completely supported by the software.
- Not always easy to use.
- Can't be transferred to another machine easily.
- Unreliable software.
- Difficult maintenance.
- High optimistic estimation regarding development time and cost.
- Lack of understanding of problem and its environment.
- Duplication of efforts due to absence of automation in cost software development activities.

There are **other reasons** due to which the following software crisis arises :

(a) Software is Expensive : Over the past decades, with the advancement of technology, the cost of hardware has consistently decreased. On the other hand, the cost of software is increasing. The main reasons for the high cost of software is that software development is still labor-intensive. As the main cost of producing software is the manpower employed, the cost of developing software is generally measured in terms of person-months of effort spent in development and productivity is frequently measured in the industry in terms of DLOC (Delivered Line of Code) per person-month.

(b) Late, Costly and Unreliable : There are many

instances quoted about software projects that are behind schedule and have heavy cost overruns. The software industry has gained a reputation of not being able to deliver product on time and within budget.

(c). Problem of Change and Rework : When the software is delivered and deployed, it enters the maintenance phase. All system need maintenance, but for other system, it is largely due to problems that are introduced due to aging. Software needs to be maintained not because some of its components wear out and need to be replaced, but because there are often some residual errors remaining in the system that must be removed as they are recovered.

There are three dimensions to manage software crisis :

1. Multiple programmers.
2. Application complexity.
3. Maintenance.

Q10. What are the symptoms of the present software crisis ? What factors have contributed to making of the present solution for software crisis ? What are possible solutions to the present software crisis ?

Ans. Software crisis attributes a set of problems that are encountered in the development of computer software. Statistics shows that only 2% of the projects were used as they are delivered, 3% of the projects used after modifications, 47% of the software was never used only delivered, 19% of the software rejected or reworked and 29% was not even delivered. The problems increased because of increased dependence of business on software and lack of systematic approach to build the software. Developers and researchers realized that development of software was not an easy and straight forward task; instead it required lot of engineering principles.

In other words, the software crisis is characterised by an inability to develop software on time, on budget and within requirements. **There are many factors that have contributed to making of the present software crisis :**

1. **Understanding the Gap :** One of the most difficult aspects of development is in getting both customers and the developers to understand each other's view.
2. Software takes long time to develop.
3. Larger problem sizes.
4. Lack of adequate training in software engineering.
5. No assurance of quality.
6. Difficult to monitor and control.
7. Difficult to maintain.
8. Increasing skill shortage.
9. Low productivity improvements.

Solution to Present Software Crisis :

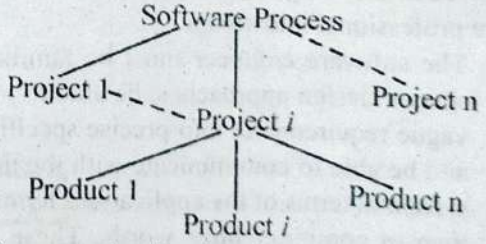
Software engineering ensures that development team is able to detect and correct errors. The only satisfactory solution is to practice software engineering among engineers, coupled with further advancements to the software engineering discipline itself. It also reduces the trial and error aspect in software development. It also helps in the maintenance of the software. It provides set of rules to the project manager to take full control of project.

Q11. Define Software Engineering Paradigm (Models).

Ans. Paradigms of Software Engineering (Models):

Software engineering is essentially a set of steps that comprises of processes, methods and tools and to solve actual problems. A software engineer incorporate a development strategy that encompasses these processes, methods and tools layer and the generic phases. These steps or strategy is referred to as process model or software engineering paradigms.

Software Process refers to the method of developing software. It specifies the abstract set of activities that should be performed to go from user needs to final product.



Software project is a development project in which software process is used. Software products are the outcomes of software project.

Characteristics of Software Process :

1. **Optimality :** The software process should be able to produce high quality software at low cost.

2. **Scalability** : It should also be applicable for large software projects.
3. **Predictability** : Predictability of a process determines how accurately the outcome of following process in a project can be predicted before the project is completed.

There are varieties of software engineering paradigms available. Selection of model depends on the nature of the project, the type of the application, the tools proposed to be used and the kind of controls and documentation that would be required.

There are variety of software process models or process paradigms or software engineering paradigm. Software models as some of them are :

1. Waterfall Model (Linear Sequential Model).
2. Prototyping model.
3. Spiral model.
4. Rapid application development model.
5. Incremental model.
6. Evolutionary model.

Q12. Explain the role of software Engineer?

OR

List out the role and responsibilities of software engineer?

Ans. The evolution of the software engineering field has defined the role of the software engineer. A software engineer must of course be a **good programmer**, be **well-versed in data structure and algorithms** and **be fluent in one or more programming languages**. These are requirements for the small-scale software development to be done by a single individual. But a software engineer is also involved in very large scale software, which requires more professional knowledge.

1. The software engineer must be familiar with several design approaches, be able to translate vague requirements into precise specifications and be able to communicate with the user of a system in terms of the applications terms rather than in computer buzz words. These in turn ***require the flexibility and openness to grasp and become conversant with the essentials of different applicant areas.***
2. The software engineer needs the ability to move among several levels of abstraction at different stages of the project from specific application procedures and requirements to abstractions for the software system to a specific design for the system and finally to the detailed design, testing and delivery level.
3. **Modeling** is another requirement that the software engineer must be able to build and use a model of the application to guide choices of the many trade-offs that he or she will face. The model is used to answer questions about both the behaviours of the system and its performance. Preferably, the model can be used by the engineer as well as the user.
4. The software engineer is a member of a team and therefore needs communication skills and interpersonal skills.
5. The software engineer also needs the ability to schedule work, both of his or her own and that of others.

As discussed above, a software engineer is responsible for many things. In practice, many organisation divide the responsibilities among several specialists with different titles. For example, an analyst is responsible for deriving the requirements and a programmer is responsible for coding the system. A rigid fragmentation of role, however, is often counter productive.

Q13. What do you mean by SDLC?

OR

Briefly discuss various phases of software development life cycle.

OR

What do you mean by SDLC? Write various stages of SDLC.

Ans. A software life cycle model is either a descriptive or prescriptive characterization of how software is or should be developed. A software life cycle model (also called process model) is a descriptive and diagrammatic representation of the software life cycle. A descriptive model describes the history of how a particular software system was developed. Descriptive models may be used as the basis for understanding and improving software development processes or for building empirically grounded perspective models.

A life cycle model represents all the activities required to make a software product transit through its life cycle phases. It also captures the order in which these activities are to be undertaken. In other words,

a life cycle model maps the different activities performed on a software product from its inception to retirement.

Needs a software development life cycle model:

The development team must identify a suitable life cycle model for the particular project and then adhere to it. Without using of a particular life cycle model, the development of software product would not be in a systematic and disciplined manner. When a team is developing a software product, there must be a clear understanding among team members about when and what to do. Otherwise, it would lead to confusion and project failure.

There are various software development approaches defined and designed which are used/employed during development process of software. Each phase has an phase "output".

Phase	Output
Requirement analysis	Software Requirements Specification (SRS) use cases.
Design	Design Document, Design Classes
Implementation	Code
Test	Test Report, Changes Requests.

The SDLC consists of the following phases during its life :

1. Project Identification and Selection
2. Project Initiation and Planning.
3. Analysis
4. Design
5. Coding.
6. Testing.
7. Implementation.
8. Maintenance.

Although any life cycle appears as to be a sequentially ordered set of phases, but this is not actually so. The steps and their sequences are meant to be adapted as required for a project, consistent with management approaches. Thus, the steps in SDLC can be performed in parallel, or iteratively in circular manner.

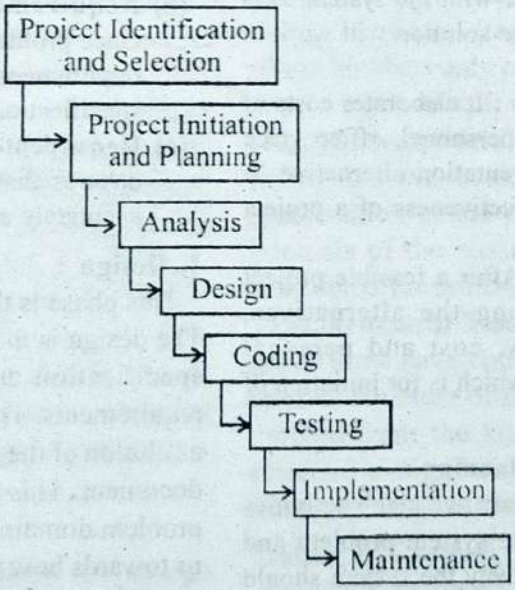


Fig : Software Development Life Cycle

1. Project Identification and Selection :

This is the first phase that identifies the need for a new or enhanced system. Information needs of the

organisation as a whole are examined and project to meet these needs are identified. The organisation's information system needs may result from requests to deal with problems in current procedures, from the desire to perform additional tasks or from the realisation that information technology could be used to capitalize on an existing opportunity.

When the request is made, the first step that takes place is being divided into three different parts :

(a) Request Clarification : Many requests from users in the organisation are not clearly defined. Therefore, it becomes necessary that project requests must be examined and clarified properly before considering system investigation.

(b) Feasibility Study : Feasibility is the analysis of risks, costs and benefits relating to economic condition, technology and user organisation. The problem to be automated is analyzed in sufficient detail to ensure that all aspects of feasibility are evaluated. **There are several types of feasibilities:**

(i) Technical Feasibility : In this analysis, alternatives for hardware, software and general design approach are determined to be available, appropriate and functional :

- Does the necessary technology exist?
- Can the system be expanded, if developed?

(ii) Operational Feasibility : It is a measure of how people are able to work with the system. It is also measure of how well the solution will work in the organisation.

(iii) Economic Feasibility : It elaborates costs of special hardware, software, personnel, office space and so on for each implementation alternative. It involves estimating cost-effectiveness of a project underline.

(c) Request Approval : After a feasible project request is approved among the alternatives, completion time, priority, cost and personal requirements are estimated which is for initiation of development.

2. Project Initiation and Planning :

In this second phase, there are two major activities – formal investigation of the system problem and presentation of reasons that why the system should or should not be developed by the organisation. A critical step at this point is determining the scope of the proposed system. The project leader and initial team of the system analysts also produce a specific plan, which the leader will follow along with the SDLC step. This baseline project plan customizes and standardises the SDLC and specifies the time and resources needed for its execution. This phase gives a detailed plan for conducting remaining phases of SDLC for the proposed system.

3. Analysis or (Requirement Analysis) :

In this phase, the requirements of the "to be developed software" are established, i.e. one must know what the problem is, before it can be solved. The requirements are usually the services, it will provide, its constraints and goals of software.

Requirement analysis provides appropriate mechanism for understanding what customer wants, analysing needs, accessing feasibility, negotiating reasonable solution, specifying solution unambiguously, validating specification and managing requirement as they are transformed into an operational system.

The goal of this phase is to produce SRS document or Requirement Document that describes all the requirements. ***The three major activities in this phase are :***

(a) Problem Understanding or Analysis : During this activity, the requirements of new system to be developed are analysed. This requires the th ough understanding of the existing system.

(b) Requirement Specification Document : Once problem is analysed and understand, the requirements must be specified on requirement specification document or SRS.

(c) Requirement Validation : This activity ensures that SRS reflect actual requirements accurately and clearly.

4. Design :

This phase is the most creative and challenging. The design is the bridge between the requirement specification and final solution for satisfying requirements. The purpose of this phase is to plan a solution of the problem specified by requirement document. This phase is the first in moving from problem domain to solution domain. Design takes us towards how to satisfy the needs. The output of this phase is the design document used later during implementation, testing and maintenance. The goal of this phase is to produce a model or representation of system which can be used later to build that

system. Software design sits at the Kernel of the software engineering and applied regardless of software process model that is used. It deals with transforming customer requirement as described in SRS document into form that is implementable using a programming language. The produced model is called Design of System.

The goals of this phase are :

- To determine which system components will cover which requirements in system specification?
- How these system components will work together?

In this, the Important Activities are :

- Designing system architecture.
- Designing the underlying logical data model.
- Designing the algorithmic structure of system component.
- Validating the system architecture and algorithms to realize the individual system components.

The Outcomes of this Phase are :

- Description of logical data model.
- Description of system architecture.
- Description of algorithmic structure of system components.
- Documentation of design-decisions.

A design can be of two types :

(a) Function-Oriented Design : The design consists of module definition with each module supporting a final abstraction. In this approach, system is viewed as a transformation function transforming input to desired output.

(b) Object-Oriented Design : Basic system component is a module that support data abstraction. The goal of design is to identify objects that system contains and relationship and interaction between objects, so that system implements specification.

5. Coding :

Once, the design phase is complete and design document have been reviewed, every module identified and specified in design document is The goal of coding phase is to translate the design of the system into code in a given programming language. This phase affects both testing and implementation and thus increases software cost. This can be reduced by well-written code. During this phase, the focus should be on developing programs that are easy to read and understand and this can be achieved using structured programming language.

6. Testing :

Once a source code has been generated, software must be tested to uncover and correct as many errors as possible before delivering it to the customer. Software testing is a major quality control measure used during software development.

Testing is an extremely critical and time-consuming activity. It requires proper planning of the overall testing process. The testing process starts with a Test Plan that identifies all the testing-related activities that must be performed and specifies the schedule, allocates the resources and specifies guidelines for testing. The test plan specifies conditions that should be tested, different units to be tested and the manner in which the modules will be integrated together.

There are Two Basic Approaches to Testing :

(a) Functional Testing : It refers to testing which involves only observation of the output for certain input values. In this, the structure of the program is not considered. Test cases are decided solely on the basis of the requirement or specification of the program or module and the internals of the module or the program are not considered for selection of test cases.

(b) Structural Testing : In this approach, test group must have complete knowledge about the internal structure of the software. In this, test are derived from the knowledge of the software's structure and implementation. It is concerned with testing the implementation of the program.

Testing is Carried Out at 3 Levels :

(a) Unit Testing : In this, a module is tested separately and is often performed by coder himself simultaneously alongwith the coding of module

modules, this testing is performed to detect design errors by focusing on testing the interconnection between modules.

(c) System Testing : After system is put together, this testing is performed. Here system is tested against the system's requirement to see, if all requirements are met and if system perform as specified by the requirement.

7. Implementation :

The implementation phase is less creative than the design phase. It is primarily concerned with user training, site preparation and file conversion. During the final testing, user acceptance is tested. The acceptance testing is called **end-user testing** in which software is tested in the real world by the intended audience. It is the formal means by which we ensure that the software does actually meet the essential user requirements. Depending on the nature of the system, extensive end-user training may be required. It prepares the user for testing and converting the system. It is crucial for minimizing resistance to change. The implementation process converts the new software design into operation. The conversion includes review of the project plan and implementation plan, convert files, conduct parallel processing, discontinue old system, user training, plan for post-implementation review.

There are three types of implementation :

(a) Implementation of a computer system to replace a manual system.

(b) Implementation of new computer system to replace an existing one.

(c) Implementation of a modified application or software to replace an existing one.

8. Maintenance :

After implementation is complete and user staff is adjusted to the system, this phase begins. This phase is also known as **Post-Implementation**. It describes activities following the delivery of the initial working version of the software system. Upon completion, a software system is handed over to the client. Any changes after the client has accepted the system are categorised as maintenance.

Changes to code, documentation, manuals or any other components are examples of maintenance.

It is the most difficult aspect of software development. The first step after maintenance request is received, is to identify the type of maintenance. Software maintenance follows conversion to the extent that changes are necessary to maintain satisfactory operation relative to changes in the user's environment.

There are four types of maintenance :

(a) **Corrective Maintenance :** It aims to correct any remaining errors regardless of where they may arise – specification, design, coding, testing, documentation etc.

(b) **Adaptive Maintenance :** It changes the system to react to changes in the environment in which the system operates. **For Example,** porting to a new compiler, operating system &/or hardware.

(c) **Preventive Maintenance :** It involves the activities to update the software in anticipation of any future problems.

(d) **Perfective Maintenance :** It enhances the performance or modify the programs to respond to the user's additional or changing needs.

Q14. Explain Linear Sequential Model / Waterfall Model. Also list its advantages & disadvantages.

OR

Which are the major phases in the waterfall model of software development? Which phases consumes the maximum effort for developing a typical software product ?

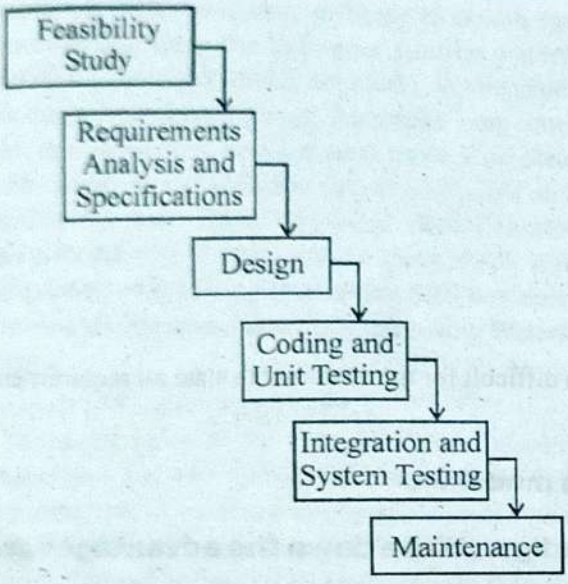
Ans. The waterfall model is an established approach that dominated software development for a number of years and is widely used. It has the virtue of simplicity.

In the waterfall model, software development is split up into a number of independent steps. These steps are carried out in sequence one after the other. Each stage produces a product which is the input into the next stage. It is important to realize that each stage is pursued, until its conclusion before the next stage is begun.

Different Phases of the Waterfall Model :

The waterfall model is well-designed and intuitively obvious, it is not a practical model in the sense that it can not be used in actual software development projects.

Thus, this model can be considered to be a theoretical way of developing software. Waterfall model divides the life cycle into the following phases:



1. Feasibility Study : The main aim of feasibility study is to determine whether it would be financially and technically feasible to develop the product.

At first, project managers or team leaders try to have a rough understanding of what is required to be done by visiting the client side. They study different input data to the system and output data to be produced by the system.

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

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

2. Requirement Analysis and Specification : Requirements are the set of functionalities and constraints that the end-user expects from the system. So, the aim of the requirement analysis and specification phase is to understand the exact requirements of the customer and to document them, properly. It consist of two activities :

(a) Requirement gathering and analysis : The goal of the requirement gathering activity is to collect all relevant information from the customer regarding the product to be developed. This is done to clearly understand the customer requirements so that incompleteness and inconsistencies are removed. The requirement analysis activity is begun by collecting all relevant data regarding the product to be developed from the user of the product and from the customer through interviews and discussions.

(b) Requirement specification : It is necessary to identify all ambiguities and contradictions in the requirements and resolve them through further discussions with the customer. After problems are resolved and all the requirements are properly understood, this phase start. During this activity, the user requirements are systematically organised into a SRS document (Software Requirement Specification).

3. Design : The goal of this phase is to transform the requirements specification in the SRS document into a structure that is suitable for implementation in some programming language. Two approaches are available as :

(a) Traditional design approach, consist of two activities i.e. first a **structured analysis** where the detailed structure of the problem is examined, followed by a **structured design,** in which the result of structured analysis are transformed into the software design.

(b) Object-oriented approach, various objects that occur in the problem domain and the solution domain are first identified and relationship among these objects. The object structure is further refined to obtain a detailed design.

4. Coding and Unit Testing : After design phase, the work is divided in modules/units and actual coding is started, it is first developed into small programs called **units.** Each unit is developed and tested for its functionality.

5. Integration and System Testing : In the above phase, units are integrated into a complete system during integration phase and tested to check, if all modules coordinate between each other and the whole system behaves as per requirement or not. System

testing laid down all requirements of SRS document. It consist of three activities :

(a) **α-testing** : It is performed by the development team.

(b) **β-testing** : It is performed by a friendly set of customers.

(c) **Acceptance testing** : It is done itself by customer after the product delivery.

6. Maintenance : As system is received on customer end, the issues related to the system are solved after deployment of the system, all problems arise time to time and needs to be solved.

Advantages :

1. Easy to understand, easy to use.
2. Clear project objectives.
3. Provide structure to inexperience staff.
4. Stable project requirements.
5. Milestones are well understood.
6. Program of system is measurable.

Disadvantages :

1. Time consuming.
2. Can't go backward.
3. Difficult in responding to changes.
4. Difficult in iteration.
5. It requires all requirements explicitly, but is often difficult for the customer to state all requirements explicitly.

Q15. What do you understand by prototyping model ?

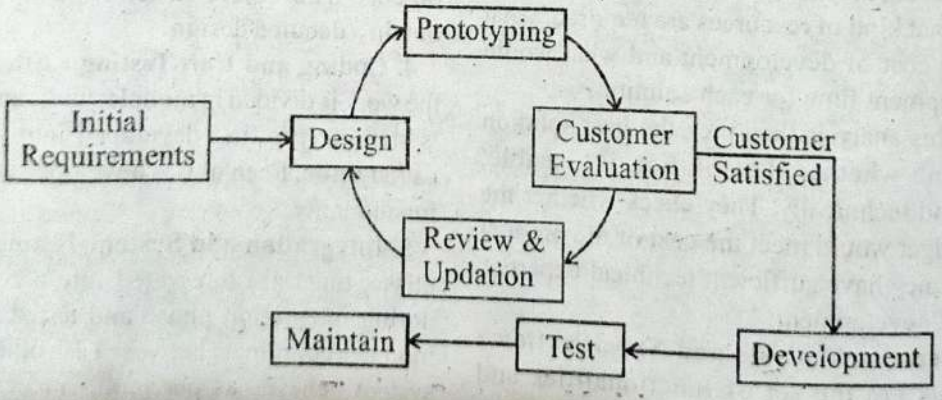
OR

State the process of Prototyping model/Paradigm. Write down the advantages and its disadvantages?

OR

What do you mean by Prototype? What are the major advantages of first constructing the prototype before developing the actual product?

Ans. Software prototyping, refers to 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. The original purpose of a prototype is to allow users of the software to evaluate developers' proposals for the design of the eventual product by actually trying them out, rather than having to interpret and evaluate the design based on descriptions. Prototyping can also be used by end users to describe and prove requirements that developers have not considered and that can be a key factor in the commercial relationship between developers and their clients.



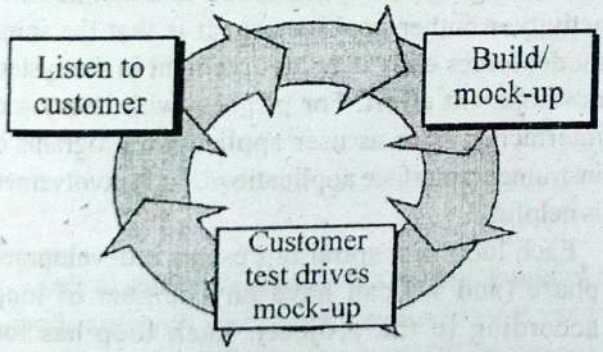
Here, a prototype is made first and based on it, final product is developed. A prototype is a model or a program which is not based on strict planning, but is an early approximation of the final product or software system. A prototype acts as a sample to test the process.

Need of Prototyping Model :

This type of System Development Method is employed, when it is very difficult to obtain exact requirements from the customer (unlike waterfall model, where requirements are clear). While making the model, user keeps giving feedbacks from time to time and based on it, a prototype is made. Completely built sample model is shown to user and based on his feedback, the SRS (System Requirements Specifications) document is prepared. After completion of this, a more accurate SRS is prepared and now development work can start using Waterfall Model.

Process of Prototyping Model :

Developer and customer meet and define the overall objectives for the software, identify whatever requirements are known and outline areas where further definition is mandatory. A "quick design" then occurs. The quick design focuses on a representation of those aspects of the software that will be visible to the customer/user (e.g., input approaches and output formats). The quick design leads to the construction of a prototype. The prototype is evaluated by the customer/user and used to refine requirements for the software to be developed. Iteration occurs as the prototype is tuned to satisfy the needs of the customer, while at the same time enabling the developer to better understand what needs to be done.



Advantages :

1. When prototype is shown to the user, he gets a proper clarity and 'feel' of the **functionality** of the software and he can suggest changes and modifications.
2. This type of approach of developing the software is used for non-IT-literate people. They usually are not good at specifying their requirements, nor can tell properly about what they expect from the software.
3. When client is not confident about the developer's capabilities, he asks for a small prototype to be built. Based on this model, he judges capabilities of developer.
4. Sometimes, it helps to demonstrate the concept to prospective investors to get funds for project.
5. It reduces risk of failure, as potential risks can be identified early and mitigation steps can be taken.
6. Iteration between development team and client provides a very good and conductive environment during project.
7. Time required to complete the project after getting final SRS reduces, since the developer has a better idea about how he should approach the project.

Disadvantages :

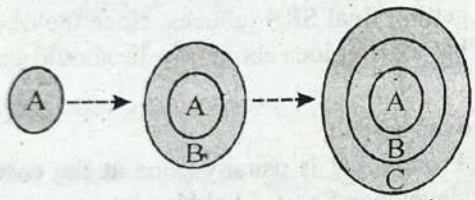
1. Prototyping is usually done at the cost of the developer. So, it should be done using minimal resources. It can be done using Rapid Application Development (RAD) tools. Please note sometimes the start-up cost of building the development team, focused on making prototype, is high.
2. Once we get proper requirements from client after showing prototype model, it may be of no use. That is why, sometimes we refer to the prototype as "**Throw-away**" prototype.
3. It is a slow process.
4. Too much involvement of client, is not always preferred by the developer.
5. Too many changes can disturb the rhythm of the development team.
6. It's reliability is low.

Q16. Explain the process of evolutionary model / Incremental Model.

Ans. Evolutionary Model /Incremental Model:

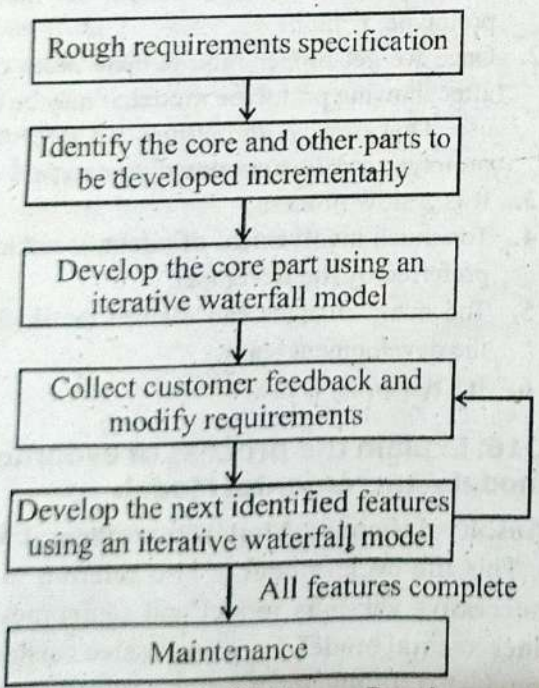
This life cycle model is also referred to as the **successive versions** model and sometimes as the **incremental model**. Each successive version of the product is a fully functioning software capable of

performing more useful work than the previous versions. In this life cycle model, the software is first broken down into several modules (or functional units) which can be incrementally constructed and delivered. The development team first develops the core modules of the system. This initial product skeleton is refined into increasing levels of capability by adding new functionalities in successive versions. Each evolutionary version may be developed using an iterative waterfall model of development. In this model, the user gets a chance to experiment with a partially developed software much before the complete version of the system is released. Therefore, the evolutionary model helps to accurately elicit user requirements during the delivery of the different versions of the software and the change requests, therefore after delivery of the complete software are minimized.



A, B, C are modules of a software product that are incrementally developed and delivered

Evolutionary development of a Software product



Evolutionary Model of Software Development

This model obviates the need to commit large resources in one go for development of the system problems, it is difficult to divide the problem into several functional units which can be incrementally implemented and delivered. Therefore, this model is normally used for large products.

Q17. Explain why Spiral Model is considered to be a meta model?

OR

Define Spiral Model. List the advantages and Disadvantages of spiral model in details?

Ans. Spiral Model :

This model defined by **Barry Boehm** in 1986 is an Evolutionary software process model that attempts to combine the strength of various other models. The spiral model combines the idea of iterative development (prototyping) with the systematic, controlled aspects of the waterfall model. It allows for incremental releases of the product, or incremental refinement through each time around the spiral. The spiral model also explicitly includes risk management within software development. Identifying major risks, both technical and managerial and determining how to lessen the risk helps keep the software development process under control. The spiral lifecycle model allows for elements of the product to be added in, when they become available or known. This assures that there is no conflict with previous requirements and design. This method is consistent with approaches that have multiple software builds and releases and allows for making an orderly transition to a maintenance activity. Another positive aspect is that the spiral model forces early user involvement in the system development effort. For projects with heavy user interfacing, such as user application programs or instrument interface applications, such involvement is helpful.

Each loop in a spiral represents a development phase (and we can have any number of loops according to the project). Each loop has four sections or quadrants :

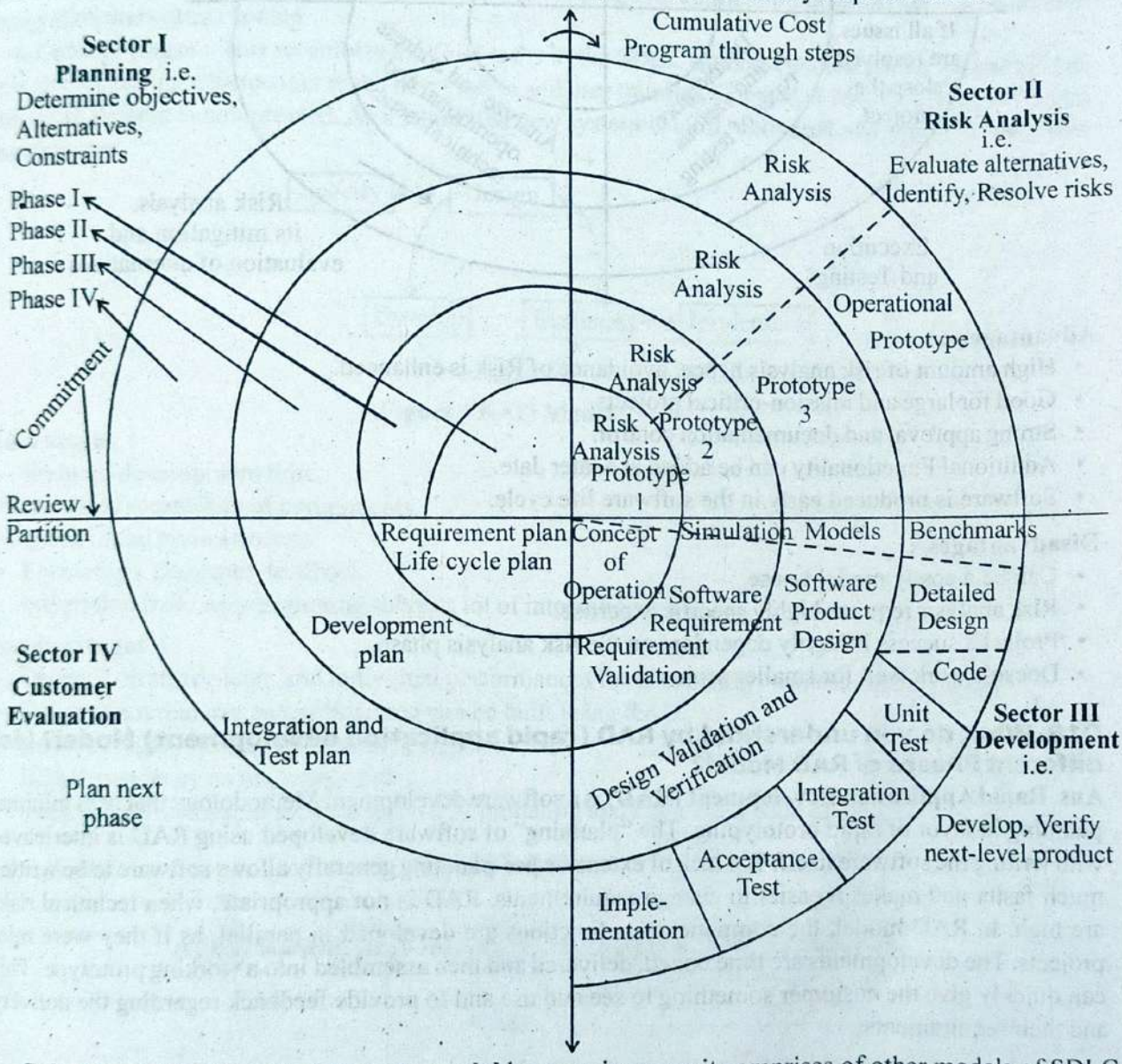
1. ***To determine the objectives, alternatives and constraints***. We try to understand the produc

objectives, alternatives in design and constraints imposed because of cost, technology, schedule, etc.

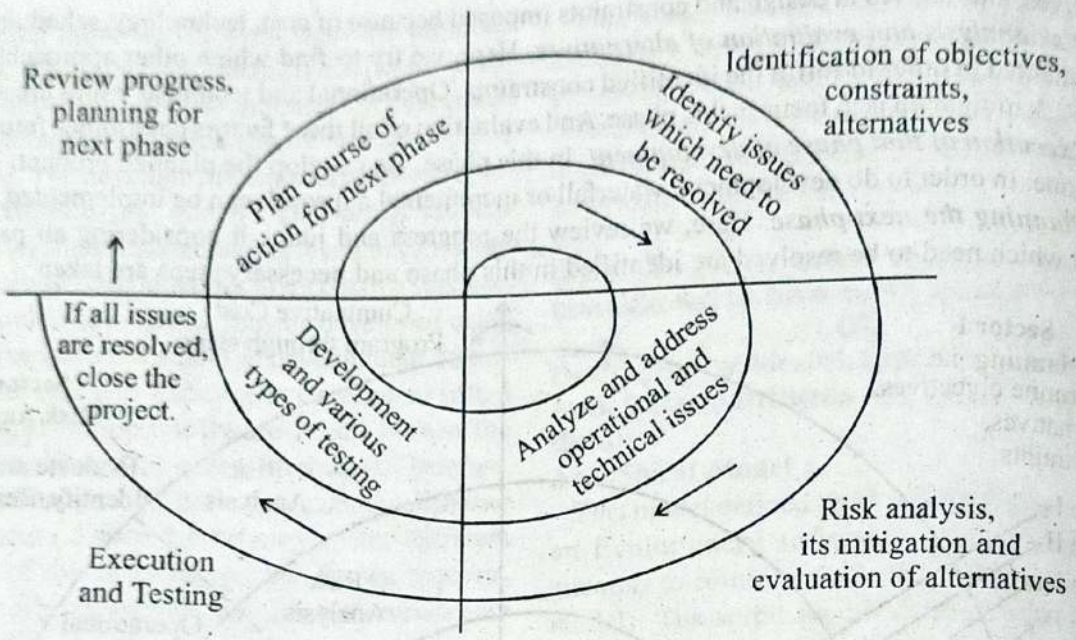
2. ***Risk analysis and evaluation of alternatives.*** Here, we try to find which other approaches can be implemented in order to fulfill the identified constraints. Operational and technical issues are addressed here. Risk mitigation is in focus in this phase. And evaluation of all these factors determines future action.

3. ***Execution of that phase of development.*** In this phase, we develop the planned product. Testing is also done. In order to do development, waterfall or incremental approach can be implemented.

4. ***Planning the next phase.*** Here, we review the progress and judge it considering all parameters. Issues which need to be resolved are identified in this phase and necessary steps are taken.



Spiral model is also called as meta-model because in a way it comprises of other models of SDLC. Both waterfall and prototype models are used in it. Here, we do software development systematically over the loops (adhering to waterfall approach) and at the same time, we make a prototype and show it to user after completion of various phases (just in case of prototype model). This way, we are able to reduce risks as well as follow systematic approach.



Advantages :

- High amount of risk analysis hence, avoidance of Risk is enhanced.
- Good for large and mission-critical projects.
- Strong approval and documentation control.
- Additional Functionality can be added at a later date.
- Software is produced early in the software life cycle.

Disadvantages :

- Can be a costly model to use.
- Risk analysis requires highly specific expertise.
- Project's success is highly dependent on the risk analysis phase.
- Doesn't work well for smaller projects.

Q18. What do you understand by RAD (rapid application development) Model? List different Phases of RAD Model?

Ans. Rapid Application Development (RAD) is a software development Methodology that uses minimal planning in favor of rapid prototyping. The "planning" of software developed using RAD is interleaved with writing the software itself. The lack of extensive pre-planning generally allows software to be written much faster and makes it easier to change requirements. RAD is not appropriate, when technical risks are high. In RAD model, the components or functions are developed in parallel, as if they were mini projects. The developments are time boxed, delivered and then assembled into a working prototype. This can quickly give the customer something to see and use and to provide feedback regarding the delivery and their requirements.

Four phases of RAD :

1. Requirements Planning phase : It combines elements of the system planning and systems analysis phases of the Software development Life Cycle (SDLC). Users, managers and IT staff members discuss and agree on business needs, project scope, constraints and system requirements. It ends when the team agrees on the key issues and obtains management authorization to continue.

2. User design phase : During this phase, users interact with system analysts and develop models and prototypes that represent all system processes, inputs and outputs. The RAD groups or subgroups typically use a combination of Joint Application Development (JAD) techniques and CASE tools to translate user needs into working models. *User Design* is a continuous interactive process that allows users to understand, modify and eventually approve a working model of the system that meets their needs.

3. Construction phase : It focuses on program and application development task similar to the SDLC. In RAD, however, users continue to participate and can still suggest changes or improvements as actual screens or reports are developed. Its tasks are programming and application development, coding, unit-integration and system testing.

4. Cutover phase : This resembles the final tasks in the SDLC implementation phase, including data conversion, testing, changeover to the new system and user training. Compared with traditional methods, the entire process is compressed. As a result, the new system is built, delivered and placed in operation much sooner.

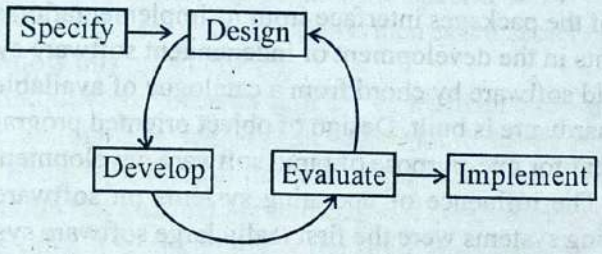


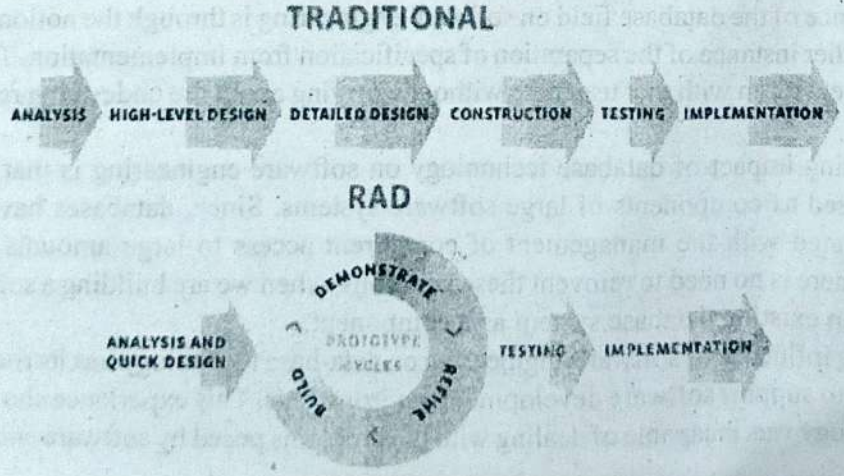
Figure : RAD Model

Advantages :

- Reduced development time.
- Increased reusability of components.
- Quick initial reviews occur.
- Encourages customer feedback.
- Integration from very beginning solves a lot of integration issues.

Disadvantages :

- Depends on strong team and individual performances for identifying business requirements.
- Only system that can be modularized can be built using RAD.
- Requires highly skilled developers/designers.
- High dependency on modeling skills.
- Inapplicable to cheaper projects as cost of modeling and automated code generation is very high.



Q19. Explain the relationship of software engineering to the other fields of computer Science i.e. Programming Languages, Operating System, Database?

Ans. Software engineering has emerged as an important field within computer science. Indeed, there is a relationship between it and many other areas in computer science: these areas both influence and get influenced by software engineering. In the following subsections, we explore the relationship between software engineering and some of the other important fields of computer science.

Programming Languages : The influence of software engineering on programming languages is rather evident. Programming languages are the central tools used in software development. As a result they gave profound influence on how well we can achieve our software engineering goals. In turn, these goals influence the development of programming languages. The most notable example of this influence in recent programming languages is the support of modularity features, such as separate and independent compilation and the separation of specification from implementation in order to support team development of large software. The ADA programming language, for example, supports the development of packages allowing the separation of the packages interface from its implementation and librarian of packages that can be used as components in the development of independent software systems. This is a step towards making it possible to build software by chord from a catalogue of available components and combining them similar to the way hardware is built. Design of object oriented programming languages like C++ is another major development for one-purpose of rapid software development.

Operating Systems : The influence of operating systems on software engineering is quite strong primarily because operating systems were the first really large software systems built and therefore they were the first instances of software that need to be engineered . Many of the first software design ideas originated from early attempts at building operating systems.

Examples of the influence of software engineering techniques on the structures of operating systems can be seen in portable operating systems and operating systems that are structured to contain a small protected kernel that provided a minimum of functionality for interfacing with the hardware and a non-protected part that provided the majority of the functionality previously associated with operating systems. For example, the non-protected part may allow the user to control the paging scheme, which has traditionally been viewed as an integral part of the operating system.

Similarly, in early operating systems, the command language interpreter was an integral part of the operating system. Today, it is viewed as just another utility program. This allows, for example each user to have a personalized version of the interpreter. On many UNIX systems, there are atleast three different such interpreters.

Database : Database represent another class of large software systems whose development has influenced software engineering through the discovery of new design techniques. Perhaps, the most important influence of the database field on software engineering is through the notion of data-indigence which is yet another instance of the separation of specification from implementation. The database allows applications to be written with that user data without worrying about the underlying representation of the data.

Another interring impact of database technology on software engineering is that it allows database systems to be used as components of large software systems. Since, databases have solved the many problems associated with the management of concurrent access to large amounts of information by multiple users, there is no need to reinvent these solutions, when we are building a software systems - we can simply use an existing database system as a component.

One interesting influence of software engineering on data-base technology has its roots in early attempts to use databases to support software development environments. This experience showed that traditional database technology was incapable of dealing with the problems posed by software engineering processes

For example, the following requirements are not handled well by traditional data bases : strong large structured objects such as source programs or user manuals; storing large unstructured objects such as object code and executable code; maintaining different versions of the same object and strong objects such as a product with many large structured and unstructured fields such as source code, object code and a user manual.

There is presently considerable work going on in the database area to address such problems, ranging from introducing new models for databases to adapting current data-base models.

Artificial Intelligence : Artificial intelligence is another field that has exerted influence on software engineering. The technique supported by artificial intelligence include the use of logic in both software specifications and programming languages.

The logic orientation seems to be filling the gap between specification and implementation by raising the level of implementation language higher than before. The logic approach to specification and programming is also called declarative. The idea is that we declare the specifications or requirements rather than specifying them procedurally; the declarative description is then executable. Logic programming languages such as PROLOG help us follow this methodology.

Software engineering techniques have been used in newer artificial intelligence systems - for example systems. These systems are modularised, with a clear separation between the facts known by the expert systems and the rules used by the systems for processing the facts - for example, a rule to decide on a course of action. The separation has enabled the building and commercial availability of expert system shells that include the rules only. A user can apply the shell to an application of interest in supplying application-specific facts. The idea is that the expertise about the application is provided by the user and the general principles of how to apply expertise to any problem are provided by the shell.

A different kind of symbiosis is currently taking place at the intersection of software engineering and artificial intelligence. Techniques of artificial intelligence are being applied to improve the software engineering tasks. For example, programming assistant are being developed to act as consultants to the programmer, watching for common programming idioms or the system requirements. Such assistants are also being developed to help in the testing activities of the software development, to debug the software.

The problem of providing interfaces for non-expert users - for example, through the use of natural languages - was first attacked by artificial intelligence. Cognitive models were also undertaken to model the user. These works have influenced the very active area of user-interface design in software engineering.

Unit - II

Requirements Analysis

2

Q1. What are the steps for Requirement Analysis?

OR

Explain the requirement analysis task.

Ans. **Steps for Requirement Analysis :**

Following are the steps that are performed during Requirement Analysis phase :

1. **Requirement Elicitation :** It is the activity during which software requirements are discovered, articulated and revealed from stakeholders or derived from system requirements. Sources are documentation, customers, end-users, domain specialist etc. it ask customers, the relevant questions in understanding the problem, analyse it, confirm their understanding and create requirement statements.

2. **Requirement Analysis and Negotiation :** Analyse, categorizes requirement and organises them into related subsets, explores each requirement in relation to other, examine requirements for consistency, omissions and ambiguity and ranks the requirements based on needs of customer/users.

3. **Requirement Specification :** Specification can be written document, graphical model, mathematical model, collection of usage scenarios, prototype or combination of these. Standard templates can also be developed for system specification in a consistent and understandable manner.

4. **System Modeling :** To fully specify what is to be built, we need a meaningful model of application, which is built in this step.

5. **Requirement Validation :** The objective is to certify that the SRS document is an acceptable document of the system to be implemented. In this, the requirement errors are fixed. The various techniques are plan the review, distribute SRS document, read it, organise the review meeting, follow-up actions, revise SRS document.

6. **Requirement Management :** It is the systematic approach for eliciting, organising and documenting the requirements of the system and process that establishes and maintains the agreement between customer and project team on the changing requirements of the system.

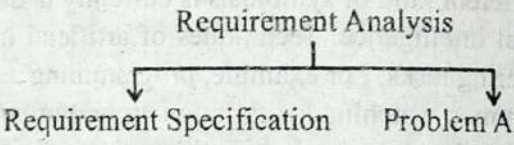
It is done in order to understand the problem, the software system is to solve. The problem could be automating.

- ⇨ An existing manual process.
- ⇨ Developing a new automated system.
- ⇨ Combination of the above two.

The emphasis in requirement analysis is on identifying "what" is needed from the system, not "how" the system will work.

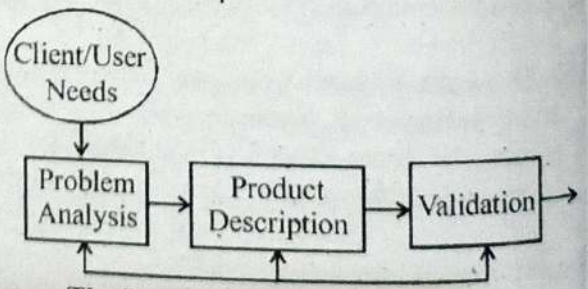
Q2. Explain various activities of Requirement Analysis. Also give three basic approaches to problem analysis.

Ans. Two parties involve in software development i.e. Client and Developer. The developer has to develop the system and satisfy the need of customers. There are basically two major activities :



1. The Requirement Specification :

It is the sequence of activity that need to be performed in the requirements phase and that cultimate in producing a high-quality document containing the software requirements. The system and its requirements for a complex system, this is a hard task and the time-tested method of "divide and conquer" i.e. decomposing the problem or system into the smaller parts and then understanding the parts and their relationships.



The Requirement Analysis Process

2. The Problem Analysis :

The basic aim of problem analysis is to obtain a clear understanding of the needs of the clients and the users, what exactly is desired from the software and what the constraints of the solution are?

Analysis leads to the actual specification.

Analysis involve some fact-finding techniques such as :

1. Interviewing : It is a face to face interpersonal meeting designed to identify relations and capture information. It is an art of arranging the interview, setting the stage, avoiding arguments and evaluating the outcome.

2. On-site Observation : It is to get close to the real system being studied. Observation helps the analyst in detecting problems that exists in the current system. Limitation of observation is the difficulty in observing attitudes and motivation and many unproductive hours that often are spent in observing one-time activities.

3. Questionnaires : It is a self-administrated tool that is more economical and requires less skills to administer than the interview. It examines the large number of respondents at the same time, provides standardized wording and interactions and places less pressure for response.

4. Meeting, Group-discussion etc. through which the requirements are gathered.

As analysis is the activity that feeds information to the specific activity; it is essential that during analysis, a complete and consistent set of specifications emerge for the system.

There are three basic approaches to Problem Analysis :

1. **Informal approaches based on structured communication :** In informal approach, don't use any "methodology" for problem analysis ; the analyst relies on his experience and use questionnaires, forms, Interviews etc.
2. **Conceptual Model :** A formal model is built for Problems such approaches are the principle of partitioning for building the model, it produces some formal structures representing some aspects of the problem.
3. **Prototyping Model :** The analyst understood problem through the feedback from the users and clients

Q3. What is requirement engineering ? What are various functions/task of requirement engineering ?

Ans. Requirement Engineering :

Requirement engineering provides the appropriate mechanism for understanding what the customer wants, analyzing need, accessing feasibility, negotiating a reasonable solution, specifying the solution unambiguously, validating the specification and managing the requirements as they are transformed into an operational system. The requirement engineering process is accomplished through the execution of seven distinct functions :

1. Inception : In some cases, a casual conversation is all that is needed to precipitate a major software engineering effort. But in general, most projects begin, when a business need is identified or a potential new market or service is discovered. Stakeholders from the business community defines a business case for the idea, try to identify the breadth and depth of the market, do a rough feasibility analysis and identify a working descriptive of the project's scope.

The intent is to establish a basic understanding of the problem, the people who want a solution, the nature of the solution that is desired and the effectiveness of preliminary communication and collaboration between the customer and developer.

2. Elicitation : It certainly seems simple enough—ask the customer, user and the others what the objectives for the system or product are; what is to be accomplished, how the system or product fits into the need of the business and finally, how the system or product is to be used on a day-to-day basis.

Christel & Kang identify a number of problems that helps us to understand why elicitation is difficult :

(a) **Problem of scope :** The boundary of the system is ill-defined or the customers/users specify unnecessary technical details that may confuse rather than clarity, overall system objectives.

(b) **Problem of understanding :** Users are not sure what they need due to poor understanding of the capabilities and limitations.

(c) **Problem of volatility :** The requirements change over time.

the customer during inception and elicitation is expanded and refined during elaboration. This activity focuses on developing a refined technical model of software functions, features and constraints.

4. Negotiation : It is not unusual for customers to ask for more than can be achieved, given limited business resources. It is also relatively common for different customers or users to propose conflicting requirements, arguing that their vision is "essential for our special needs."

The engineer must reconcile these conflicts through a process of "Negotiation".

5. Specification : The specification is the final work product produced by the requirement engineer. It serves as the foundation for subsequent software engineering activities. It describes the function and performance of a computer-based system and the constraints that will govern its development.

6. Validation : It examines the specification to ensure that all software requirements have been stated unambiguously, that inconsistencies, omissions and errors have been detected and corrected.

7. Management : It is a set of activities that help the project team identify, control and track requirements and changes to requirements at any time as the project proceeds.

Q4. What do you mean by system analysis? Who are the system analyst? Explain the function of system analyst? What are the duties and responsibilities of a system analyst?

Ans. System Analysis :

Systems Analysis is the study of sets of interacting entities, including computer systems analysis. This field is closely related to requirements analysis or operations research. It is also "an explicit formal inquiry" carried out to help someone (referred to as the decision maker) identify a better course of action and make a better decision than he might otherwise have made.

System Analyst :

A systems analyst researches problems, plans solutions, recommends software and systems, at least at the functional level and coordinates development to meet business or other requirements. Although they may be familiar with a variety of programming languages, operating systems and computer hardware platforms, they do not normally involve themselves in the actual hardware or software development. Because they often write user requests into technical specifications, the systems analysts are the **liaisons** between vendors and information technology professionals. They may be responsible for developing cost analysis, design considerations, staff impact amelioration and implementation time-lines.

Functions Performed by System Analyst :

A systems analyst may perform following functions to identify, understand and plan for organizational and human impacts of planned systems and ensure that new technical requirements are properly integrated with existing processes and skill sets as:

- Plan a system flow from the ground up.
- Interact with internal users and customers to learn and document requirements that are then used to produce business requirements documents.
- Write technical requirements from a critical phase.
- Interact with designers to understand software limitations.
- Help programmers during system development, provide use cases, flowcharts or even database design.
- Perform system testing.
- Deploy the completed system.
- Document requirements or contribute to user manuals.
- Whenever a development process is conducted the system analyst is responsible for designing components and providing that information to the developer.

The Role of System Analyst :

System analyst is the person who is responsible for the analysis of the system. He is the person who is responsible to deal with the customers and gathe his requirements. He gathers the information abou the key requirements of customer by askin questions, writing down the answers, organizin mock interview with the customers etc. He prepare the analysis model to achieve three primary objectives

- To describe what the customer requires.
- To establish a basis for creation of softwar design.
- To define a set of requirements that can b validated once the software is built.

He collaborates a set of requirements that can be established during earlier requirement engineering tasks and build model that depict user scenarios, functional activities, problem classes and their relationships, system and classes behaviour and flow of data as it transformed. In early stages, he focuses on what and not how. He decides what objects must system manipulate, what function must the system performs, what behaviour does the system exhibit, what interfaces are defined and what constraints to apply. If customer is precisely unsure of what he wants, then analyzer decides to apply specific approach to accomplish function and performance.

- Design, develop, modify and implement middleware software engineering components on application servers used for web or n-tier applications.
- Develop and evaluate technical design, architecture and framework.
- Prepare technical specifications as well as documentation for software engineering analysis.
- Contribute to entire software development lifecycle inclusive of collection, design, analysis, development technologies and version control etc.
- Conduct application testing of parts, modifications and entire systems.
- Prepare and execute training plus documentation for software engineering applications.
- Support task plans, estimation, schedules and staffing.
- Direct and review software engineers plus consultants work on project teams.
- Develop and debug C language firmware for entire T.I. Processors utilizing Code Composer Studio.
- Design and develop technical solutions on basis of functional specifications.
- Design code, install, maintain, unit test and retrofit software programs.
- Evaluate code non-conformities along with prepared code enhancements.
- Develop documents for use through internal and external clients.
- Provide technical support and information to various team members.
- Review software design ideas and present work estimates.

Q5. What are Requirement Elicitation Techniques?

OR

What are the communication principles of requirement?

Ans. Requirement Elicitation :

Requirement Elicitation is the activity during which software requirements are discovered, articulated and revealed from stakeholders or derived from system requirements. This is perhaps the most difficult, most critical, most error-prone and most communication intensive aspect of software development. Sources of requirement elicitation may be system requirements documentation, customers, end-users, domain specialist or market analysts.

Requirement elicitation includes the following activities :

1. Knowledge of the general area where the systems are applied.
2. The details of the specific customer problem where the system will be applied must be understood.
3. Interaction of the system with external environment.
4. Detailed investigation of user needs.
5. Define the constraints for system development.

Techniques for Requirement Elicitation :

1. Interviews : Once a problem statement is received from a customer, than comes the stage, where the meeting with a customer is arranged. Interviews are one of the most popular techniques for understanding the problem domains, interviews may be open-ended or structured. In open-ended interview, there is no pre-set agenda. Context-free questions may be asked to understand the problem and to have an overview of the situation. In structured interview, agenda of a fairly open questions is prepared.

2. Brainstorming : It is a group technique that can be used during requirements elicitation to understand the requirements. It has become very popular and is being used by most of the companies. It promotes creative thinking, generating new ideas and provides platform to share views, getting expectations and difficulties of implementation.

3. Facilitated Application Specification Technique (FAST) : It is an approach in which joint team of customers and developers work together to

identify the problem, propose elements of solution, negotiate different approaches and prepare a specification for preliminary set of solution requirements. This process is continued till a consensus is reached :

- In fast meeting, each FAST attendant is asked to prepare a list of objects, services and a list of constraints.
- The objects list consists of all objects, objects produced by the system and objects that surround the system.
- The list of services contains all the required information that manipulates with the objects, etc.

4. Quality Function Development (QFD) : It is a requirement engineering approach that focuses on quality. It means producing technical requirements from the customer requirements in the production and development of a product. These stages include:

- Identifying the customer (stakeholders).
- Gathering high-level customer requirements.
- Constructing a set of system features that can satisfy customer needs.
- Creating a matrix to evaluate system features against satisfaction of customer needs.

5. Joint Application Development (JAD) : It is specially designed for the development of large computer system. The goal is to involve all stakeholders in the design phase of the product via highly structured and focused meetings. In the preliminary phase of JAD, the requirement engineering team is tasked with fact finding and information gathering. Typically, the output is, as applied to security requirement elicitation, as security goals and artifacts.

Q6. Write down the principles for requirement analysis modelling ?

Ans. Over the past three decades, a large number of analysis modelling methods have been developed. Investigators have identified analysis problems and their causes and have developed a variety of modelling notations and corresponding sets of heuristics to overcome them. Each analysis method has a unique point of view. All analysis method are related by a set of operational principles :

1. The information domain of a problem must the data that flow into the system (from end-users, other systems or external devices), the data that flow out of the system and the data stores that collect and organises persistent data objects.

2. The functions that the software performed must be defined : Software functions provide direct benefit to end-uses and also provide internal support for those features, that are user visible. Some functions transform data that flow into the system.

3. The behaviour of the software (as a consequence of external events) must be represented : The behaviour of computer software is driven by its interaction with the external behaviour. Input provided by end-users, control data provided by an external system, or monitoring data collected over a network all causes the software to behave in a specific way.

4. The models that depict information, function and behaviour must be partitioned in a manner that uncover details in a layered fashion : Analysis modeling is the first step in software engineering for problem solving. It allows the practitioners to better understand the problem and establishes a basic for the solution. A large, complex problem is divided into sub-problems, until each sub-problem is relatively easy to understand.

5. The analysis task should move from essential information toward implementation detail : Analysis modelling begins by describing the problem from the end-user's perspective. The "essence" of the problem is described without any consideration of how a solution will be implemented.

Q7. What are the methods for Requirement analysis? Explain in detail?

Ans. Methods for Requirement Analysis :

Any requirements analysis method combines a set of distinct heuristics and a unique notation to analyze information, function and behavior for a computer-based system. Through the application of the fundamental analysis principles, each method creates a model of the problem and the requirements for its solution. ***There are various methods for requirement analysis that are :***

1. Structured Analysis :

Structured Analysis (SA) in software engineering and its allied technique Structured Design (SD) ar

requirements into specifications and ultimately, computer programs, hardware configurations and related manual procedures. Structured analysis is part of a series of structured methods, that "represent a collection of analysis, design and programming techniques that were developed in response to the problems", faced by the software world from the 1960s to the 1980s. Structured Analysis views a system from the perspective of the data flowing through it. The function of the system is described by the processes that transform the data flows. Structured analysis takes advantage of information hiding through successive decomposition (or top down) analysis. The result of structured analysis is a set of related graphical diagrams, process descriptions and data definitions. They describe the transformations that need to take place and the data required to meet a system's functional requirements.

It consists of the following objects:

(a). Context diagram : Context diagrams are diagrams that represent the actors outside a system that could interact with that system. This diagram is the highest level view of a system, similar to Block Diagram showing a possibly software-based system as a whole and its inputs and outputs from/to external factors. System context diagram are related to Data Flow Diagram and they show the interactions between a system and other actors with which the system is designed to work. System context diagrams can be helpful in understanding the context in which the system will be part of software engineering.

(b). **Data dictionary :** A data dictionary or *database dictionary* is a file that defines the basic organization of a database. A database dictionary contains a list of all files in the database, the number of records in each file and the names and types of each data field. Most database management systems keep the data dictionary hidden from users to prevent them from accidentally destroying its contents. Data dictionaries do not contain any actual data from the database and have only bookkeeping information for managing it. Without a data dictionary, however, a database management system cannot access data from the database.

(c). **Data Flow Diagrams :** A Data Flow Diagram (DFD) is a graphical representation of the "flow" of data through an information system. It differs from the system flowchart, as it shows the flow of data through processes instead of computer hardware. Data flow diagrams were invented by Larry Constantine, developer of structured design based on Martin and Estrin's "data flow graph" model of computation.

It is common practice to draw a System Context Diagram first which shows the interaction between the system and outside entities. The DFD is designed to show how a system is divided into smaller portions and to highlight the flow of data between those parts. This context-level Data flow diagram is then "exploded" to show more detail of the system being modeled.

(d). Structure Chart : A Structure Chart (SC) is a chart, that shows the breakdown of the configuration system to the lowest manageable levels. This chart is used in structured programming to arrange the program modules in a tree structure. Each module is represented by a box which contains the name of the modules. The tree structure visualizes the relationships between the modules.

In structured analysis, structure charts are used to specify the high-level design, or architecture of a computer program. As a design tool, they aid the programmer in dividing and conquering a large software problem, that is, recursively breaking a problem down into parts that are small enough to be understood by a human brain. The process is called top-down design or functional decomposition. Programmers use a structure chart to build a program in a manner similar to how an architect uses a blueprint to build a house. In the design stage, the chart is drawn and used as a way for the client and the various software designers to communicate. During the actual building of the program (implementation), the chart is continually referred to as the **master-plan**.

(e). Structured Design : Structured Design (SD) is concerned with the development of modules and the synthesis of these modules in a so called "module hierarchy". In order to design optimal module structure and interfaces, two principles are crucial:

(i) ***Cohesion*** which is "concerned with the grouping of functionally related processes into a particular module".

(ii) ***Coupling*** relates to "the flow of information, or parameters, passed between modules. Optimal coupling reduces the interfaces of

modules, and the resulting complexity of the software".

(f). Structured Query Language : The structured query language (SQL) is a standardized language for querying information from a database. SQL was first introduced as a commercial database system in 1979 and has since been the favorite query language for database management systems running on minicomputers and mainframes. Increasingly, however, SQL is being supported by PC database systems because it supports distributed databases.

2. Object Oriented Analysis :

Object-oriented analysis and design (OOAD) is a software engineering approach that models a system as a group of interacting objects. Each object represents some entity of interest in the system being modeled and is characterized by its class, its state (data elements) and its behavior. Various models can be created to show the static structure, dynamic behavior and run-time deployment of these collaborating objects. There are a number of different notations for representing these models, such as the Unified Modeling Language (UML).

Object-oriented analysis (OOA) applies object-modeling techniques to analyze the functional requirements for a system. Object-oriented design (OOD) elaborates the analysis models to produce implementation specifications. OOA focuses on *what* the system does, OOD on *how* the system does it. It is of various forms :

(a). Object-oriented systems : An object-oriented system is composed of objects. The behavior of the system results from the collaboration of those objects. Collaboration between objects involves them sending messages to each other. Sending a message differs from calling a function in that, when a target object receives a message, it decides on its own what function to carry out to service that message. The same message may be implemented by many different functions, the one selected depending on the state of the target object. The implementation of "message sending" varies depending on the architecture of the system being modeled and the location of the objects being communicated with.

(b). Object-oriented analysis : Object-oriented analysis (OOA) is the process of analyzing a task (also known as a problem domain) to develop a conceptual model that can then be used to complete the task. A typical OOA model would describe computer software that could be used to satisfy a set of customer-defined requirements. During the analysis phase of problem-solving, the analyst might consider a written requirements statement, a formal vision document, or interviews with stakeholders or other interested parties. The task to be addressed might be divided into several subtasks (or domains), each representing a different business, technological, or other areas of interest. Each subtask would be analyzed separately. Implementation constraints, (e.g., concurrency, distribution, persistence, or how the system is to be built) are not considered during the analysis phase; rather they are addressed during object-oriented design (OOD). The conceptual model that results from OOA will typically consist of a set of use cases, one or more UML class diagrams and a number of interaction diagrams. It may also include some kind of user interface mock-up.

(c). Object-oriented design : During object-oriented design (OOD), a developer applies implementation constraints to the conceptual model produced in object-oriented analysis. Such constraints could include not only constraints imposed by the chosen architecture but also many non-functional technological or environmental – constraints such as transaction throughput, response time, run-time platform, development environment, or those inherent in the programming language. Concepts in the analysis model are mapped onto implementation classes and interfaces resulting in a model of the solution domain, i.e., a detailed description of *how* the system is to be built.

Q8. What are DFD's? Describe its symbols? Illustrate with a suitable example?

Ans. Data Flow Diagram (DFD) :

A data flow diagram is a graphical representation of the "flow" of data through an information system, modeling its *process* aspects. Often, they are a preliminary step used to create an overview of the system which can later be elaborated. DFDs can also be used for the visualization of data processing (structured design).

A DFD shows what kinds of information will be input to and output from the system, where the data will come from and go to and where the data will be

stored. It does not show information about the timing of processes, or information about whether processes will operate in sequence or in parallel (which is shown on a flowchart).

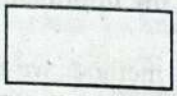
Components of DFD :

Following are the components or primitive symbols used for constructing DFDs :

• **Process :** A function or process or a bubble is represented using a circle. It transforms incoming data flows into outgoing data flows.

Process

• **External Entity :** A rectangle represents an external entity. It defines a source (originator) or destination of system data.

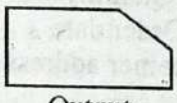


External Entity

• **Data Flows :** An arrow is used as a data flow symbol. It identifies data in motion. It represents data flow occurring between 2 processes or between external entity and a process in the direction of data flow arrow.

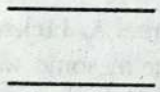
Data Flow

• **Output :** Output box is used when a hard copy is produced and the user of the copies cannot be clearly specified or there are several users of the output.



Output

• **Data Store :** Open box represents data stores. Data store represents a logical file, a data structure or a physical file or disc. Each data store is connected to a process by means of a data flow symbols.



Data Store

Q9. Explain the concept of data Structure Oriented Methods?

Ans. Data structure-oriented analysis methods represent software requirements by focusing on data structure, rather than data flow. Although each data structure-oriented method has a distinct approach and notation, all have some characteristics in common:

1. Each assists the analyst in identifying key *information objects* (also called *entities* or *items*) and *operations* (also call *actions* or *processes*).
2. Each assumes that the structure of information is hierarchical.
3. Each requires that the data structure be represented using the sequence, each provides a set of steps for mapping a hierarchical data structure into a program structure.

Like their flow-oriented counterparts, data structure-oriented analysis methods lay the foundation for software design. In every case, an analysis method may be extended to encompass architectural and procedural design for software.

Important Data Structure Oriented Analysis Methods :

1. Data Structured Systems Development :

Data Structured Systems Development (DSSD), also called the **Warnier-Orr methodology,** evolved from pioneering work on information domain analysis conducted by **J. D. Warnier**. Warnier developed a notation for representing information hierarchy using the three constructs for sequence, selection and repetition and demonstrated that the software structure could be derived directly from the data structure.

Ken Orr has extended **Warnier's** work to encompass a somewhat broader view of the information domain that has evolved into *Data Structured Systems Development.* DSSD considers information flow and functional characteristics as well as data hierarchy.

(a) Warnier Diagrams: This enables the analyst to represent information hierarchy in a compact manner. The information domain is analyzed and the hierarchical nature of the output is represented. To illustrate, let us consider an automated composition system used by a newspaper to prepare each day's edition. The general organization of the paper takes the following form.

Front Section :

- Headline news
- National news
- Local news

Editorial Section :

- Editorials columns
- Letters to the editor
- Satirical cartoon

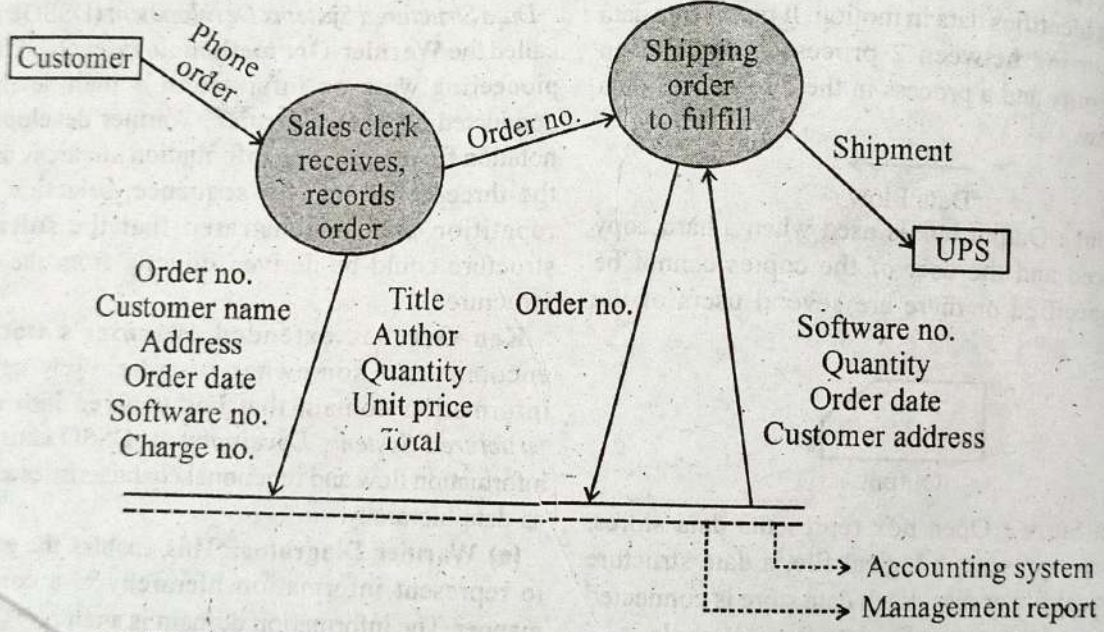
Second Section :

- Sports news
- Business news
- Classified

The Warnier diagram may be used to further partition the information domain by refining composite data items. The *exclusive-or* symbol (⊕) indicates a conditional occurrence (**selection**) of an information item; in this case, the business news section will contain either a management profile or a labour profile, but not both.

(b) The DSSD Approach : Rather than beginning analysis by examining information hierarchy, DSSD first examines the *application context,* that is, how data moves between producers and consumers of information from the perspective of one of the producers or consumers. Next, *application functions* are assessed with a Warnier-like representation that depicts information items and the processing that must be performed on them (this is similar in concept to the data flow diagram). Finally, application results are modeled using the Warnier diagram. Using this approach, DSSD encompasses all attributes of the information domain : data flow, content and structure.

To illustrate DSSD notation and to provide an overview of the analysis method, we present a simple example. A mail/phone order business, called *The Software Store,* sells personal computer software. A computer-based order processing system is to be specified for the business.



2. Jackson Syste... Development :

Jackson System Deve...lopment (JSD) evolved out of work conducted by **Michael A. Jackson** on information domain analysis and its re...lationship to program and system design. Similar in some ways to **Warnier's** approach and DSSD, JSD focuse... on models of the "real world" information domain.

To conduct JSD, the analyst applies the following steps:

1. ***Entity Action Step :*** Using an approach that is quite similar to the object oriented analysis technique, *entities* (people, objects or organizations that a system needs to produce or use information) and *actions* (the events that occur in the real world that affect entities) are identified.
2. ***Entity Structure Step :*** Actions that affect each entity are ordered by time and represented with *Jackson*

Diagrams (a tree-like notation).

3. ***Initial Modeling Step.*** Entities and actions are represented as a process model; connections between the model and the real world are defined.
4. ***Function Step :*** Functions that correspond to defined actions are specified.
5. ***System Timing Step :*** Process scheduling characteristics are assessed and specified.
6. ***Implementation Step :*** Hardware and software are specified as a design.

Q10. What is Formal Technique? Explain its advantages and disadvantages?

Ans. A **formal specification** is a specification based on a mathematical theory (for example, first-order logic with extensions for sets and other simple structures). A **formal technique** used to specify a hardware and/or a software system, verify whether a specification is realizable, verify whether an implementation satisfies its specification, proves properties of a system without necessarily running the system and so on the mathematical basis of a formal method is provided by its specification language. More precisely, a formal specification language consist of two sets **syn** and **sem**, and a relation **sat** between them. The set **syn** is called as **syntactic** domain, the set **sem** is called the **semantic** domain, and the relation **sat** is called **satisfaction** relation.

Syntactic Domain : The syntactic domain of a formal specification language consists of a alphabet of symbols and a set of formation rules to construct well-formed formulas from the alphabet. The well-formed formulas are used to specify a system.

Semantic Domain : Formal techniques can have considerably different semantic domains. Abstract data type specification languages are used to specify algebras, theories and programs. Programming languages are used to specify functions from input to output values. Concurrent and distributed system specification languages are used to specify state sequences, event sequences, state-transition sequences, synchronization trees, partial orders, state-machines, etc.

Satisfaction Relation : Given the model of a system, it is important to determine whether an element of the semantic domain satisfies the specification. This satisfaction is determined by using a homomorphism known as **semantic abstraction function**. The semantic abstraction function maps the elements of the semantic domain into equivalent classes. There can be different aspects of a system model, using different specification language.

Advantages and Disadvantages of Formal Specification:

- Formal specification requires a detailed and thorough study of the design. This study may be the greatest benefit of formal specification.
- Languages such as Z and VDM provide full first-order logic with set theory, consequently, they are powerful and expressive.
- We can use the formal specification to prove properties of the design.

In general, we cannot prove that a formal specification is correct, because the requirements are stated informally.

- Many programmers do not like formal notation (or may be afraid of it).
- Some things are hard to specify.

There is a trade-off between model-based specification languages, such as Z and VDM and property-based specification languages such as OBJ, Clear and ActOne. Paradoxically, the property-based languages are in some senses more abstract, but they are also executable.

V. Imp. → SRS

Q11. Explain about Software Requirement specification and its structure?

Ans. A **Software requirements specification** (SRS), a requirements specification for a software system, is a complete description of the behavior of a system to be developed and may include a set of use cases that describe interactions, the users will have with the software. In addition, it also contains non-functional requirements. Non-functional requirements impose constraints on the design or implementation (such as performance engineering requirements, quality standards, or design constraints). The software requirements specification document enlists all necessary requirements that are required for the project development. To derive the requirements, we need to have clear and thorough understanding of the products to be developed. This is prepared after detailed communications with the

project team and customer.

The SRS may be one of a contract deliverable Data Item Descriptions or have other form of organizationally-mandated content. An example of organization of an SRS is as follows :

- Introduction
- Purpose
- Definitions
- System overview
- References
- Overall description
- Product perspective
- System Interfaces
- User Interfaces
- Hardware interfaces
- Software interfaces
- Communication Interfaces
- Memory Constraints
- Operations
- Site Adaptation Requirements
- Product functions
- User characteristics
- Constraints, assumptions and dependencies
- Specific requirements
- External interface requirements
- Functional requirements
- Performance requirements
- Design constraints
- Standards Compliance
- Logical database requirement
- Software System attributes
- Reliability
- Availability
- Security
- Maintainability
- Portability
- Other requirements

Q12. What is Software Quality? Discuss software quality attributes.

OR

Write down the important qualities of software product and process.

Ans. Quality as "a characteristics or attribute of something". As an attribute of an item, quality refers to measurable characteristics – things we are able to compare to know standards such as length, colour, electrical properties etc.

The basic goal of software engineering is to produce quality software. The quality can be defined as those product features which meets the needs of the customers and thereby provide product satisfaction.

Software quality is conformance to explicitly stated functional and performance requirements, explicitly documented development standards and implicit characteristics that are expected of all professionally developed software.

When we examine an item based on its measurable characteristics, two kinds of quality may be encountered :

1. Quality of Design : It refers to the characteristics that designer specify for an item. The grade of materials, tolerance and performance specifications all contributes to the quality of design.

2. Quality of Conformance : It is the degree to which the design specifications are followed during manufacturing process. Again, the greater the degree of conformance, the higher is the level of quality of conformance.

Software Quality Attributes :

There are many important qualities of software products. Major software quality attributes are as follows :

1. Correctness : It establishes the equivalence between the software and its specification. The correctness of a software system refers to -

(i) Agreement of program code with specifications.

(ii) Independence of the actual application of the software system.

2. Reliability : Reliability of a software system is defined as the portability that this system fulfills function for a specified number of input trials under specified input conditions in a specified time interval.

3. User Friendliness : A software system is user friendly, if its human users find its easy to use. User friendliness of any software system is characterised by

(i) Adequacy : Factors for the requirements of adequacy are –

(a) The input required by the user should be limited to only, what is necessary.

(b) The results that a software system delivers should be output in a clear and well structured form and easy to interpret.

(c) Error messages must be provided in a form

that is comprehensible for the user.

(ii) Learnability : It is of a software system depends on –

(a) Design of user interfaces.

(b) Clarity and the simplicity of the user instructions.

(iii) Robustness : A program is robust, if it behaves "reasonably", even in circumstances that were not anticipated in the requirements specifications. Robustness reduces the impact of operational mistakes, erroneous input data and hardware errors.

4. Maintainability : Maintainability of a software system is defined as the suitability for debugging and ease of modification and extension of functionality.

5. Verifiability : A software system that is verifiable in its properties can be verifiable easily. For example, it is important to be able to verify the correctness or the performance of a system. Verifiability is usually a internal quality, although it sometimes becomes an external quality also. A common technique for improving verifiability is the use of software monitors.

6. Reusability : In product evolution, we modify a product to build a new version of the same product, in product reuse, we use it with minor changes to build another product. It appears to be more applicable to software components than to whole products, but it certainly seems possible to build products that are reusable.

7. Portability : It is said to be portable, if it can be easily made to work in different operating system environments, in different machines, with other software products, etc.

8. Efficiency : The amount of computing resources and code required by software to perform a function.

Q13. Explain the difficulties of measuring program maintainability. Describe why maintainability is not simply related to a single complexity metric?

Ans. The most important objectives of the development project should be to produce software that is easy to maintain and the process should be such that it ensures maintainability. When systems are installed, they generally are used for bug periods. However, this period of use brings with it, the need to continually maintain the system. Because of the use a system receives after it is fully implemented, analyst must take precautions to ensure that the need for maintenance is controlled through design and testing and the ability to perform, it is provided through proper design practices.

Few points which explain the maintenance are :

1. Software demand is drawing at a faster rate than supply. Many programmers are spending more time on system maintenance than a new development.
2. From 60 to 90% of the over all cost of software during the life of a system is spent on maintenance.

Maintenance Designs :

The keys for reducing the need for maintenance, while making it possible to do essential tasks more efficiently are these:

1. Managing the system engineering process effectively.
2. Assembling better systems documentation.
3. Making better use of existing tools and techniques.
4. More accurately defining user's requirements during system development.

Types of System Maintenance :

1. Corrective : Corrective maintenance of a software product becomes necessary to rectify the bugs observed, while the system is in use.

Activity : Emergency fixes, routine debugging.

2. Adaptive: A software product might need maintenance, when the customer needs the product to run on new platform, on new operating system or when they need the product to be interfaced with new hardware or software.

Activity : Accommodation of changes to data and files and to hardware and system software.

3. Perfective : A software product needs maintenance to support the new features that user want it to support, to change different functionalities of the system according to customer demands or to enhance the performance of the system.

Activity : User enhancement, improved documentation, recording for computation efficiency.

Gilb gives a number of maintainable matrix that relate to the effort spent during maintenance.

1. Problem recognition time.
2. Administrative delay time.

3. Maintenance tools collection time.
4. Problem analysis time.
5. Change specification time.
6. Active correction time.
7. Total recovery time, etc.

Metrics for maintenance have been explicitly designed and proposed. IEEE standard suggests a software maturity index that provides an indication of the stability of a software product. The software maturity index is specified as :

$$SMT = [MT - (Fa + Fc + Fd)] / MT$$

MT = The number of modules in the current release.

Fc = The number of modules in the current release that have been changed.

Fa = The number of modules in the current release that have been added.

Fd = The number of modules from the proceeding releases that were detected in the current release.

Q14. Define Abstraction in terms of Object-Oriented Design.

Ans. In the object-oriented design, complexity is managed using abstraction.

Abstraction is the elimination of the irrelevant and the amplification of the essentials.

Abstraction is the selective examination of certain aspects of a problem. In other words, the main purpose of abstraction is to consider only those aspects of the problem that are relevant for certain purpose and to suppress aspects of the problems that are not relevant for the given purpose. Thus, the abstraction mechanism allows us to represent a problem in a simpler way by omitting unimportant details. Many different abstractions of the same problem are possible depending on the purpose fc which they are made. Abstraction not only helps the development engineers to understand and appreciate the problem better, it also leads to better comprehension of the system design by the end-users and the maintenance team. Abstraction is supported at two levels in an object-oriented design (OOD) :

1. A class hierarchy can be viewed as defining an abstraction level, where each base class is an abstraction of its subclasses.
2. An object itself can be looked upon as a data abstraction entity, because it abstracts out the exact way in which various private data ite of the object are stored and provides only a s of well-defined methods to other objects access and manipulate these data items.

Abstraction is a powerful mechanism for reduci the complexity of software. In fact, it has been sho from data collected from several softwa development projects, that software productivity inversely proportional to software complexi Therefore, abstraction can be viewed as a way increasing software productivity.

Abstractions can be of **four** types :

1. **Data Abstraction :** It is a named collection data or we can define it as the combination attributes.
2. **Procedural Abstraction :** It is a nam sequence of instruction. This sequence ha specific & limited function.
3. **Control Abstraction :** In this abstraction program control mechanism is created with specifying internal details.
4. **Functional Abstraction :** In this type abstraction, a module is specified by function. It acts as the basis of partitioning

Q15. Define Modularity. Classify t various categories of module.

Ans. The software is divided into separately nam and addressable components that are called **Modules.** Creating such module bring the modula in software.

The main purpose of modularity is that it all the principles of separation of concerns to be app in two phases – when dealing with the details of e module in isolation (and ignoring details of o modules) and when dealing with the ove characteristics of all modules and their relation in order to integrate them into a coherent syste the two phases are executed in the order mentio then we say that the system is designed bottom the converse denotes top-down design.

Modules that may be created during prog modularization are :

1. **Process support modules :** In it, all functions and data items that are require support a particular business process grouped together.
2. **Data abstraction :** These are abstract

that are created by associating data with processing components.

3. **Functional modules :** In it, all the function that carried out similar or closely related tasks are grouped together.
4. **Hardware modules :** In it, all the functions which controls particular hardware are grouped together.

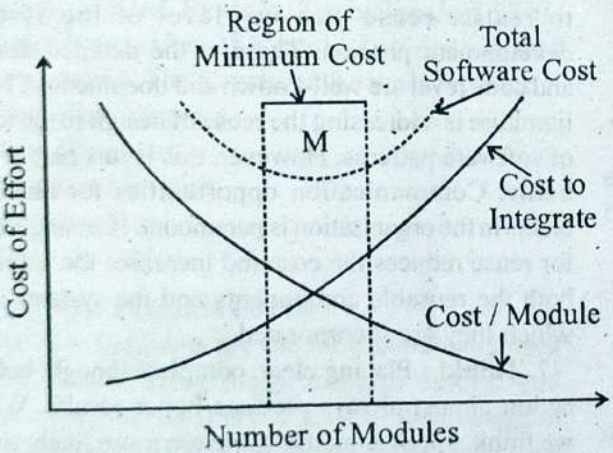
Classification of Modules :

A module can be classified into three types depending on activation mechanism.

1. **An incremental module :** It is activated by an interruption and can be interrupted by another interrupt during the execution prior to completion.
2. **A sequential module :** It is a module that is referenced by another module and without interruption of any external software.
3. **Parallel module :** Parallel modules are executed in parallel with another modules.

Modularity enhances design clarity, which in turn eases implementation, debugging, testing, documenting and maintenance of the software product.

It is possible to conclude that if we subdivide software indefinitely, the effort required to develop it will become negligibly small. But, unfortunately other forces comes into play.



Refer the figure, the effort (cost) to develop an individual software modules does decrease as the total number of modules increases. However, as the number of modules grow, the effort (cost) associated with integrating the modules also grows. This leads to a total cost or effort curve. There is a number M of modules that would result in minimum development cost.

Q16. What do you mean by Modular System ? What are the advantages of modular system ?

Ans. A system is considered modular, if it consists of discrete components so that each component can be implemented separately and a change to one component has minimal impact on other components. In other words, a module is a software component with parts at any level of abstraction. At the highest level of abstraction, there is an entire system of which it is composed.

Desirable properties of a modular system include:

- Each function in each abstraction has a single, well-defined purpose.
- Each function manipulates not more than one major data structure.
- Functions share global data selectively. It is easy to identify all routines that share a major data structure.
- Functions that manipulate instances of abstract data types are encapsulated with the data structure being manipulated.

Advantages :

- Modular systems are easier to understand and explain because their parts made are functionally independent.
- They are easier to document because each part can be documented as an independent unit.
- Programming individual modules is easier because the programmer can focus on just one small, simple problem rather than a long complex problem.
- Testing and debugging individual modules is easier because they can be dealt with isolation from the rest of the program.
- Bugs are easier to isolate and understand and they can be fixed without fear of introducing problems outside the module.

Q17. Write down the principles of Software Engineering.

Ans. The dictionary defines the word **principle** as "an important underlying law or assumption required in a system of thought".

Some focus on software engineering as a whole,

others consider it as a specific framework activity, for example customer communication and still other's focus on software engineering actions for example architectural design or technical task or write a usage scenario. Regardless of their level of focus, principles help us to establish a mindset for a software engineering practice.

David Hooker has proposed seven core principles that focus on software engineering practice as a whole.

1. The Reason it all Exists : A software system exists for one reason : to provide value to its users. All decisions should be made with this in mind, before specifying a system requirement, before noting a piece of system functionality, before determining the hardware platforms or development process. If our software doesn't do it all, leave out other principles & keep this one in mind.

2. KISS (Keep it Simple Stupid!) : Software design is not a haphazard process. There are many factors to consider in any design effort. All design should be as simple as possible but not simpler. This facilitates having a more easily understood & easily maintained system. This is not to say that features, even internal features, should be discarded in the more of simplicity. Indeed, the mare elegant design are usually the simple ones. Simple also doesn't mean "Quick & dirty". It takes lots of thought and work over multiple iterations to simplify.

3. Maintain the Vision : A clear vision is essential for the success of a software project. Without one, a project almost unfailingly ends up being of "two or more minds" about itself without conceptual integrity, a system threatens to become a patchwork of incompatible design held together by the wrong kind of screws.

Compromising the architectural vision of a software, system weakens & will eventually break even a well-designed system having a empower architect who can hold the vision & enforce compliance helps ensures a very successful software project.

4. What you Produce others will consume : Seldom is an industrial-strength software system constructed & used in a vacuum. In some way or other, someone else will use, maintain document or otherwise depend on being able to understand our system. So, always specify, design and implement knowing someone else have to understand what we are doing. The audience for any product of software development is potentially large. Specify with an eye to the users. Design, keeping the implementers in minds, code with concern for those who must maintain and extend the system, thus making their job easier adds value to the system.

5. Be open to the Future : A system with a long lifetime has more value. In today's computing environments, where specifications change on a moment's notice and hardware platforms are obsolete after few months, software lifetimes are typically measured in months instead of years. However, true "industrial-strength" software systems must endure far longer. To do this successfully, these systems must be ready to adapt to these and other changes, system that do this successfully are those that have been designed this way from the start. Never design ourself into a corner. Always ask "what if" and prepare for all possible answers by creating systems that solve the general problem, not just the specific one.

6. Plan ahead for Reuse : Reuse saves time and effort. Achieving a high-level reuse is arguably the hardest goal to accomplish in developing a software system with the reuse of code and oriented technology. However, the return on this investment is not automatic. To leverage the reuse possibilities that object-oriented programming provides requires forethought and planning. There are many techniques to realise reuse at every level of the system development process. These at the detailed design and code level are well-known and documented. New literature is addressing the reuse of design in the form of software patterns. However, this is just part of the bottle. Communication opportunities for reuse to others in the organisation is paramount. Planning ahead for reuse reduces the cost and increases the value of both the reusable components and the systems into which they are incorporated.

7. Think! : Placing clear, complete thought before action almost always produce better results. When we think, about something, we are more likely to do it right. We also gain knowledge about how to do it, right again. If we do think about something and still do it wrong, it becomes valuable experience. A side effect of thinking is learning to recognize, when we don't know something, a which has gone into a system, value comes out.

Unit - III

3 Structured Methodology

Q1. What is Structured Analysis? Discuss the tools available for Structured Analysis?

Ans. Structured Analysis :

Structured Analysis is the method for analysis modeling. It is a model building activity. ***It is a set of techniques and graphical tool that allow the analyst to develop new kind of system specification that are easily understandable to the user***. It usually restricts itself to the "what needs to be done" aspects of the problem and carefully avoids discussing the "how to do it" aspects.

The aim of structured analysis is to transform a textual problem description into a graphical model.

It is used to carry out the top-down decomposition of the set of high-level functions depicted in the problem description and to represent them graphically. It is based on the principles like **Top-Down Approach, Divide and Conquer** and **Graphical Representation of Analysis.**

Tools of Structured Analysis :

The Structured Analysis tools is used to build a new document called **System Specification**. This document provides the basis for design and implementation. The Structured Analysis tools are :

1. Data Flow Diagram
2. Data Dictionary
3. Structured English
4. Decision Tree
5. Decision Tables

1. Data Flow Diagram (DFD) :

Data flow Diagrams are the most commonly used way of documenting the process of current and required systems. DFD is a Graphical representation of a system's data and how the processes transform data. It is also known as **'Bubble Chart'**. Unlike flowchart, DFDs, do not give detailed description of modules, but generally describes a system's data and how the data interact with the system.

Following are the important points regarding DFD :

1. A DFD is a simple graphical formalism that can be used to represent a system in terms of input data to the system, various processing carried out on these data and the output generated by the system.
2. It is hierarchical graphical model that shows the different processing activities or functions that the system performs and the data interchange among these functions.
3. It represents the flow of data in a system.
4. It describes what data flow (logical) rather than how they are processed.
5. It enables the development of models of information domain and functional domain at the same time.

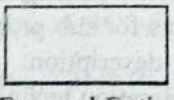
Components of DFD :

Following are the components or primitive symbols used for constructing DFDs :

- **Process :** A function or process or a bubble is represented using a circle. It transforms incoming data flows into outgoing data flows.

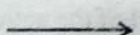
Process

- **External Entity :** A rectangle represents an external entity. It defines a source (originator) or destination of system data.



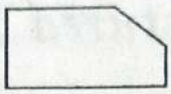
External Entity

- **Data Flows :** An arrow is used as a data flow symbol. It identifies data in motion. It represents data flow occurring between 2 processes or between external entity and a process in the direction of data flow arrow.



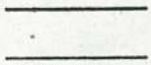
Data Flow

• **Output :** Output box is used when a hard copy is produced and the user of the copies cannot be cle specified or there are several users of the output.



Output

• **Data Store :** Open box represents data stores. Data store represents a logical file, a data structure physical file or disc. Each data store is connected to a process by means of a data flow symbols.



Data Store

2. Data Dictionary : Data Dictionary has been defined as an ordered listing of all data elements that pertinent to the system with precise, rigorous definitions so that both Users and System Analyst(s) will h a common understanding of Inputs, Outputs, Components of stores and even intermediate calculations.

Following are the important points in this regard :

1. It is a structured (control) repository of data about data.
2. It is a list of terms with their definitions and description for all data items and data stores of a syst
3. It contains definitions of all data objects consumed or produced by the software.
4. It just give a simple description of the data.

Importance of Data Dictionary :

Data dictionary is an important tool for structured analysis as it offers following advantages :

1. It is a valuable reference for designing the system. It is used to build the database and write progr during design phase.
2. It assists in communicating meanings of different elements, terms and procedures.
3. It facilitates analysis in determining additions and changes in the system.
4. It helps the analyst to record the details of each element and data structure.
5. It is used to locate errors in the system descriptions.
6. It is also a useful reference document during implementation of the system.

3. Structured English : Structured English is a popular tool for defining process specifications, becaus language constructs are English like and easy to understand. Structured English is easy to relate wi popular programming language tool – pseudo code.

Structured English is used to document the logic of the processes. Structured English is "**English structure**". Structured English borrows heavily from structured programming. It uses logical construc and imperative sentences designed to carry out instructions for action.

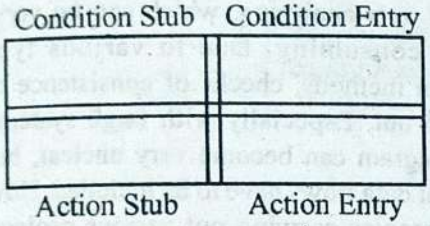
Syntax of Structured English include :

(a). Constructs for sub program definition.
(b). Interface description.
(c). Data declaration techniques for block structuring.
(d). Condition construct.
(e). Repetition construct.
(f). Input/Output construct.

4. Decision Tree : It gives a graphic view of the processing logic involved in decision making and corresponding action taken. It is a network type chart that is equivalent to a decision table. It describes log rules, showing all of the action that results from various combinations of conditions. A decision tree prov a very easily understood picture, since the branches can be read to show how all of the major and m logical components go together. It helps to show the paths that are possible in a design following an actio decision by the user.

5. Decision Table :

It provide a mechanism for specifying complex decision logic. It is formal, table-based notation that can also be automatically processed to check for qualities like completeness and lack of ambiguity. It is a table of contingencies for defining a problem and the action to be taken. It is a single regression table of the relationship between condition and action. A decision table causes the analyst to look at every possible combination of condition which he might omit. The decision table can only represent a single decision to select the appropriate action for execution.



1. **Condition Stub Quadrant :** Condition stub quadrant is used to set forth in question form, the condition that may exist.
2. **Action Stub Quadrant :** Action stub quadrant outlines in narrative form, the action to be taken to meet each condition.
3. **Condition Entry Quadrant :** Condition entry quadrant provide answers to question asked in condition stub quadrant.
4. **Action Entry Quadrant :** Action entry quadrant is used to indicate the appropriate action resulting from answers to conditions in condition entry quadrant.

Q2. What is structured design methodology? Explain the various design principles?

Ans. Structured Design Methodology :

Structured design is a data-flow based methodology. The approach begins with a system specification that identifies inputs and outputs and describes the functional aspects of the system. The system specifications are then used as a basis for the graphic representation-data flow diagram of data and processes. From the DFD, the next step is the definition of the modules and their relationships to one another in a form called **structure chart** using a data dictionary and other structured tools.

Structure design partitions a program into small independent modules. It is an attempt to minimize complexity and make a problem manageable by sub-dividing it into smaller segments which is called **modularization or decomposition.**

Design Principles :

The design process needs to follow certain principles and practices in order to be effective. ***These principles states :***

- A design should clearly be verifiable, complete, traceable, consistent, efficient and simple.
- **The design architectural structure should not suffer from "lack of vision" :** A good design plan must consider all the possible design approaches, judging each against system requirements, availability of resources and the identified system constraints.
- **An ideal design process should not reinvent the wheel :** If reusable design components are available within a system. Then, they must be used, instead of wasting precious time and efforts involved in recreating them.
- **The design should exhibit uniformity :** The design process should follow the design standards for coding, presentation etc.
- **Design is not coding and coding is not design:** The design decision made at the coding level address the small implementation details that enable the procedural design to be coded.
- **The design must be readable :** It should be an understandable guide for those who generate the design code and also for those who list and maintain the system thereafter.
- **The design should be reviewed to minimise conceptual errors :** The designer must necessarily check all the conceptual elements of design such as ambiguity, omission, and see that inconsistencies have been taken care of before worrying about the syntax of the design model.

Q3. What are the various techniques of structured methodology? Discuss its advantages and disadvantages?

Ans. The three most important techniques that are used in Structured Systems Analysis & Design Method (SSADM) are :

1. Logical Data Modeling : The process of identifying, modeling and documenting the data requirements of the system being designed. The result is a data model containing **entities** (things about

which a business needs to record information), **attributes** (facts about the entities) and **relationships** (associations between the entities).

2. Data Flow Modeling : The process of identifying, modeling and documenting how data moves around an information system. Data Flow Modeling examines **processes** (activities that transform data from one form to another), **data stores** (the holding areas for data), **external entities** (what sends data into a system or receives data from a system) and **data flows** (routes by which data can flow).

3. Entity Event Modeling : A two-stranded process Entity Behavior Modeling, identifying, modeling and documenting the events that affect each entity and the sequence (or life history) in which these events occur and Event Modeling designing for each event, the process to coordinate entity life histories.

Advantages :

1. **Timelines:** Theoretically, SSADM allows one to plan, manage and control a project well. These points are essential to deliver the product on time.
2. **Usability:** Within SSADM, special emphasis is put on the analysis of user needs. Simultaneously, the system model is developed and a comprehensive demand analysis is carried out. Both are tried to see, if they are well suited to each other.
3. **Respond to changes in the business environment:** As in SSADM, documentation of the project's progress is taken very seriously, issues like business objectives and business needs are considered, while the project is being developed. This offers the possibility to tailor the planning of the project to the actual requirements of the business.
4. **Effective use of skills:** SSADM does not require very special skills and can easily be taught to the staff. Normally, common modelling and diagramming tools are used. Commercial CASE tools are also offered in order to be able to set up SSADM easily.
5. **Better quality:** SSADM reduces the error rate of IS by defining a certain quality level in the beginning and constantly checking the system.
6. **Improvement of productivity:** By encouraging on-time delivery, meeting business requirements, ensuring better quality, using human resources effectively as well as trying to avoid bureaucracy, SSADM improves the overall productivity of the specific project and the company.
7. **Cuts costs:** SSADM separates the logical and the physical systems design. So the system does not have to be implemented again with new hardware or software.

Disadvantages :

SSADM puts special emphasis on the analysis of the system and its documentation. This causes the danger of over-analysing, which can be very time and cost consuming. Due to various types of description methods, checks of consistence cannot be carried out. Especially with large systems, the outline diagram can become very unclear, because all relevant data flows have to be included. However, large companies carrying out various projects can profit from the fact that SSADM gives the possibility to reuse certain techniques and tools for other projects. This reduces cost and time spent enormously in the long run. So, the danger of spending too much money on analysis can be compensated by the reuse of the developed systems and experience gained.

Q4. What developmental activities are carried out during structured design?

Ans. Several developmental activities are carried out during structured design. They are :

1. Database design : This activity deals with the design of the physical database. A key is to determine how the access paths are to be implemented. A physical path is derived from a logical path. It may be implemented by pointer chains, or other mechanisms.

2. Program design : In conjunction with database design, is a decision on the programming language to be used and the flowcharting, coding, and debugging prior to conversion.

3. System and program test preparation : Each aspect of the system has a separate test requirement. System testing is done after all programming and testing are completed. The test cases cover every aspect of the candidate system-actual operations, user interface and so on. System and program test requirements become a part of the design

specifications- a perquisite to implementation.

4. **System interface specification :** This phase specifies for the user, how information should enter and leave the system. The designer offers the user various options. By the end of the design, formats have to be agreed upon, so that machine-machine and human-machine protocols are well-defined prior to implementation.

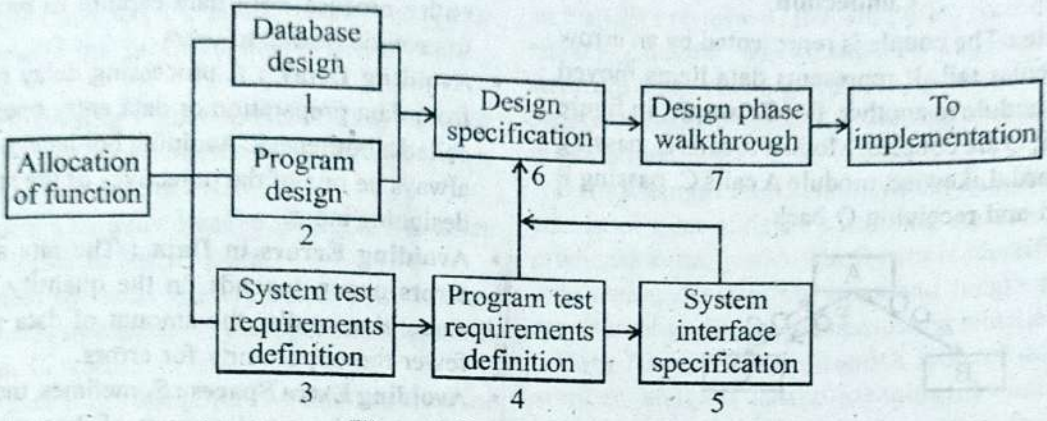


Figure : System Design Activities

Q5. Explain the process of structured design.

Ans. **Structured Design :**

The approach begins with a specification that identifies inputs and outputs and describes the functional aspects of the system. The system specifications are used as a basis for the graphic representation (i.e. DFD) of the data flows and processes. After DFD, modules are defined and their relationship to one another in a form called 'structure chart' using a data dictionary and other structured costs.

(a) **Data Flow Diagram (DFD) :** It is a graphical representation of system in terms of input data to the system, various processing carried out on these data and the output generated by the system. It represents system requirements in a graphical form and identify major transformation that will become programs in system design.

(b) **Data Dictionary :** It is a structured repository of data about data. It is a set of rigorous definition of all DFDs, data elements and data structures.

(c) **Decision Table :** It is a tabular technique for describing logical rules. It is a table of contingencies for defining a problem and action to be taken. It is a single representation of the relationship between condition and action.

(d) **Decision Trees :** It is a network type chart, that is equivalent to a decision table. It helps to show the path, that are possible in a design following an action or decision by the user. It describes logical rules showing all of the actions, that result from various combinations of the conditions.

(e) **Structured English :** It uses logical construction and imperative sentences designed to carry out instruction for action. It uses standard logical terms such as IF, THEN and ELSE.

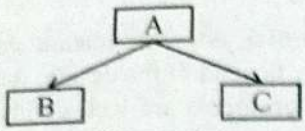
Q6. Briefly explain documentation tool for structured design.

Ans. The documentation tool for structured design is the **hierarchy** or **structure chart**. It is a graphical tool for representing hierarchy, and it has three elements :

1. **Module :** The module is represented by a rectangle with a name. It is a contiguous set of statements.

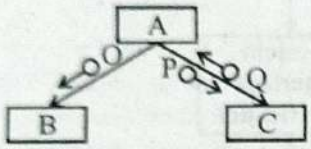
A Module

2. **Connection :** The connection is represented by a vector linking two modules. It usually means one module has called another module. **For Example,** module A calls module B; it also calls module C.



Connection

3. Couple : The couple is represented by an arrow with a circular tail. It represents data items moved from one module to another. **For Example,** in figure below O, P, Q are couples. Module A calls B, passing O downward. Likewise, module A calls C, passing P downward and receiving Q back.



Q7. Write a detail note on Input Design.

Ans. Input Design :

Input design is the process of converting user-originated inputs to a computer-based format. Input design is the link that ties the information system into the world of users. Input design consist of :

1. Developing specifications and procedures for data preparation.
2. Those steps necessary to put transaction data into a usable form for processing and data entry.
3. The activity of putting the data into the computer for processing.

Input design features can ensure the reliability of the system and produce results from accurate data.

Goals of Input Design :

(i). To make data entry easy,
(ii). To make data entry logical,
(iii). To make data entry free from errors.

Inaccurate input data are the most common cause of errors in data processing. Errors entered by data entry operators can be controlled by input design. "Input Design" is the process of converting user-originated inputs to a computer-based formats.

In entering data, operators need to know the following :

(i). The allocated space for each field,
(ii). Field sequence, which must match that in the source document,
(iii). The format in which data-fields are entered.

Objective of Input Design :

The five objectives for guiding the design of input are :

- **Controlling Amount of Input :** By reducing the input requirements, the analyst can speed up the entire process from data capture to processing to provide results to users.
- **Avoiding Delay :** A processing delay resulting from data preparation or data entry operation is called a bottleneck. Avoiding bottlenecks should always be one of the objectives of the analyst in designing input.
- **Avoiding Errors in Data :** The rate at which errors occur depends on the quantity of data, since, the smaller the amount of data to input, fewer the opportunity for errors.
- **Avoiding Extra Spaces :** Sometimes, the volume of transactions and amount of data preparation or data entry jobs resulting from them cannot be controlled. When the volume of transactions cannot be reduced, the analyst must be sure the process is as efficient as possible.
- **Keeping the Process Simple :** The best-designed system fits the people, who will use it in a way that is comfortable for them while providing error-control methods that management must have.

Steps of Input Design :

The steps in input design are :

1. Input Data : The goal of designing input data is to make data entry as easy, logical and free from errors as possible. In entering data, operators need to know the following :

⇨ The allocated space for each field.
⇨ Field sequence.
⇨ The format in which data are entered.

When we approach to input data design, we design the source documents that capture the data.

2. Source Documents : Source data are captured initially on original paper or a source document. It is a source document. The next step is to encode that source document they are entered into the system. It could be entered with the help of punch card, from diskettes, or through keyboard. A source document may or may not be retained in the candidate system.

3. Input Media & Device : Source data are input into system in variety of ways.

(a). Punch Card : Use a keypunch to copy data from source document, it is either 80 or 96

column under data arranged in sequential & logical order.

(b). **Key-to-diskette** : It is modelled after keypunch process. A diskette replaces the card and stores upto 325000 character of data.

(c). MICR translates the special fonts printed in magnetic ink or check into direct computer input.

(d). Mark sensing reader automatically convert pencil mark in pre-determined locations on a card to punched holes.

(e). Optical Character Recognition (OCR) reader are similar to NICR reader.

(f). Optical bar code readers detect combination of marks that represent data, which codes retail items in stores.

(g). Cathode Ray Tube (CRT) screens are used for online data entry. CRT screen display 20, 40 & 80 characters simultaneously.

Q8. Write a detail note on Output Design.

Ans. Output Design :

Computer output is the most important and direct source of information to the user. Efficient, intelligible output should improve the system's relationships with the user and help in decision making.

Goals of Output Design :

(i). It should be Efficient,
(ii). It should be intelligible,
(iii). It should improve system's relationship with user,
(iv). It should help in decision-making,

A major form of "output design" is a hard copy from the printer. Printouts should be designed around the output requirements of the user. The following media-based devices are available for providing computer based output :

(i). MICR Readers,
(ii). Line, Matrix and daisy-wheel Printers,
(iii).Computer Output Microfilm (COM),
(iv). CRT Screen display,
(v). Graph Plotters,
(vi). Audio Response

In addition to deciding on output device, the system analyst must consider the print format and the editing for the final printout.

Q9. How data structure is related to data elements to process data flows and data stores?

Ans. Since, data descriptions are used repeatedly throughout an investigation and later during design, an easy to use format, that simplifies recording and retracing details, when needed is suggested. Data structure dilates the organization, methods of access, degree of associativity and processing alternatives for information.

Data Element : Each entry in the data dictionary consist of a set of detail describing the data used or produced in the system. Each item is identified by a data name, description, alias and length and has specific value, that are permissible for it in the system.

Data Name : To distinguish item of data from another, analysts assign meaningful names. The names are used to refer to each element, throughout the entire system development process. Therefore, meaningful and understandable data names should be given. Some organizations prescribe standards for developing data names. A common standard specifies that data names should not exceeds 30 characters and should contain no blanks.

Data Description : The data description briefly states what data item represents in the system. **For example,** description for Date_of_invoice indicates that, it is the date on which the statement was prepared.

Data descriptions should be written with the assumption that the person, who will be reading the description does not know anything about the system. Jargon or special terms should be avoided, all words should be understandable to the reader.

Aliases : Often, the same data item can be referred by different names, depending on who is using the data, the additional names is called aliases.

On the other hand, when data are augmented through processing and reflected in the data names. The data item are no longer aliases. **For example**, if invoice become **Approved Invoice,** which in turn becomes **Audited Invoice,** the items are not aliases for each other. The processing that occurs in the form of approving the invoice or auditing the invoice changes the data by adding additional details. In a sense the process adds value, since the additional details add to, what is known about the invoice.

A meaningful data dictionary will include all aliases.

Length : When system design features are developed later in the systems development process, it i important to know the amount of space needed for each data item. Analysts can capture these details, whe developing data flow diagrams; length identifies the number of space required but without concern for hov they are stored. If a customer name can be upto 30 characters long, when written on a sales form, the dat dictionary entry should show a length of 30.

Data Values : In some processes, only specific data values are permissible. For example, purchase orde numbers in many organizations are often given a one letter prefix to indicate the department or origin. Thi details belongs in the data dictionary description of department number. The following table shows th purchase order number prefixes for one company :

Prefix	Department
A	Accounting Department
B	Purchasing Department
M	Manufacturing Division
P	Personnel Section
S	Sales Department
T	Transportation

The system may later be designed so that the above prefixes are the only ones, that are acceptable inpu

If data values are restricted to a specific range, that information belongs in the data dictionary entry for th data item as well. **For example,** if the price of any product sold by a firm does not exceed $ 25, tha annotation belongs in the data dictionary similarly, the fact that all purchase order numbers should have fiv digit identification numbers should also be stated. These details help the analysts at the time, when the design systems control.

Data Flows and Data Stores : A complete explanation of all elements in the data flow diagram include description of each data flow, data structure and process.

All the details related to data flow, data structure and process are captures in sample data flow form. Ea data flow is named and briefly described. The names and identification of processes associated with this da flow are also included. To complete the definition of this data flow, the appropriate data structure are liste It is not necessary to define the contents of the data structure, that is handled elsewhere in the data dictiona

The data store entry describes data flows to and from the data store as well as the volume, an indication how busy this position of the system.

Example :

Student data :

Name + street address + city + state + postal code + Telephone number + [student number social security number] + {course number + course name + credits + section number + time + d +instructor}+ term + year+advisor

Name : First Name + (middle initial) + Last Name

Data Structure for Student Data

Q10. Explain the current generations of software development tools?

Ans. Programming languages have been classified into several **programming language generatio** Historically, this classification was used to indicate increasing power of programming styles.

1. First Generation :

A first-generation programming language is a machine-level programming language. Originally, no transla was used to compile or assemble the first-generation language. The first-generation programming instructi were entered through the front panel switches of the computer system.

A first generation (programming) language (1GL) is a grouping of programming languages that are mach level languages used to program first-generation computers. The instructions were given through the f

panel switches of these computers directly to the CPU. There was originally no compiler or assembler to process the instructions in 1GL. The instructions in 1GL are made of binary numbers, represented by 1s and 0s. This makes the language suitable for the understanding of the machine, but very much more difficult to interpret and learn by the human programmer, also known as a **1st generation language.**

The **main advantage** of programming in 1GL is that the code can run very fast and very efficiently, precisely because the instructions are executed directly by the CPU. One of the **main disadvantages** of programming in a low level language is that when an error occurs, the code is not as easy to fix.

The program is written as binary instructions, consisting of zeros and ones. This language is very much adapted to a specific computer and CPU and code portability is therefore significantly reduced in comparison to higher level languages.

2. Second Generation :

Second-generation programming language is a generational way to categorise assembly languages. The term was coined to provide a distinction from higher level third-generation programming languages (3GL) such as COBOL and earlier machine code languages. Second-generation programming languages have the following properties:

- The code can be read and written by a programmer. To run on a computer, it must be converted into a machine readable form, a process called **assembly**.
- The language is specific to a particular processor family and environment.

Second-generation languages are sometimes used in kernels and device drivers (though C is generally employed for this in modern kernels), but more often find use in extremely intensive processing such as games, video editing, graphic manipulation/rendering.

One method for creating such code is by allowing a compiler to generate a machine-optimised assembly language version of a particular function. This code is then hand-tuned, gaining both the brute-force insight of the machine optimizing algorithm and the intuitive abilities of the human optimiser.

3. Third Generation :

A third-generation programming language (3GL) is a refinement of a second-generation programming language. The second generation of programming languages brought logical structure to software. The third generation brought refinements to make the languages more programmer-friendly. This includes features like improved support for aggregate data types, and expressing concepts in a way that favours the programmer, not the computer (e.g. no longer needed to state the length of multi-character (string) literals in Fortran). A third generation language improves over a second generation language by having the computer take care of non-essential details, not the programmer. "High level language" is a synonym for third-generation programming language. First introduced in the late 1950s, Fortran, ALGOL, and COBOL are early examples of this sort of language.

Most popular general-purpose languages today, such as C, C++, C#, Java, BASIC and Pascal are also third-generation languages, although C++, Java and C# follow a completely different path as they are object-oriented in nature. Third generation focused on structured and had no meaning for the object encapsulation concepts. Such languages are considered **high-level** because they are closer to human languages and farther from machine languages. In contrast, assembly languages are considered low-level because they are very close to machine languages.

The main **advantage** of high-level languages over low-level languages is that they are easier to read, write and maintain. Ultimately, programs written in a high-level language must be translated into machine language by a compiler or interpreter.

4. Fourth Generation :

The term fourth-generation programming language (1970s-1990) (abbreviated as **4GL**) is better understood to be a fourth generation **environment**, packages of systems development software including very high level programming languages. A very high level programming language and a development environment or 'Analyst Workbench' designed with a central data dictionary system, a library of loosely coupled design patterns, a CRUD generator, report generator, end-user query language, DBMS, visual design tool and integration API. Historically, often used for prototyping and evolutionary development of commercial business software. In the history of computer science, the 4GL followed the 3GL in an upward trend toward higher abstraction and statement

power. The 4GL was followed by efforts to define and use a 5GL. All 4GLs are designed to reduce programming effort, the time it takes to develop software and the cost of software development. They are not always successful in this task, sometimes resulting in inelegant and unmaintainable code.

5. Fifth Generation :

A fifth-generation programming language (abbreviated as **5GL**) is a programming language based on solving problems using constraints given to the program, rather than using an algorithm written by a programmer. Most constraint-based and logic programming languages and some declarative languages are fifth-generation languages. Fifth-generation languages are designed to make the computer solve a given problem without the programmer. This way, the programmer only needs to worry about what problems need to be solved and what conditions need to be met without worrying about how to implement a routine or algorithm to solve them. Fifth-generation languages are used mainly in artificial intelligence research. **Prolog, OPS5,** and **Mercury** are examples of fifth-generation languages.

These types of languages were also built upon Lisp, many originating on the Lisp machine such as ICAD. Then, there are many frame languages such as KL-ONE.

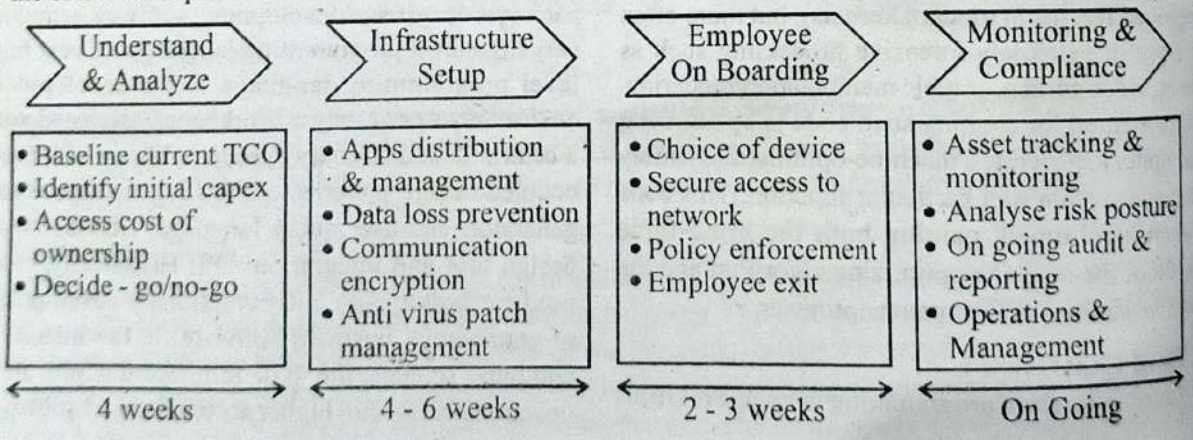
Q11. What are end-User computing?

Ans. End-user computing (EUC) refers to systems in which non-programmers can create working applications. EUC is a group of approaches to computing that aim at better integrating end users into the computing environment. These approaches attempt to realize the potential for high-end computing to perform in a trustworthy manner in problem-solving.

End-user computing can range in complexity from users simply clicking a series of buttons to writing scripts in a controlled scripting language to being able to modify and execute code directly.

Examples of end-user computing are systems built using fourth-generation programming languages, such as **MAPPER** or **SQL**, or one of the fifth-generation programming languages such as **ICAD**.

EUC applications should not be evolved by accident, but there should be a defined EUC strategy. Any Application Architecture Strategy / IT Strategy should consider the white spaces in automation (Enterprise Functionality not automated by ERP / Enterprise Grade Applications). These are the potential areas where EUC can play a major role. Then, **ASSIMPLER** parameters should be applied to these white spaces. ASSIMPLER stands for Availability, Scalability, Security, Interoperability, Maintainability, Performance, Low Cost of Ownership, Extendibility and Reliability to develop the EUC Strategy. Some of the factors contributing to the need for further EUC research are knowledge processing, pervasive computing, issues of ontology, interactive visualization and the like. Some of the issues related to end-user computing concern architecture (iconic versus language interface, open versus closed, and others). Other issues relate to IP configuration and maintenance. End-user computing allows more user input into system affairs that can range from personalization to full-fledged ownership of the system. The end user computing assurance is to be shown by the below example :



Q12. Write a short note on:
(a) Relational Database.
(b) Non-procedural language.

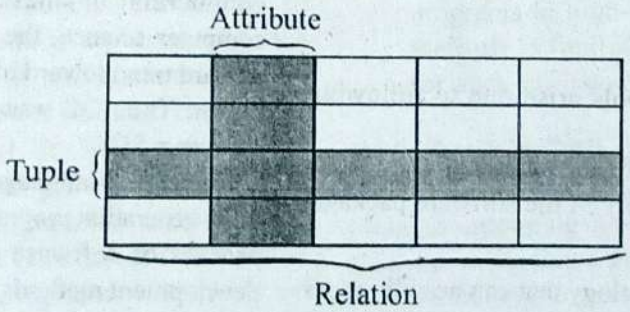
Ans. (a) Relational Database :

A **relational database** is a database that has a collection of tables of data items, all of which is formally described and organized according to the relational model. The term is in contrast to only one table as *the database* and in contrast to other models which also have many tables in one database.

In the relational model, each table schema must identify a primary column called the ***primary key***, used for identifying a row. Tables can relate by using a ***foreign key*** that points to the primary key of another table. The relational model offers various levels of refinement of the table relations called **database normalization**. The database management system(DBMS) of a relational database is called an **RDBMS**, and is the software of a relational database. Relational database theory uses mathematical terminology, which are roughly equivalent to the SQL database terminology concerning ***normalization***. The table below summarizes some of the most important relational database terms and their SQL database equivalents. It was first introduced in 1970 following the work of **E.F.Codd**.

A row or tuple has a *relation* schema, but an entire database has a *relational* schema.

SQL	Relational database	Description
Row	***Tuple***	*Data set* with specific instances in the range of each member.
Column name; Column data	***Attribute*** name; Attribute value	Labeled member in the set of elements common to all *data sets*; A name, word, number, phrase, etc.
Table	***Relation***; Base relvar	Formal data structure.
Set of column names	Relation scheme; Set of attributes	A schema.
View; Query result; Result set	Derived relvar	A data report from the RDBMS in response to a query.



(b) Non-Procedural Language :

NPL (Non Procedural Language) was a relational database language developed by **T.D. Truitt et al.** in 1980, for Apple II and MS-DOS. In general, a non-procedural language (also called a declarative language) requires the programmer to specify *what* the program should do, rather than (as with a procedural language) providing the sequential steps indicating *how* the program should perform its task(s). Non-procedural language also refers to a computer language that does not require writing traditional programming logic. It is also known as a **declarative language** as users concentrate on defining the input and output rather than the program steps required in a procedural language.

Example: Procedural programming is classic programming where the program language is used to tell the computer **EXACTLY** what to do - step by step. For examples are Assembler, Fortran, Cobol, C, etc. It is very detailed, very difficult and time consuming to write, but very efficient from the computers point of view.

Non-procedural programming is where we tell the computer, what we want and it figures out how to get it. Non/P programming is often used for database manipulations like SQL, Visual Basic, etc.

A SQL command like "Select name, address, city, state, zip order by zip" is non-procedural. That one line replaces dozens, or hundred of lines that would be needed to do the same thing in Cobol or C to get data out of a database.

Q13. Discuss the CASE tools.

Ans. CASE (Computer Aided Software Engineering) Tools :

CASE is a tool, which aids a software engineer to maintain and develop a software. It assists software engineering managers and practitioners in every activity associated with the software process. They automate project management activities, manage all work products produced throughout the process and assist engineers in their analysis, design, coding and test work.

The workshop for software engineering is called an Integrated Project Support Environment (IPSE) and the tool set that fills the workshop is called Computer Aided Software Engineering (CASE) tool. CASE tools add to the software engineers tool box and automate the manual activities to improve development process.

Need of CASE Tools :

The need of CASE tools arise due to following reasons :

- Increase in the size of the software.
- Increase in complexity of the software package.

CASE Tool Technology :

CASE tool has a technology that can be split into 3 levels. These levels are:

- **Production-Process Support Technology :** This range of tools support process activities such as specification, design, implementation and testing.
- **Process Management Technology :** This range of tools supports process modeling and process management.
- **Meta CASE Technology :** These tools are used to create production process and process management support tools.

Benefits of CASE Tools :

1. Smooth transfer of information (models, programs, documents, data) from one tool t another and one software engineering step t the next.
2. Reduction in the effort required to perfor umbrella activities such as Softwar Configuration Management, Quality Assurance and document production.
3. An increase in project control, that is achieve through better planning, monitoring, an communication.
4. Improved coordination among staff member who are working on a large software project.

Q14. What is fourth-generatio programming languages?

Ans. The term **fourth-generation programmin language** (1970s-1990) (abbreviated as **4GL**) is bett understood to be a fourth generation **environmen** packages of systems development software includin very high level programming languages. A very hig level programming language and a developme environment or 'Analyst Workbench' designed with central data dictionary system, a library of loose coupled design patterns, a CRUD generator, repo generator, end-user query language, DBMS, visu design tool and integration API. Historically, often us for prototyping and evolutionary development commercial business software. In the history computer science, the 4GL followed the 3GL in upward trend toward higher abstraction and stateme power. The 4GL was followed by efforts to defi and use a 5GL.

The natural-language, block-structured mode of third-generation programming languages improved process of software development. However, 3(development methods can be slow and error-prone became clear that some applications could developed more rapidly by adding a higher-le programming language and methodology, which wo generate the equivalent of very complicated 3 instructions with fewer errors. In some sens software engineering arose to handle 3GL developm 4GL and 5GL projects are more oriented tow problem solving and systems engineering.

All 4GLs are designed to reduce programm effort, the time it takes to develop software, and cost of software development. They are not alw successful in this task, sometimes resultin

inelegant and unmaintainable code. However, given the right problem, the use of an appropriate 4GL can be spectacularly successful as was seen with MARK-IV and MAPPER (Santa Fe real-time tracking of their freight cars, the productivity gains were estimated to be 8 times over COBOL). The usability improvements obtained by some 4GLs (and their environment) allowed better exploration for heuristic solutions than did the 3GL.

A quantitative definition of 4GL has been set by **Capers Jones**, as part of his work on function point analysis. Jones defines the various generations of programming languages in terms of developer productivity, measured in function points per staff-month. A 4GL is defined as a language that supports 12–20 function points per staff month. This correlates with about 16–27 lines of code per function point implemented in a 4GL.

Fourth-generation languages have often been compared to domain-specific programming languages (DSLs). Some researchers state that 4GLs are a subset of DSLs.

A number of different types of 4GLs exist :

1. Table-Driven(codeless) programming, usually running with a runtime framework and libraries. Instead of using code, the developer defines his logic by selecting an operation in a pre-defined list of memory or data table manipulation commands. In other words, instead of coding, the developer uses **Table-Driven Algorithm.** A good example of this type of 4GL language is **PowerBuilder**. These types of tools can be used for business application development usually consisting in a package allowing for both business data manipulation and reporting, therefore they come with GUI screens and report editors. They usually offer integration with lower level DLLs generated from a typical 3GL for when the need arise for more hardware/OS specific operations.

2. Report Generator programming languages take a description of the data format and the report to generate and from that they either generate the required report directly or they generate a program to generate the report.

3. Similarly, forms generators manage online interactions with the application system users or generate programs to do so.

4. More ambitious 4GLs (sometimes termed as *fourth generation environments*) attempt to automatically generate whole systems from the outputs of CASE tools, specifications of screens and reports and possibly also the specification of some additional processing logic.

5. Data management 4GLs such as SAS, SPSS and Stata provide sophisticated coding commands for data manipulation, file reshaping, case selection and data documentation in the preparation of data for statistical analysis and reporting.

Q15. Discuss the advantages and disadvantages of using 4GLs?

Ans. The various advantages and disadvantages are as follows :

Advantages :

1. Simplified the programming process.
2. Use non-procedural languages that encourage users and programmers to specify the results they want, while the computers determines the sequence of instructions that will accomplish those results.
3. Use natural languages that impose no rigid grammatical rules.

Disadvantages :

1. Less flexible than other languages.
2. Programs written in 4GLs are generally far less efficient during program execution than programs in high-level languages. Therefore, their use is limited to projects that do not call for such efficiency.

Q16. Explain fifth-generation languages?

Ans. A fifth generation (programming) language (5GL) is a grouping of programming languages build on the premise that a problem can be solved and an application built to solve it by providing constraints to the program (constraint-based programming), rather than specifying algorithmically how the problem is to be solved (imperative programming). Most constraint-based and logic programming languages and some declarative languages are fifth-generation languages.

While fourth-generation programming languages are designed to build specific programs, fifth-generation languages are designed to make the computer solve a given problem without the programmer. This way, the programmer only needs to worry about what problems need to be solved and what conditions need

to be met, without worrying about how to implement a routine or algorithm to solve them. Fifth-generation languages are used mainly in artificial intelligence research. **Prolog**, **OPS5**, and **Mercury** are examples of fifth-generation languages.

These types of languages were also built upon Lisp, many originating on the Lisp machine such as ICAD. Then, there are many frame languages such as KL-ONE.

In the 1980s, fifth-generation languages were considered to be the wave of the future, and some predicted that they would replace all other languages for system development with the exception of low-level languages. Most notably from 1982 to 1993, Japan put much research and money into their fifth generation computer systems project, hoping to design a massive computer network of machines using these tools.

Q17. Give a brief description of the Prototyping.

Ans. Prototyping :

It is based on the idea of developing an initial implementation for user feedback and then refining this prototype through many versions, until a satisfactory system emerges. In prototyping, instead of freezing the requirements before any design or coding can proceed, a throwaway prototype is built to help understand the requirements.

The process model of prototyping approach is shown in figure below :

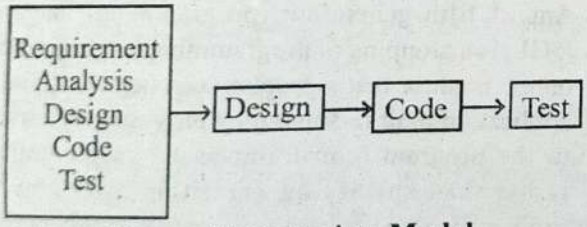


Figure: Prototyping Model

Prototyping is an attractive idea for complicated and large systems for which there is no manual process and existing system to help determine the requirements.

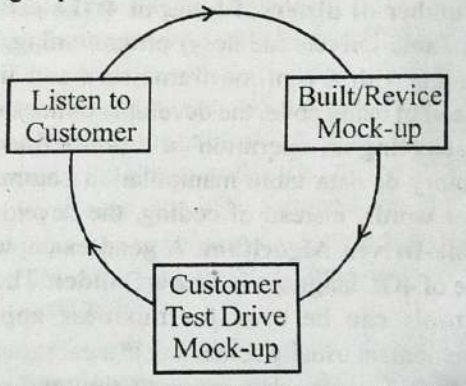
For prototyping for the purposes of requirement analysis to be feasible, its cost must be kept low. In prototyping, only minimal documentation needs to be produced. Prototyping is well suited for projects where requirements are hard to determine and t confidence in obtained requirements is low.

The Prototyping Paradigms begin wi requirement gathering.

- ⇨ Developer and customer meet and define t overall objectives for the Software.
- ⇨ Identify whatever requirements are know outline area where further definition mandatory.
- ⇨ A "Quick Design" occurs which focuses on representation of those aspects of the softwa that will be visible to the customer or user (F **Example**, input approaches and outp formats).

A quick design leads to the construction of Prototype. A prototype is evaluated by the custome user and used to refine requirement for the softwa to be developed. Interaction occurs as the prototyp is tuned to satisfy the needs of the customer.

The customer and developer, both, must agree th the prototype is built to serve as a mechanism f defining requirement.



This model is well suited for routine types projects, where requirements are well understood

Advantages of Prototyping Model :

1. It may be a usable program.
2. Experience gathered from developin prototyping model helps in developing actu system.
3. Since a toy model is developed before han serious errors are discovered at initial stage
4. Well suited for projects, where requiremen are hard to determine.
5. It is an excellent technique for reducing son types of risks associated with the project.

Disadvantages of Prototyping Model :

1. It has limited functional capabilities.
2. It's reliability is low.
3. It is not suited as final software product.
4. It's performance and overall quality is low.
5. Development of prototype may sometimes involve extra cost.

Q18. What are the method/stages of Prototyping?

Ans. **Prototyping has Three Methods or Stages:**

1. **Mock-Ups :** Mock-ups of a proposed system are produced for demonstration purpose at the early stages of a project, so as to assist the users and designers to make decisions about what is required at this stage. Prototype is the final appearance of the system, but lacks real processing or storage capability.
2. **Trial Prototyping :** In this, a simplified working model is produced. It might be a prototype of method of real processing.
3. **Rapid Prototyping :** Also called RAD (Rapid Application Development). It is not frequently used for small scale system which are required urgently. A simplified version of system is built and reviewed and the prototype is extended and upgraded. It is normally incomplete at this stage, but may have all essential features in a simplified form. The rapid results provided by this method give rise to its name.

Q19. Write a short note on Prototyping and Interface.

Ans. **Prototyping and Interface :**

Prototyping is an attractive idea for complicated and large systems for which, there is no manual process or existing system to help determine the requirements. In such situations, letting the client "play" with the prototype provides invaluable and intangible inputs, that help determine the requirement for the system. It is also an effective method of demonstrating the feasibility of a certain approach. The prototype is evaluated by the customer/user and used to refine requirements for the software to be developed. Iteration occurs as, the prototype is tuned to satisfy the need of the customer, while at the same time, enabling the developer to better understand, what needs to be done.

The prototype can serve as "the first system". Both customers and developers like the prototyping paradigms, users get a feel for the actual system and developers get to build something immediately prototyping can also be problematic for the following reasons.

1. The customer sees what appears to be a cracking version of the software unware that the prototyping is held together "with chewing gum and balling wire" unware that in the rush to get it working, no one has considered over all software quality or long term maintainability. When informed that the product must be rebuilt, so that high level of quality can be maintained, the customer arise foul and demands that "a few fixes" be applied to make the prototype a working product.
2. The developers often makes implementation, compromises in order to get a prototype working quickly. An inappropriate operating system or programming language may be used simply because, it is available and known, an inefficient algorithm may be implemented simply to demonstrate capability. After a time, the developer may become familiar with these choices and forget all the reasons why they were inappropriate.

Although the problem can occur, prototyping can be an effective paradigm for software engineering. The key is to define the rules of the game at the beginning, that is the customer and developer must both agree that the prototype is built to serve as a mechanism for defining requirements. It is then discarded and the actual software is engineered with an eye towards quality and maintainability.

Q20. Discuss the Problems that occur in application development?

Ans. Mistake is "an act or judgement that is misguided or wrong". And what is a problem? A problem is "a matter or situation regarded as unwelcome or harmful and needed to be dealt with and overcome".

A misguided judgement or an act for a purpose then will naturally create situations that would be regarded as unwelcome or needed to be dealt with. In other words, the roots of a problem arised during execution of any task lie in the mistakes committed while executing that task. In software development, the common problems arise due to the mistakes that are committed repeatedly due to external and internal influences such as competition, lack of right skills, lack of budget etc.

There are various problems which occured during the software development :

Problem stage #1 : Requirements gathering

Many projects still follow Waterfall SDLC. If the requirements are not very clear and exhaustive, then the team will find it difficult to proceed on schedule. The team will find it especially difficult to provide deliverables as per customer's expectations, since the requirements were unclear or incomplete at the beginning.

We may be saved, if we decide to go for Agile SDLC, but then that again requires that the end-user involvement be high. Else, requirements gathering will suffer.

Problem stage #2 : Scheduling & Estimation

Many a time project managers cram too much work within too less man-hours either because of inexperience or because of other constraints such as limited budget or lack of clarity in customer's business. Bad scheduling is bound to have a ripple effect on development, testing and deployment stages of an SDLC.

Problem stage #3 : Development

Mistakes committed during the above two stages cause problems that are inevitable during the development stage. Unclear requirements at the beginning may mean that we have to be ready for lot of feature creep-in. The business side of the customer may consider adding features during development stage as minor changes. But it may throw the whole development team very much off the schedule and cost estimates if the team has to keep revising the code to add features very often. Similarly, bad project schedule estimation will build pressure on the developers, which means more scope for bugs.

Problem stage #4 : Testing

Due to bad scheduling, it may be possible tha there is not enough time for testing, bug fixing and re-testing. It is not possible to have bug free software development. So, an overly optimistic schedule (one of the mistakes considered to have serious to catastrophic impact on software development projects) will most certainly create problems, as there will not be adequate testing done before deployment.

Problem stage #5 : Communication and Collaboration

Though communication and collaboration are no the stages of an SDLC, communication is the backbone for collaborating and getting things done Whether communication is done through documentation, stand-up meetings and virtua brainstorming or through Web based SDL(management tools, information sharing an knowledge exchange is required for the project t meet quality standards and deliver on schedule.

Q21. What is System Investigation? Wh we need to study system investigation

Ans. **System Investigation :**

System Investigation is the purpose to investiga and improve our knowledge about something b searching for extra information on the particul subject. In order to reach this stage in the Softwa Life Cycle, management would have listened to t alternative solutions presented by the system analy and have decided to either commission a brand ne IT system or have changes made to their curre system. During the earlier 'definition' phase, t analysts looked superficially at the current syste and the potential benefits of a new system.

During this **'investigation and analysis'** pha they will carry out very detailed investigations order to fully understand the current system a the proposed new system.

The analyst needs to have a plan of how t information is to be gathered. This needs to presented to management, so they can agree or si off the process. This is because carrying out t

information gathering phase is likely to cause disruption to staff which may not be acceptable at critical times of the month.

For example, imagine that payroll is run on the last Tuesday of the month, so the accounting staff are extremely busy on that day. They are unlikely to be able to spend an hour or two providing information on that day. It is likely that there will be limits as to what the analyst is able to carry out in the time available.

Q22. Explain the Life Cycle of Information System?

OR

Describe System Development Life Cycle and its various Phases?

Ans. Systems Development Life Cycle (SDLC) :

The **systems development life cycle (SDLC)** or **software development process** or **Software Development Life Cycle** in systems engineering, information systems and software engineering, is a process of creating or altering information systems and the models and methodologies that people use to develop these systems. In software engineering, the SDLC concept underpins many kinds of software development methodologies. These methodologies form the framework for planning and controlling the creation of an information system.

The System Development Life Cycle framework provides a sequence of activities for system designers and developers to follow. It consists of a set of steps or phases in which each phase of the SDLC uses the results of the previous one. A Systems Development Life Cycle (SDLC) adheres to important phases that are essential for developers such as planning, analysis, design and implementation and are explained in the section below. It includes evaluation of present system, information gathering, feasibility study and request approval. A number of system development life cycle (SDLC) models have been created that are **waterfall**, **fountain**, **spiral**, **build** and **fix**, **rapid prototyping**, **incremental** and **synchronize** and **stabilize**. The oldest of these and the best known is the waterfall model which is a sequence of stages in which the output of each stage becomes the input for the next.

These stages can be characterized and divided up in different ways, including the following :

Preliminary Analysis : The objective of phase 1 is to conduct a preliminary analysis, propose alternative solutions, describe costs and benefits and submit a preliminary plan with recommendations.

Conduct the preliminary analysis : In this step, we need to find out the organization's objectives and the nature and scope of the problem under study. Even if a problem refers only to a small segment of the organization itself, then we need to find out what the objectives of the organization itself are. Then we need to see how the problem being studied fits in with them.

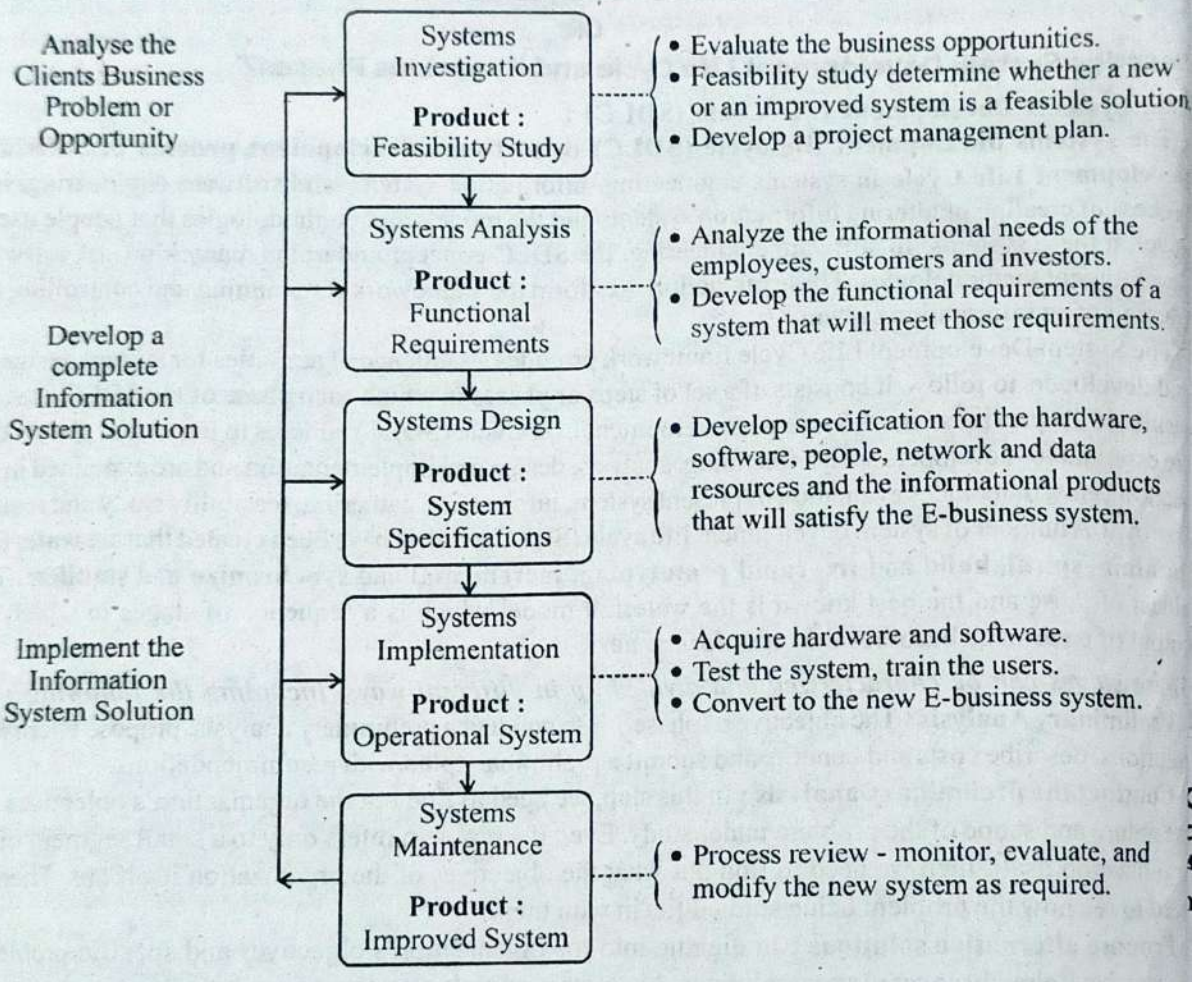
Propose alternative solutions : In digging into the organization's objectives and specific problems, we may have already covered some solutions. Alternate proposals may come from interviewing employees, clients, suppliers, and/or consultants. We can also study what competitors are doing. With this data, we will have three choices: leave the system as is, improve it, or develop a new system.

It describe the costs and benefits :

- **Systems analysis, requirements definition :** It defines project goals into defined functions and operation of the intended application. Analyzes end-user information needs.
- **Systems design:** It describes desired features and operations in detail, including screen layouts, business rules, process diagrams, pseudocode and other documentation.
- **Development:** The real code is written here.
- **Integration and testing:** It brings all the pieces together into a special testing environment, then checks for errors, bugs and interoperability.

- **Acceptance, installation, deployment:** It is the final stage of initial development, where t software is put into production and runs actual business.
- **Maintenance:** What happens during the rest of the software's life changes, correction, additio moves to a different computing platform and more. This is often the longest of all the stages.

The Information Systems Development Cycle



Q23. Explain various phases of System Investigation?

Ans. The system investigation stage addresses the needs or opportunities that can be achieved b sponsor or IT proposal. Preliminary investigation is the first step in the system development project. I a way of handling the user's request to change, improve or enhance an existing system. ***Syste investigation includes the following two stages :***

1. Problem Definition :

The first responsibility of a system analyst is to prepare a written statement of the objectives of t problem. Based on interviews with the user, the analyst writes a brief description of his/her understandi of the problem and reviews it with both the groups. People respond to written statements. They ask f clarifications and they correct obvious errors or misunderstandings. That is why, a clear statement objectives is important. In other words, proper understanding of the problem is essential to discover

cause of the problem and to plan a directed investigation by asking questions like what is being done? Why? Is there an underlying reason different from the one the user identifies? Following are some possible definitions of problems :

(a) The existing system has a poor response time.
(b) It is unable to handle the workload.
(c) The problem of cost, that is the economic system is not feasible.
(d) The problem of accuracy and reliability.
(e) The required information is not produced by the existing system.
(f) The problem of security.

2. Feasibility Study :

The actual meaning of feasibility is **viability**. This study is undertaken to know the likelihood of the system being useful to the organization. The aim of feasibility study is to assess alternative systems and to propose the most feasible and desirable system for development. Thus, feasibility study provides an overview of the problem and acts as an important checkpoint that should be completed before committing more resources. The feasibility of a proposed system can be assessed in terms of four major categories as given below :

(a) **Organizational feasibility :** The extent to which a proposed information system supports the objective of the organization's strategic plan for information systems determines the organizational feasibility of the system project.
(b) **Economic feasibility :** In this study, costs and returns are evaluated to know whether returns justify the investment in the system project.
(c) **Technical feasibility :** Whether reliable hardware and software, capable of meeting the needs of the proposed system can be acquired or developed by the organizations in the required time is a major concern of the technical feasibility.
(d) **Operational feasibility :** The willingness and ability of the management, employees, customers, suppliers, etc to operate, use and support a proposed system come under operational feasibility. In other words, the test of operational feasibility asks, if the system will work when it is developed and installed.

Q24. List out the persons who are involved in a system investigation phase of software development Cycle?

Ans. There are various people involved in system investigation phases such as IS personnel, users, managers, stakeholders etc. Their roles and responsibilities are as follows :

- **User :** The purpose of the user is to refine the document as per the need. Its responsibilities are:
  - To Eliminate Misunderstandings.
  - To Reduce Errors.
  - To Gain User agreement.
- **Stakeholder :** Stakeholder is a person, group, organization, member or system who affects or can be affected by an organization's actions. Stakeholder is an entity that can be affected by the results of that in which they are said to be stakeholders, i.e., that in which they have a stake.
- **Manager :** A manager must be proficient in a number of areas to be an effective leader, one who can motivate employees to perform at their highest capabilities. A manager differs from his subordinates because he must measure his success by what he can get others to accomplish and not solely by what he can do on his own. While opinions vary about a manager's specific top responsibilities, performing certain key functions effectively will ensure long-term success in management.

Q25. What are various methods which carry out the system investigation?

Ans. The main methods used to carry out the investigation of a system are :

1. Observation
2. Interview
3. Document Analysis
4. Questionnaire

1. Observation : This method involves examining procedures as they are carried out. The analy observes how work and procedures are carried out with the existing system and this enables the analy to witness at first hand, how the work is actually done and what it involves.

- **Advantages :** The existing system can be observed throughout the whole input, processing an output cycle. The recording system will show clearly what data is collected and what the informatio output is.
- **Disadvantages :** Can be very time consuming. The observer may not fully understand what the are observing without asking further questions. It can be difficult to record the observations in useful way.

2. Interview : Interviewing is a face-to-face method used to gather facts directly from the users of th system under investigation. The interviewer will ask some specific questions in order to get usefu information from the interviewee.

- **Advantages :** Much more flexible than a questionnaire as the responses to questions can lead t new questions as the interview takes place and it is much easier to clarify any difficult points.
- **Disadvantages :** The interview can take time to set up and may involve travelling (*although telephone or video-conferencing could be used*). The person carrying out the interview wi need to record the answers through the interview, unless it is recorded.

3. Document Analysis : The document analysis method involves examining existing data, record documentations as well as procedure manuals used for the existing system.

This method enables the analyst to obtain realistic and actual information about the system.

4. Questionnaire : The questionnaire consists of a standard set of questions. They can be paper bas or electronic.

- **Advantages:** Quick to distribute to a number of users, either by post or email. A range of clos and open questions can easily be asked.
- **Disadvantages:** Some questions and/or answers may not be clear particularly, if the person w created the questionnaire is totally unfamiliar with the existing system.

Q26. Write down the general principles of System Investigation?

Ans. In the Systems investigation, problems and opportunities are identified and considered in light of t goals of the business.

- Summarizes the results of systems investigation and the process of feasibility analysis.
- Recommends a course of action, continue systems analysis, modify the project in some manner drop it.
- State the purpose of systems investigation.
- Discuss the importance of performance and cost objectives.
- State the purpose of systems analysis and discuss some of the tools and technique used in this ph of systems development.
- Identify several factors that influence the success or failure of a systems development project.
- Effective systems development requires a team effort from stakeholders, users, managers, syste development specialists and various support personnel and it starts with careful planning.

Unit - IV

PERT, SDLC & CASE Tools

4

Q1. Explain the Project Evaluation and Review Technique (PERT).

Ans. PERT :

There are projects such as research project, design of a new machine etc. in which the time values are not deterministic. Such projects are handled by PERT.

PERT was developed by Navy sponsored research team in 1950. It is a method in which we try to exercise logical discipline in planning and controlling projects.

PERT is designed for scheduling complex projects that involve many inter-related tasks. It improves the planning process because :

(a) It forms the planner to define the projects, various component activities and events logically.

(b) It provides a basis for normal time estimates.

(c) It shows the effects of changes to the overall plan as they contemplated.

(d) It provides a built-in means for on-going evaluation of the plan.

(e) It identifies likely troubles before they are encountered.

(f) It provides all parties involved in a common basis of progress reporting both within organization and outside of it.

The main objective in the analysis through PERT is to find out whether a job could be finished by a given date. In PERT, we assume that the expected time of any operation can never be determined exactly.

In this approach, we use the following three time values associated with each operation (or activity). These are :

1. **Optimistic Time :** It is the shortest possible time in which the operation can be computed, if everything goes very well.
2. **Most Likely Time :** It is the estimate of the normal time, the operation would take.
3. **Pessimistic Time :** It is the longest time that an operation could take under adverse conditions.

Let, t_0 = optimistic time

t_m = most likely time

t_p = pessimistic time

Then, the expected value (t) connected with any operation is given by

$$t = \frac{t_0 + 4t_m + t_p}{6}$$

and the **variance** of t is given by

$$v(t) = \left(\frac{t_p - t_0}{6}\right)^2$$

Limitations of the PERT :

Following are the important limitations of PERT :

1. The difficultly arises, while seeking the realistic time estimates. In case of the new and non-repetitive type of projects, the time estimates produced are often more guess.
2. The natural tendency to oppose changed results in the difficulty of persuading the management to accept these techniques.
3. Determination to the level of network detail is another troublesome area. The level of detail varies from planner to planner and depends upon judgement and experience.

Q2. Explain procedure for PERT techniques.

Ans. PERT Procedure :

Step 1 : Draw the project network.

Step 2 : Compute the expected duration of each activity using the formula.

$$t_e = \frac{t_o + 4t_m + t_p}{6}$$

Also calculate the expected variance σ^2 of each activity.

Step 3 : Compute the earliest start, earliest finish, latest start, latest finish and total float of each activity.

Step 4 : Find the critical path and identify the critical activities.

Step 5 : Compute the project length variance σ^2,

which is the sum of the variance of all the critical activities and hence, find the standard deviation of the project length σ.

Step 6 : Calculate the standard normal variable

$$Z = \frac{T_S - T_e}{\sigma}$$

where T_S is the scheduled time to complete the project.

T_e = Normal expected project length duration

σ = Expected standard deviation of the project length.

Using the normal curve, we can estimate the probability of completing the project within a specified time.

Q3. Explain various methodologies for software development?

Ans. A software development methodology or system development methodology in software engineering is a framework that is used to structure, plan, and control the process of developing an information system.

There are the following methodologies :

1. Agile Software Development Methodology :

Agile software development is a conceptual framework for undertaking software engineering projects. There are a number of agile software development methodologies e.g. **Crystal Methods, Dynamic Systems Development Model (DSDM),** and **Scrum**.

Most agile methods attempt to minimize risk by developing software in short time boxes called **iterations**, which typically last for one to four weeks. Each iteration is like a miniature software project of its own and includes all the tasks necessary to release the mini-increment of new functionality: planning, requirements analysis, design, coding, testing and documentation. While iteration may not add enough functionality to warrant releasing the product, an agile software project intends to be capable of releasing new software at the end of every iteration. At the end of each iteration, the team re-evaluates project priorities.

Agile methods emphasize real time communication, preferably face-to-face, over written documents. At a minimum this includes programmers and the people who define the product such as product managers, business analysts or actual customers. The bullpen may also include testers, interface designers, technic writers and management. It also emphasize workin software as the primary measure of progres Combined with the preference for face-to-fa communication, agile methods produce very lit written documentation relative to other methods.

2. Joint Application Development (JA Methodology :

JAD is a requirements-definition and user-interf design methodology in which end-users, executi and developers attend intense off-site meetings work out a system's details. So, the Joint Applicat Development (JAD) methodology aims to involve client in the design and development of an applicat This is accomplished through a series of collabora workshops called **JAD sessions.**

JAD focuses on the business problem rather t technical details. It is most applicable to development of business systems, but it can be u successfully for systems software. It produces savings by shortening the elapsed time required gather a system's requirements and by gather requirements better, thus reducing the numbe costly, downstream requirements changes. success depends on effective leadership of the J sessions; on participation by key end-us executives and developers and on achieving gr synergy during JAD sessions.

3. Rapid Application Development (RA Methodology :

"Rapid-development language" is a general term refers to any programming language that off speedier implementation than the traditional th generation languages such as C/C++, Pascal Fortran. Rapid-Development Languages (RD produce their savings by reducing the amoun construction needed to build a product. Although savings are realized during construction, the ab to shorten the construction cycle has projectw implications, shorter construction cycles m incremental lifecycles such as Evolution Prototyping practical. Because RDLs often lack fi rate performance, constrained flexibility and limited to specific kinds of problems, they are usu better suited to the development of in-house busi software and limited-distribution custom soft than systems software.

RAD (rapid application development) proposes that products can be developed faster and of higher quality by :

- Using workshops or focus groups to gather user requirements.
- Prototyping and user testing of designs.
- Re-using software components.

Following a schedule that defers design improvements to the next product version, keeping review meetings and other team communication informal. There are commercial products that include requirements gathering tools, prototyping tools, software development environments such as those for the Java platform, groupware for communication among development members and testing tools. RAD usually embraces object-oriented programming methodology which inherently fosters software re-use. The most popular object-oriented programming languages, C++ and Java are offered in visual programming packages often described as providing rapid application development.

4. Spiral Methodology :

The Spiral Lifecycle Model is a sophisticated lifecycle model that focuses on early identification and reduction of project risks. A spiral project starts on a small scale, explores risks, makes a plan to handle the risks and then decides whether to take the next step of the project - to do the next iteration of the spiral. It derives its rapid development benefits not from an increase in project speed, but from continuously reducing the projects risk level - which has an effect on the time required to deliver it. Success at using the Spiral Lifecycle Model depends on conscientious, attentive and knowledgeable management. It can be used on most kinds of projects and its risk-reduction focus is always beneficial.

The spiral methodology extends the waterfall model by introducing prototyping. It is generally chosen over the waterfall approach for large, expensive, and complicated projects.

At a high-level, the steps in the spiral model are as follows :

1. The new system requirements are defined in as much detail as possible. This usually involves interviewing a number of users representing all the external or internal users and other aspects of the existing system.
2. A preliminary design is created for the new system.
3. A first prototype of the new system is constructed from the preliminary design. This is usually a scaled-down system and represents an approximation of the characteristics of the final product.
4. A second prototype is evolved using four steps:
   (a) Evaluate the first prototype and identify its strengths, weaknesses and risks.
   (b) Define the requirements of the second prototype.
   (c) Plan and design the second prototype.
   (d) Construct and test the second prototype.
5. At the project sponsor's option, the entire project can be aborted, if the risk is deemed too great. Risk factors might involve development cost overruns, operating-cost miscalculation or any other factor that could result in a less-than-satisfactory final product.
6. The existing prototype is evaluated in the same manner as was the previous prototype, and if necessary, another prototype is developed from it according to the fourfold procedure outlined above.
7. The preceding steps are iterated, until the customer is satisfied that the refined prototype represents the final product desired.
8. The final system is constructed based on the refined prototype.
9. The final system is thoroughly evaluated and tested. Routine maintenance is carried out on a continuing basis to prevent large-scale failures and to minimize downtime.

5. Systems Development Life Cycle (SDLC) Methodology :

The systems development life cycle (SDLC) is a conceptual model used in project management that describes the stages involved in an information system development project from an initial feasibility study through maintenance of the completed application.

In general, an SDLC methodology follows these steps :

1. If there is an existing system, its deficiencies are identified. This is accomplished by interviewing users and consulting with support personnel.

2. The new system requirements are defined including addressing any deficiencies in the existing system with specific proposals for improvement.
3. The proposed system is designed. Plans are created detailing the hardware, operating systems, programming and security issues.
4. The new system is developed. The new components and programs must be obtained and installed. Users of the system must be trained in its use and all aspects of performance must be tested. If necessary, adjustments must be made at this stage.
5. The system is put into use. This can be done in various ways. The new system can phased in according to application or location and the old system is gradually replaced. In some cases, it may be more cost-effective to shut down the old system and implement the new system all at once.
6. Once, the new system is up and running for a while, it should be exhaustively evaluated. Maintenance must be kept up rigorously at all times. Users of the system should be kept up-to-date concerning the latest modifications and procedures.

6. Waterfall (Traditional) Methodology :

The waterfall model is a popular version of the systems development life cycle model for software engineering. Often considered the classic approach to the systems development life cycle, the waterfall model describes a development method that is rigid and linear. Waterfall development has distinct goals for each phase of development, where each phase is completed for the next one to be started and there is no turning back.

The perceived advantages of the waterfall process are that it allows for departmentalization and managerial control. A schedule is typically set with deadlines for each stage of development and a product can proceed through the development process. In theory, this process leads to the project being delivered on time because each phase has been planned in detail.

In practice, waterfall development often falls short of expectations as it does not embrace the inevitable changes and revisions that become necessary with most projects. Once, an application is in the testing stage, it is very difficult to go back and change something that was not thought of in the con stage. Alternatives to the waterfall model include application development (JAD), rapid applica development (RAD), sync and stabilize, build an and the spiral model.

Q4. What is Problem Analysis? What the different methods of probl analysis?

Ans. **Problem Analysis :**

Problem analysis is a systematic proce analysing and obtaining a clear understandi the needs of the clients and the users, what ex is desired from the software and what ar constraints.

The basic principle used in analysis is **Divide Conquer** i.e. Partitioning the problem subproblems and then try to understand subproblem and its relationship to other subprob in an effort to understand the total problem.

Few Methods of problem analysis are :

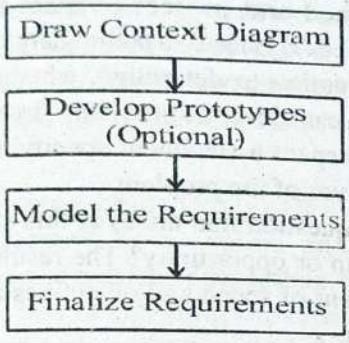
1. Informal Approach : The informal approa analysis is one where no defined methodology is Like in any approach, the information abou system is obtained by interaction with the client users and questionnaires, study of existing docum brainstorming etc. However, with this approac formal model is built of the system. The problem the system model are essentially built in the min the analysts and are directly translated from the n of the analysts to SRS.

2. Data Flow Modeling : Data flow modeling, referred to as the **structured analysis** technique function based decomposition, while modelin problem. It focuses on the functions perform the problem domain and the data consumed produced by these functions.

3. Object Oriented Modeling : In object or modeling, a system is viewed as a set of objects objects interacts with each other through the se they provide. Some objects also interact with the through their services such that the users g desired services. Hence, the goal of modeling identify the objects that exist in the problem do define the classes by specifying what state inform they encapsulate and what services they provi identify relationship that exist between obje different classes, such that the overall model

that it supports the desired user services.

4. Prototyping : Prototyping takes a different approach to problem analysis as compared to modeling based approaches. In prototyping, a partial system is constructed, which is then used by clients, users and developers to gain a better understanding of the problem and the needs.



Q5. Describe ISO 9000 Quality Standard.

Ans. ISO 9000 Quality Standard :

ISO(International Standard Organization) is a *family* of standards for quality management systems. ISO 9000 is maintained by ISO, the International Organization for Standardization and is administered by accreditation and certification bodies. The rules are updated, as the requirements motivate changes over time. Some of the requirements in ISO 9001:2008 (which is one of the standards in the ISO 9000 family) include :

- a set of procedures that cover all key processes in the business;
- monitoring processes to ensure that they are effective;
- keeping adequate records;
- checking output for defects, with appropriate and corrective action, where necessary;
- regularly reviewing individual processes and the quality system itself for effectiveness; and
- facilitating continual improvement.

A company or organization that has been independently audited and certified to be in conformance with ISO 9001 may publicly state that it is "ISO 9001 certified" or "ISO 9001 registered". Certification to an ISO 9001 standard does not guarantee any quality of end products and services rather, it certifies that formalized business processes are being applied.

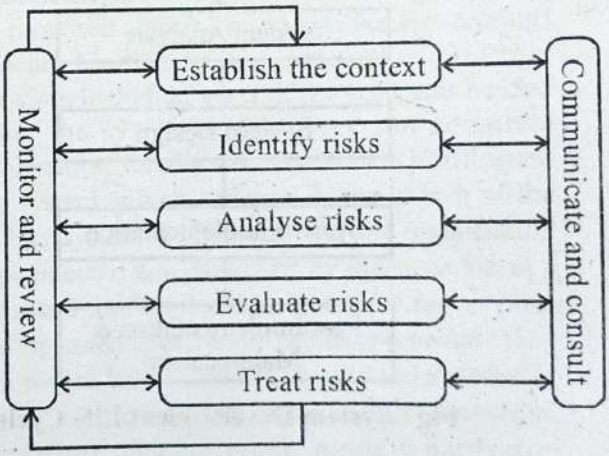
Although the standards originated in manufacturing, they are now employed across several types of organizations. A "product", in ISO vocabulary can mean a physical object, services or software.

ISO 9001 :

The ISO 9001 standard can be applied to any type or size of organization, from small family businesses to the world's largest corporations and government institutions. It provides a structured yet flexible framework for a customer-focused quality management system (QMS) that drives performance improvement.

The implementation of ISO 9001 can provide organizations with measurable improvements in their overall business performance, effectiveness and efficiency. ISO 9001 is based on the following eight Quality Management Principles, which are incorporated within the requirements of the standard and can be applied to improve organizational performance :

- Customer focus
- Leadership
- Involvement of people
- Process approach
- System approach to management
- Continual improvement
- Factual approach to decision making and
- Mutually beneficial supplier relationships.



Q6. What is system development life cycle? Explain the various phases of System Development Life Cycle in detail.

Ans. System Development Life Cycle (SDLC) :

The SDLC is a set of specified activities that system analyst, designers and users carry out to develop and implement a system. It encompasses method, tools and processes that ends in the development of

a system. Each phase or step performs a well-defined activity leading towards satisfaction of goal with output of one phase forming input of next phase. System development revolves around a life cycle that has some stages. The life cycle is not a procedure that deals with hardware and software. It is building computer-based system to help the users operate a business or make decisions effectively and manage an enterprise successfully. It is phased approach to analysis and design that holds that systems are best developed through the use of a specific cycle of analyst and user activities.

Various Phases or Stages or Steps of SDLC are listed below :

1. Recognition of Need
2. Feasibility Study
3. Analysis
4. Design
5. Implementation
6. Post-Implementation and Maintenance.

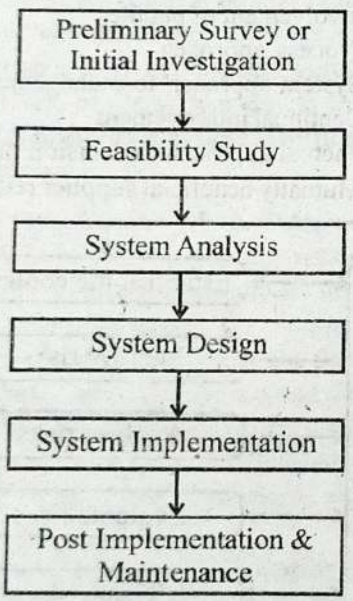


Fig : System Development Life Cycle

Each of the phase in SDLC is of great importance, because each phase perform various activities in the development of a system. Each phase takes some input, such as specification document from its previous phase and delivers an output, such as design document or some report.

Phases of SDLC :

I. Recognition of Need or Preliminary Surve Initial Investigation : This is the first phas SDLC, in which the problem to be solved is identi for improving the information system. It ident the needs for a new or enhanced syst Information needs of the organisation as a whol examined and project to meet these need identified. This leads to preliminary survey or an i investigation to determine, whether an alter system can solve the problem. The analysts firs is to prepare a statement specifying the scope objectives of the problem.

The question that arises at this stage is : Wh problem or opportunity? The result of this pha statement of scope and objectives and perform criteria.

II. Feasibility Study :

After the initial investigation is done, the feasi study on its result is conducted. It is a test of a sy proposal obtained from first phase according workability, impact on the organisation, ability to user needs, and effective use of resources. Feasi study takes into account, various constraints w which the system should be implemented.

The activity performed during this sta evaluation of existing system, and proced analysis of alternative candidate systems, estimation of the candidate system and final sele of the system. It focuses on three major questi

- What are the users demonstrable needs and does a candidate system meet them?
- What resources are available for given cand systems? Is the problem worth solving?
- What are the likely impact of the candi system on the organisation? How well do fit within the organisation's master MIS pl

The feasibility checks, whether it is feasibl implement the system. It describes and eval candidate system and provides for the selectio the best system, that meets system's perform requirements. ***There are three major type feasibility study:***

1. **Technical Feasibility :** It invo determining, whether or not a system actually be constructed to solve the pro at hand.

2. **Economical Feasibility** : It involves estimating cost-effectiveness of a project underline.

3. **Behavioural Feasibility** : It is a measure of how people feel about the system/project.

The result of this study is a formal proposal i.e., a port detailing the nature and scope of the proposed lution. It summarises, what is known and what is ing to be done. This formal agreement paves a y for actual design and implementation.

I. Analysis :

This phase performs a detailed study of the various erations performed by a system and their ationship within and outside of the system. System alysis is the process of gathering and interpreting cts, diagnosing problems and using the information recommend improvements for the system. Its mphasis is on identifying what is needed from the stem, not how system will achieve its goals.

The question focused here is :

- What must be done to solve the problem?
- What are the facts?

The activities performed here are – detailed valuation of the present system and data collection. The analysis phase defines the boundaries of the stem and determines whether or not a candidate stem should consider other related systems. During nalysis, data are collected on the available files, ecision points, and transaction handled by the resent system like DFD, interview, on-site bservation and questionnaire some, logical system odels and tools are used in this phase. Once analysis s completed, analyst has a firm understanding of hat is to be done.

V. Design :

The next step is to decide, how the problem might e solved. In system design, we move from logical o the physical aspects of the system. It is the most reative and challenging phase of the system life ycle. It describes a final system and the process by hich, it is developed. It refers to the technical pecifications, that will be applied in implementing he candidate system. This phase is the first in moving rom problem domain to solution domain. It also ncludes the construction of programs and testing.

The activities of this phase are producing general lesign specification, detailed design specifications, showing output, input, files and procedures, program construction and testing.

The question focused here are :

- How must the problem be solved?
- What is the system (processing) flow?

The result of this phase is design of alternative solutions, final cost/benefit analysis, hardware specifications, cost estimates, implementation specifications, implementation schedule, approval of system by user, programs, test plan, security audit and operating procedures, actual hardware used, formal system test.

This phase goes through two phases of development – logical design and physical design. Logical design reviews the present physical system, prepares input and output specifications, makes edit, security and control specification, details of implementation plan and prepares a logical design walkthrough.

Physical design maps out the details of the physical system, plans the system implementation, devises a test and implementation plan and specifies any new hardware and software.

This phase determines, how the output is to be produced and in what format. Input data and master files (database) are designed to meet the requirements of the proposed output. The operational (processing) phases are handled through program construction and testing, including a test 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 organisation are documented and evaluated by management as a step toward implementation. The final report prior to the implementation phase includes procedural flow charts, record layouts, report layouts and a workable plan for implementing the candidate system. Information on personnel, money, hardware, facilities, and their estimated cost must also be available. At this point, projected cost must be close to actual costs of implementation. Operating procedures and documentation must be completed. Security and auditing procedures must also be developed.

V. Implementation :

The implementation phase is less creative than the system design. It is primarily concerned with user

training, site preparation and the file conversion. Telecommunication network and tests of the network along with the system are also included under implementation. During the final testing, user acceptance is tested followed by user training. Conversion takes place normally parallel to the user training process. System testing is performed to check the readiness and accuracy of the system to access, update and retrieve data from new files. Once, the programs became available, test data are read into the computer and processed against the files for testing. If successful, the program is then run with *"live"* data. In most conversions, parallel run is conducted, where the new system runs simultaneously with the *"old"* system. After the candidate system itself, the old system is phased out.

So, the activities discussed during this phase are user training and file/system conversion.

The question that was focused are :

- What is the actual operation?
- Are user manuals ready?
- Are there delays in loading files?

The result of this phase is training program and user-friendly documentation.

Conversion is the final shifting of one system to another system. This stage of implementation is the main part of making the system functional. System can be converted in various ways depending on cost and risk that the management is ready to incur.

Various conversion methods are :

1. **Parallel Conversion :** It involves simultaneous operation of both the old as well as new system. Both are compared and when both generate similar results, the new system is displayed.
2. **Phased Conversion :** In this shifting from old system to new is done in a phased manner, where there is a gradual replacement of old system with the new in parts.
3. **Direct/Immediate Conversion :** It consist of immediate shifting or change from the old system to the new system.

VI. Post-Implementation and Maintenance :

After the installation phase is completed and the user staff is adjusted to the changes created by the candidate system, evaluation and maintenance begins. Any changes after the client has accepted the system are categorised as maintenance. The importance of maintenance is to bring the new system to standards. It is the most difficult aspect of system development. The first step after maintenance request is received is to identify the type of maintenance.

The activities performed during this phase are: Evaluation, Maintenance and Enhancements.

The key questions focused are :

- Is the key system running?
- Should the system be modified?

The results are user requirements met, user standards met and satisfied user.

There are Four Types of Maintenance :

1. **Corrective Maintenance :** It aims to correct any remaining errors regardless of, where they may arise – specification, design, coding etc.
2. **Adaptive Maintenance :** It changes the system to react to changes in the environment in which the system operates. **For Example,** porting to a new compiler, operating system and/or hardware.
3. **Preventive Maintenance :** It involves the activities to update the software in anticipation of any future problems.
4. **Perfective Maintenance :** It enhances the performance or modifying the programs to respond to the user's additional or changing needs.

Q7. What is feasibility study? Explain its types.

Ans. **Feasibility Study :**

The feasibility study activity involves the analysis of the problem and collection of all relevant information relating to the product such as the different data items that are to input to the system processing required to be carried out on these data output data required to be produced by the system as well as various constraints on the behaviour of the system.

Types of Feasibility Study :

1. Economic Feasibility : This part of feasibility study gives the top management, the economic justification of the new system. It involves simple economic analysis of the whole system that gives the actual comparison of costs and benefits statin whether system is economically feasible or not.

2. Technical Feasibility : It involves

(a). Understanding different technologies involve

in the proposed system.
(b). Find out whether the organization currently possesses the required technologies.
(c). Finds out whether the necessary expertise exists to use the techniques and technologies.

3. **Legal Feasibility :** The development of certain sensitive systems may have some legal ramifications. In that case, it has to be ascertained whether the development of new system, may result in any infringement, violation, contracts or liabilities. For that situation, legal experts have to be consulted.

4. **Organizational Feasibility :** It involves getting familiar with the industry in which the organization operates, performance of the organization. In this, one has to get a comprehensive understanding of what are the issues on hand and what are the management expectations.

Once, the feasibility study is completed and best option is chosen, then the outcome of same has to presented in the form of a project proposal. Based on feasibility report, the management decide whether or not to accept the proposal.

Q8. What are software requirements? Also describe its type ?

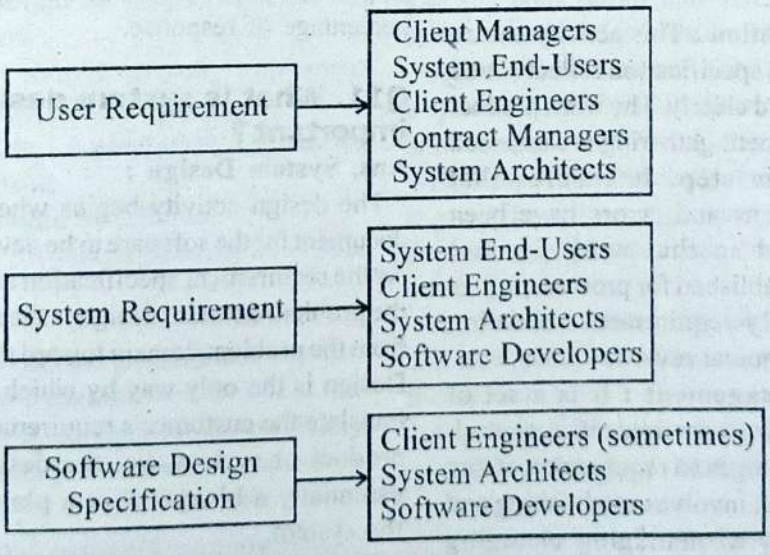
Ans. Software Requirement :

The requirements for system are the descriptions of the services provided by the system and its operational constraints. These requirements reflect the need of customers for the system that helps solve some problems such as controlling a device, placing an order or finding information. The process of finding out, analysing, documenting and checking these services and constraints is called **Requirement Engineering (RE).**

A software requirement is a property which must be exhibited by software developed or adopted to solve a particular problem. The problem may be to automate a part of a task of someone who will use the software to support the business processes of the organisation that has commissioned the software to correct shortcomings of existing software and many more.

Types of Requirement :

1. **User Requirement :** They are statements in a natural language plus diagrams of what services, the system is expected to provide and the constraints under which it must operate.
2. **System Requirement :** They set out the system's functions, services and operational constraints. The document should be precise. It should define exactly what is to be implemented. It may be a part of the contract between the system buyer and the software developer.
3. **Software Specification :** A detailed software description which can serve as a basis for design or implementation. It is written for developers.



Q9. What are the steps for Requirement Analysis?

Ans. **Steps for Requirement Analysis :**

Following are the steps that are performed during Requirement Analysis phase of System Development Life Cycle :

1. Requirement Elicitation : Ask customer, user, other object for system, what is to be accomplished, how system or product fits into needs of business, how system is to be used on day to day basis. The problems faced during this step are : problem of scope, problem of understanding and problem of volatility. To avoid such problem, requirement gathering activity must be followed in organised manner.

2. Requirement Analysis and Negotiation: Analysis, categorizes requirement and organises them into related subsets, explores each requirement in relation to each other, examine requirement for consistency, omissions and ambiguity and ranks requirement based on needs of customer/users.

3. Requirement Specification : Specification can be written document, graphical model, mathematical model, collection of usage scenarios, prototype or combination of these. Standard templates can also be developed for system specification in a consistent and understandable manner. System specification is final work product that serves as foundation for hardware engineering, software engineering and database engineering.

4. System Modeling : To fully specify what is to be built, we need a meaningful model of application, which is built in this step.

5. Requirement Validation : This activity ensure that software requirement specification reflect actual requirement accurately and clearly. The work product produced during requirement gathering are assessed for quality during this step. It ensures that inconsistencies, omissions and errors have been detected and corrected so that work product conforms to standards established for process, project and product. The primary requirement validation mechanism is formal technical review.

6. Requirement Management : It is a set of activities that help in project team to identify, control, track requirement and changes to requirement at any time as project proceeds. It involves establishing and maintaining an activity of managing changing requirements during development of software system.

Q10. What are the techniques for Requirement Elicitation?

Ans. We use four tools for Requirement Elicitation (Information Gathering). The primary information gathering tools are record review, on site observations, interviews and questionnaires. The most commonly used tool is interview.

1. Record Review : Procedure manuals, forms and text books are useful sources for the analyst. These describe format and functions of the present system. Upto date manuals save hours of information gathering time. It is a technique mostly used in all projects.

2. On-Site Observation : On-site observation is to get close to the real system being studied. Observation helps the analyst in detecting problems that exist in the current system. Main Limitation of observation is the difficulty of observing attitudes and motivation and many unproductive hours that often are spent in observing one time activities.

3. Interview : Interview is a face-to-face interpersonal meeting designed to identify relations and capture information as it exists. It is an art that requires experience in arranging the interview, setting the stage, avoiding arguments and evaluating the outcome. It may be structured or unstructured.

4. Questionnaires : The questionnaires is a self administered tool that is more economical and requires less skill to administer than the interview. It examines a large number of respondents at the same time, provides standardized wording and interactions and places less pressure on subjects for immediate response. Main limitation of questionnaire is low percentage of response.

Q11. What is system design ? Why it is important ?

Ans. System Design :

The design activity begins when the requirement document for the software to be developed is available. As the requirement specification activity is entirely in the problem domain, design is the first step in moving from the problem domain toward the solution domain. Design is the only way by which we can accurately translate the customer's requirements into a finished product or system. So, the design of a system is essentially a **blueprint** or a **plan** for a solution for the system.

The goal of the design process is to produce a model

or representation of a system, which can be used later to build that system.

The design process for software systems often has two levels. At first, the focus is on deciding which modules are needed for the system with the specifications interconnected. This is called "**System-design**" or "**Top-level design**". In second, the internal design of the modules or how the specifications of the modules can be satisfied is decided. This design level is often called "**Detailed design**" or "**Logic design**".

According to **Stevens,** ***"Software design is the process of inventing and selecting programs that meet the objectives for a software system."***

Input includes an understanding of the following :

1. Requirements
2. Environmental constraints
3. Design criteria.

The output of the design effort is composed of the following :

1. Architecture design which shows how pieces are interrelated.
2. Specifications for any new pieces.
3. Definitions for any new data.

Design is Important :

The definition of a "good software design" can vary depending on the application for which it is being designed. The notion of goodness of a design itself varies widely across software engineers and academicians. Most researchers and software engineers agree on a few desirable characteristics that every good software design for general applications must possess. These are :

1. **Correctness :** It should correctly implement all the functionalities of the system.
2. **Understandability :** A good design should be easily understandable.
3. **Efficiency :** It should be efficient.
4. **Maintainability :** It should be easily amenable to change.

Q12. Explain the design process.

Ans. **Software Design Process :**

Software design process can be decomposed mainly into following 3 levels (or phases) of design:

(i). Interface Design :

It describes how the software communicates within itself, with system that interoperate with it and with humans who use it. It is the specification of the interaction between a system and its environment. This phase proceeds at a high level of abstraction with respect to the inner workings of the system i.e., during interface design, the internals of the system are completely ignored in favour of the relationship between the system and its environment. An interface implies a flow of information (**Example:** Data and/or Control) and a specific type of behaviour.

Interface design should usually include following:

- Precise description of events in the environment, or message from agents, to which the system must respond.
- Precise description of the events or messages that, the system must produce.
- Specification of the data, and the formats of the data, coming into and going out of the system.
- Specification of the ordering and timing relationship between incoming events or messages and outgoing events or outputs (often called protocols).

The major task of interface design is user interface design, especially for system with complex graphical user interfaces.

(ii) Architecture Design Process :

The **Architecture Design** defines the relationship between structured elements of the software, the 'design patterns' that can be used to achieve the requirements that have been defined for the system and the constraints that affect the way in which architectural design patterns can be applied.

It is the specification of the major components of a system, their responsibilities, properties, interfaces and the relationships and interactions between them.

Data flow oriented design is an architectural design method that allows a convenient transition from the analysis model to design description of program structure.

Architectural design initiates with the data design and then proceeds to representation of structure of the system.

Issues in Architectural Design include :

- Gross decomposition of the system into major components.
- Allocation of functional responsibilities to components.

- Component interfaces.
- Component scaling and performance properties, resource consumption properties, reliability properties and so forth.
- Communication and interaction between components.

(iii). Detailed Design :

It is the specification of the internal elements of all major system components, their structure, properties, relationship, processing and often their algorithms and data structures. It fills in all design specifications still left open after architectural design. Its focus is on designing the (internal) logic for each of the modules specified in system design. The outcome of this is called module specification document.

It may include :

- Decomposition of major system components into program units.
- Allocation of functional responsibilities to units.
- Unit interfaces.
- Unit states and state changes.
- Data and control interactions between units.
- Unit packaging and implementation, including issues of scope and visibility of program elements.
- Algorithms and data structures.

There is a design procedure called Procedural Design or Component Level Design.

The procedural design occurs after data, architectural and interface designs have been established. The intent is to translate the design model into operational software. All the data, architectural and interface design must be translated into operational software. To accomplish this, the design must be represented at a level of abstraction, that is closer to the code.

The procedural design represents the software in a way that allows us to review the details of the design for correctness and consistency with earlier design representation (i.e., the data, architectural and interface design). It provides a means for accessing whether data structure, interfaces and algorithms will work.

A software design procedure can also be of 2 types :

- Function-Oriented Design.
- Object-Oriented Design.

Function-Oriented Design : In this, design consists of module definition with each module supporting a functional abstraction. The basic output of system design, when this approach is being followed, is definition of all major data structure, all major modules of system and how module interact with each other.

Object-Oriented Design : In this design, the modules in the design represent data abstraction. Its goal is to identify object that system contains and relationship and interaction between objects so that system implements specification.

Q13. What is structured programming? Explain important elements of good programming style.

Ans. Structured Programming :

Structured programming is often regarded as "**goto-less**" programming. Although extensive use of gotos is certainly not desirable, structured programs can be written with the use of gotos.

A program has a static structure as well as dynamic structure. The static structure is the structure of the text of the program, which is usually just a linear organisation of statements of the program. The dynamic structure of the program is the sequence of statements executed during the execution of the program. The objective of structured programming is to write program, so that the sequence of statements executed during the executing of the program is same as sequence of steps in the text of that program. So, the objective becomes developing programs whose control flow during execution is linearized and follows the linear organisation of the program text.

The readability of a program depends on the programming language used and on the programming style of the implementer.

Important elements of good programming style

Most important elements of good programming style are :

1. Structuredness :

(a) Structuring in the large : The structure is provided by classes and methods using object-oriented decomposition and by modules and procedures assigned to the modules. During implementation and the components defined in the design must be realised in such a way that they are executable on a computer.

(b) Structuring in the small : Selecting appropriate

programs components in the algorithmic formulation of subsolutions. The idea behind this is to use only control flow structures with one input and one output in the formulation of algorithms.

2. **Expressive Power :** The implementation of software system encompasses the naming of the objects and the descriptions of actions that manipulate these objects. The choice of names becomes particularly important in writing an algorithm.

3. **Outward Form :** Beyond name selection and connecting, the readability of a software systems also depends on its outward form.

Rules :

(a) For every program component, the declaration should be distinctly separated from the statement section.

(b) The declaration section should have a uniform structure, e.g., using following sequence :
   (i). Constant
   (ii). Data type
   (iii). Class and modules
   (iv). Methods and procedures.

(c) The interface description should separate input, output and I/O parameters.

(d) Keep comments and source code distinctly separate.

Q14. What do you understand by System testing?

Ans. System Testing :

Testing is vital for success of the system. Testing is the process of executing a program with the intention of finding errors.

Testing a program consists of subjecting the program to a set of test inputs and observe, if the program behaves as expected.

A system is tested for online response, volume of transactions, stress, recovery from failure and usability.

System testing requires a test plan that consists of several key activities and steps for program, string, system and user acceptance testing.

System testing makes a logical assumption that if all the parts of the system are correct, the goal will be successfully achieved. Software testing is the process of testing the software product. Effective software testing contributes to the delivery of higher quality software products and more satisfied users.

Objectives of Testing :

There are the various testing objectives for the testing phase of software engineering. Important of them are :

(a) Testing is a process of executing a problem with the intent of finding an error.

(b) A good test case is one that has a high probability of finding an as-yet undiscovered error.

(c) A successful test is one that uncovers an as-yet undiscovered error.

Our objective is to design tests that systematically uncover different classes of errors and do so with a minimum amount of time and effort.

Testing Principles :

There are many principles that guide software testing. Following are the main principles for testing:

1. All tests should be traceable to customer requirements.
2. Tests should be planned before testing begins.
3. Testing should begin "in the small" and progress towards testing "in the large".
4. Exhaustive testing is not possible.
5. To be most effective, testing should be conducted by an independent third party.

Types of System Testing :

During system testing, we should evaluate a number of attributes of the software that are vital to the user. There are some special types of system tests which are peak load test, storage testing, performance testing, recovery testing, procedure testing, security testing and stress testing, etc.

1. Peak Load Testing : This is used to determine whether the system will handle the volume of activities that occur when the system is at peak of its processing demand. For instance, when all terminals are active at the same time, this test applies mainly for on-line systems.

2. Storage Testing : This test is to be carried out to determine the capacity of the system to store transaction data on a disk or in other files. Capacities here are measured in terms of the number of records that a disk will handle or a file can contain. If this test is not carried out, then there are possibilities that during installation, one may discover that there is not enough storage capacity for transactions and

master file records.

3. Performance Testing : This test refers to the response time of the system being installed. Performance time testing is conducted prior to implementation to determine how long it takes to receive a response to an inquiry, make a backup copy of a file or send a transmission and receive a response.

Performance testing is designed to test run-time performance of software within the context of an integrated system. Performance testing occurs throughout all steps in the testing process.

4. Recovery Testing : Recovery testing involves a system test that forces the software to fail in a variety of ways and verify that recovery is properly performed. Many computer-based systems must recover from faults and resume processing within a pre-specified time.

5. Procedure Testing : Documentation and run manuals telling the user how to perform certain functions are tested quite easily by asking the user to follow them exactly through a series of events. It is surprising how not including instructions about aspects such as when to depress the enter key, removing the diskettes before putting off the power and so on could cause problems. This type of testing brings out what is not mentioned in documentation and also the errors in them.

6. Security Testing : Security testing attempts to verify that protection mechanism built into a system will in fact protect it from improper penetration. During security testing, the tester plays the role of the individual who desire to penetrate in the system. Given enough time and resources, good security testing will ultimately penetrate a system. The role of a system designer is to make penetration cost greater than the value of the information that will be obtained.

7. Stress Testing : During earlier software testing steps, white box and black box techniques resulted in thorough evaluation of normal program functions and performance. Stress tests are designed to confront programs with abnormal situation. Stress testing executes a system in a manner that demands resources in abnormal quantity, frequency or volume. A variation of stress testing is a technique called **sensitivity testing.**

8. Human Factors : In case during processing if the screen goes blank, the operator may start to wonder as to what is happening and he could just about do anything such as press the enter key a number of times or switch off the system and so on, but if a message is displayed saying that the processing is in progress and asking the operator to wait, these types of problems can be avoided.

Q15. Explain the testing methods of structural testing.

Ans. **Structural Testing / White-Box Testing :**

It is also called Clear Box Testing, Open–Box Testing, Logic-Driven Testing, Glass-Box Testing and Path-Oriented Testing.

Structural Testing Covers Following Types of Testing Methods :

(a) Path Testing : It requires us to design test cases such that all linearly independent paths in the program are executed atleast once. It allows the design and definition of a basis set of execution paths. The starting point for path testing is a program flow graph.

(b) Condition Testing : It is a test case design method, that exercises the logical conditions contained in a program module. Its purpose is to detect not only errors in the conditions of a program but also other errors in the program.

(c) Data-Flow Testing : It focus on the points at which variable receives values and the points at which these values are used. It is useful for selecting test paths of a program containing nested if and loop statements. This approach is effective for error protection.

(d) Loop Testing : It focuses exclusively on the validity of loop constructs. There may be different classes of loop such as simple loops, nested loops and concatenated loops.

(e) Mutation Testing : In this, the software is first tested by using an initial test suite built up from different white-box strategies. The idea is to make a few arbitrary changes to a program at a time. Each time a program is changed, it is called mutated program and the change effected is called a mutant. A mutated program is tested against the full test suite of the program. It is a fault-based technique.

Q16. Explain the testing methods of functional testing.

Ans. **Functional Testing Covers Following Types of Testing Methods :**

(a) Graph-Based Testing : This testing method verifies that all the objects modelled in a software have the expected relationship to one another. This testing begins by creating a graph that has collection of nodes that represent objects, links that represent relationship between objects, node weights that describe properties of a node and link weights that describe some characteristics of a link.

(b) Equivalence Testing : It divides the input domain of a program into classes of data from which test cases can be derived. The main idea behind the equivalence classes is that testing the code with any one value belonging to an equivalence class is as good as testing the software with any other value belonging to that equivalence class.

(c) Boundary Value Analysis : In this method, the test case designer choose values at the extremes of the class for partitioning. It encourages test case designers to look at output conditions and design test cases for the extreme conditions in output. Boundary value test cases are called extreme test cases.

(d) Comparison Testing : It is done on the software whose reliability is critical and thus software is developed redundantly. Thus, separate SE team develops independent version of the applications using same specification. These version form basis of this testing (also called back-to-back testing). If the specification from which all versions have been developed is in error, all versions will likely reflect the error.

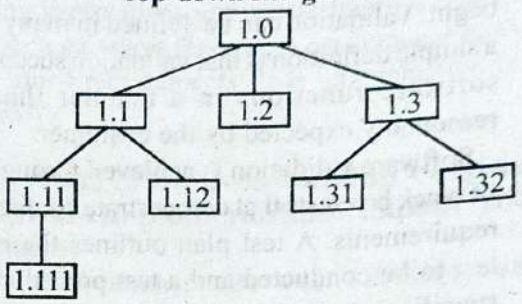
(e) Orthogonal Array Testing : It can be applied to problems in which the input domain is relatively small but too large to accommodate exhaustive testing. It is particularly useful in finding errors associated with region faults.

Q17. What are top-down and bottom-up strategies for integration testing?

Ans. **Top-Down and Bottom-Up Strategies for Integration Testing :**

Top-down and bottom-up test strategies reflect different approaches to system integration. In top-down integration, the high-level components of a system are integrated and testing before their design and implementation has been completed. In bottom-up integration, low-level components are integrated and tested before the higher-level components have been developed.

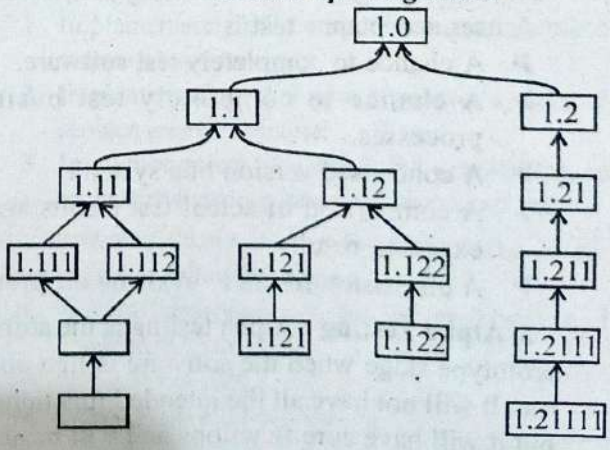
Top-down Integration



Top-down testing is an integral part of a top-down development process where the development process starts with high level components and works down the component hierarchy. After the top-level component has been programmed and tested, its sub-components are implemented and tested in the same way. This process continues until the bottom-level components are implemented. The whole system has then been completely tested.

By contrast, bottom-up testing involves integrating and testing the modules at the lower levels in the hierarchy and then working up the hierarchy of modules until the final module is tested. This approach does not require the architectural design of the system to be complete so it can start at an early stage in the development process.

Bottom-up Integration



Q18. Explain validation testing with its types.

Ans. **Validation Testing :**

At the culmination of integration testing, software is completely assembled as a package, interfacing errors have been uncovered and corrected and a final series of software test-validation testing may begin. Validation can be defined in many ways but a simple definition is that validation succeeds when software functions in a manner that can be reasonably expected by the customer.

Software validation is achieved through a series of black box tests that demonstrate conformity with requirements. A test plan outlines the classes of test to be conducted and a test procedure defines specific test cases that will be used in an attempt to uncover errors in conformity with requirements. Both the plan and procedure are designed to ensure that all functional requirements are satisfied.

Types of Validation Testing :

1. Acceptance Testing : In software development acceptance testing also called end user testing, application testing is a phase of software development in which the software is tested in the real world by the intended audience. User acceptance testing is a key feature of projects to implement new systems or processes. It is the formal means by which we ensure that the new system or process does actually meet the essential user requirements. Each module to be implemented will be subject to one or more user acceptance tests before being signed off as meeting user needs.

A user acceptance test is :

- A chance to completely test software.
- A chance to completely test business processes.
- A condensed version of a system.
- A comparison of actual test results against expected results.
- A discussion forum to evaluate the process.

2. Alpha Testing : Alpha testing is the software prototype stage when the software is first able to run. It will not have all the intended functionality, but it will have core functions and will be able to accept inputs and generate outputs. An alpha test usually takes place in the developer's offices on a separate system. Any software that will be run on customer equipment must first be reviewed.

The alpha test is conducted at the developer's site by a customer. The software is used in a natural setting with the developer "looking over the shoulder" of the user and recording errors and usage problems. Alpha tests are conducted in a controlled environment.

The alpha testing is also called simulated test because it is conducted in a simulated environment using simulated data.

3. Beta Testing : The beta test is conducted at one or more customer sites by the end users of the software. Unlike alpha testing, the developer is generally not present. Therefore, the beta test is a live application of the software, in an environment that cannot be controlled by the developer. The customer records all problems, that are encountered during beta testing and reports them to the developer at regular intervals. As a result of problems reported during beta test, the software developer makes modifications and then prepares for release of the software product to customer.

A test for a computer product i.e., software prior to commercial release is known as Beta test. Beta testing is the last stage of testing and normally will involve sending the product to beta test sites outside to company for real world exposure or offering the product for a free trial. Beta testing is often preceded by a round of testing called alpha testing.

4. System Testing : When unit testing is conducted, we test the components i.e., various modules of the software system. During integration of these modules we perform integration testing. When all components of the system are integrated we have a complete computer based system and then we perform system testing. System testing is actually a series of different tests, primary purpose of which is to fully exercise the computer based system.

Systems testing must also verify that file structures are adequate and that indexes have been built properly. Sorting and reindexing procedures are assumed to be present in lower-level modules. These must be tested at the system level to see that they, in fact, exist and achieve the results the module expect.

Q19. What do you understand by Software Implementation?

Ans. **Software Implementation :**

A crucial phase in the software life cycle is the successful implementation of new software design. Implementation phase is less creative than design phase. It is the process of converting a new or existing old system's design into operation. It is mainly concerned with user training, site selection and preparation and file conversion. Implementation is concerned with those tasks, leading immediately to a fully operational system. It includes the final testing of complete system to user satisfaction and supervision of initial operation of the system. It also includes providing security to the system so that it may not be misused by any person.

Types of Implementation :

There are basically three types of implementation that are as follows :

1. Implementation of a computer system to replace a manual system: The problems encountered are converting files, training users, creating accurate files and verifying printouts for integrity.

2. Implementation of a new computer system to replace an existing one : This is usually a difficult conversion. If, not properly planned, there can be many problems. Some large computer systems have taken as long as a year to convert.

3. Implementation of a modified application to replace an existing one : This type of conversion is relatively easy to handle, provided there are no changes in the files.

There are 3 Aspects or Issues Associated with Software Implementation :

- Training Personnel.
- Conversion Procedure.
- Post-Implementation Review.

1. Training Personnel : During the final testing, user acceptance is tested followed by user training. Depending on the nature of the system, extensive user training may be required. The quality training received by the personnel involved with the system helps or hinders and may even prevent the successful implementation of software.

2. Conversion Procedure : Conversion is the process of changing from the old system to new one. It include review of the project plan and implementation plan, conversion of files, conduct parallel processing, discontinue old system and then plan for post implementation review.

3. Post-Implementation Review : After system is implemented and conversion is done, a review is conducted to determine, whether the system is meeting expectations and where improvements are needed. It measures the system performance against pre-defined requirements. It is an evaluation of a system.

Q20. What is the relationship between design and implementation? Explain with example.

Ans. **Relationship Between Design and Implementation :**

Design Phase : The design phase focuses on the detailed implementation of the system recommended in the feasibility study. The design phase is a transition from a user- oriented document to a document oriented to the programmers or database personnel .

The system design consists of the following steps :

1. Design the physical system.
2. Plan system implementation.
3. Devise a test and implementation plan and specify any hardware / software.
4. Update benefits, costs, conversion date and system constraints.

Implementation Phase : The implementation phase is the process of converting a new or a revised system design into an operational one. **There are three types of implementation.**

1. Implementation of a computer system to replace a manual system.
2. Implementation of a new computer system to replace an existing one.
3. Implementation of a modified application to replace an existing one using the same computer.

Conversion of design phase to implementation phase consist of the following steps :

1. Review the project plan, test documentation and implementation.
2. Convert the files.
3. Conduct parallel processing.
4. Log the computer run for reference.
5. Discontinue the old system.

6. Plan for post-implementation review.

A Simple Example : A software for Library called Library Management System.

The library management system software automates all manual activities performed in the libraries by the librarian.

To develop the software, number of phases are involved.

The first phase is the system analysis, in which initial investigation, information gathering is done for the library. It includes information related to library activities, like issue and return of books, fine details, and other categories of tasks. The control flow, data flow and relationship between member and library books is shown using structured tools like flow chart, data flow diagram and entity relationship diagram respectively. This phase also involves feasibility study.

The second phase is the systems design that translate the user-oriented document i.e., SRS for library to a document oriented to the programmers or database personnel. The logical and physical design is developed. The logical design involves reviewing the current physical system i.e., register based library system – its data flows, file content, volumes etc. It prepares input/output specification i.e., format, content, frequency of report on issue and return of books by member. It also involves database design, file organisation output, input file and screen layouts. Structured chart is drawn to show different modules like issue return, fine and their relationship in a top-down manner. The physical design defines the design specifications, that tells the programmer exactly what the library management system must do. The programmer write the necessary programs or modifies the software package that accepts input from the user, performs necessary calculation through existing file or database, produces the hard copy or displays it on a screen and maintains an updated databases at all times.

Then, comes the implementation phase. After the programs have been coded, they are tested for quality assurance. The test plans are prepared. Levels of testing are performed on different modules of library software. After testing the modules, the system is implemented in the library for operation by the librarian for user acceptance testing. Training is given to the librarian to help him/her to run the system. Implementation involves conversion i.e., converting a new system design into operation. It involves installing hardware, terminals, network, to setup library system and run it. Different kinds of conversions are there and one of them is used chosen by the organisation. Like parallel conversion means keeping the old manual library registers as well as running the library management software.

After the system is setup, it is maintained at regular intervals and modifications are done when necessary.

Q21. What is software review? Explain different types of review.

Ans. **Software Review :**

A Software Review is *"A process or meeting during which a software product is examined by project personnel, managers, users, customers, user representatives or other interested parties for comment or approval."*

Software reviews may be divided into three categories:

- **Software Peer Reviews :** Software Peer Reviews are conducted by the author of the work product.
- **Software Management Reviews :** Software Management Reviews are conducted by management representatives to evaluate the status of work done and to make decisions regarding breakdown team activities.
- **Software Audit Reviews :** Software Audit Reviews are conducted by personnel external to the software project to evaluate compliance with specifications, standards, contractual agreements or other criteria.

Different Types of Reviews :

A. Code Review
B. Pair programming
C. Inspection
D. Walk-through
E. Technical review

A. Code Review : It is systematic examination (also known as peer review of computer source code) intended to find and fix mistakes overlooked in the initial development phase, improving both the overall quality of software and the developer's skills. Code reviews can often find and remove common vulnerabilities such as format string exploits, race conditions, memory leaks and buffer overflows, thereby improving software security.

Code Review Fall Into Two Main Categories:

- **Formal Code Review :** It involves a careful and detailed process with multiple participants and multiple phases.
- **Lightweight Code Review :** It typically requires less overhead than formal code inspection, though it can be equally effective when done properly. It is often conducted as a part of the normal development process.

B. **Pair Programming :** It is a software development technique in which two programmers work together at one work station. One types in the code, while the other reviews each line of code as it is typed in. The person typing is called the 'driver'. The person reviewing is called the '**observer**' or '**navigator**'. The two programmers switch over their roles frequently.

While reviewing, the observer also considers the strategic direction of the work, coming up with ideas for improvements and likely, future problems to address.

C. **Inspection :** In software engineering, it refers to peer review of any work product by trained individuals who look for defects using a well defined process.

The goal of the inspection is for all of the inspectors to reach consensus on a work product and approve it for use in the project. Common inspected work product include software requirements specifications and test plans. The goal of inspection is to identify defects. In an inspection, a defect is any part of the work product that will keep an inspector from approving it.

D. Software Walkthrough : Software walkthrough or walk-through is a form of software peer review, in which a designer or programmer leads members of the development team and other interested parties through a software product, and the participants ask questions and make comments about possible errors, violation of development standards and other programs.

In general, a walkthrough has one or two broad objectives : to gain feedback about the technical quality or content of the document and/or to familiarize the audience with the content. A walkthrough is normally organized and directed by the author of the technical document.

E. Software Technical Review : It is a form of peer review in which "a team of qualified personnel examines the suitability of the software product for its intended use and identifies discrepancies from specifications and standards. Technical reviews may also provide recommendations of alternatives and examination of various alternatives." The purpose of a technical review is to arrive at a technically superior version of the work product reviewed, whether by correction of defects or by recommendation or by introduction of alternative approaches.

Q22. What is coding style? Describe various rules for coding a program?

Ans. **Coding Style :**

A well-written program is more easily read and understood, by both the author and by others who work with that program. The program itself should help the reader to understand, what it does quickly because, only a small fraction of the developers, if any, are maintaining the program they wrote. Bad programming style makes program difficult to understand, hard to modify, and impossible to maintain over a long period of time, even by the person who coded it originally.

Some General Rules That Usually Apply :

Names : Most variables in a program reflect some entity in the problem domain, and the modules reflect some process. Variables names should be closely related to the entity they represent, and modules names should reflect their activity. It is bad practice to choose cryptic names, just to avoid typing or totally unrelated names. It is also bad practice to use the same name for multiple purpose.

Control Constructs : As much as possible, single entry, single-exit constructs should be used. And, also should use a few standard control constructs rather than using a wide variety of constructs, just because they are available in the language.

Information Hiding : Information hiding should be supported where possible. Only the access functions for the data structures should be made visible, while hiding the data structure behind these functions.

User Defined Types : Modern languages allow user to define types like the enumerated type. When, such facilities are available, they should be exploited where

applicable.

Nesting : The different control constructs, particularly, the if-then-else, can be nested. If the nesting becomes too deep, the programs become harder to understand. In case of deeply nested if-then-else, it is often difficult to determine the, if statement to which, a particular else clause is associated. Where possible, deep nesting should be avoided, even if it means a little inefficiency.

Module Size : Large modules often will not be functionally cohesive, and too-small modules might incur unnecessary overhead. There can be no hard and fast rule about module size. The guiding principle should be cohesion and coupling.

Module Interface : As a rule of thumb, any module, whose interface has more than five parameters should be carefully examined and broken into multiple modules with a simpler interface, if possible.

Program Layout : Automated tools are available to "pretty print" a program, but it is good practice to have a clear layout of programs.

Side Effect : When a module is invoked, it sometimes has side effects of modifying the program state, such side effects should be avoided where possible, and if a module has side effects, they should be properly documented.

Robustness : A program is robust, if it does something planned even for exceptional conditions. A program might encounter exceptional conditions in such forms as incorrect input, the incorrect value of some variable, and overflow. A program should try to handle such situations.

Q23. What are the different characteristics of coding ?

Ans. There are different characteristics of coding:

1. **Simplicity :** A code should be easy to understand. Many inexperienced engineers actually take pride in writing an incomprehensive code. This should be avoided, because it hampers the understanding.
2. **Readability :** This type of code is written in simple code to be readable. So, we write simple code method, which is easily readable by the users.
3. **Good Documentation :** It enhances understandability and maintainability of software product.
4. **Changeability :** Programs should be well documented, so that further enhancements can be made without efforts.
5. **Predictability :** Program should provide expected results.
6. **Consistency in input and output :** Program should generate the expected results based on the input data and, it should not change with time.
7. **Module Independence :** It state that the modules of the program should be independent and this can be attained by making the degree of cohesion high and that of coupling low.
8. **Good Structure :** It often leads to better control over data and it makes program simple and readable. Good structure includes :
   - ➢ Use of initialized variables.
   - ➢ Do not jump into loops.
   - ➢ Terminating loops.

Q24. What are different coding technique for procedural design?

Ans. Coding-Technique for Procedural Design :

Component-level design, also called procedural design, occurs after data, architectural, and interface design have been established. The level of abstraction of the existing design model is relatively high and the abstraction level of the operational program is low.

Coding-Techniques :

(a). Graphical Design Notation :

There are so many graphical tools, such as the flowchart or box diagram, provide useful pictorial patterns that readily depict procedural detail. A flow chart is quite simple pictorially. A box id is used to indicate a processing step. A diamond represents a logical condition, and arrows shows the flow control.

Another graphical design tool, the box diagram evolved from a desire to develop a procedural design representation, that would not allow violation of the structured constructs.

(b). Tabular Design Notation :

Decision tables provide a notation, that translates action and conditions into a tabular form. The table is difficult to misinterpret and may even be used as machine readable input to a table driven algorithm. a comprehensive treatment of this design.

Basically, the table is divided into 4 sections. The

upper left-hand quadrant contains a list of all conditions. The lower left-hand quadrant contains a list of all actions that are possible based on combinations of condition. Right-hand quadrant form a matrix for a specific combination.

	Condition	1	2	3	4	5
Condition Stub	Condition 1	✓		✓		
	Condition 2		✓	✓		
Action Stub	Action					
	Action 1	✓		✓		
	Action 2		✓	✓		

(c). Program Design Language(PDL) :

Program Design Language, also called Structured English or Pseudo code, is a "pidgin language" in that, it uses the vocabulary of one language (i.e, English) and the overall syntax of another (i.e., a structured programming language).

The difference between PDL and a real programming language lies in the use of narrative text (For Example, English), embedded directly within a syntactical structure.

Q25. What do you mean by software errors? Also explain its types and prevention methods.

Ans. Software Errors : A software bug is a defect, flaw or fault in a computer program or system that produces an incorrect or unexpected result or causes it to behave in unintended ways. Most bugs arise from mistakes and errors made by people in either a program's source code or its design and a few are caused by compilers producing incorrect code. A program that contains a large number of bugs and/or bugs that seriously interfere with its functionality is said to be *buggy*.

Types of Errors :

1. **User Interface Errors:** Missing/Wrong Functions, Doesn't do what the user expects, Missing information, Misleading, Confusing information, Wrong content in Help text, Inappropriate error messages. Performance issues - Poor responsiveness, Can't redirect output, inappropriate use of key board.
2. **Error Handling:** Inadequate - protection against corrupted data, tests of user input, version control; Ignores – overflow, data comparison, Error recovery – aborting errors, recovery from hardware problems.
3. **Boundary related errors:** Boundaries in loop, space, time, memory, mishandling of cases outside boundary.
4. **Calculation errors:** Bad Logic, Bad Arithmetic, Outdated constants, Calculation errors, Incorrect conversion from one data representation to another, Wrong formula, Incorrect approximation.
5. **Initial and Later states:** Failure to set data item to zero, to initialize a loop-control variable or re-initialize a pointer, to clear a string or flag, Incorrect initialization.
6. **Control flow errors:** Wrong returning state assumed, Exception-handling based exits, Stack underflow/overflow, Failure to block or un-block interrupts, Comparison sometimes yields wrong result, Missing/wrong default, Data Type errors.
7. **Errors in Handling or Interpreting Data:** Un-terminated null strings, Overwriting a file after an error exit or user abort.
8. **Race Conditions:** Assumption that one event or task finished before another begins, Resource races, Tasks starts before its prerequisites are met, Messages cross or don't arrive in the order sent.
9. **Load Conditions:** Required resources are not available, No available large memory area, Low priority tasks not put off, Doesn't erase old files from mass storage, Doesn't return unused memory.
10. **Hardware:** Wrong Device, Device unavailable, Underutilizing device intelligence, Misunderstood status or return code, Wrong operation or instruction codes.
11. **Source, Version and ID Control:** No Title or version ID, Failure to update multiple copies of data or program files.
12. **Testing Errors:** Failure to notice/report a problem, Failure to use the most promising test case, Corrupted data files, Misinterpreted specifications or documentation, Failure to make it clear how to reproduce the problem, Failure to check for unresolved problems just before release, Failure to verify fixes, Failure to provide summary report.

Prevention Methods :

The software industry has put much effort into finding methods for preventing programmers from inadvertently introducing bugs, while writing software. These include :

1. Programming style : While typos in the program code are often caught by the compiler, a bug usually appears when the programmer makes a logical error. Various innovations in programming style and defensive programming are designed to make these bugs less likely or easier to spot. In some programming languages, so-called typos, especially of symbols or logical/mathematical operators, actually represent logic errors, since the mistyped constructs are accepted by the compiler with a meaning other than that which the programmer intended.

2. Programming techniques : Bugs often create inconsistencies in the internal data of a running program. Programs can be written to check the consistency of their own internal data, while running. If an inconsistency is encountered, the program can immediately halt, so that the bug can be located and fixed. Alternatively, the program can simply inform the user, attempt to correct the inconsistency, and continue running.

3. Development methodologies : There are several schemes for managing programmer activity, so that fewer bugs are produced. Many of these fall under the discipline of software engineering (which addresses software design issues as well). For example, formal program specifications are used to state the exact behavior of programs, so that design bugs can be eliminated. Unfortunately, formal specifications are impractical or impossible for anything but the shortest programs, because of problems of combinatorial explosion and indeterminacy.

4. Programming language support : Programming languages often include features which help programmers prevent bugs such as static type systems, restricted name spaces and modular programming among others. For example, when a programmer writes (pseudocode) LET REAL_VALUE PI = "THREE AND A BIT", although this may be syntactically correct, the code fails a type check. Depending on the language and implementation, this may be caught by the compiler or at run-time. In addition, many recently invented languages have deliberately excluded features which can easily lead to bugs at the expense of making code slower than it need be: the general principle being that, because of Moore's law, computers get faster and software engineers get slower; it is *almost always* better to write simpler, slower code than "clever", inscrutable code, especially considering that maintenance cost is considerable. For example, the Java programming language does not support pointer arithmetic; implementations of some languages such as Pascal and scripting languages often have runtime bounds checking of arrays, at least in a debugging build.

5. Code analysis : Tools for code analysis help developers by inspecting the program text beyond the compiler's capabilities to spot potential problems. Although in general, the problem of finding all programming errors, given a specification is not solvable (halting problem), these tools exploit the fact that human programmers tend to make the same kinds of mistakes when writing software.

6. Instrumentation : Tools to monitor the performance of the software as it is running, either specifically to find problems such as bottlenecks or to give assurance as to correct working, may be embedded in the code explicitly (perhaps as simple as a statement saying PRINT "I AM HERE"), or provided as tools. It is often a surprise to find where most of the time is taken by a piece of code, and this removal of assumptions might cause the code to be rewritten.

Q26. What is CASE Tool?

Ans. CASE (Computer Aided Software Engineering) Tools :

CASE is a tool, which aids a software engineer to maintain and develop a software. It assists software engineering managers and practitioners in every activity associated with the software process. They automate project management activities, manage all work products produced throughout the process and assist engineers in their analysis, design, coding and test work.

The workshop for software engineering is called an Integrated Project Support Environment (IPSE) and the tool set that fills the workshop is called Computer Aided Software Engineering (CASE) tool. CASE tools add to the software engineers tool box and automate the manual activities to improve development process.

Need of CASE Tools :

The need of CASE tools arise due to following reasons :

- Increase in the size of the software.
- Increase in complexity of the software package.

CASE Tool Technology :

CASE tool has a technology that can be split into 3 levels. These levels are:

- **Production-Process Support Technology :** This range of tools support process activities such as specification, design, implementation and testing.
- **Process Management Technology :** This range of tools supports process modeling and process management.
- **Meta CASE Technology :** These tools are used to create production process and process management support tools.

Benefits of CASE Tools :

1. Smooth transfer of information (models, programs, documents, data) from one tool to another and one software engineering step to the next.
2. Reduction in the effort required to perform umbrella activities such as Software Configuration Management, Quality Assurance and document production.
3. An increase in project control that is achieved through better planning, monitoring and communication.
4. Improved coordination among staff members who are working on a large software project.

Q27. What are the primary objectives of CASE tools ?

OR

What are the reasons for using CASE tools?

Ans. The primary reason for using a CASE tool are:

- To increase productivity.
- To help produce better quality software at lower cost.

CASE provides the software engineer with the ability to automate manual activities and to improve engineering insight. Examples of activities, which can be automated using CASE tools include :

1. The development of graphical system models as part of the requirement specification or the software design.
2. Understanding the design using a data dictionary, this holds information about the entities and relations in a design.
3. The generation of user interfaces from a graphical interface description, which is created interactively by the user.
4. Program debugging through the provision of information about an executing program.
5. The automated translation of programs from old version of a programming language to be more recent version.

Other major benefits of using CASE tools include the following :

(a) **Improved Productivity :** CASE tools provide automation and reduce the time to complete many tasks, especially those involving diagramming and associated specification. Estimates of improvements in productivity after application range from 355 to more than 200%.

(b) **Better Documentation :** By using CASE tools, vast amount of documentation are produced along the way. Most tools have revision for comments and notes on system development and maintenance.

(c) **Reduced Life Time Maintenance :** As a result of better design, better analysis and automatic code generation, automatic testing and debugging, overall systems quality improves. There is better documentation also. Thus, the net effort and cost involved with maintenance is reduced.

(d) **Improved Accuracy :** CASE tools can provide ongoing debugging and error checking which is very vital for early defect removal, which actually played a major role in shaping modern software.

(e) **Intangible Benefits :** It can be used to allow for greater user participation, which can lead to better acceptance of the new system.

(f) **Opportunity to Nonprogrammers :** With the increased movement towards object oriented technology and client server basis, programming can also be done by people who don't have complete programming background, important to understand logic and posses the ability to analyse.

Q28. Discuss CASE tool's evolution, environment and reason of its failure.

Ans. Evolution of CASE :

Earlier, CASE tools had diagrammatic capabilities, i.e., drawing Data Flow Diagram, Entity Relationship Diagram etc., that could store basic information about each of the diagrams. But, they could not integrate with one another and could not perform cross-check of diagram for validity purpose. In the next stage, CASE tools were updated and were capable of performing cross- diagram checks through a repository i.e., data dictionary. Then, CASE tool evaluation saw the integration of project management and system life cycle activities.

CASE Environment :

CASE can be as simple as a single tool that supports a specific software engineering activity or as complex as a complete environment that encompasses tools, a database, people, hardware, a network, operating systems, standards and other components.

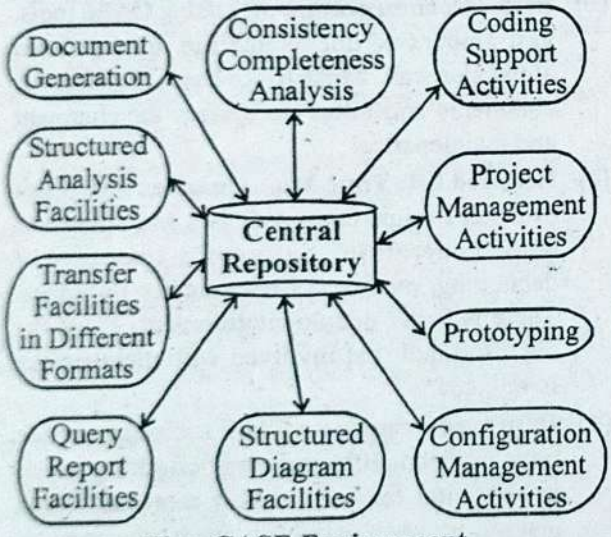


Fig : CASE Environment

CASE Tools include :

- **Graphic Tool** for data flow diagram, flow chart, entity relationship diagram, structure chart, state-transition diagram.
- **Dictionary Tool** for contents of files, inputs and outputs, properties of data elements, logic rules for processes.
- **Prototyping Tool** for external design of inputs, screens, forms, outputs.
- **Quality Checking Tool** for dictionary specification, correctness of DFD, correctness of ERD, consistency errors, completeness errors.
- **Code Generating Tools**
- **Cost Benefit Analysis Tools**
- **Project Management Tools**
- **Documentation Assemblers**
- **Test Data Generators**
- **Path Coverage Analysers.**

Some Popular CASE Tools are :

- Excelerator.
- Design Aid.
- Information Engineering Facility.
- Structured Architect.
- Analyst/Designer Toolkit.
- Profit-workbench.
- Foundation.
- Teamworks.

Failure of CASE Tools :

- Low management involvement.
- Poor documentation of tools.
- Misuse of CASE tools.
- Too much emphasis on tools as total solution

Q29. Explain various classifications of CASE tools.

Ans. CASE Tools :

Today, a wide range of CASE tools are available in the market. Different types of CASE tools address different areas of software engineering.

Most of the CASE tools offered by various vendors consist of some or most of these components. The basic advantage of such tools is that, they are user-friendly and attempt to automate the process of system analysis and design. Such automation not only helps in improving productivity of development team, but also reduces the time required to complete the project within a shorter span of time. Since, the entire information generated during the process of system analysis and design is stored in the form of repository, it is possible to link up information regarding various components to integrate them. This, along with the automation, also provides flexibility to the system.

List of CASE Tools :

Application	CASE Tool	Purpose of Tool
1. Planning	Excel spreadsheet, MS Project, PERT/CPM Network, Estimation tools.	Function applications : Planning, Scheduling, Control.
2. Editing	Diagram editors, text editors, word processors.	Speed and efficiency.
3. Testing	Test data generators, file comparators.	Speed and efficiency.
4. Prototyping	High-level modelling language, user interface generators.	Confirmation and certification of RDD & SRS.
5. Documentation	Report generators, publishing imaging, PPT presentation.	Faster structural documentation with quality of presentation.
6. Programming & Language processing integration	Program generators, code generators, compilers, interpreters, interface connectivity.	Programming of high quality with no errors, system integration.
7. Templates	–	Guided systematic development.
8. Reengineering tools	Cross – reference system, program restructuring systems.	Reverse – engineering to find structure, design and design information.
9. Program analysis tools	Cross – reference generators, static analyzers, dynamic analyzers.	Analyzes risks, functions, features.

Classification of CASE Tools :

CASE Tools can be classified on the basis of :

1. Based on Function Criteria.
2. Based on Problems on which these CASE Tools.

1. Based on Function Criteria :

- Process Modeling and Management Tools.
- Project Planning Tools.
- Risk Analysis Tools.
- Project Management Tools.
- Requirement Tracing Tools.
- Documentation Tools.
- System Software Tools.
- Quality Assurance Tools.
- Software Configuration Management Tools.
- Metrics and Measurement Tools.
- Analysis and Design Tools.
- Interface Design and Development Tools.
- Programming Tools.
- Integration and Testing Tools.
- Test Management Tools.
- Re-engineering Tools.

2. Based on Problems on which these CASE Tools :

This classification is according to the CASE problems on which these focus :

- **Front-End CASE Tools or Upper CASE Tools:** These tools deal with high-level design, specification and analysis of software and requirements.

- **Back-End CASE Tools or Lower CASE Tools :** These deal with the detailed design, coding, assembly and testing of software.
- **Maintenance Tools :** These may assist in tracking bug fixes and enhancement requests, porting to new requests or performing new releases.
- **Support-Software and Framework :** These provide basic functionality required in above types of tools.

Another Classifications of CASE Tools :

CASE tools can be broadly classified into the following categories :

(a) System planning, analysis and design tools.
(b) System documentation tools.
(c) System construction tools.
(d) Data-base management system.
(e) Fourth generation languages.
(f) Program templates.
(g) Testing and implementation tools.

(a) System Planning, Analysis and Designing Tools : These tools are meant for improving the productivity of systems analyst. These help in planning the system, system analysis and designing process and management of the project for system development. These also help the system analysts to design data dictionaries, which links these processes to each other in such a way, that any modification during the process makes changes in the dependent elements of the process.

(b) System Documentation Tools : These tools are very useful in documenting the information collected during the system analysis and the specification made during the systems design. These tools aim at providing help and automating the process of drawing the data flow diagrams, structure charts, system flowchart, Entity Relationship charts and maintaining data dictionaries.

(c) System Construction Tools : These tools aim at enhancing the speed of generating computer programs for the information system. It is achieved by :

(i) Automating the process of generating codes for programs;
(ii) Reusing the existing codes generated for other applications by permitting modifications conveniently;
(iii) Improving the productivity of the programmer.

The code generation and screen generation tools automatically generate codes in high level language such as COBOL or C++ for specified procedures and input/output screens. Code translation tools convert a program written in one high level language to another high level language. Code restructuring tools reorganise programs into a better structured programs, so that changes can be made quickly and conveniently. Such restructuring makes these programs more flexible and thus, reusable in future.

(d) Database Management System : These tools help in developing, organising, updating and maintaining database. Such databases are organised in such a way that different programs can share the data contained in these databases. This eliminates the problem of data duplication and helps in reducing redundancy of data. These systems are also very useful for developing prototypes of application. In fact, prototyping became a popular option due to facilities available in Database Management System.

(e) Fourth Generation Languages : These tools offer user-friendly programming environment with more powerful commands for procedures that would otherwise require long programs in high-level languages. They improve the productivity of the programmer and help in developing prototypes for different system modules.

(f) Program Templates : These tools are pre-written programs for common procedures. These templates are easily modifiable to suit specific applications.

(g) Testing and Implementation Tools : These tools provide for online debugging for removing program and data errors. They manage libraries of programs and import or export data for other information system.

Q.50. What are merits and demerits of CASE tools?

Ans. **Merits of Computer Aided Software Engineering Tools :**

(a) **Standardization :** CASE tools make it imperative to use standard methods in each stage of application development. Such standardization helps in maintenance of software and in reducing system development life cycle time.

(b) **Rigour :** The rigour is the ability of the system to always produce a current and predictable solution, as well as to provide correct, consistent and complete information. CASE tools helps in building up higher level of rigour in information system.

(c) **Integration :** CASE tools generally, store all the information regarding systems design and specification in the form of a repository. This information helps in integrating each component of the application with orders.

(d) **Automation :** One of the major benefits of CASE tools is the automation of each activity in the process of system development life cycle.

(e) **Graphics :** The extensive use of graphics to represent information regarding the processes in application and their specifications help in making them more understandable. With the help of animation techniques, the processes are properly displayed for improving communication between the development team and the user of the system.

Conclusion : These merits helps the organisation to foster development and more productivity.

Limitations or Demerits of CASE Tools :

After seeing the various merits of CASE tools, one is forced to see the other side, which may result in loss, if proper care is not exercised. The following are the demerits :

(a) **Expensive Tools :** Good CASE tools are quite expensive. These cost anywhere between 20,00,000 and the price is an important indicator of functionality of these tools. The good and functional ones require huge investment in these tools.

(b) **Risk of Tool Driven Solutions :** The application development process is likely to be tool driven than solution driven. This happens because these tools are quite effective and convenient, if the application design fits into their structure and any deviation from the required structure becomes very difficult, or at least inconvenient.

(c) **Poor Documentation :** Most of the CASE tools come with inadequate documentation as regards the way, they function. It is natural as these tools are complex software systems modified quite regularly to incorporate changing industry standards. As a result, it becomes very difficult to use them to their full potential.

(d) **Lack of Standard Methods :** It is very likely that the business processes do not follow any standard methodologies. The CASE tools are designed assuming the presence of well defined, standard methodologies in business processes. As a result, they might not be suitable for enterprises, that do not follow standard methodologies in business processes.

(e) **Lack of follow-up :** Once the application of CASE tools is taken in many cases, company using these tools do not bother about the latest changes in the tools.

(f) **Cost :** The cost of outfitting every software developer with a preferred CASE tool kit can be quite high.

(g) **Learning Curve :** Users need time to learn the technology. Therefore, programmer's productivity may fall in the initial phase of implementation.

(h) **Tool Mix :** Important to make appropriate selection of tool mix to get cost advantage. CASE integration and data integration across all platforms is also very important. The ability to share the results of work done on one CASE tool with another CASE tool is perhaps the most important type of CASE integration.