## What do you mean by software?

Software is nothing but a collection of computer programs and related documents that are intended to provide desired features, functionality and better performance.

Software products may be-

Generic – That means developed to be sold to a range of different customers.

Custom – That means developed for a single customer according to their specification.

## Is there any difference between software engineering and computer science? Justify your answer

There are difference between software engineering and computer science. Some difference are given bellow:

| Software engineering | Computer Science |
|---|---|
| The software engineering is concerned with the practical problems of product software. | The computer science deals with the theories and methods used by the computers and software systems. |
| Focuses on how to design and build software in teams. | Focuses on the concept and technologies involved with how to make a computer do something |
| Software engineering concentrates on the process, specification, analysis, design, implementation and maintenance of software. | Computer Science concentrates on data, data transformation, and algorithms. |
| Software engineering is more structured and disciplined. | Computer science is less structured and disciplined. |
| Software Engineering is a sub category of Computer Engineering. | Computer science is the root of computer based voation. |

## What is software engineering?
## Or,
## Give IEEE definition of software engineering.

Software engineering is a discipline in which theories, methods and tools are applied to develop professional software product.

IEEE defines software engineering as following:

Software Engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

The definition of software engineering is based on two terms:

Discipline: for finding the solution to the problem an engineer applies appropriate theories, methods and tools.

Product: The software product gets developed after following systematic theories, methods and tools along with the appropriate management activities.

## State the attribute of good software.

## Or,

## State and explain some qualities that are used to assess software

Satisfying the user requirement is not the only goal of good software. There are some essential attributes of good software and those are

**Maintainability:** Sometimes there is a need to make some modification in the existing software. A good software is a software which can be easily modified in order to meet the changing needs of the customer.

**Usability:** It is the ability of the software being useful. For making the software useful it is necessary that is necessary that it should have proper GUI and documentation.

**Dependability:** The dependability is a property that includes reliability, security and safety of software. In other words the developed software product should be reliable and safe to use; it should not cause any damage or destruction.

**Efficiency:** The software should be efficient in its performance and it should not waste the memory.

## What are the objectives of software engineering?

While developing software following are common objectives.

1. Satisfy user requirements.
2. High reliability.
3. Low maintenance costs.
4. Delivery on time.
5. Low production costs.
6. High performance.
7. Ease of reuse

## What are the differences between software engineering and system engineering?

| System engineering | Software engineering |
|---|---|
| System engineering is concerned with all aspects of computer-based systems development including hardware, software and process engineering. | Software engineering is part if this process concerned with developing the software infrastructure, control, applications and databases in the system. |
| System engineering is not part of software engineering | Software engineering is a part of system engineering. |
| The system engineering discipline covers the development of total systems, which may or may not include software. | The software engineering discipline covers the development of software systems. |
| Focuses on transforming customer needs, expectations, and constraints into production solutions and supporting those product solutions throughout the product life cycle. | Focuses on applying systematic, disciplined, and quantifiable approaches to the development, operation and maintenance of software. |

## State and Explain the generic activities followed in all the software process.

A set of activities whose goal is the development or evolution of software is called Generic software process. Generic activities in all software processes are:

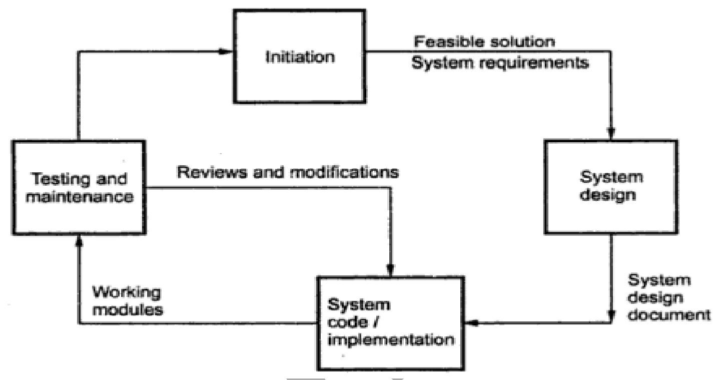
- Specification-what the system should do and its development constraints
- Development-production of the software system
- Validation-checking that the software is what the customer wants
- Evolution-changing the software in response to changing demands.

## Describe each step of System development life cycle (SDLC) / software development life cycle.

The system development life cycle is the logical process of developing any system. Using SDLC one can develop a system which satisfies customer needs, can be developed within the predefined schedule and cost. Normally the system analyst makes use software development life cycle for developing the information systems. The SDLC is a linear or sequential model in which output of previous phase is given as input to next sub-sequence stage.

Various phases of software development life cycle are –

1. Initialization
2. Requirement gathering and analysis
3. Design
4. Coding or implementation
5. Maintenance



Each step is described following:

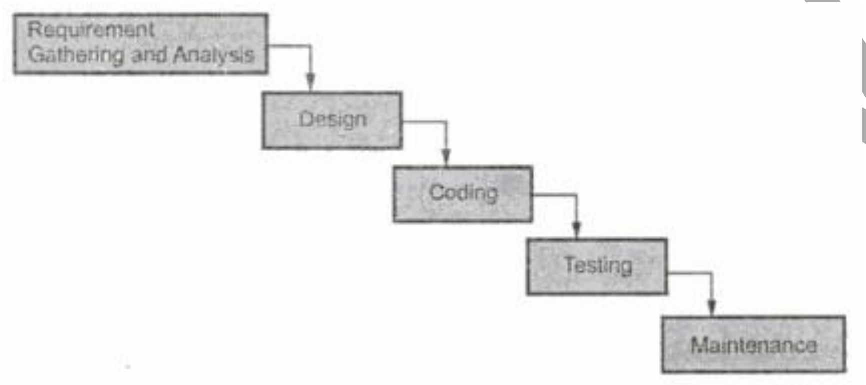
**Initialization:** This is the first stage in SDLC in which it is checked whether or not the system will be within the given budget and whether or not the system will be developed in specific time. Sometime preliminary survey is made to find the alternative solution. This is the phase of software development in which the development process is planned out.

**Requirement gathering and analysis:** Once it is decided to develop the system, the first essential thing is to gather and analyze the requirements of the system. The information domain, function, behavioral requirements of the system are understood.

**Design:** After collecting and analyzing all the necessary requirements, the design architecture for the system is prepared. Sometimes this system design can be discussed with the customer for ensuring his needs and demands.
**Coding:** after getting approved system design, one can develop the code for the system using sme suitable programming language. During coding the design is translated into machine readable form.
**Testing:** Testing is a phase that begins after the coding or the implementation phase. The purpose of testing is to uncover errors, fix bugs and meet the customer requirements.
**Maintenance:** When the system is installed and put in practical use then errors may get introduced, correcting such errors and putting it in use is the major purpose of the phase

## Explain waterfall model with merits and demerits.

The waterfall model is also called linear-sequential model or classic life cycle model. It is the oldest software paradigm. This model suggests a systematic, sequential approach to software development.
The software development starts with requirements gathering phase. Then progresses through analysis, design, coding, testing and maintenance.



**Requirement Gathering and Analysis:** In this phase the basic requiremets of the system must be understood by software engineer called software analyst. The information domain, function, behavioral requirements of the system are understood. All these requirements are then well documented and discussed further with the customer, for reviewing.
**Design:** Th design is an intermediate step between requiremetns analysis and coding.
Design focuses on program attributes such as-

- Data structure
- Software architecture
- Interface representation
- Algorithm details.

The requirements are translated in some easy to represent form using which coding can be done effectively, and efficiently.
**Coding:** Coding is a step which design is translated into machine readable form.
**Testing:** Testing begins when coding is done. While performing testing the major focus is one logical terminals of the software. The testing ensures execution of all paths, functional behaviors. The purpose of testing is to uncover errors, fix, the bugs and meet the customer requirements.

**Maintenance:** Maintenance is the longest life cycle phase. When the system is installed and put in practical use, then errors may get introduced, correcting such errors and putting it in use is the major purpose. Similarly enhancing system's service as new requirements are discovered is again.
Merits of waterfall model:

- This model is simple and easy to understand and use.
- It is easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process.
- In this model phases are processed and completed one at a time. Phases do not overlap.
- Waterfall model works well for smaller projects where requirements are very well understood.

Demerits of Waterfall model:

- Once an application is in the testing stage, it is very difficult to go back and change something that was not well-thought out in the concept stage.
- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing.

## What are the key challenges facing in software engineering?

The key challenges facing software engineering are:

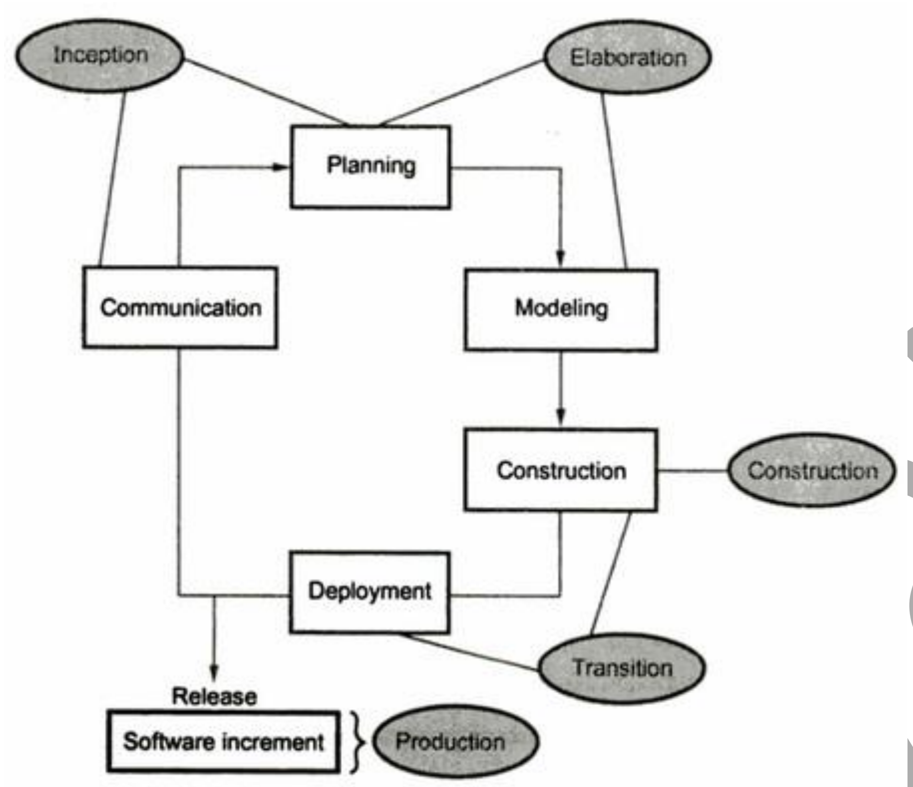
- **Copying with legacy:** Old, valuable systems must be maintained and updated. Hardware is evolved faster than software. If original developer have moved on managing maintaining or integrating of software becomes a critical issue.
- **Heterogeneity challenge:** Sometimes systems are distributed and include a mix of hardware and software systems must cleanly integrate with other different software systems, built by different organizations and teams using different hardware and software platforms.
- **Delivery time challenge:** There is increasing pressure for faster delivery of software. As the complexity of systems that we develop increases, this challenge becomes harder.

## Describe rational Unified process (RUP) model including core disciplines and phases.

The Unified process is a framework for object oriented models. This model is also called rational Unified process (RUP). There are 5 phases of unified process model and those are

- Inception
- Elaboration
- Construction
- Transition
- Production

Let us discuss all phases as following



**Inception:** In this phase there are two major activities that are conducted: communication and planning. By having customer communication business requirements can be identified. Then a rough architecture of the system is proposed. Using this rough architecture it then becomes easy to make a plan for the project.

**Elaboration:** Elaboration can be done using two activities: planning and modeling. In this phase using case model, analysis model, design model, implementation model and development model an architectural model has represented and a plan is created.

**Construction:** The main activity in this phase is to make the use cases operational. The analysis and design activities that are started in elaboration phase are completed and a source code is developed.

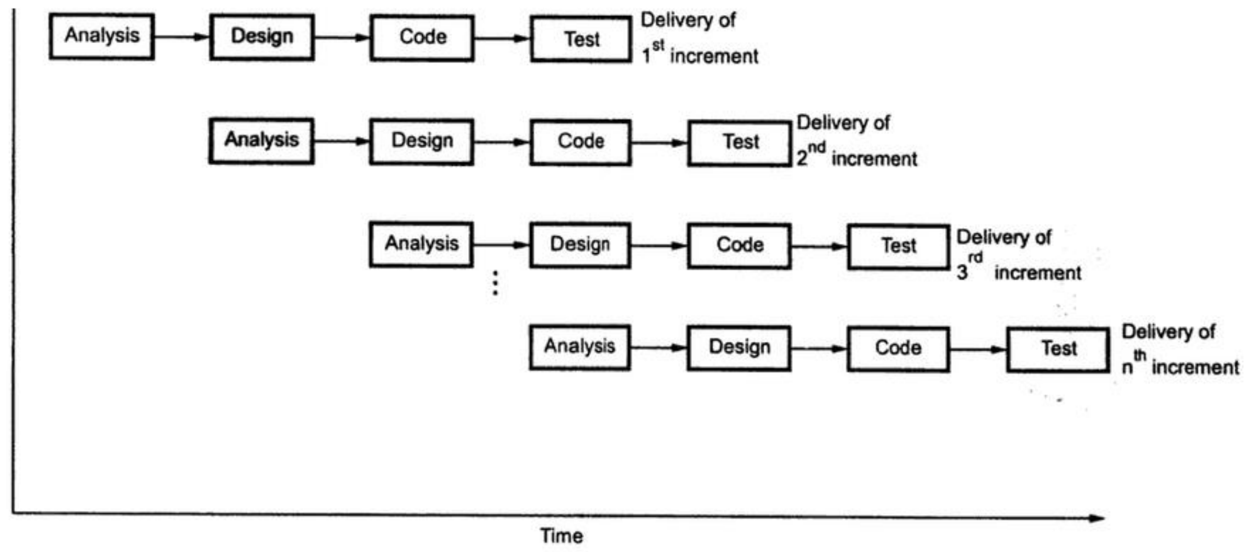
**Transition:** In this phase all the activities that are required at the time of deployment of the software product are carried out. Beta testing, user feedback report is used to remove defects. This makes the software more usable at the time of release.

**Production:** This is the final phase of this model. In this mainly maintenance activities are conducted in order to support the user in operational environment.

## Explain incremental software process model with merits and demerits.

The incremental model has same phase that are in waterfall model. But it is iterative in nature. The incremental model has following phases.

1. Analysis
2. Design
3. Code
4. Test


Analysis
Design
Code
Test
Delivery of 1st increment
Delivery of 2nd increment
Delivery of 3rd increment
Delivery of nth increment
Time

The incremental model delivers series of releases to the customer. These releases are called increments. More and more functionality is associated with each increment. The first increment is called core product. In this release the basic requirements are implemented and then in subsequent increments, new requirements are added.

**[Add description of phases from waterfall model]**

Advantages of Incremental model:

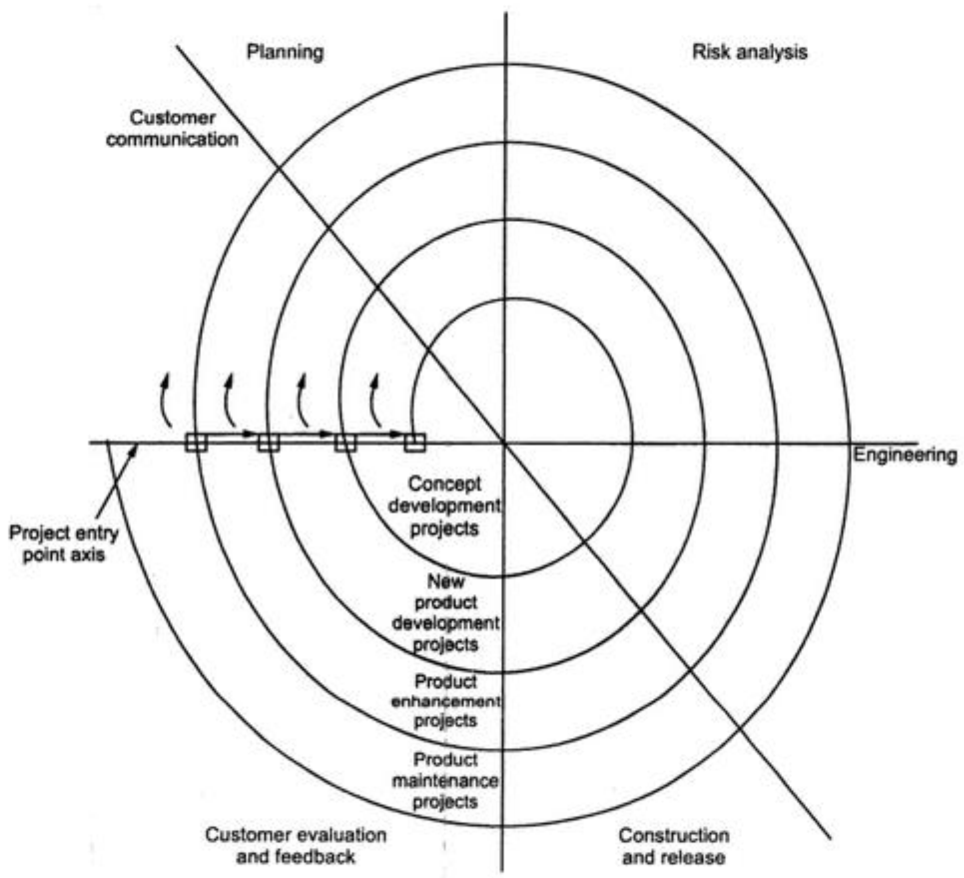
- Generates working software quickly and early during the software life cycle.
- This model is more flexible – less costly to change scope and requirements.
- It is easier to test and debug during a smaller iteration.
- In this model customer can respond to each built.
- Lowers initial delivery cost.
- Easier to manage risk because risky pieces are identified and handled during it’d iteration.

Disadvantages of Incremental model:

- Needs good planning and design.
- Needs a clear and complete definition of the whole system before it can be broken down and built incrementally.
- Total cost is higher than waterfall.

## Describe Spiral model.

Spiral model possess the iterative nature of prototyping model and controlled and systematic approaches of the linear sequential model. This model gives efficient development of incremental version of software. In this model, the software is developed in series of increments. The spiral model is divided into a number of framework activities. These framework activities are denoted by task regions. Usually there are six task regions.


Planning
Risk analysis
Customer communication
Engineering
Project entry point axis
Concept development projects
New product development projects
Product enhancement projects
Product maintenance projects
Customer evaluation and feedback
Construction and release

The task regions can be described as:

i. Customer Communication: In this region it is suggested to establish customer communication.
ii. Planning: All planning activities are carried out in order to define resources time line and other project related activities.
iii. Risk Analysis: The task required to calculate technical and management risks carried out.
iv. Engineering: In this task region, tasks required to build one or more representations of applications are carried out.
v. Construct and release: All the necessary tasks required constructing, testing, installing the application are conducted. Some tasks that are required to provide user support are also carried out in this task region.
vi. Customer evaluation: Customer's feedback is obtained and based on customer evaluation required tasks are performed and implemented at installation stage.

In each region, numbers of work tasks are carried out depending upon the characteristics of project. For instance concept development project will start at core of spiral and will continue along the spiral path. If the concept has to be developed inti actual project the at entry point the product development process starts. This entry point is called project entry point.

Advantage:

1) Requirement change can be made every stage.
2) Risks can be identified and rectified before the got problematic.
3) This is realistic approach to development of large scale system.

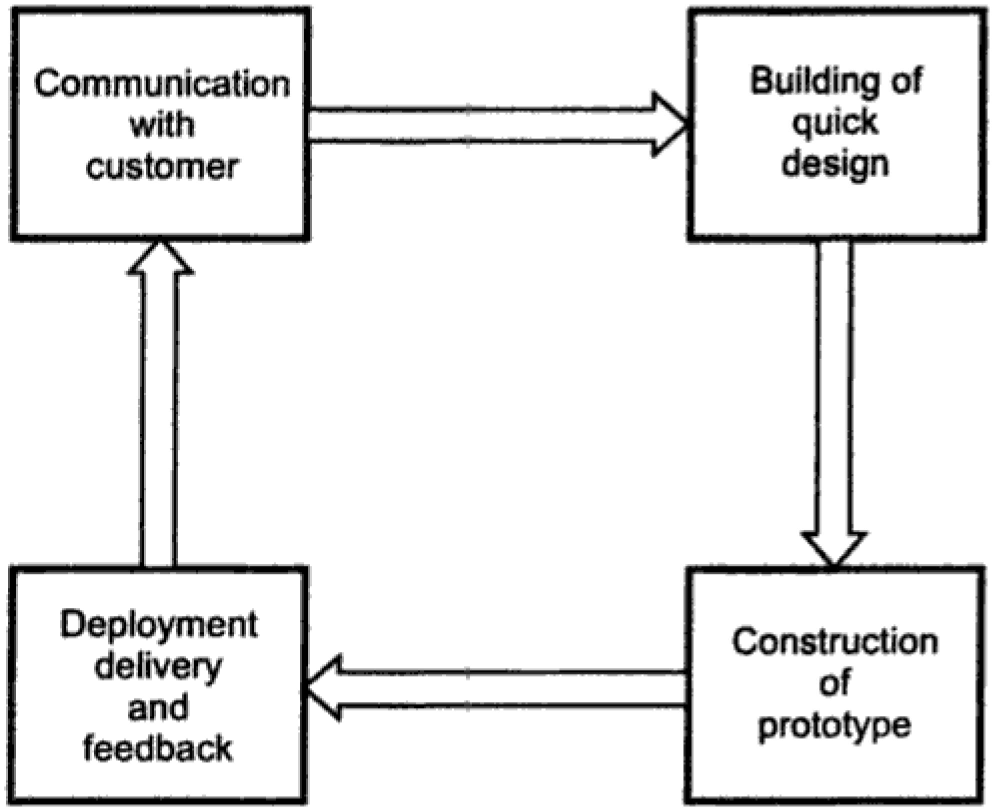
Disadvantage:

1) It is based on customer communication. If the communication is not proper the software product that gets developed will not be up to the mark.
2) It demands considerable risk assessment. If the risk assessment is done properly then only the successful product can be obtained.

## Give a brief description of software prototyping and discuss the various prototyping techniques in a nutshell.

Prototyping is the process of implementing the presumed software requirements with an intention to learn more about the actual requirements or alternative design that satisfies the actual set of requirements. It is likely news interview where journalists presume some questions to ask. Here in prototyping, requirement analysts presumed software requirements with an intention to learn more about the actual requirements.

In prototyping model initially the requirement gathering is done. Developer and customer define overall objectives. Then a quick design is prepared. From the quick design a prototype is prepared. Then customer evaluates the prototype and gives a feedback. Using this feedback, developer gets some requirement again and builds a quick design. Prototyping model cycle continue till final delivery.


Communication with customer
Building of quick design
Construction of prototype
Deployment delivery and feedback

There are different types of software prototypes used in the industry. Following are the major software prototyping types used widely:

**Throwaway/Rapid Prototyping:** Throwaway prototyping is also called as rapid or close ended prototyping. This type of prototyping uses very little efforts with minimum requirement analysis to build a prototype. Once the actual requirements are understood, the prototype is discarded and the actual system is developed with a much clear understanding of user requirements.

**Evolutionary Prototyping:** Evolutionary prototyping also called as breadboard prototyping is based on building actual functional prototypes with minimal functionality in the beginning. Using evolutionary prototyping only well understood requirements are included in the prototype and the requirements are added as and when they are understood.

**Incremental Prototyping:** Incremental prototyping refers to building multiple functional prototypes of the various sub systems and then integrating all the available prototypes to form a complete system.

**Extreme Prototyping:** Extreme prototyping is used in the web development domain. It consists of three sequential phases. First, a basic prototype with all the existing pages is presented in the html format. Then the data processing is simulated using a prototype services layer. Finally the services are implemented and integrated to the final prototype.

Advantage:

- Increased user involvement in the product even before implementation
- Since a working model of the system is displayed, the users get a better understanding of the system being developed.
- Reduces time and cost as the defects can be detected much earlier.
- Quicker user feedback is available leading to better solutions.
- Missing functionality can be identified easily
- Confusing or difficult functions can be identified

Disadvantage:

- Risk of insufficient requirement analysis owing to too much dependency on prototype
- Users may get confused in the prototypes and actual systems.
- Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans.
- Developers may try to reuse the existing prototypes to build the actual system, even when its not technically feasible
- The effort invested in building prototypes may be too much if not monitored properly

## Write down the principle of agile process method.

Agile method is adopted from waterfall approach because it provides a structured, sequential and phased lifecycle. It places emphasis on face to face communication and collaboration with teams of designers, developers and testers working with iterations of the production. This iteration is then demonstrated to stakeholders and feedback incorporated into the subsequent iteration. It follows 12 principles. Those are

i.) Highest priority is to satisfy the customer through early and continuous delivery of valuable software.

ii.) Welcome changing requirements, even late in development.

iii.) Deliver working software frequently.

iv.) Business people and developers must work together daily throughout the project.

v.) Provide developers the environment and support the need and trust them to get the job done.
vi.) The most efficient and effective method of conveying information to and within a development team is face to face conversation.
vii.) Working software sis the primary measure of progress.
viii.) The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
ix.) Continuous attention to technical excellence and good design enhances agility.
x.) The art of maximizing the amount of work not done.
xi.) The best architectures, requirements, and designs emerge from self-organizing teams.
xii.) At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

## Discuss about the professional and ethical responsibilities of a software engineer.

Software engineering involves wider responsibilities than simply the application of technical skills. Software engineers must behave in an honest and ethical responsible way if they are to be respected as professionals. Ethical behavior is are than simply upholding the law. Some issues of professional and ethical responsibilities are

- **Confidentiality:** Engineers should normally respect the confidentiality of their employers or clients irrespective of whether or not a formal confidentiality agreement has been signed.
- **Competence:** Engineers should not misrepresent their levels of competence. They should not knowingly accept work which is outwith their competence.
- **Intellectual property rights:** Engineers should be aware of local laws governing the use of intellectual property such as patents, copyright etc. They should be careful to ensure that the intellectual property of employers and clients is protected.
- **Computer misuse:** Software engineers should not use their technical skills to misuse other people's computers. Computer misuse ranges from relatively trivial to extremely serious.

## Write down the umbrella activities of software engineering.

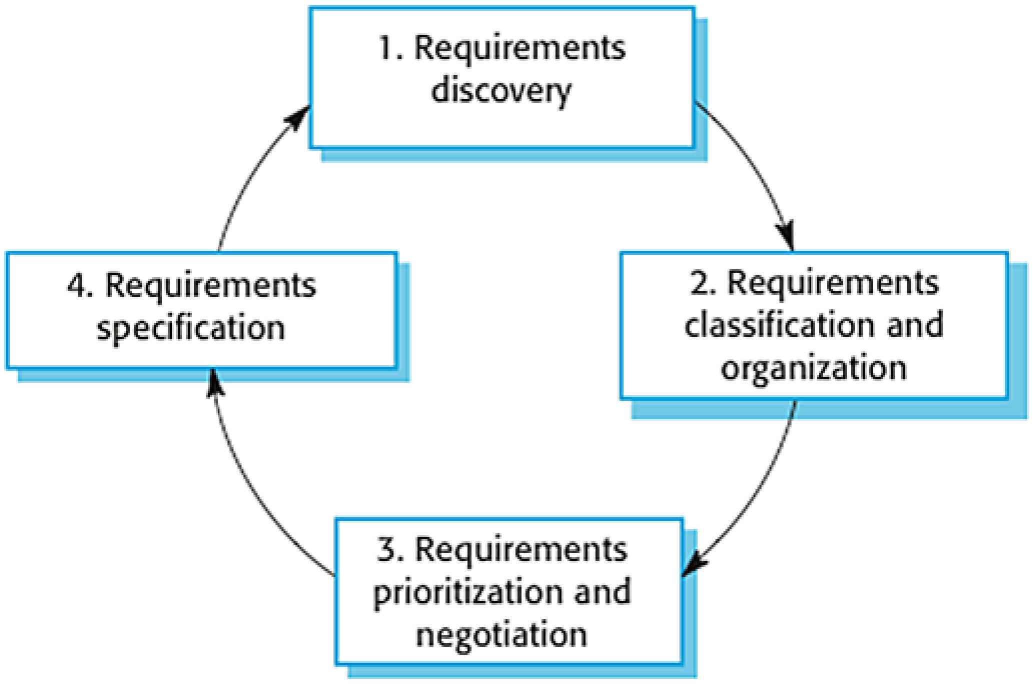
Umbrella activities: The umbrella activities are some steps in software process which focuses on project management, tracking and control. The umbrella activities are:

1. **Software project tracking and control:** This is an activity in which software team can assess progress and take corrective action to maintain schedule.
2. **Risk management:** The risks that may affect project outcomes or quality can be analyzed.
3. **Software quality assurance:** These activities required to maintain software quality.
4. **Formal technical reviews:** It is required to assess engineering work products to uncover and remove errors before they propagate to next activity.
5. **Software configuration management:** managing of configuration process when any changes in the software occur.
6. **Work product preparation and production:** The activities to create models, documents, logs, forms and lists are carried out.

7. **Re-usability management:** It defines criteria for work product reuse.
8. **Measurement:** In this activity, the process can be defined and collected. Also project and product measures are used to assist he software team in delivering the required software.

## Brief the process activities of requirements elicitation and analysis with figure.

The requirement elicitation means requirements discovery. This involves technical staff working with customers to find out about the application domain, the services that the system should provide and the system's operational constraints. Process activities of requirement elicitation and analysis are as following:


1. Requirements discovery
2. Requirements classification and organization
3. Requirements prioritization and negotiation
4. Requirements specification

**Requirements discovery**: Interacting with stakeholders to discover their requirements. Domain requirements are also discovered at this stage.
**Requirements classification and organization**: Groups related requirements and organizes them into some class.
**Prioritization and negotiation**: Prioritizing requirements and resolving requirements conflicts.
**Requirements documentation**: Requirements are documented and input into the next round of the spiral.

## Explain non-functional and functional requirements of software.
## Or,
## What are the different types of user requirement process?

User requirements are two of types; functional and non-functional.

Non-functional requirement: The non-functional requirements define system properties and constraints. Various properties of a system can be: Reliability, response time, storage requirements. And constraints of the system can be Input and output device capability, system representations. Non-functional requirements are three of types.

Product requirement: These requirements specify how a delivered product should behave in a particular way. For instance: execution speed, reliability.

Organizational requirements: The requirements which are consequences of organizational policies and procedures come under this category. For instance: process standards used implementation requirements.

External requirements: These requirements arise due to the factors that are external to the system and its development process.

In short, non functional requirements arise through:

a) User needs
b) Because of budget constraints
c) Organizational policies
d) The need for interoperability with other software or hardware systems.
e) Because of external factors such as safety regulations.

Functional requirement: Functional requirement should describe all the required functionality system services. Functional requirements are heavily dependent upon the type of software expected users and the type of system where the software is used. This may be high-level statements of what the system should do but functional system requirements should describe the system services in detail.It may be:

- Calculations
- Technical details
- Data manipulation
- Data processing
- Other specific functionality

## State and explain maturity levels in SEIs CMM

The Software Engineering Institute (SEI) has developed a process model emphasizing process maturity. It is called capability maturity model (CMM). This is used in assessing how well an organization's processes allow to compete and manage new software projects.

There are 5 levels in SEIs capability maturity model (CMM) to measure quality. These are:

I. Initial
II. Repeatable
III. Defined
IV. Managed
V. Optimizing

These levels are described following:

**Initial:** Few processes are defined and individual efforts are taken.

**Repeatable:** To track cost schedule and functionality basic project management processes are established. Depending on earlier successes of projects with similar applications, necessary discipline can be repeated.

**Defined:** The process is standardized, documented and followed in developing and supporting software.

**Managed:** Both the software process and product are quantitatively understood and controlled using detailed measures.

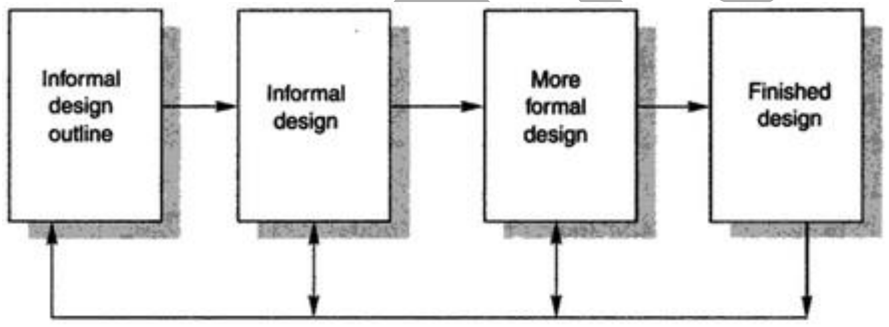
**Optimizing:** Establish mechanisms to plan and implement change.

## What are the objectives of software design?

Design is essentially the bridge between requirements specification and the final solution for satisfying the requirements. The objective of the design process is to produce a model or representation of a system, which can be used later to build that system. The produced model is called the design of the system. The design of a system is essentially a blueprint or a plan for a solution for the system. The purpose of the software design phase is to specify the components for this transformation function, so that each component is also a transformation function.

## How do we transform an informal design to a detailed design?

Software designer do not arrive at a finished design document immediately but developed the design iteratively through a number of different phases. The design process involves adding details as the design is developed with consistent backtracking to correct earlier, less formal, designs. The starting point is an informal design which is refined by adding information to make it consistent and complete detailed design.



## How is software design different from coding?

"Design is not coding and coding is not design" this is one of the design principles that are to be applied while designing the software systems. This means that level of abstraction of design model should be higher that the source code. At the time of coding, whatever design, decisions are made can be implemented. Hence design should be such that it can be transformed into source code. Similarly, during coding it is not desired to do the design to fulfill the requirements. This is how software design is different from coding.

## What problem arises if two modules have high coupling?

Coupling means the interconnection of dissimilar modules with each other or we can say, it tells about the interrelationship of dissimilar modules of a system. A system with high coupling means there are strong interconnections among its modules. If two modules are concerned in high coupling, it means their interdependence will be very high. Any changes applied to single module will affect the functionality of the other module. As per the type of coupling, the test coverage and structural analysis becomes more difficult especially when one component is having any issue or problem other module also faces some problem.

## What are the factors consider in the case of requirement validation?

Requirement validation is a process in which it is checked that whether the gathered requirements represent the same system that customer really wants. Following factors are consider in requirement validation:

**Validity:** Does the system provide the functions which best support the customer's needs?
**Consistency:** Are there any requirements conflicts?
**Realism:** Can the requirements be implemented according to budget technology?
**Verifiability:** Can the requirements be checked?

## Explain different types of design principles of software.

Software design principles represent a set of guidelines that helps us to avoid having a bad design. General design principles are

a) Modularization: Modularization is the process of continuous decomposition of the software system until fine-grained components are created.
b) Abstraction: Abstraction is a view of an object that focuses on the information relevant to a particular purpose and ignores the remainder of the information.
c) Encapsulation: Encapsulation deals with providing access to services of abstracted entities by exposing only the information that is essential to carry out such services while hiding details of how the services are carried out.
d) Coupling: Coupling refers to the manner and degree of interdependence between software modules. Measurement of dependency between units.
e) Cohesion: The manner and degree to which the tasks performed by single software modules related to one another. Measures how well design units are put together for achieving a particular tasks.
f) Separation of interface and implementation: The principle involves defining a component by specifying a public interface that is separate from the details of how the component is realized.
g) Completeness and sufficiency: Completeness measures how well designed units provide the required services to achieve the intent. Sufficiency measures how well the designed units are at providing only the services that are sufficient for achieving the intent.

## Define requirement engineering.

Requirement engineering is the process of establishing the services that the customer requires from a system and the constraints under which it operates and is developed.

## What do you know about software requirements specification (SRS)?

A software requirements specification (SRS) is a document that captures complete description about how the system is expected to perform. The SRS is useful in estimating cost, planning team activities, performing tasks and tracking the team's progress throughout the development activity. The standard template for writing SRS is given below:

**Document Title**

Author(s)

Affiliation

Address

Date

Document Version

1. **Introduction**
    a. Purpose of this document: Describe the purpose of the document.
    b. Scope of this document: Describe the scope of this requirement, schedules and costs.
    c. Overview: Provides a brief overview of the product defined as a result of the requirements elicitation process.
2. **General Description:** Describes the general functionality of the product such as similar system information, user characteristics, user objective, general constraints placed on design team.
3. **Functional requirement:** This section lists the functional requirements in ranked order. A functional requirement describes the possible effects of a software system.

**Short functional requirement**

    a. **Description:** A full description of the requirement.
    b. **Criticality:** Describes how essential this requirement is to the overall system.
    c. **Technical issues:** Describes any design or implementation issues involved in satisfying this requirement.
    d. **Cost and schedule:** Describe the relative or absolute costs of the system.
    e. **Risks:** Describe the circumstances under which this requirement might not able to be satisfied.
    f. **Dependencies with other requirements:** Describes interactions with other requirements.
    g. **Any other appropriate.**

4. **Interface Requirements:** This section describes how the software interfaces with other software products r user for input or output.
    a. User interface: Describes how this product interfaces with the user.
    b. GUI: Describe the graphical user interface of present.

c. CLI: Describes the command line interface of present.
d. API: Describes the application programming interface if present.
e. Hardware interface: Describes interface to hardware devices.
f. Communications interfaces: Describes network interfaces.
g. Software interfaces: Describes any remaining software interfaces not included above.

5. **Performance Requirements:** Specifies speed and memory requirements.
6. **Design Constraints:** Specifies any constraints for the design team such as software or hardware limitations.
7. **Other non functional attributes:** Specifies security, binary compatibility, reliability, maintainability, Portability, extensibility, reusability etc.
8. **Operational Scenarios:** This section describes a set of scenarios that illustrate from user's perspective.
9. **Preliminary Schedule:** This section provides an initial version of the project plan.
10. **Preliminary budget:** This section provides an initial budget details.
11. **Appendices:** Definitions, acronyms, abbreviations and references are provided.

## Why software architecture is needed?

The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements and the relationships among them. Here are the three main reasons why good software architecture is so important when it comes to development.

**A basis for communication:** Software architecture is a sort of plan of the system for the understanding and the communication between all the stakeholders. In fact, it makes it easier to understand the whole system and therefore makes the decisions process more efficient.

**The earliest decisions:** The first decisions taken are at this stage. Those early decisions have a huge importance on the rest of the project and become very difficult to change the more we advance in the process

**Transferability of the model:** Software architecture defines the model of the software and how it will function. Having it makes it possible to reuse this model for other softwares; code can be reused as well as the requirements. All the experience we get while doing that architecture is also transferred. This mean that we know and can reuse the consequences of the early decisions we took on the first place.

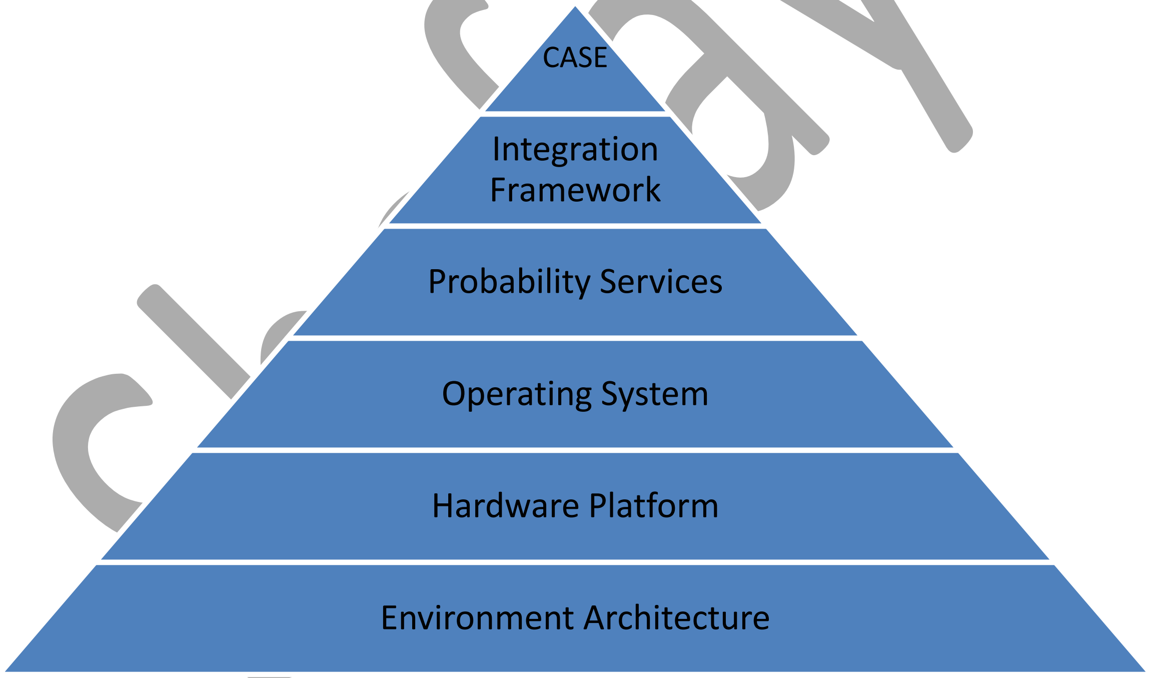
In other words, the architecture will define the problems you might encounter when it comes to implementation. It also shows the organizational structure and makes it much easier to take decisions and manage all sort of change.

## Describe layer based software architecture model.

Software engineering is a layered technology. Any software can be developed using these layered approaches. Various layers on which the technology is based are quality focus layer, process layer, method layer, tools layer.
Quality focus: A disciplined quality management is backbone of software engineering technology.
Process layer: Process layer is a foundation of software engineering. Basically, process defines the framework for timely delivery of software.
Method layer: In Method layer the actual method of implementation is carried out with the help of requirement analysis, designing, coding using desired programming constructs and testing.
Tools: Software tools are used to bring automation in software development process.

## What do you mean by CASE? Show with diagram the basic building blocks of CASE.

CASE: The Computer Aided Software Engineering (CASE) tools automate the project management activities, manage all the work products. The CASE tools assist to perform various activities such as analysis, design, coding and testing.
**Basic Building Blocks:**



There are six blocks in building structure of CASE. Each building blocks forms a foundation for the next, with tools sitting at the top of the heap. Each blocks are described below:
<u>Environment Architecture & Hardware Platform:</u> The environment architecture composed of the hardware platform and system support. It lays the ground work for CASE.
<u>Operating System:</u> Operating system supports database and object management services.
<u>Probability Services</u>: This allows CASE tools and their integration framework to migrate across different operating systems and hardware platforms without significant adoptive maintenance.

Integration framework: This is specialized programs allowing CASE tools to communicate with one another.
All of these blocks build CASE tools.

## Mention CASE tools with brief description.

CASE: The Computer Aided Software Engineering (CASE) tools automate the project management activities, manage all the work products. The CASE tools assist to perform various activities such as analysis, design, coding and testing. Software engineers and project managers make use of CASE tools. The use of CASE tools in the software development process reduces the significant amount of efforts. Case tools help software engineers to produce the high quality software efficiently.
The CASE tools are classified using three approaches. Fuggetta has suggested a classification of CASE tools as

- Tools: The tools may be general purpose tools for checking the consistency in the design, compiling the program or result.
- Workbenches: The CASE tools for workbenches are for supporting various activities of software development life cycle
- Environments: The CASE tools for environments support the substantial part of the software process. It generally includes the integrated environment.

## Explain each term of the CASE life cycle in an organization.

## Suggest potential benefits and practical problems of integrating CASE tools.

## With example, explain the use of viewpoint template and service template in VORD method.

| Viewpoint Template | Service Template |
|---|---|

| |
|---|
| Reference: The name of viewpoint<br>Attributes: The information about viewpoints is provided by the attributes.<br>Events: Reference to set of events and description about their interactivity with the system.<br>Services: References to set of services and t heir interactivity with the system.<br>Sub VPs: The name of sub |

| |
|---|
| Reference: The name of service<br>Rationale: The purpose of the service<br>Specification: Reference to the set of service specifications.<br>Viewpoints: References to set of viewpoints receiving the service.<br>Sub VPs: The names ↑ ↑ of sub viewpoints |

For Example: Viewpoint template for ATM

| Viewpoint Template |
|---|
| Reference: Account holder<br>Attributes: Account number<br>PIN<br>Events: Enters the account no. and PIN<br>Starts transaction<br>Select service<br>Cancel service<br>Performs the transaction<br>Close Transaction<br>Services: Customer withdraw<br>Balance inquiry<br>Sub VPs: Joint account holder |

| Service Template |
|---|
| Reference: Balance inquiry<br>Rationale: To allow the user to check the amount of money present in his account.<br>Specification: The user selects this service by pressing the balance inquiry button. After knowing the amount present in his account, he is then asked to get printout of mini statement. The user select the service accordingly.<br>Viewpoints: Account holder<br>Sub VPs: |

## Discuss about different types of cohesion in the context of software design.

Cohesion: Cohesion refers to the degree to which the elements of module belong together.Thus cohesion measures the strength if relationship between pieces of functionality within a given module.

Types of cohesion:

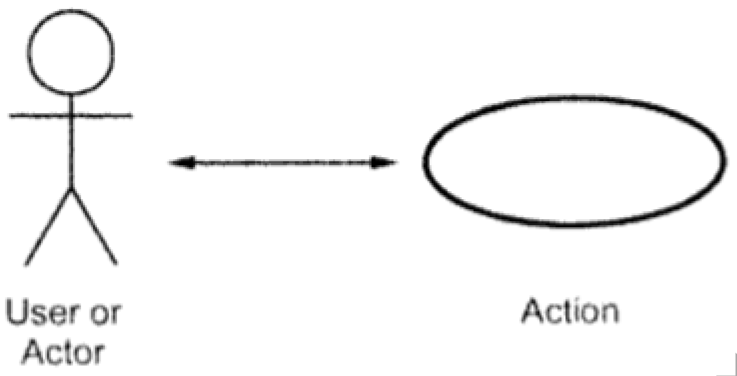
1. Coincidental cohesive: The modules in which the set of tasks are related with each other loosely then such modules are called coincidentally cohesive.
2. Logically cohesive: A module that performs the tasks that are logically related with each other is called logically cohesive.
3. Procedural cohesion: When processing elements of a module are related with one another and must be executed in some specific order then such module is called procedural cohesive.
4. Temporal cohesion: A module in which the tasks need to be executed in some specific time span is called temporal cohesive.
5. Communicational cohesion: When the processing elements of a module share the data then such module is communicational cohesive.

## Distinguish between requirement definition and requirement specification.

| Requirement Definition | Requirement Specification |
|---|---|
| Determine the needs or conditions to meet for a new or altered product, taking account of the possibly conflicting requirements of the various stakeholders. | They provide a precise idea of the problem to be solved so that they can efficiently design the system and estimate the cost of design alternatives. |
| It defines what the system should do. | It defines component, subsystem or system IS. |
| Requirement definition are customer oriented. | Requirement specifications are oriented towards the software designer. |

## Define the terms “Use case” and “Stakeholder”.

Use case are the fundamental units of modeling language, in which functionality are distinctly presented. The use case is a scenario based technique. Use case help to identify individual interactions with the system. Use case are extensively used for requirements elicitation. By designing the proper use cases for different scenarios major requirements of the system can be identified. The typical notations used in the use cases are



Stakeholder: This is a commonly used term in software engineering. The stakeholder means the person who will be affected by the system. For example: end-user, system maintenance engineers or software engineers can be stakeholders.

## What are the different types of user requirement process?

## What are the factors consider in the case of requirement validation?

Requirement validation is a process in which it is checked that whether the gathered requirements represent the same system that customer really wants. In requirement validation the requirement errors are fixed. Requirements error costs are high so validation is very important.

Factors considered in the case of requirement validation are:

1) Validity: Does the system provide the functions which best support the customer's needs?
2) Consistency: Are there any requirements conflicts?
3) Completeness: Are all functions required by the customer included?
4) Realism: Can the requirements be implemented according to budget and technology?
5) Verifiability: Can the requirements be checked?

## Define software design process. State the principle of software design.

Software design process is the phase in which based on functionalities of the system various components of the system are designed. A number of principles are followed while designing the software. Some of the commonly followed design principles are as following:

1. Software design should correspond to the analysis model: We must know how the design model satisfies all the requirements represented by the analysis model.
2. Choose the right programming paradigm: A programming describes the structure of the software system. Depending on the nature and type of application, different programming paradigms can be used The paradigm should be chosen keeping constraints in mind such as time, availability of resources and nature of user's requirements.
3. Software design should be uniform and integrated: Software design is considered uniform and integrated. For this rules, format and styles are established before the design team starts designing the software.
4. Software design should be flexible: Software design should be flexible enough to add changes easily.
5. Software design ensure minimal conceptual errors: The design team must ensure that major conceptual errors of design such as ambiguousness and inconsistency are addressed in advance before dealing with errors present in design model.

6. Software design should be structured to degrade gently: Software should be designed to handle unusual changes and circumstances, and if the need arise for termination, it must do so in a proper manner so that functionality of the software is not affected.
7. Software design should represent correspondence between the software and real-world problem: The software design should be structured in such away that it always relates with the real-world problem.
8. Software reuse: Software engineers believe on the phrase: 'do not reinvent the wheel'. Therefore, software components should be designed in such a way that they can be effectively reused to increase the productivity.
9. Prototyping: Prototyping should be used when the requirements are not completely defined in the beginning. Using prototyping, a quick 'mock-up' of the system can be developed. This mock-up can be used as a effective means to give the users a feel of what the system will look like and demonstrate functions that will be included in the developed system.

## Explain various object oriented concept used in software engineering.

There are some Object Oriented Concepts, which are used in software engineering.

- Messages
- Encapsulation
- Inheritance
- Method
- Polymorphism
- Data abstraction
- Object Composition

Messages: A message is a request for performing an operation by some object in the system. Objects communicate through passing messages.

Encapsulation: Encapsulation is also known as information hiding concept. The data and operations are combined into a single unit. This concept may make the data safe and secure from external interventions.
Inheritance: Inheritance brings reusability at design. It gives the complete knowledge about domains and the system used.
Method: Method is the sequence of steps to be performed to fulfill the assigned task. There may be many methods available to design software.
Polymorphism: Polymorphism gives same operations many meanings in different situations. This makes software design flexible.
Data Abstraction: Data abstraction is to collect essential elements composing to a compound data.
Object Composition: It creates “has a” relationship between objects. This is important in software design.

## Discuss different types of coupling in context of software engineering.

Coupling means the interconnection of dissimilar modules with each other or we can say, it tells about the interrelationship of dissimilar modules of a system. There are 5 types of coupling:

- Content Coupling
- Common Coupling
- Control Coupling
- Stamp Coupling

- Data Coupling

Content coupling - When a module can directly access or modify or refer to the content of another module, it is called content level coupling.

Common coupling- When multiple modules have read and write access to some global data, it is called common or global coupling.

Control coupling- Two modules are called control-coupled if one of them decides the function of the other module or changes its flow of execution.

Stamp coupling- When multiple modules share common data structure and work on different part of it, it is called stamp coupling.

Data coupling- Data coupling is when two modules interact with each other by means of passing data (as parameter). If a module passes data structure as parameter, then the receiving module should use all its components.

Ideally, no coupling is considered to be the best.

## What are the factors for effective modular design?

The software is divided into separately named and addressable components that called as modules. Creating such modules bring the modularity in software. There are five factors for effective modular design.

Modular Decomposability: A design method provides a systematic mechanism for decomposing the problem into sub problems. This reduce complexity of the problem and the modularity can be achieved.

Modular Composability: A design method enables existing design components to be assembled into a new system.

Modular Understandibility: A module can be understood as a standalone unit. Then it will be easier to build and easier to change.

Modular Continuity: Small changes to the system requirements result in changes to individual modules, rather than system wide changes.

Modular Protection: An aberrant condition occurs within a module and its effects are constrained within the module.

## What are the objectives of testing?

Objectives of testing are

1. Testing is a process of executing a program with the intent of finding an error.
2. A good test case is one that has high probability of finding an undiscovered error.
3. A successful test is one that uncovers an as-yet undiscovered error.

The major testing objective is to design tests that systematically uncover types of errors with minimum time and effort.

## What testing principles the software engineer must apply while performing software testing?

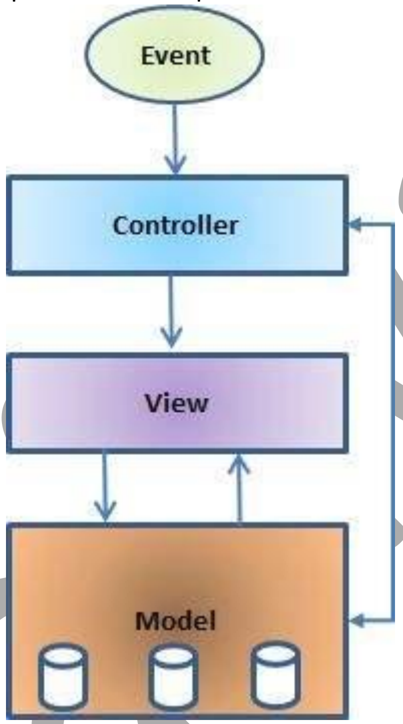
Every software engineer must apply following testing principles while performing the software testing –

1. All tests should be traceable to customer requirements.
2. Tests should be planned long before testing begins
3. The Pareto principle can be applied to software testing – 80% of all errors uncovered during testing will likely be traceable to 20% of all program modules.
4. Testing should begin "in the small" and progress toward testing "in the large".

5. Exhaustive testing is not possible,
6. To be most effective, testing should be conducted by an independent third party.

## Describe the Model View Controller (MVC) architecture.

Model View Controller or MVC as it is popularly called, is a software design pattern for developing web applications. MVC is popular as it isolates the application logic from the user interface layer and supports separation of concerns. Here the Controller receives all requests for the application and then works with the Model to prepare any data needed by the View. The View then uses the data prepared by the Controller to generate a final presentable response.



A Model View Controller pattern is made up of the following three parts:
The model: The model is responsible for managing the data of the application. It responds to the request from the view and it also responds to instructions from the controller to update itself.
The view: A presentation of data in a particular format, triggered by a controller's decision to present the data.
The controller: The controller is responsible for responding to user input and performs interactions on the data model objects. The controller receives the input, it validates the input and then performs the business operation that modifies the state of the data model.

## What is user interface?

User Interface (UI) is the junction between a user and a computer program. It is the front-end application view to which user interacts in order to use the software. User can manipulate and control the software as well as hardware by means of user interface.

## Explain the principles of user interface design.

Principles of User Interface Design are:

User familiarity: The interface should use terms and concepts which are drawn from the experience of the people who will make most use of the system. The interface should be based on user-oriented terms and concepts rather than computer concepts.
Consistency: The system should display an appropriate level of consistency. Commands and menus should have the same format, command punctuation should be similar, etc.
Minimal Surprise: Users should never be surprised by the behavior of a system. If a command operates in a known way, the user should be able to predict the operation of comparable commands.
Recoverability: The interface should include mechanisms to allow users to recover from errors. This might include an undo facility, confirmation of destructive actions, 'soft' deletes, etc.
User Guidance: The interface should provide meaningful feedback when errors occur and provide context-sensitive user help facilities. Some user guidance such as help systems, on-line manuals, etc. should be supplied.
User Diversity: The interface should provide appropriate interaction facilities for different types of system user. Interaction facilities for different types of user should be supported. For example, some users have seeing difficulties and so larger text should be available.

## What are the different types of architectural styles exist for software design?

There are several types of architectural styles. These are:

- Three tier
- Multilayered architecture
- Model view controller
- Domain driven design
- Micro kernel
- Blackboard pattern
- Sensor controller actuator
- Presentation abstraction control

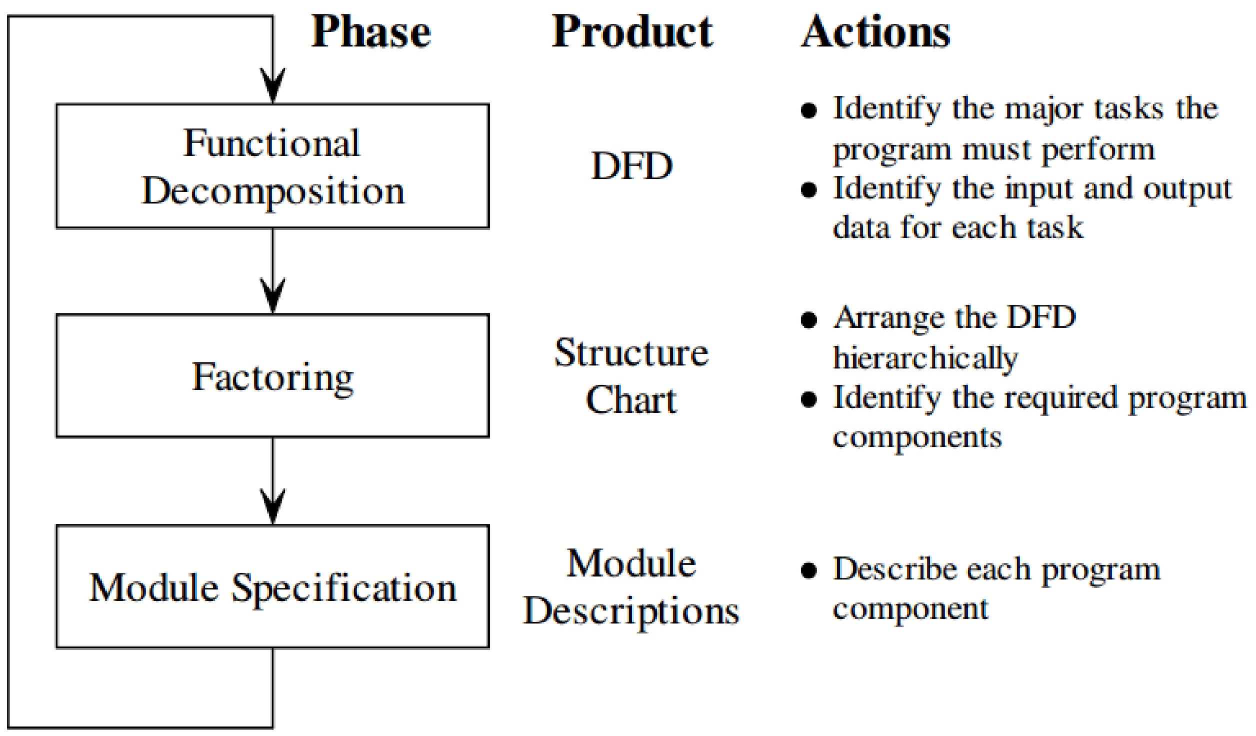
## What are the guidelines that will lead to a good design?

## What do you mean by Design Process?

Design process is a sequence of steps carried through which the requirements are translated into a system or software model. Many times the solution involves designing a product (like a machine or computer code) that meets certain criteria and/or accomplishes a certain task.

## Describe procedural design with diagram.

A design methodology combines a systematic set of rules for creating a program design with diagramming tools needed to represent it. Procedural design is best used to model programs that have an obvious flow of data from input to output. It represents the architecture of a program as a set of interacting processes that pass data from one to another.
Here are the basic steps we follow to create a procedural design.


Phase
Product
Actions
Functional Decomposition
DFD
Identify the major tasks the program must perform
Identify the input and output data for each task
Factoring
Structure Chart
Arrange the DFD hierarchically
Identify the required program components
Module Specification
Module Descriptions
Describe each program component

**Functional Decomposition:** Decomposition is the process of dividing a large entity into more manageable pieces. For a procedural design, this means dividing tasks into sequences of smaller tasks, which is functional decomposition. One technique for doing this, called data flow analysis. IT involves:

1. Identifying a major data flow.
2. Following it from input to output.
3. Determining where it undergoes a major transformation
4. Dividing the process at that point.

**Factoring:** Factoring is the second phase of procedural design in which we create a structure chart that shows what program components need to be implemented. We do this in two passes. First arrange our DFD hierarchically. Second, identify exactly which conceptual processes are to be implemented as physical components in the program.

**Module specification** Module specification is the act of documenting our program design by fully describing each of its module. Module is a general term that can refer to any manner of computer program components, including a single method, a single class, a single object or a collection of related methods.

## Draw work flow diagram for top-down and bottom-up design.

## Describe logical construction of design.

## What factors should be considered when designing interface for human software user?

Here are 8 things I consider a good user interface needs to be:

1. Clear
2. Concise

3. Familiar
4. Responsive
5. Consistent
6. Attractive
7. Efficient
8. Forgiving

Clear: Clarity is the most important element of user interface design. Indeed, the whole purpose of user interface design is to enable people to interact with your system by communicating meaning and function. If people can't figure out how your application works they'll get confused and frustrated.
Concise: Interface must be clear, so that user can easily understand and use the software. But to clarify the interface, we should not use extra description or clarification. Otherwise user may get bored. So interface should be concise.
Familiar: User feels comfortable using an interface if they found familiar with something. So interface must be familiar.
Responsive: Responsive means couple of things. First, responsive means fast. The interface must be fast. It must be load quickly. Secondly, it means interface provides some form of feedback. It must tell to the user what is happening?
Consistent: Consistent interfaces allow users to develop usage patterns – they'll learn what the different interface elements look like and will recognize them and realize what they do in different contexts. They'll also learn how certain things work, and will be able to work out how to operate new features.
Attractive: Attractive in a sense that it makes the use of that interface enjoyable. When software is pleasant to use, your customers or staff will not simply be using it – they'll look forward to using it.
Efficient: What you designer really need to do to make an interface efficient is to figure out what exactly the user is trying to achieve, and then let them do exactly that without any fuss. UI designer have to identify how application should 'work' – what functions does it need to have? Implement an interface that lets people easily accomplish what they want instead of simply implementing access to a list of features.
Forgiving: A forgiving interface is one that can save your users from costly mistakes. For example, if user trying to access a nonexistent page on website, they do not get any cryptic error. They get helpful suggestion.

## What are the concepts of following design pattern when coding?

## What is the objective of testing?

Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design and coding. The purpose of software testing is to ensure whether the software functions appear to be working according to specifications and performance requirements.
According to Glen Myers the testing objectives are:

1. Testing is a process of executing a program with the intend of finding an error.
2. A good test case is one that has high probability of finding an undiscovered error.
3. A successful test is one that uncovers an as-yet undiscovered error.

The major testing objective is to design tests that systematically uncover types of error with minimum time and effort.

## Explain different testing strategy in brief.

Various testing strategies for conventional software are

- Unit testing

- Integration testing
- Validation testing
- System testing

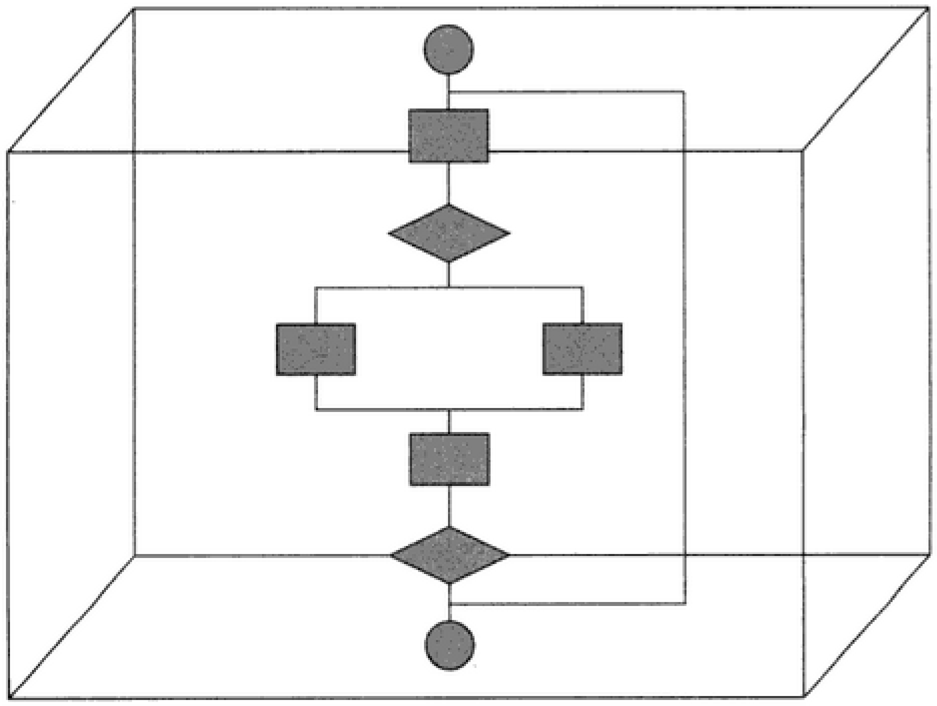
Unit testing: In this type of testing techniques are applied to detect the errors from each software component individually.
Integration testing: It focuses on issues associated with verification and program construction as components begin interacting with one another.
Validation testing: It provides assurance that the software validation criteria meet all functional, behavioral and performance requirements.
System testing: In system testing all system elements forming the system is tested as whole.

## What do you mean by white box testing?

The white box testing is a testing method which is based on close examination of procedural details. Hence it is also called as glass box testing. In white box testing the test cases are derived for:

1. Examining all the independent paths within a module.
2. Exercising all the logical paths with their true and false sides.
3. Executing all the loops within their boundaries and within operational bounds.
4. Exercising internal data structures to ensure their validity.

In white box testing, test cases are selected on the basis of examination of the code, rather than specification. White box testing is illustrated in following:



## Describe the basis path testing.

Path testing is a structural testing strategy. This method is intended to exercise every independent execution path of a program at least once.
Following steps that are carried out while performing path testing:
Step 1- Design the flow graph for the program or a component: Flow graph is a graphical representation of logical control flow of the program. Such a graph consists of circle called a flow graph node representing procedural statements and arrows called as edges which basically represent control flow.

Step 2- Calculating the cyclomatic complexity: The cyclomatic complexity defines the number of independent paths in the basis set of the program that provides that upper bound for the number of tests that must be conducted to ensure that all the statements have been executed at least once.
Step 3- Select a basis set of path: In this step all set of path by following any one we can get required result, is selected.
Step 4- Generate test cases for these paths: After computing cyclomatic complexity and finding paths, the test case has to be executed for these paths.

## Distinguish between (i) Alpha and Beta testing (ii) Unit and Module testing

| Alpha test | Beta test |
|---|---|
| Improve the quality of the product and ensure beta readiness. | Improve the quality of the product, integrate customer input on the complete product and ensure release. |
| Performed at the end of the development when product is near fully usable state. | Performed at the prior to launch. |
| Usually very long lasting with many iterations. It takes 3-5 times duration of bête test. | Usually only few weeks lasts with few major iteration. |
| Cares about quality or engineering. | Cares about product marketing, support, docs, quality and engineering the entire product team. |
| Tester expects plenty of bugs, crashes, missing docs and features from alpha test. | Tester expects some bugs, fewer crushes, most docs, feature complete. |
| After alpha test, beta test is performed. | After beta test, release occurs. |
| Tested by tester. | Tested by tester and End user. |
| Conducted in controlled environment. | Conducted in uncontrolled environment. |

| Unit testing | Module testing |
|---|---|
| Unit testing is a type of testing to check if the small piece of code is doing what it is suppose to do. | Module testing is a type of testing to check if a small part of a function or signature of a function doing what it is suppose to do. |
| Unit testing checks a single component of an application. | Module testing checks a single module of a component. |
| Unit testing is conducted during application testing. | Module testing is conducted during unit testing. |
| | |
| | |
| | |
| | |

## What is software testing strategy? Which strategies are followed to test software?

## Distinguish between white box testing with black box testing.

| Black box testing | White box testing |
|---|---|
| Black box testing is called behavioural testing. | White box testing is called glass box testing. |
| Black box testing examines some fundamental aspect of the system with little regard for internal | In white box testing the procedural details, all the logical paths, a;; the internal data structures are |

| | |
|---|---|
| logical structure of the software. | closely examined. |
| During black box testing the program cannot be tested 100%. | White box testing lead to test the program thoroughly. |
| This type of testing is suitable for large projects. | This type of testing is suitable for small projects. |

## Why verification and validation is important in testing?

Verification refers to the set of activities that ensure that software correctly implements a specific function. Validation refers to a different set of activities that ensure that the software that has been built is traceable to customer requirements. According to Boehm:
Verification: “Are we building product right?”
Validation: “Are we building right product?”
Verification and validation involve large number of Software Quality Assurance activities such as formal technical reviews, quality and configuration audits, performance monitoring, feasibility study, documentation review, database review, algorithm analysis, development testing and installation testing. To assure a right product we must need these SQA activities. So verification and validation is important in testing to assure a quality product.

## What are the testing principles the software engineer must apply while performing the software testing?

## What is black box testing?

Black box testing is also called as behavioral testing. Black box testing methods focuses on the functional requirements of the software. Tests sets are derived that fully exercise all functional requirements. The black box testing is not an alternative to white box testing and it uncovers different class of errors than white box testing. Black box testing uncovers following types of errors:

- Incorrect or missing functions
- Interface errors
- Errors in data structures
- Performance errors
- Initialization or termination errors

## What are the reasons behind to perform white box testing?

White box testing is carried out to test whether:

- All paths in a process are correctly operational
- All logical decisions are executed with true and false conditions
- All loops are executed with their limit values tested
- To ascertain whether input data structure specifications are tested a used for other processing.

## Distinguish between validation and verification.

| Verification | Validation |
|---|---|
| Ensures that product is being build according to the requirements and design specifications. | Ensures that the product actually meets the user’s needs. |
| Describe whether the outputs are according to | Describe whether the software is accepted by the |

| | |
|---|---|
| inputs or not. | user or not. |
| Testing type static. | Testing type dynamic. |
| Verification is done by QA team. | Validation is done by tester/ business users with the help of QA team. |
| Generally carried out before validation. | Generally follows verifications. |
| Test level is low. | Test level is high. |
| It does not involve executing the code. | It involves executing the code. |
| Uses method like walkthroughs, reviews, inspections etc. | Uses methods like black box testing white box testing etc. |

## Define debugging. What are the common approaches in debugging?

Debugging is a process of removal of a defect. It occurs as a consequence of successful testing. Debugging process starts with execution of test cases. The actual test results are compared with the expected result. The debugging process attempts to find the lack of correspondence between actual and expected results. Common approaches in debugging are:
Brute force method: The memory dumps and run-time traces are examined and program with write statements is loaded to obtain clues to error causes. This is the least efficient method of debugging.
Backtracking method: This method is applicable to small programs. In this method, the source code is examined by looking backwards from symptom to potential causes of errors.
Cause elimination method: This method uses binary partitioning to reduce the number of locations where errors can exist.

## Describe the checklist of software testability.

Software testability refers to the ease with which software can be made to demonstrate its faults through testing. Specifically, testability refers to the probability, assuming that the software has at least one fault that it will fail on its next test execution.
Checklist of software testability is as follows:

1. Operability – the better it works the more efficiently it can be tested.
2. Observability – what you see is what you test.
3. Controllability – the better software can be controlled the more testing can be automated and optimized.
4. Decomposability – by controlling the scope of testing, the more quickly problems can be isolated and retested intelligently.
5. Simplicity – the less there is to test, the more quickly we can test.
6. Stability – the fewer the changes, the fewer the disruptions to testing.
7. Understandability – the more information known, the smarter the testing.

## What are the factors affecting the software pricing?

Factors which are affecting software pricing are:

| Factor | Description |
|---|---|
| Market opportunity | A development organization may quote a low price because it wishes to move into a new segment of the software market. Accepting a low profit on one project may give the organization opportunity to make a greater profit later. The experience gained may also help it develop new product. |

| | |
|---|---|
| Cost estimate uncertainty | If an organization is unsure of its cost estimate, it may increase its price by some contingency over and above its normal profit. |
| Contractual terms | A customer may be willing to allow the developer to retain ownership of the source code and reuse it in other projects. The price charged may be less than if the software source code is handed over to the customer. |
| Requirements volatility | If the requirements are likely to change, an organization may lower its price to win a contract. After the contract is awarded, high prices can be charged for changes to the requirements. |
| Financial health | Developers in financial difficulty may lower their price to gain a contract. It is better to make a smaller than normal profit or break even than go out of business. |

## Explain the COCOMO model

COCOMO is one of the most widely used software estimation models in the world. COCOMO predicts the efforts and schedule of a software product based on size of the software. COCOMO stands for COnstructive COst MOdel.
COCOMO has three different models that reflect the complexity:

- Basic Model
- Intermediate Model
- Detailed Model

Basic Model: The basic COCOMO model estimates the software development effort using only lines of code. Various equations in this model are-

$$E = a_b(KLOC)^{b_b}$$

$$D = C_b(E)^{d_b}$$

$$P = \frac{E}{D}$$

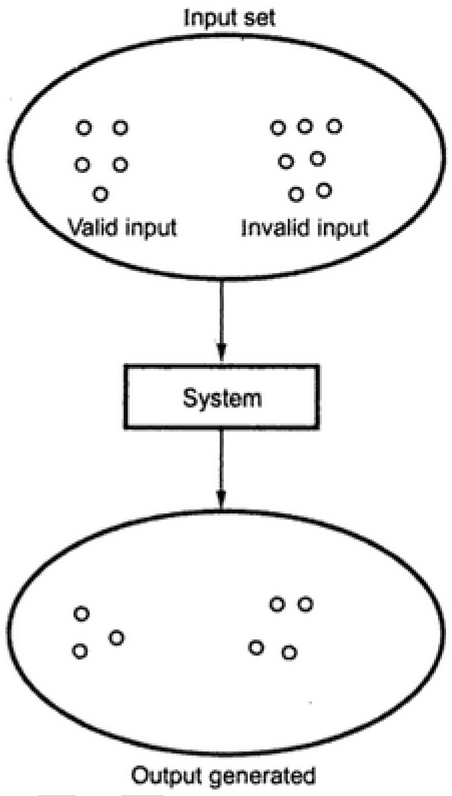
Where E is the effort applied in person – months.
D is the development time in chronological months.
KLOC means kilo line of code for the project.
P is total number of persons required to accomplish the project.
Intermediate Model: This is an extension of basic COCOMO model. This estimation model makes use of set of “cost driver attributes” to compute the cost of software.
Detailed COCOMO Model: The detailed model uses the same equations for estimation as the intermediate model. But detailed model can estimate the effort, duration and persons of each of development phases, subsystems, modules.

Equivalence class guidelines can be as given below:

- If input condition specifies a range, one valid and two invalid equivalence classes are defined.

- If an input condition requires a specific value, one valid and two invalid equivalence classes are defined.
- If an input condition specifies a member of a set, one valid and one invalid equivalence class is defined.
- If an input condition is Boolean, one valid and one invalid equivalence class is defined.

  Example:



## What do you mean by software quality? List characteristics of software quality.

Software quality in the traditional sense means conformity to specification or the absence of defects.
Some definitions of quality are given below:
Conformance to specification. – Philip B Crosby
Fitness for use. – J M Juran
Predictable degree of uniformity and dependability at low cost and suited to market. – W Edward Deming
The definition given by ISO is generally considered as the most acceptable definition of software quality.
"Totality of features and characteristics of product/service that has ability to satisfy stated and implied needs of customer."
Some characteristics of software quality are:

| Attribute | Description |
|---|---|
| Correctness | Extent to which a program meets system specifications and user objectives. |

| | |
|---|---|
| Reliability | Degree to which the system performs its intended functions over a period of time. |
| Efficiency | Amount of computer resources required by a program to perform a function. |
| Ease of Use | Effort required to learn and operate system. |
| Accuracy | Required precision in input editing, computations and output. |
| Portability | Ease of transporting a program from one hardware configuration to another. |
| Control and audit | Control of access to the system and the extent to which it permits data and processing to be audited. |
| Maintainability | Ease with which errors are located and corrected. |
| Expandability | Ease of adding or expanding the existing data base. |

## What is a boundary value analysis?
A boundary value analysis is a testing technique in which the elements at the edge of the domain are selected and tested. Using boundary value analysis, instead of focusing in input conditions only, the test cases from output domain are also derived. Boundary value analysis is a test case design technique that complements equivalence partitioning technique.

## What are the reasons behind to perform black box testing.
Black box testing is unbiased because the designer and the tester are independent of each other. The tester does not need knowledge of any specific programming languages. The test is done from the point of view of the user, not the designer. Test cases can be designed as soon as the specifications are completed. These are the reasons behind to perform black box testing.

## What are the conditions that exist after performing validation testing?
After performing the validation tests, there exists two conditions:
1. The function or performance characteristics are according to the specifications and are accepted.
2. The requirement specifications are derived and the deficiency list is created. The deficiencies then can be resolved by establishing the proper communication with the customer.

## What is cyclomatic complexity? How to compute cyclomatic complexity?
Cyclomatic complexity is a software metric that gives the quantitative measure of logical complexity of the program. The cyclomatic complexity defines the number if independent paths in the basis of the program that provides the upper bound for the number of tests that must be conducted to ensure that tall the statements have been executed at least once.
There are three method of computing cyclomatic complexities:
**Method 1:** The total number of regions in the flow graph is a cyclomatic complexity.
**Method 2:** The cyclomatic complexity, V(G) for a flow graph G can be defined as

V(G) = E – N + 2

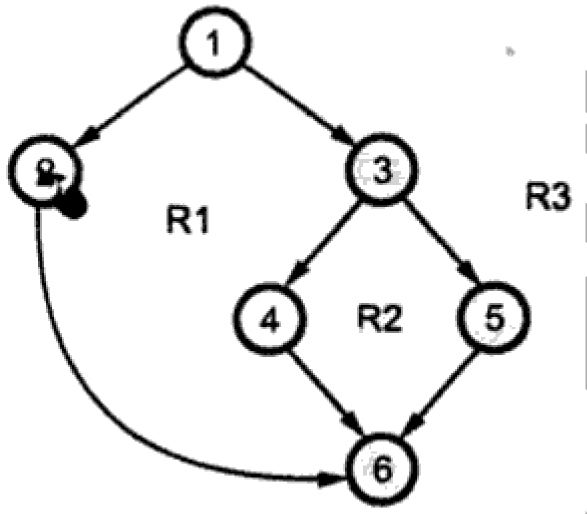
Where E is total number of edges in the flow graph. N is the total number of nodes in the flow graph.

**Method 3:** The cyclomamtic complexity V(G) for a flow graph G can be defined as

V(G) = P + 1

Where P is the total number of predicate nodes contained in the flow graph.

For example consider the following graph:



There are 3 regions denoted by R1, R2 and R3.
Node 1 and 3 are predicate node because which branch to be followed is decided at these points.
Total edges = 7
Total nodes = 6
Applying method 1, cyclomatic complexity = total number of regions = 3
Applying method 2, cyclomatic complexity = 7 – 6 + 2 = 3
Applying method 3, cyclomatic complexity = 2 + 1 = 3

## What is software maintenance?

Software maintenance is an activity in which program is modified after it has been put into use. In software maintenance usually it is not preferred to apply major software changes to system's architecture. Maintenance is a process in which changes are implemented by either modifying the existing system's architecture or by adding new components to the system.

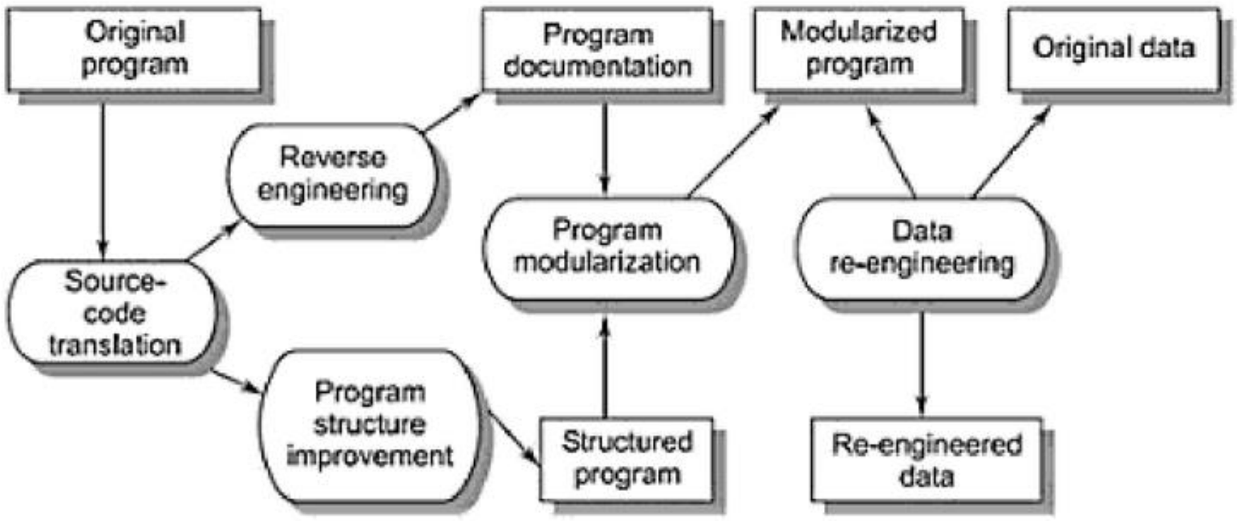
## Describe different types of software maintenance.

Various types of software maintenance are:

1. **Corrective maintenance:** Means the maintenance for correcting the software faults.
2. **Adaptive maintenance:** Means maintenance for adopting the change in environment (Different computers or different operating systems.)
3. **Perfective maintenance:** Means modifying or enhancing the system to meet the new requirements.
4. **Preventive maintenance:** Means changes made to improve future maintainability.

## Explain software re-engineering process activities with figure.

Re-engineering means to re-implement systems to make them more maintainable.
In re-engineering, the functionality and system architecture remains the same but it includes re-documenting, organizing and restricting, modifying and updating the system. It is a solution to the problems of system evolution.
Figure bellow illustrates a possible re-engineering process:



The re-engineering process includes the following activities:

- Source-code translation: In source-code translation the programming language of an old program is converted into modern version of the same language or to a new language.
- Reverse engineering: In reverse engineering the program is analyzed and important and usefyl information is extracted from it which helps to document its functionality.
- Program structure improvement: In program structure improvement the control structure of the program is analyzed and modified to make it easier to read and understand.
- Program modularization: In program modularization redundancy of any part is removed and related parts are grouped together.
- Data re-engineering: In data re0engineering the data processes by the program is changed to reflect program changes.

## Discuss about different types of software cost estimation techniques.
## Or,
## What are the metrics used for estimating cost?

The software cost estimation is the process of predicting the resources required for software development process. Various estimation techniques are:

1. Algorithmic cost modeling: The cost estimation is based on the size of softeware.
2. Expert judgment: The expert from software development and the application domain use their experience to predict software costs.
3. Estimation by analog: The cost of a project is computed by comparing the project to a similar project in the same application domain and then cost can be computed.
4. Parkinson’s law: The cost is determined by available resources rather than by objective assessment.

5. Pricing to win: The project cost whatever the customer ready to spend on it.

## What is SQA?

Software quality assurance is a set of activities designed to evaluate the process by which software is developed and/or maintained. SQA is a planned and systematic pattern of all actions necessary to provide adequate confidence that the item or product conforms to established technical requirements.

## Write down the responsibilities of SQA group.

Responsibilities of the SQA group are:

- Perform audits of projects, subcontractors and departments.
- Provide visibility into processes/ products
- Report results to management and affected groups.
- Act as a partner in quality with the projects.
- Evaluate extent to which quality is built into the project.
- Evaluate effectiveness of the defined processes.
- Work with groups to facilitate improvement to processes.
- Verify implementation and institutionalization of key process areas
- Set quantitative goals for project quality.
- Participate in process improvement activities (on SEPG).
- Participate in evaluation of capability of subcontractor.
- Support/champion implementation of CMM.
- Promote awareness and a culture of quality.
- Facilitate definition of metrics for groups including project management, requirements, development, testing etc.
- Implement a regular management quality review.
- Participate as a member of the project team.
- Review project plan.
- Participate in post-implementation reviews of the projects to learn lessons.
- Facilitate evaluation of new/innovative technologies.
- Implement a test process.

## Write down the guidelines of FTR (Formal Technical Review).

A formal technical review (FTR) is a software quality assurance activity performed by software engineers with the following objectives:
- uncover errors in function, logic or implementation of the software.
- verify that the software meets its requirements.
- ensure that the software has been developed according to the standards.
- achieve uniform software.
- make projects manageable.

There are few guidelines while conducting formal technical reviews. They are:
- Work product should be reviewed and not the developer.
- Make a practice to write down notes while conducting reviews.
- Agenda should be planned.
- Minimize the debate and discussions.
- Keep the number of participants to a minimum and insist on preparing for the review.

- The defect areas should be pointed but no solution should be provided.
- A checklist that is to be reviewed is provided.
- Schedule the reviews as part of the software process and ensure that resources are provided for each reviewer
- Check the effectiveness of review.

## Explain how software cost estimation can be achieved using function point method.

## What is software configuration management?

Software configuration management is a set of activities carried out for identifying, organizing and controlling changes throughout the life cycle of computer software. During the development of software change must be managed and controlled in order to improve quality and reduce error. Hence software configuration management is a quality assurance activity that is applied throughout the software process.

## Distinguish between re-engineering and reverse engineering.

| **Reverse Engineering** | **Re-Engineering** |
|---|---|
| Reverse engineering is the process followed in order to find difficult, unknown and hidden information about a software system. | Re-engineering means to re-implement systems to make them more maintainable. |
| Purpose of reverse engineering is to recover information from the existing code or any other intermediate documents. | Purpose of re-engineering is to updating information in an existing code or any other intermediate documents. |
| Reverse engineering is finding out how a product works from the finished product. | Re-engineering is to examine the finished product and build it again, but better. |
| Reverse engineering is to take a bridge apart to see how it was built. | Re-engineering is to throw a bridge away and rebuild it from scratch. |

## Describe software quality factors according to McCall.

McCall's Quality Factors called as General Electrics Model. This model was mainly developed for US military to bridge the gap between users and developers. It mainly has 3 major representations for defining and identifying the quality of a software product, namely
**Product Revision :** This identifies quality factors that influence the ability to change the software product.
(1) Maintainability : Effort required to locate and fix a fault in the program within its operating environment.
(2) Flexibility : The ease of making changes required as dictated by business by changes in the operating environment.
(3) Testability : The ease of testing program to ensure that it is error-free and meets its specification, i.e, validating the software requirements.
**Product Transition :** This identifies quality factors that influence the ability to adapt the software to new environments.
(1) Portability : The effort required to transfer a program from one environment to another.
(2) Re-usability : The ease of reusing software in a different context.

(3) Interoperability: The effort required to couple the system to another system.
**Product Operations :** This identifies quality factors that influence the extent to which the software fulfills its specification.
(1) Correctness : The extent to which a functionality matches its specification.
(2) Reliability : The systems ability not to fail/ the extent to which the system fails.
(3) Efficiency : Further categorized into execution efficiency and storage efficiency and generally means the usage of system resources, example: processor time, memory.
(4) Integrity : The protection of program from unauthorized access.
(5) Usability : The ease of using software.

## What is CCB?

In software development, a Change Control Board (CCB) or Software Change Control Board (SCCB) is a committee that makes decisions regarding whether or not proposed changes to a software project should be implemented. In short any changes to the Baseline Requirements agreed with the client, should be taken up by project team on approval from this committee. If any change is agreed by the committee, it is communicated to the project team and client and the requirement is Baselined with the change.

## Distinguish between configuration management and change management.

| **Configuration Management** | **Change Management** |
|---|---|
| Configuration management plan only guides in making changes which are specific to the product configuration | Change management plan is a generic plan that guides the project manager in term of making any kind of change in the project, especially the ones that can impact the baselines. |
| The configuration management oversees how any change to the product should done. | The change management plan oversees how any change to the process should be done. |
| Configuration management is an generic, umbrella term for all the activities that reduce the risk of integration failure due to component changes on the project. | Change management is a generic procedure to manage all kinds of changes on the project. |

**Write short notes on**
- **Top down and bottom up design**
- **Object oriented software engineering**
- **Software errors and faults**
- **The design implementation of large multi-modules programs systems**
- **Software re-engineering**
- **ISO 9126 software quality factors**
- **CASE**
- **Real time design**
- **Debugging**
- **DFD (Data Flow Diagram)**
- **Loop Testing**
- **Software cost estimation technique**
- **Spiral Model**

- **Software Requirement Specification**
- **Software testing strategy**
- **V & V software process model**
- **Top down and Bottom up integration testing**
- **Legacy software**
- **Capability Maturity Model**
- **CASE building block**
- **Software configuration management**
- **RAD software process model**
- **Software prototyping**

**Top Down and bottom up design**: We know that a system is composed of more than one sub-systems and it contains a number of components. Further, these sub-systems and components may have their on set of sub-system and components and creates hierarchical structure in the system.

Top-down design takes the whole software system as one entity and then decomposes it to achieve more than one sub-system or component based on some characteristics. Each sub-system or component is then treated as a system and decomposed further. This process keeps on running until the lowest level of system in the top-down hierarchy is achieved.
The bottom up design model starts with most specific and basic components. It proceeds with composing higher level of components by using basic or lower level components. It keeps creating higher level components until the desired system is not evolved as one single component. With each higher level, the amount of abstraction is increased.

**Object Oriented Software Engineering:** Object-Oriented Software Engineering (OOSE) is a software design technique that is used in software design in object-oriented programming**.** It is a system of building software using object oriented paradigm. In this paradigm, to build software, all computations are performed in context of object. The objects are instances of programming constructs, normally classes, which are data abstractions and which contain procedural abstractions that operate on the objects. OOSE includes a requirement, an analysis, a design, an implementation and a testing model.

**Software errors and faults:** Fault is a defect within the system. For example: software bug, random hardware fault, memory bit stuck, omission or commission fault in data transfer etc.
Error is a deviation from the required operation of system or subsystem. A fault may lead to an error. Error is a mechanism by which the fault becomes apparent. Fault may stay dormant for a long time before it manifests itself as an error. For example: Memory bit get stuck but CPU does not access this data.

**The design implementation of large multi-modules programs systems**

**Software re-engineering: See previous pages.**

**ISO 9126 software quality factors:** The ISO 9126-1 software quality model identifies 6 main quality characteristics, namely:

1. Functionality
2. Reliability
3. Usability

4. Efficiency
5. Maintainability
6. Portability

**Functionality:** System should interact with other systems. Attributes of software that bear on its ability to prevent unauthorized access, whether accidental or deliberate, to program data.
**Reliability:** System should be able to maintain a specific level of performance in case of software faults. Also system should be capable of re-establishing its level of performance and recovering the data directly affected in case of failure.
**Efficiency:** This characteristic is concerned with the system resources used when providing the required functionality. The amount of disk space, memory, network etc. provides a good indication of this characteristic.
**Usability:** Attributes of software that bear on the users' effort for recognizing the logical concept and its applicability.
**Maintainability:** The ability to identify and fix a fault within a software component is what the maintainability characteristic addresses. In other software quality models this characteristic is referenced as supportability. Maintainability is impacted by code readability or complexity as well as modularization. Anything that helps with identifying the cause of a fault and then fixing the fault is the concern of maintainability.
**Portability:** This characteristic refers to how well the software can adopt to changes in its environment or with its requirements.

**CASE: see previous pages.**

**Real Time design:** A real-time system is a system whose specification includes both logical and temporal correctness requirements.
Logical Correctness: Produces correct outputs.
Temporal Correctness: Produces outputs at the right time.
Real time design is the process of designing a real time system. It is two of kinds. Hard real time and soft real time.

**Debugging: see previous pages.**

**DFD (Data Flow Diagram):** Data flow diagram is a graphical representation for representing the information flow and the transforms that are applied as data move from input to output. The data flow diagram is a collection of process, data store, flow of data and external entity. The data flow diagrams are used to represent the system at any level of abstraction, and the increasing levels are used to expose more and more functionalities in the system.

**Loop testing:** Loop Testing is the variant of testing that completely focuses on the validity of the loop constructs. It is one of the part of Control Structure Testing (path testing, data validation testing, condition testing).
Loop testing is a White box testing. This technique is used to test loops in the program.
Types of loop Tested
The types of loop tested are,

- Simple loop
- Nested loop
- Concatenated loop
- Unstructured loop

Loop Testing is done for the following reasons

- Testing can fix the loop repetition issues
- Loops testing can reveal performance/capacity bottlenecks
- By testing loops, the uninitialized variables in the loop can be determined
- It helps to identify loops initialization problems.

**Software cost estimation technique: See previous pages**

**Spiral Model: See previous pages**

**Software Requirement Specification: See previous pages**

**Software testing strategy: See previous pages**

**V & V software process model: See previous pages for verification and validation testing**

**Top down and bottom up integration testing:** Top-down integration testing is an integration testing technique used in order to simulate the behavior of the lower-level modules that are not yet integrated. Stubs are the modules that act as temporary replacement for a called module and give the same output as that of the actual product.
The replacement for the 'called' modules is known as 'Stubs' and is also used when the software needs to interact with an external system.
In bottom up integration the modules at the lowest levels are integrated at first, then integration is done by moving upward through the control structure. The bottom up integration process can be carried out using following steps:

1. Low-level modules are combined into clusters that perform a specific software sub-function.
2. A driver program is written to coordinate test case input and output.
3. The whole cluster is tested.
4. Drivers are removed and clusters are combined moving upward in the program structure.

**Legacy Software:** The definition of legacy software is an old and outdated program that is still used to perform a task for a user, even though newer and more efficient options are available.
In particular:

- It cannot be simply discarded because it contains experienced and validated knowledge
- It was developed to answer precise needs that are not covered by existing software
- The system might be running on old hardware
- The system might be currently used by other people

**Capability Maturity Model: See previous pages**

**CASE building block: See previous pages**

**Software configuration management: See previous pages**

**RAD software process model: See previous pages**

**Software prototyping: See previous pages**

## List four reasons why it is difficult to improve software process.

It is difficult to improve software process due to following reasons:
1. Lack of knowledge-Many software developers are not aware of best practices of industry. In fact best practices available in literature are not being used widespread in software development.
2. Not enough time-There is always a shortage of time because upper management are always demanding more software of higher quality in minimum possible time. Unrealistic schedule sometimes leave insufficient time to do the essential project work.
3. Wrong motivations-The process improvement initiatives are taken for wrong reasons like sometimes contractor is demanding achievement of CMM or sometimes senior management is directing the organization to achieve CMM without a clear explanations why improvement was needed and its benefits.
4. Insufficient commitments-The software improvement fails due to lack of true commitment. Management sets no expectations from the development community regarding process improvement.

## Distinguish between software design and software architecture.

| Software Architecture | Software Design |
|---|---|
| Software architecture is a fundamental properties. | Software design is a detailed properties. |
| Manage uncertainty. | Avoid uncertainty |
| Communicate with stakeholders to make. | Follows guidelines to make. |
| Architecture is what we’re building. | Design is how we’re building. |
| Cross-cutting concerns of requirement. | Detailed representation of requirement. |
| Brings conceptual integrity. | Brings completeness. |
| It is a branch of software design. | It is the root of software architecture. |

## What is the role of SRS in waterfall software development model? Explain

First phase of the waterfall model is software requirement and analysis. Requirement specification is generally the most crucial phase of waterfall mode. If it succeeds then a complete failure is unlikely. The requirement specification is the basis of a contract. An SRS minimizes the time and effort required by developers to achieve desired goals and also minimizes the development cost. A good SRS defines how an application will interact with system hardware, other programs and human users in a wide variety of real-world situations. Parameters such as operating speed, response time, availability, portability, maintainability, footprint, security and speed of recovery from adverse events are evaluated. So whole process goes later in waterfall is depending on SRS. So it is clear that SRS is very important in waterfall model.

## How does agile process model involve the stakeholders with the software development?

Stakeholder is anyone who is direct user, indirect user, developer, analyzer and everyone who have direct and indirect relation or interaction with software. Agile process model provides multiple opportunities for stakeholder before, during and after each sprint. To deliver high quality software agile process mode provide a high degree of collaboration between the client and project team. In agile model, by using time boxed, fixed schedule sprints of a time, developers implement a part of software and release it. Stakeholders give feedback. Using that feedback further steps are taken. In this way agile process model involve the stakeholders with the software development.

## Depict the online flight reservation system using use case diagram.