Q. What are the attributes of a good software?

Ans.

- Reliability
  Measure if the product is reliable enough to sustain in any condition. Should give the correct results consistently. Product reliability is measured in terms of working of the project under different working environments and different conditions.
- Maintainability
  Different versions of the product should be easy to maintain. For development, it should be easy to add code to the existing system, should be easy to upgrade for new features and new technologies from time to time. Maintenance should be cost-effective and easy. The system is easy to maintain and correct defects or make a change in the software.
- Usability
  This can be measured in terms of ease of use. The application should be user-friendly. It should be easy to learn. Navigation should be simple.
- Portability
  This can be measured in terms of Costing issues related to porting, Technical issues related to porting, and Behavioral issues related to porting.
- Correctness
  The application should be correct in terms of its functionality, calculations used internally and the navigation should be correct. This means that the application should adhere to functional requirements.
- Efficiency
  It is one of the major system quality attributes. It is measured in terms of time required to complete any task given to the system. For example, the system should utilize processor capacity, disk space, and memory efficiently.
- Integrity or Security
  Integrity comes with security. System integrity or security should be sufficient to prevent unauthorized access to system functions, prevent information loss, ensure that the software is protected from virus infection, and protect the privacy of data entered into the system.
- Testability

The system should be easy to test and find defects. If required, it should be easy to divide into different modules for testing.

- Flexibility
  Should be flexible enough to modify. Adaptable to other products with which it needs interaction. Should be easy to interface with other standard 3rd party components.
- Reusability
  Software reuse is a good cost-efficient and time-saving development method. Different code library classes should be generic enough to be easily used in different application modules. Divide the application into different modules so that modules can be reused across the application.
- Interoperability
  Interoperability of one system to another should be easy for the product to exchange data or services with other systems. Different system modules should work on different operating system platforms, different databases, and protocol conditions.

Q. Explain the design methodology.

Ans

Software design is an important activity as it determines how the whole software development task would proceed including the system maintenance. The design of software is essentially a skill, but it usually requires a structure which will provide a guide or a methodology for this task. A methodology can be defined as the underlying principles and rules that govern a system. A method can be defined as a systematic procedure for a set of activities. Thus, from these definitions, a methodology will encompass the methods used within the methodology. Different methodologies can support work in different phases of the system life cycle, for example, planning, analysis, design and programming, testing, and implementation.

Q. What do you mean by requirement specification?

Ans. The production of the requirements stage of the software development process is Software Requirements Specifications (SRS) (also called a requirements document). This report lays a foundation for software engineering activities and is constructed when entire requirements are elicited and analyzed.

SRS is a formal report, which acts as a representation of software that enables the customers to review whether (SRS) is according to their requirements. Also, it comprises user requirements for a system as well as detailed specifications of the system requirements.

Some features of SRS are:

Correctness - User review is used to provide the accuracy of requirements stated in the SRS. SRS is said to be perfect if it covers all the needs that are truly expected from the system.

Completeness: The SRS is complete if, and only if, it includes the following elements:

Consistency: The SRS is consistent if, and only if, no subset of individual requirements is described in its conflict.

Unambiguousness: SRS is unambiguous when every fixed requirement has only one interpretation. This suggests that each element is uniquely interpreted.

Some properties of a good SRS are:

Concise: The SRS report should be concise and at the same time, unambiguous, consistent, and complete. An unconcise report is prone to a lot of errors.

Structured: It should be well-structured. A well-structured document is simple to understand and modify. In practice, the SRS document undergoes several revisions to cope with the user requirements. Often, user requirements evolve over a period. Therefore, to make the modifications to the SRS document easy, it is vital to make the report well-structured.

Black-box view: It should only define what the system should do and refrain from stating how to do these. This means that the SRS document should define the external behavior of the system and not discuss the implementation issues.

Q: What do you mean by the term modularity in the context of software design?

Ans. Modularity specifies the separation of concerned ‘components’ of the software which can be addressed and named separately. These separated components are referred to as “modules”. These modules can be integrated to satisfy the requirement of other software.

Modularity can be defined as a mechanism where a complex system is divided into several components that are referred to as “modules”. Now, whenever a customer requests to develop software, we can use or integrate these modules to develop new software.

The required modules can be selected and integrated to develop new software in this way, we can customize the software according to users’ needs.

Important: If you subdivide software into infinite components or modules it will reduce the efforts required to develop software. But this is going to revert somewhere.

As we are saying that effort or the development cost will get reduced with the increase in modules. But as the number, of modules, gets increased, the cost required to integrate the several modules also get increases.

So, you must be careful while modularizing the software. The software should neither be left as un-modularized, nor it must be over-modularized.

Benefits of Modularity

- Modularity let the development of software be divided into several components that can be implemented simultaneously by the team of developers. This minimizes the time that is required to develop software.
- Modularity makes the components of the software reusable.
- As modularity breaks the large complex program into components, it improves manageability. As it is easy to develop, test, and maintain the small components.
- It is also easy to debug and trace the error in modular programs.

Q. What are the challenges of software engineering?

Ans. Software engineering employs a well-defined and systematic approach to developing software. This approach is the most effective way of producing high-quality software. However, despite this systematic approach to software development, there are still some serious challenges faced by software engineering. Some of these challenges are listed below.

- The methods used to develop small or medium-scale projects are not suitable when it comes to the development of large-scale or complex systems.
- Changes in software development are unavoidable. In today’s world, changes occur rapidly, and accommodating these changes to develop complete software is one of the major challenges faced by software engineers.
- The advancement in computer and software technology has necessitated the changes in the nature of software systems. The software systems that cannot accommodate changes are not of much use. Thus, one of the challenges of software engineering is to produce high-quality software adapting to the changing needs within acceptable schedules. To meet this challenge, the object-oriented approach is preferred, but accommodating changes to software and its maintenance within the acceptable cost is still a challenge.
- The user generally has only a vague idea about the scope and requirements of the software system. This usually results in the development of software, which does not meet the user’s requirements.
- Changes are usually incorporated in documents without following any standard procedure. Thus, verification of all such changes often becomes difficult.
- The development of High-quality and reliable software requires the software to be thoroughly tested. Through thorough testing of software consumes the majority of resources, underestimating it because of any reason deteriorates the software quality.
- In addition to the above mentioned key challenges, the responsibilities of the system analyst, designers, and programmers are usually not well defined. Also, if the user requirements are not precisely defined, software developers can misinterpret the meaning.

Q. What are software requirements, how to analyze software requirements?

Ans.   The software requirements are description of features and functionalities of the target system. Requirements convey the expectations of users from the software product. The requirements can be obvious or hidden, known, or unknown, expected, or unexpected from client's point of view.

Step 1: Define Software Requirements

You may encounter vague business requirements for software development from time to time. In that case, the first step is to clarify and define them. I'd like to list out several useful techniques to elicit and define requirements as follows:

Document analysis

User manuals and process documents about the current system can become helpful in defining software requirements. Below are a few steps that you can consider for your analysis:

Collecting relevant documents

Evaluating these documents if they are appropriate to be studied

Analyzing them to identify relevant requirements

Reviewing and confirming identified details

Interview

This technique is very helpful in revealing true software needs. For example, you can make use of questions like who uses the system in what timeframe for what purposes, and how their work.

Observation

Observing can provide further information about processes, inputs, and outputs of the existing system. Sometimes, when you interview stakeholders, they cannot explain enough about their process so consider using this technique to bring out the missing information.

Workshop

Another good method to use is to organize a workshop with a group of users or stakeholders. During the workshop, you can talk with participants to discover, define, scope, and prioritize their needs in a new software system.

Brainstorming

Another practical approach is to determine requirements. Users can come up with very innovative ideas that they have seen or experienced somewhere else. Then, you need to collect all their ideas then review and keep relevant ones to include in the new system.

Prototyping

The prototyping technique is often used to present screen mock-ups to users to get their feedback. This helps them visualize how the system will be so that they can provide very useful comments to help recognize requirements that truly meet their needs.

At the end of this step, you may have a clear understanding of the system and how it should work. You should put in a Word document features and functionalities of the system, as well as any business logic like data validation or calculation. This kind of document is known as the **Software Requirements Specification (SRS).**

Q. What do you mean by capable maturity model (CMM)?

Ans. The Software Engineering Institute (SEI) Capability Maturity Model (CMM) specifies an increasing series of levels of a software development organization. The higher the level, the better the software development process, hence reaching each level is an expensive and time-consuming process.

IRDMO

Level One: Initial - The software process is characterized as inconsistent, and occasionally even chaotic. Defined processes and standard practices that exist are abandoned during a crisis. Success of the organization majorly depends on an individual effort, talent, and heroics. The heroes eventually move on to other organizations taking their wealth of knowledge or lessons learnt with them.

Level Two: Repeatable - This level of Software Development Organization has a basic and consistent project management processes to track cost, schedule, and functionality. The process is in place to repeat the earlier successes on projects with similar applications. Program management is a key characteristic of a level two organization.

Level Three: Defined - The software process for both management and engineering activities are documented, standardized, and integrated into a standard software process for the entire organization and all projects across the organization use an approved, tailored version of the organization's standard software process for developing, testing, and maintaining the application.

Level Four: Managed - Management can effectively control the software development effort using precise measurements. At this level, organization set a quantitative quality goal for both software process and software maintenance. At this maturity level, the performance of processes is controlled using statistical and other quantitative techniques and is quantitatively predictable.

Level Five: Optimizing - The Key characteristic of this level is focusing on continually improving process performance through both incremental and innovative technological improvements. At this level, changes to the process are to improve the process performance and at the same time maintain statistical probability to achieve the established quantitative process-improvement objectives.

Q. Write the advantages and the disadvantages of the LOC concept of software size estimation.

Ans. A line of code (LOC) is any line of text in a code that is not a comment or blank line, in any case of the number of statements or fragments of statements on the line. LOC clearly consists of all lines containing program header files, declaration of any variable, and executable and non-executable statements. As Lines of Code (LOC) only counts the volume of code, you can only use it to compare or estimate projects that use the same language and are coded using the same coding standards.

Features:

They are used in assessing a project’s performance or efficiency.

Advantages:

- Most used metric in cost estimation.
- Its alternates have many problems as compared to this metric.
- It is very easy in estimating the efforts.

Disadvantages:

- Increases productivity while writing more lines of code.
- Less efficient code.
- It doesn’t consider complexity.

Researches have shown a rough correlation between LOC and the overall cost and length of developing a project/ product in Software Development, and between LOC and the number of defects. This means the lower your LOC measurement is, the better off you probably are in the development of your product.

Q. What is software crisis?

Ans. Software Crisis is a term used in computer science for the difficulty of writing useful and efficient computer programs in the required time. The software crisis was due to using the same workforce, same methods, same tools even though rapidly increasing in software demand, the complexity of software, and software challenges.

With the increase in the complexity of software, many software problems arise because existing methods were insufficient. If we will use the same workforce, same methods, and same tools after the fast increase in software demand, software complexity, and software challenges, then there arise some problems like software budget problems, software efficiency problems, software quality problems, software managing and delivering problem, etc. This condition is called a software crisis.

Causes of Software Crisis:

- The cost of owning and maintaining software was as expensive as developing the software
- At that time Projects were running over-time
- At that time Software was very inefficient
- The quality of the software was low quality
- Software often did not meet user requirements
- The average software project overshoots its schedule by half
- At that time Software was never delivered
- Non-optimal resource utilization.
- Difficult to alter, debug, and enhance.
- The software complexity is harder to change.

factors contributing to the software crisis.

- Poor project management.

- Lack of adequate training in software engineering.
- Less skilled project members.
- Low productivity improvements.

Solution of Software Crisis:

There is no single solution to the crisis. One possible solution to a software crisis is Software Engineering because software engineering is a systematic, disciplined, and quantifiable approach. For preventing software crises, there are some guidelines:

- Reduction in software over budget.
- The quality of software must be high.
- Less time is needed for a software project.
- Experienced and skilled people working over the software project.
- Software must be delivered.
- Software must meet user requirements.

Q. Explain the waterfall model with advantages and limitations.

Ans. Waterfall approach was first SDLC Model to be used widely in Software Engineering to ensure success of the project.

In "The Waterfall" approach, the whole process of software development is divided into separate phases. In this Waterfall model, typically, the outcome of one phase acts as the input for the next phase **sequentially.**

RSIIDM

The sequential phases in Waterfall model are −

- **Requirement Gathering and analysis** − All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.
- **System Design** − The requirement specifications from first phase are studied in this phase and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture.
- **Implementation** − With inputs from the system design, the system is first developed in small programs called units, which are integrated in the next

phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing.

- **Integration and Testing** − All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.
- **Deployment of system** − Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.
- **Maintenance** − There are some issues which come up in the client environment. To fix those issues, patches are released. Also, to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

All these phases are cascaded to each other in which progress is seen as flowing steadily downwards (like a waterfall) through the phases. The next phase is started only after the defined set of goals are achieved for previous phase and it is signed off, so the name "Waterfall Model". In this model, phases do not overlap.

Some of the major **advantages** of the Waterfall Model are as follows −

- Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.
- Phases are processed and completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Clearly defined stages.
- Well understood milestones.
- Easy to arrange tasks.
- Process and results are well documented.

The major **disadvantages** of the Waterfall Model are as follows −

- 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.

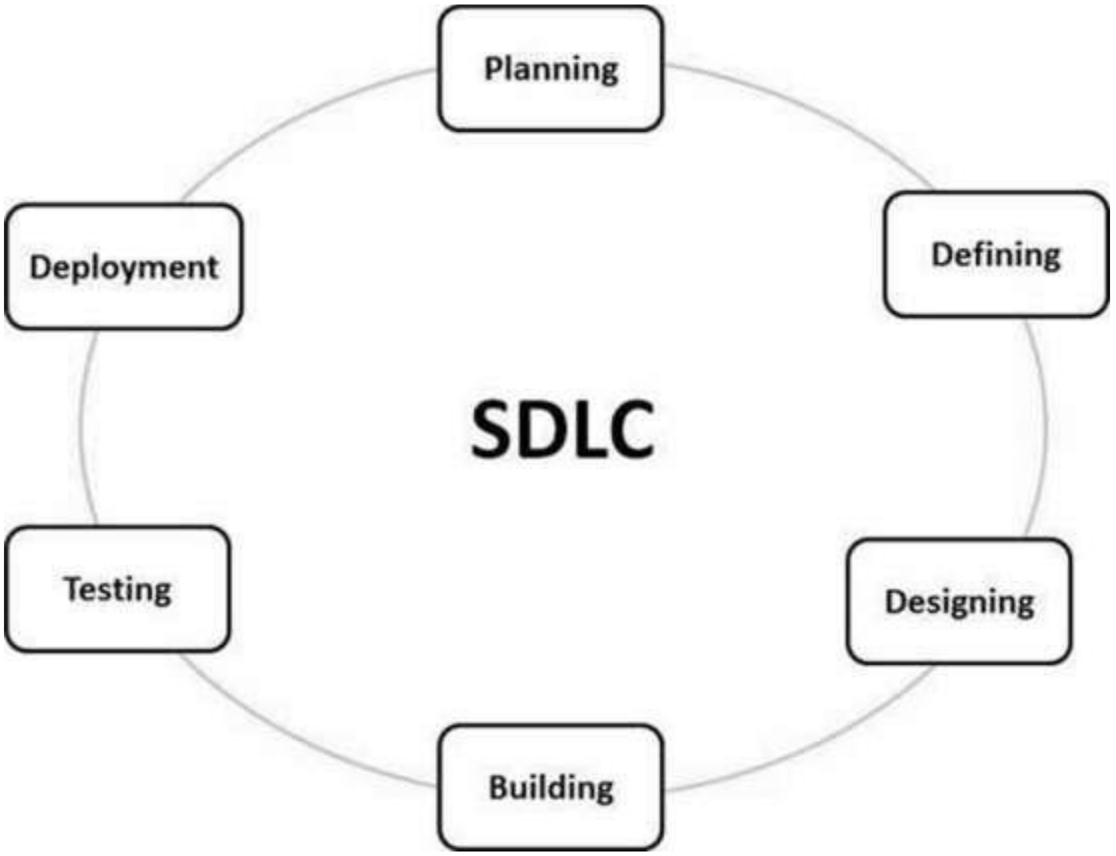
- Not suitable for the projects where requirements are at a moderate to high risk of changing. So, risk and uncertainty is high with this process model.
- It is difficult to measure progress within stages.
- Cannot accommodate changing requirements.
- Adjusting scope during the life cycle can end a project.

Q. What is the Software development life cycle and why it is required?

Ans. SDLC is a process followed for a software project, within a software organization. It consists of a detailed plan describing how to develop, maintain, replace, and alter or enhance specific software. The life cycle defines a methodology for improving the quality of software and the overall development process.

The following figure is a graphical representation of the various stages of a typical SDLC.

PDDBTD



A typical Software Development Life Cycle consists of the following stages −

Stage 1: Planning and Requirement Analysis

Requirement analysis is the most fundamental stage in SDLC. It is performed by the senior members of the team with inputs from the customer, the sales department, market surveys, and domain experts in the industry. This information is then used to plan the basic project approach and to conduct a product feasibility study in the economical, operational, and technical areas.

Planning for the quality assurance requirements and identification of the risks associated with the project is also done in the planning stage. The outcome of the technical feasibility study is to define the various technical approaches that can be followed to implement the project successfully with minimum risks.

Stage 2: Defining Requirements

Once the requirement analysis is done the next step is to clearly define and document the product requirements and get them approved by the customer or the market analysts. This is done through an SRS (Software Requirement Specification) document which consists of all the product requirements to be designed and developed during the project life cycle.

Stage 3: Designing the Product Architecture

SRS is the reference for product architects to come out with the best architecture for the product to be developed. Based on the requirements specified in SRS, usually more than one design approach for the product architecture is proposed and documented in a DDS - Design Document Specification.

This DDS is reviewed by all the important stakeholders and based on various parameters as risk assessment, product robustness, design modularity, budget and time constraints, the best design approach is selected for the product.

A design approach clearly defines all the architectural modules of the product along with its communication and data flow representation with the external and third-party modules (if any). The internal design of all the modules of the proposed architecture should be clearly defined with the minutest of the details in DDS.

Stage 4: Building or Developing the Product

In this stage of SDLC the actual development starts, and the product is built. The programming code is generated as per DDS during this stage. If the design is performed in a detailed and organized manner, code generation can be accomplished without much hassle.

Developers must follow the coding guidelines defined by their organization and programming tools like compilers, interpreters, debuggers, etc. are used to generate the code. Different high level programming languages such as C, C++, Pascal, Java, and PHP are used for coding. The programming language is chosen with respect to the type of software being developed.

Stage 5: Testing the Product

This stage is usually a subset of all the stages as in the modern SDLC models, the testing activities are mostly involved in all the stages of SDLC. However, this stage refers to the testing only stage of the product where product defects are reported, tracked, fixed, and retested, until the product reaches the quality standards defined in the SRS.

Stage 6: Deployment in the Market and Maintenance

Once the product is tested and ready to be deployed it is released formally in the appropriate market. Sometimes product deployment happens in stages as per the business strategy of that organization. The product may first be released in a limited segment and tested in the real business environment (UAT- User acceptance testing).

Then based on the feedback, the product may be released as it is or with suggested enhancements in the targeting market segment. After the product is released in the market, its maintenance is done for the existing customer base

Why is Software Development Life Cycle Important?

It is important to give structure to the phases involved in software development efforts and SDLC serves that purpose. The SDLC does not conclude until all the phases have been successfully fulfilled. All the potential needs have to be adjusted within the system. **The most visible advantage of the SDLC life cycle is that it provides control of the development process to some extent and ensures that the software system complies with all the estimated requirements**. The software development life cycle (SDLC) does not work well where there is uncertainty to some extent. There are no chances of adding creative inputs and the entire process follows from the planning phase. For all these reasons and more, organizations these days are inclined towards adopting the Agile Software Development Approach as it is incremental rather than being sequential.

Q. What are the advantages of implementing software engineering process?

Ans.

Q. Explain the flow graph notation independent program path in basis path testing with example.

Ans.

1. Control Flow Graph

The control flow graph is a graphical representation of control structure of a program. Flow graphs can be prepared as a directed graph. A directed graph (V,E)(V,E) consists of a set of vertices V and a set of edges E that are ordered pairs of elements of V. Based on the concepts of directed graph, following notations are used for a flow graph:

Node: It represents one or more procedural statements. The nodes are denoted by a circle. These are numbered or labelled.

Edges or links: They represent the flow of control in a program. This is denoted by an arrow on the edge. An edge must terminate at a node.

Decision node: A node with more than one arrow leaving it is called a decision node.

Junction node: A node with more than one arrow entering it is called a junction node.

Regions: Areas bounded by edges and nodes are called regions. When counting the regions, the area outside the graph is also considered a region.

2. Flow Graph Notations for Different Programming Constructs

Since a flow graph is prepared on the basis of the control structure of a program, some fundamental graphical notations are shown here (See Fig.1) for basic programming constructs.

Using the aforementioned notations, a flow graph can be constructed. Sequential statements having g as no conditions or loops can be merged in a single node. This

is why the flow graph is also known as the decision-to-decision-graph
or DDDD graph.

Q. What is software engineering? Explain with a proper example and a figure.

Ans. Software engineering is defined as a process of analyzing user requirements and then designing, building, and testing software application which will satisfy those requirements.

Software engineering starts when there is a demand for a specific result or output for a company, from an application. From somewhere on the IT team, typically the CIO, there is a request put into the developer to create some sort of software. The software development team breaks down the project into the requirements and steps. Sometimes, this work will be farmed out to independent contractors, vendors, and freelancers. When this is the case, software engineering tools help to ensure that all of the work done is congruent and follows best practices.

Not all software requires software engineering. Simplistic games or programs that are used by consumers may not need engineering, depending on the risks associated with them. Almost all companies do require software engineering because of the high-risk information that they store and security risks that they pose.

Software engineering helps to create customized, personalized software that should look into vulnerabilities and risks before they even emerge. Even when the software engineering principles of safety aren’t required, it can also help to reduce costs and improve customer experience.

Q. What do you mean by Agile SDLC Model?

Ans. Agile model believes that every project needs to be handled differently and the existing methods need to be tailored to best suit the project requirements. In Agile, the tasks are divided to time boxes (small time frames) to deliver specific features for a release.

Iterative approach is taken and working software build is delivered after each iteration. Each build is incremental in terms of features; the final build holds all the features required by the customer.

The Agile thought process had started early in the software development and started becoming popular with time due to its flexibility and adaptability.

The most popular Agile methods include Rational Unified Process (1994), Scrum (1995), Crystal Clear, Extreme Programming (1996), Adaptive Software Development, Feature Driven Development, and Dynamic Systems Development Method (DSDM) (1995). These are now collectively referred to as Agile Methodologies,

Agile Vs Traditional SDLC Models

Agile is based on the adaptive software development methods, whereas the traditional SDLC models like the waterfall model is based on a predictive approach. Predictive teams in the traditional SDLC models usually work with detailed planning and have a complete forecast of the exact tasks and features to be delivered in the next few months or during the product life cycle.

Predictive methods entirely depend on the requirement analysis and planning done in the beginning of cycle. Any changes to be incorporated go through a strict change control management and prioritization.

Agile uses an adaptive approach where there is no detailed planning and there is clarity on future tasks only in respect of what features need to be developed. There is feature driven development and the team adapts to the changing product requirements dynamically. The product is tested very frequently, through the release iterations, minimizing the risk of any major failures in future.

Customer Interaction is the backbone of this Agile methodology, and open communication with minimum documentation are the typical features of Agile development environment. The agile teams work in close collaboration with each other and are most often located in the same geographical location.

Agile Model - Pros and Cons

Agile methods are being widely accepted in the software world recently. However, this method may not always be suitable for all products. Here are some pros and cons of the Agile model.

The advantages of the Agile Model are as follows −

- Promotes teamwork and cross training.

- Functionality can be developed rapidly and demonstrated.
- Resource requirements are minimum.
- Suitable for fixed or changing requirements
- Delivers early partial working solutions.
- Good model for environments that change steadily.
- Minimal rules, documentation easily employed.
- Enables concurrent development and delivery within an overall planned context.
- Little or no planning required.
- Easy to manage.
- Gives flexibility to developers.

The disadvantages of the Agile Model are as follows −

- Not suitable for handling complex dependencies.
- More risk of sustainability, maintainability, and extensibility.
- An overall plan, an agile leader and agile PM practice is a must without which it will not work.
- Strict delivery management dictates the scope, functionality to be delivered, and adjustments to meet the deadlines.
- Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrong direction.
- There is a very high individual dependency, since there is minimum documentation generated.
- Transfer of technology to new team members may be quite challenging due to lack of documentation.

Q. **What do you mean by cyclometric complexity? Explain.**

Ans. Cyclomatic complexity of a code section is the quantitative measure of the number of linearly independent paths in it. It is a software metric used to indicate the complexity of a program. It is computed using the Control Flow Graph of the program. The nodes in the graph indicate the smallest group of commands of a program, and a directed edge in it connects the two nodes i.e., if second command might immediately follow the first command.

For example, if source code contains no control flow statement, then its cyclomatic complexity will be 1 and source code contains a single path in it. Similarly, if the source code contains one if condition then cyclomatic complexity will be 2 because there will be two paths one for true and the other for false.

M = E – N + 2P

where,
E = the number of edges in the control flow graph
N = the number of nodes in the control flow graph
P = the number of connected components

Q. For which SDLC phase, the UML diagram plays an important role.

Ans. Analysis Phase.

Q. What do you mean by iterative waterfall model?

Ans. The iterative waterfall model provides feedback paths from every phase to its preceding phases, which is the main difference from the classical waterfall model.

When errors are detected at some later phase, these feedback paths allow correcting errors committed by programmers during some phase. The feedback paths allow the phase to be reworked in which errors are committed and these changes are reflected in the later phases. But there is no feedback path to the stage – feasibility study, because once a project has been taken, does not give up the project easily.

It is good to detect errors in the same phase in which they are committed. It reduces the effort and time required to correct the errors.

Advantages of Iterative Waterfall Model:

- Feedback Path –
  In the classical waterfall model, there are no feedback paths, so there is no mechanism for error correction. But in the iterative waterfall model feedback path from one phase to its preceding phase allows correcting the errors that are committed and these changes are reflected in the later phases.

- Simple –
  Iterative waterfall model is very simple to understand and use. That's why it is one of the most widely used software development models.
- Cost-Effective –
  It is highly cost-effective to change the plan or requirements in the model. Moreover, it is best suited for agile organizations.
- Well-organized –
  In this model, less time is consumed on documenting and the team can spend more time on development and designing.

Drawbacks of Iterative Waterfall Model:

- Difficult to incorporate change requests –
  The major drawback of the iterative waterfall model is that all the requirements must be clearly stated before starting the development phase. Customers may change requirements after some time, but the iterative waterfall model does not leave any scope to incorporate change requests that are made after the development phase starts.

- Incremental delivery not supported –
  In the iterative waterfall model, the full software is completely developed and tested before delivery to the customer. There is no scope for any intermediate delivery. So, customers have to wait a long for getting the software.

- Overlapping of phases not supported –
  Iterative waterfall model assumes that one phase can start after completion of the previous phase, But in real projects, phases may overlap to reduce the effort and time needed to complete the project.

- Risk handling not supported –
  Projects may suffer from various types of risks. But, the Iterative waterfall model has no mechanism for risk handling.

- Limited customer interactions –
  Customer interaction occurs at the start of the project at the time of

requirement gathering and at project completion at the time of software delivery. These fewer interactions with the customers may lead to many problems as the finally developed software may differ from the customers' actual requirements.

Q. What do you mean by the boundary value analysis?

Ans. Boundary Value Analysis is based on testing the boundary values of valid and invalid partitions. The behavior at the edge of the equivalence partition is more likely to be incorrect than the behavior within the partition, so boundaries are an area where testing is likely to yield defects.

It checks for the input values near the boundary that have a higher chance of error. Every partition has its maximum and minimum values, and these maximum and minimum values are the boundary values of a partition.

Q. Justify the need for coupling and cohesion for any software. What is the relation b/w them?

Ans.

Coupling: Coupling is the measure of the degree of interdependence between the modules. A good software will have low coupling.

Cohesion: Cohesion is a measure of the degree to which the elements of the module are functionally related. It is the degree to which all elements directed towards performing a single task are contained in the component. Basically, cohesion is the internal glue that keeps the module together. A good software design will have high cohesion.

Coupling shows the relationships between modules. Cohesion shows the relationship within the module. Coupling shows the relative independence between the modules. Cohesion shows the module's relative functional strength.

A good design is one that has low coupling.

Thus, it can be said that a design with high coupling will have more errors.

While creating you should aim for high cohesion, i.e., a cohesive component/ module focuses on a single function (i.e., single-mindedness) with little interaction with other modules of the system.

Q. What is a feasibility study and requirement analysis?

Ans. Feasibility Study in Software Engineering is a study to evaluate feasibility of proposed project or system. Feasibility study is one of stage among the important four stages of Software Project Management Process. As name suggests feasibility study is the feasibility analysis or it is a measure of the software product in terms of how much beneficial product development will be for the organization in a practical point of view. Feasibility study is carried out based on many purposes to analyze whether software product will be right in terms of development, implantation, contribution of project to the organization etc.

Need of Feasibility Study :
Feasibility study is so important stage of Software Project Management Process as after completion of feasibility study it gives a conclusion of whether to go ahead with proposed project as it is practically feasible or to stop proposed project here as it is not right/feasible to develop or to think/analyze about proposed project again.

Along with this Feasibility study helps in identifying risk factors involved in developing and deploying system and planning for risk analysis also narrows the business alternatives and enhance success rate analyzing different parameters associated with proposed project development.

Software requirement analysis simply means complete study, analyzing, describing software requirements so that requirements that are genuine and needed can be fulfilled to solve problem.

Problem Recognition:
The main aim of requirement analysis is to fully understand main objective of requirement that includes why it is needed, does it add value to product, will it be beneficial, does it increase quality of the project, does it will have any other effect. All these points are fully recognized in problem recognition so that requirements that are essential can be fulfilled to solve business problems.

Evaluation and Synthesis:
Evaluation means judgment about something whether it is worth it or not and

synthesis means to create or form something. Here are some tasks are given that is important in the evaluation and synthesis of software requirement:

- To define all functions of software that are necessary.
- To define all data objects that are present externally and are easily observable.
- To evaluate that flow of data is worth or not.
- To fully understand overall behavior of system that means overall working of system.
- To identify and discover constraints that are designed.
- To define and establish character of system interface to fully understand how system interacts with two or more components or with one another.

Modeling:
After complete gathering of information from above tasks, functional and behavioral models are established after checking function and behavior of system using a domain model that also known as the conceptual model.

Specification:
The software requirement specification (SRS) which means to specify the requirement whether it is functional or non-functional should be developed.

Review:
After developing the SRS, it must be reviewed to check whether it can be improved or not and must be refined to make it better and increase the quality.

Q. Explain a decision table that is required for software design with an example.

Ans. A decision table is a brief visual representation for specifying which actions to perform depending on given conditions. The information represented in decision tables can also be represented as decision trees or in a programming language using if-then-else and switch-case statements.

A decision table is a good way to settle with different combination inputs with their corresponding outputs and is also called a **cause-effect table**. The reason to call a cause-effect table is a related logical diagramming technique called cause-effect graphing that is basically used to obtain the decision table.

CONDITIONS	STEP 1	STEP 2	STEP 3	STEP 4
Condition 1	Y	Y	N	N
Condition 2	Y	N	Y	N
Condition 3	Y	N	N	Y
Condition 4	N	Y	Y	N

Q. What do you mean by Reverse Engineering and its need.

Ans. Software Reverse Engineering is a process of recovering the design, requirement specifications and functions of a product from an analysis of its code. It builds a program database and generates information from this.

The purpose of reverse engineering is to facilitate the maintenance work by improving the understandability of a system and to produce the necessary documents for a legacy system.

- Collection Information:
  This step focuses on collecting all possible information (i.e., source design documents etc.) about the software.
- Examining the information:
  The information collected in step-1 as studied so as to get familiar with the system.
- Extracting the structure:
  This step concerns with identification of program structure in the form of structure chart where each node corresponds to some routine.
- Recording the functionality:
  During this step processing details of each module of the structure, charts are recorded using structured language like decision table, etc.

- Recording data flow:
  From the information extracted in step-3 and step-4, set of data flow diagrams are derived to show the flow of data among the processes.
- Recording control flow:
  High level control structure of the software is recorded.
- Review extracted design:
  Design document extracted is reviewed several times to ensure consistency and correctness. It also ensures that the design represents the program.
- Generate documentation:
  Finally, in this step, the complete documentation including SRS, design document, history, overview, etc. are recorded for future use.

NEED:

Reverse Engineering Goals:

- Cope with Complexity.
- Recover lost information.
- Detect side effects.
- Facilitate Reuse.

Q. What are the advantages of modularization in terms of software development?

Ans. Modularization

Modularization is a technique to divide a software system into multiple discrete and independent modules, which are expected to be capable of carrying out task(s) independently. These modules may work as basic constructs for the entire software. Designers tend to design modules such that they can be executed and/or compiled separately and independently.

Modular design unintentionally follows the rules of ‘divide and conquer’ problem-solving strategy this is because there are many other benefits attached with the modular design of a software.

Advantage of modularization:

- Smaller components are easier to maintain
- Program can be divided based on functional aspects

- Desired level of abstraction can be brought in the program
- Components with high cohesion can be re-used again
- Concurrent execution can be made possible
- Desired from security aspect

Q. What do you mean by black box testing and white box testing. Explain with example.

Ans. Black box testing is a broad category that includes a variety of techniques and technologies that check software from the outside without scanning the code. Black box testing refers to any type of software test that examines an application without knowledge of the internal design, structure, or implementation of the software project.

Black box testing can be performed at multiple levels, including unit testing, integration testing, system testing, or acceptance testing. At any of these levels, black box testing examines the input and output of an application to ensure that the software runs as intended under a variety of conditions and to uncover and remediate any errors.

#1 Functional testing

A type of black box testing that focuses on specific functions in the application. This includes sanity checks, integration testing, or system testing. Functional testing is performed by providing a certain input and checking if the output meets the software requirements and specifications.

#2 Non-functional testing

This includes a number of black box testing types that don't examine functionality. Non-functional testing focuses on other aspects and requirements, like usability, load, performance, compatibility, stress, or scalability, to name a few.

#3 Regression testing

Performed after vulnerability remediation, version updates, or other types of system upgrades and maintenance. Regression testing checks whether changes made to the software hurt the existing functional or non-functional aspects of the code.

The term 'white box' is used because of the internal perspective of the system. The clear box or white box or transparent box name denote the ability to see through the software's outer shell into its inner workings.

Developers do white box testing. In this, the developer will test every line of the code of the program. The developers perform the White-box testing and then send the application or the software to the testing team, where they will perform the black box testing and verify the application along with the requirements and identify the bugs and sends it to the developer.

The developer fixes the bugs and does one round of white box testing and sends it to the testing team. Here, fixing the bugs implies that the bug is deleted, and the particular feature is working fine on the application.

Q. Describe Spiral SDLC Model.

Ans. Spiral model is one of the most important Software Development Life Cycle models, which provides support for Risk Handling.

it looks like a spiral with many loops. The exact number of loops of the spiral is unknown and can vary from project to project. Each loop of the spiral is called a Phase of the software development process. The exact number of phases needed to develop the product can be varied by the project manager depending upon the project risks. As the project manager dynamically determines the number of phases, so **the project manager** has an important role to develop a product using the spiral model.

Advantages of Spiral Model:
Below are some advantages of the Spiral Model.

- Risk Handling: The projects with many unknown risks that occur as the development proceeds, in that case, Spiral Model is the best development model to follow due to the risk analysis and risk handling at every phase.
- Good for large projects: It is recommended to use the Spiral Model in large and complex projects.
- Flexibility in Requirements: Change requests in the Requirements at later phase can be incorporated accurately by using this model.
- Customer Satisfaction: Customer can see the development of the product at the early phase of the software development and thus, they habituated with the system by using it before completion of the total product.

- Disadvantages of Spiral Model:
  Below are some main disadvantages of the spiral model.
- Complex: The Spiral Model is much more complex than other SDLC models.
- Expensive: Spiral Model is not suitable for small projects as it is expensive.
- Too much dependability on Risk Analysis: The successful completion of the project is very much dependent on Risk Analysis. Without very highly experienced experts, it is going to be a failure to develop a project using this model.
- Difficulty in time management: As the number of phases is unknown at the start of the project, so time estimation is very difficult.

Q. What is the advantage of spiral model over prototype model?
Ans. A risk is any adverse situation that might affect the successful completion of a software project. The most important feature of the spiral model is handling these unknown risks after the project has started. Such risk resolutions are easier done by developing a prototype. The spiral model supports coping up with risks by providing the scope to build a prototype at every phase of the software development.

The Prototyping Model also supports risk handling, but the risks must be identified completely before the start of the development work of the project. But in real life project risk may occur after the development work starts, in that case, we cannot use the Prototyping Model. In each phase of the Spiral Model, the features of the product dated and analyzed, and the risks at that point in time are identified and are resolved through prototyping. Thus, this model is much more flexible compared to other SDLC models.

Q. Describe the context of unit testing, system testing, and integration testing.

Ans.

1. Unit Testing

It focuses on the smallest unit of software design. In this, we test an individual unit or group of interrelated units. It is often done by the programmer by using sample input and observing its corresponding outputs.

Example:

a) In a program we are checking if loop, method or

function is working fine

b) Misunderstood or incorrect, arithmetic precedence.

c) Incorrect initialization

2. Integration Testing

The objective is to take unit tested components and build a program structure that has been dictated by design. Integration testing is testing in which a group of components is combined to produce output.

Integration testing is of four types: (i) Top-down (ii) Bottom-up (iii) Sandwich (iv) Big-Bang
Example

(a) Black Box testing:- It is used for validation.

3. System Testing

This software is tested such that it works fine for the different operating systems. It is covered under the black box testing technique. In this, we just focus on the required input and output without focusing on internal working.
In this, we have security testing, recovery testing, stress testing, and performance testing
Example:

This includes functional as well as non functional

testing

this we ignore internal working mechanism and

focus on what is the output?.

(b) White Box testing:- It is used for verification.

In this we focus on internal mechanism i.e.

how the output is achieved?

Q. Describe basis path testing. What are the advantages and limitation of testing process?

Ans. Basis Path Testing is a white-box testing technique based on the control structure of a program or a module. Using this structure, a control flow graph is prepared and the various possible paths present in the graph are executed as a part of testing. Therefore, by definition,

Basis path testing is a technique of selecting the paths in the control flow graph, that provide a basis set of execution paths through the program or module.

Advantages of testing:

#1 QA Saves You Money

#2 QA Prevents Catastrophic Corporate Emergencies

#3 QA Inspires Client Confidence

#4 QA Maintains Great User Experience

#5 QA Brings In More Profit

#6 QA Boosts Customer Satisfaction

#7 QA Promotes Organization, Productivity, and Efficiency

Testing Limitations

You cannot test a program completely

We can only test against system requirements

– May not detect errors in the requirements.
– Incomplete or ambiguous requirements may lead to inadequate or incorrect testing.

Exhaustive (total) testing is impossible in present scenario.

Time and budget constraints normally require very careful planning of the testing effort.

Compromise between thoroughness and budget.

Test results are used to make business decisions for release dates.

Even if you do find the last bug, you'll never know it

You will run out of time before you run out of test cases

You cannot test every path

You cannot test every valid input

You cannot test every invalid input

Q. What is project management?

Ans. Software Project Management (SPM) is a proper way of planning and leading software projects. It is a part of project management in which software projects are planned, implemented, monitored, and controlled.

Need for Software Project Management: Software is a non-physical product. Software development is a new stream in business and there is very little experience in building software products. Most of the software products are made to fit clients' requirements. The most important is that the basic technology changes and advances so frequently and rapidly that experience of one product may not be applied to the other one. Such type of business and environmental constraints increase risk in software development hence it is essential to manage software projects efficiently. It is necessary for an organization to deliver quality products, keep the cost within the client's budget constrain and deliver the project as per schedule. Hence in order, software project management is necessary to incorporate user requirements along with budget and time constraints.

Q. Why is testing important wrt to software?

Ans.

- To identify defects
- To reduce flaws in the component or system

- Increase the overall quality of the system

There can also be a requirement to perform software testing to comply with legal requirements or industry-specific standards. These standards and rules can specify what kind of techniques should we use for product development. For example, the motor, avionics, medical, and pharmaceutical industries, etc., all have standards covering the testing of the product.

The points below shows the significance of testing for a reliable and easy to use software product:

- The testing is important since it discovers defects/bugs before the delivery to the client, which guarantees the quality of the software.
- It makes the software more reliable and easier to use.
- Thoroughly tested software ensures reliable and high-performance software operation.

Q. Explain the need of software measures and describes different metrices.

Ans. Software Measurement: A measurement is a manifestation of the size, quantity, amount, or dimension of a particular attribute of a product or process. Software measurement is a titrate impute of a characteristic of a software product or the software process. It is an authority within software engineering. The software measurement process is defined and governed by ISO Standard.

Need of Software Measurement:
Software is measured to:

- Create the quality of the current product or process.
- Anticipate future qualities of the product or process.
- Enhance the quality of a product or process.
- Regulate the state of the project in relation to budget and schedule.

Metrics:
A metric is a measurement of the level that any impute belongs to a system product or process.

Classification of Software Metrics:
There are 3 types of software metrics:

Product Metrics:
Product metrics are used to evaluate the state of the product, tracing risks and

undercovering prospective problem areas. The ability of team to control quality is evaluated.

Process Metrics:
Process metrics pay particular attention on enhancing the long term process of the team or organization.

Project Metrics:
The project matrix describes the project characteristic and execution process.

Number of software developer

Staffing pattern over the life cycle of software

Cost and schedule

Productivity

Q. Write a short note on requirement engineering process.

Ans. Requirements engineering (RE) refers to the process of defining, documenting, and maintaining requirements in the engineering design process. Requirement engineering provides the appropriate mechanism to understand what the customer desires, analyzing the need, and assessing feasibility, negotiating a reasonable solution, specifying the solution clearly, validating the specifications and managing the requirements as they are transformed into a working system. Thus, requirement engineering is the disciplined application of proven principles, methods, tools, and notation to describe a proposed system's intended behavior and its associated constraints.

Requirement Engineering Process

It is a four-step process, which includes -

- Feasibility Study
- Requirement Elicitation and Analysis
- Software Requirement Specification
- Software Requirement Validation
- Software Requirement Management

Q. What is the need of quality assurance in case of software development?

Ans.

Q. Explain incremental process model.

Ans. Incremental Model is a process of software development where requirements divided into multiple standalone modules of the software development cycle. In this model, each module goes through the requirements, design, implementation and testing phases. Every subsequent release of the module adds function to the previous release. The process continues until the complete system achieved.

When we use the Incremental Model?

- When the requirements are superior.
- A project has a lengthy development schedule.
- When Software team are not very well skilled or trained.
- When the customer demands a quick release of the product.
- You can develop prioritized requirements first.

Advantage of Incremental Model

- Errors are easy to be recognized.
- Easier to test and debug
- More flexible.
- Simple to manage risk because it handled during its iteration.
- The Client gets important functionality early.

Disadvantage of Incremental Model

- Need for good planning
- Total Cost is high.
- Well defined module interfaces are needed.

Q. What are the design principles of a good software design.

Ans. Principles Of Software Design:

- Should not suffer from “Tunnel Vision” –
  While designing the process, it should not suffer from “tunnel vision” which means that is should not only focus on completing or achieving the aim but on other effects also.
- Traceable to analysis model –
  The design process should be traceable to the analysis model which means it

should satisfy all the requirements that software requires to develop a high-quality product.

- Should not "Reinvent The Wheel" –
  The design process should not reinvent the wheel that means it should not waste time or effort in creating things that already exist. Due to this, the overall development will get increased.
- Minimize Intellectual distance –
  The design process should reduce the gap between real-world problems and software solutions for that problem meaning it should simply minimize intellectual distance.
- Exhibit uniformity and integration –
  The design should display uniformity which means it should be uniform throughout the process without any change. Integration means it should mix or combine all parts of software i.e. subsystems into one system.
- Accommodate change –
  The software should be designed in such a way that it accommodates the change implying that the software should adjust to the change that is required to be done as per the user's need.
- Degrade gently –
  The software should be designed in such a way that it degrades gracefully which means it should work properly even if an error occurs during the execution.
- Assessed or quality –
  The design should be assessed or evaluated for the quality meaning that during the evaluation, the quality of the design needs to be checked and focused on.
- Review to discover errors –
  The design should be reviewed which means that the overall evaluation should be done to check if there is any error present or if it can be minimized.
- Design is not coding and coding is not design –
  Design means describing the logic of the program to solve any problem and coding is a type of language that is used for the implementation of a design.

Q. Difference b/w Software engineering and system engineering.

Ans.

What is Software Engineering?

Software engineering deals with designing and developing software of the highest quality. A software engineer does analyzing, designing, developing and testing software. Software engineers carry out software engineering projects, which usually have a standard software life cycle. For example, the Water Fall Software Life cycle will include an analysis phase, design phase, development phase, testing and verification phase and finally the implementation phase. Analysis phase looks at the problem to be solved or the opportunities to be seized by developing the software. Sometimes, a separate business analyst carries out this phase. However, in small companies, software engineers may do this task. Design phase involves producing the design documents such as UML diagrams and ER diagrams depicting the overall structure of the software to be developed and its components. Development phase involves programming or coding using a certain programming environment. Testing phase deals with verifying that software is bug free and also satisfies all the customer requirements. Finally, the completed software is implemented at the customer site (some times by a separate implementation engineer). In recent years, there has been a rapid growth of other software development methodologies in order to further improve the efficiency of the software engineering process. For example, Agile methods focus on incremental development with very short development cycles. Software Engineering profession is a highly rated job because of its very high salary range.

What is Systems Engineering?

System Engineering is the sub discipline of engineering which deals with the overall management of engineering projects during their life cycle (focusing more on physical aspects). It deals with logistics, team coordination, automatic machinery control, work processes and similar tools. Most of the times, System Engineering overlaps with the concepts of industrial engineering, control engineering, organizational and project management and even software engineering. System Engineering is identified as an interdisciplinary engineering field due to this reason. System Engineer may carry out system designing, developing requirements, verifying requirements, system testing and other engineering studies.

What is the difference between Software Engineering and Systems Engineering?

The difference between System Engineering and Software Engineering is not very clear. However, it can be said that the System Engineers focus more on users and domains, while Software Engineering focus more on n implementing quality software. System Engineer may deal with a substantial amount of hardware engineering, but typically software engineers will focus solely on software components. System Engineers may have a broader education (including Engineering, Mathematics and Computer science), while Software Engineers will come from a Computer Science or Computer Engineering background.

Q. What do you mean by software designing tools and technology?

Ans.