1.Define software. How does software differ from other built products?

Ans: Software is **a set of instructions** which takes data as input and produces information as output for its users. It is the opposite of hardware, which describes the physical aspects of a computer. Software is a generic term used to refer to applications, scripts and programs that run on a device.

Difference between Product and Software

1. Tangibility: Software is intangible, existing as lines of code and digital information, whereas other engineering products, such as bridges or cars, are physical and tangible.
2. Iterative nature: Software can be easily updated and changed, often through iterative development processes, while physical engineering products may require significant rework to implement changes.
3. Flexibility: Software can be easily customized and adapted to different use cases, while physical engineering products may have more limited flexibility once they are manufactured.
4. Software is engineered. Product is manufactured

2. Define software engineering and explain that software engineering is a layered technology.

Software Engineering: The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software.

Software engineering is a layered technology. Any engineering approach (including software engineering) must rest on an organizational commitment to quality. You may have heard of total quality management (TQM) or Six Sigma, and similar philosophies that foster a culture of continuous process improvement. It is this culture that ultimately leads to more effective approaches to software engineering. The bedrock that supports software engineering is a quality focus.

The foundation for software engineering is the process layer. The software engineering process is the glue that holds the technology layers together and enables rational and timely development of computer software. Process defines a framework that must be established for effective delivery of software engineering technology. The software process forms the basis for management control of software projects and establishes the context in which technical methods are applied, work products (models, documents, data, reports, forms, etc.) are produced, milestones are established, quality is ensured, and change is properly managed.

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

Software engineering tools provide automated or semi-automated support for the process and the methods. When tools are integrated so that information created by one tool can be used by another, a system for the support of software development, called computer-aided software engineering, is established.

3. Illustrate the software engineering practices. Name the general principles of Software Engineering.

The Software Engineering Practices-

The essence of software engineering practice:

1. Understand the problem (communication and analysis).

2. Plan a solution (modeling and software design).

3. Carry out the plan (code generation).

4. Examine the result for accuracy (testing and quality assurance).

General Principles

The First Principle: The Reason It All Exists

A software system exists for one reason: to provide value to its users. All decisions should be aligned with this goal. Before defining requirements or functionalities, ask whether they genuinely add value to the system.

The Second Principle: KISS (Keep It Simple, Stupid!)

Aim for simplicity in design, making systems easy to understand and maintain. While simplicity is essential, it should not sacrifice necessary features. Elegant designs often emerge from thoughtful simplification, resulting in more maintainable and less error-prone software.

The Third Principle: Maintain the Vision

A clear vision is essential to the success of a software project. Without conceptual integrity, a system threatens to become a patchwork of incompatible designs, held together by the wrong kind of screws . . . Compromising the architectural vision of a software system weakens and will eventually break even the well

designed systems. Upholding the architectural vision, often guided by an empowered architect, ensures project success.

The Fourth Principle: What You Produce, Others Will Consume

Always specify, design, document, and implement knowing someone else will have to understand what you are doing. The audience for any product of software development is potentially large. Specify with an eye to the users. Design, keeping the implementers in mind. Code with concern for those that must maintain and

extend the system.

The Fifth Principle: Be Open to the Future

Design software systems to adapt to changing specifications and evolving hardware platforms. Avoid designing yourself into a corner by anticipating future changes and creating systems that solve general problems rather than specific ones.

The Sixth Principle: Plan Ahead for Reuse

Though there are lots of challenges in achieving a high level of reusable components, it reduces the cost and increases the value of both the reusable components and the systems into which they are incorporated.

The Seventh Principle: Think!

Clear, complete thought before action leads to better results in software development. Intense thought, combined with applying the preceding principles, yields significant rewards.

4. Graphically describe the generic process flow models.

Ans: Generic Process Model is a definitive description of processes. eneric Process Model consists of 5 activities. The software process is a collection of various activities. These activities include communication, planning, modeling, construction, and deployment. Each of these activities includes a set of engineering actions and each action defines a set of tasks that incorporate work products, project milestones, and SQA, Software Quality Assurance points.

Generic Process model includes the below five framework activities,

- **Communication:** This is the first step in the software development process which starts with the communication between the customer and the developer.
- **Planning:** This step consists of a complete estimation of the project, scheduling for development of the project, and tracking.
- **Modeling:** This step consists of complete requirement analysis and project design like algorithm, flowchart, etc.

- **Construction:** This step consists of code generation and its testing. Coding will implement the design details using appropriate programming languages.

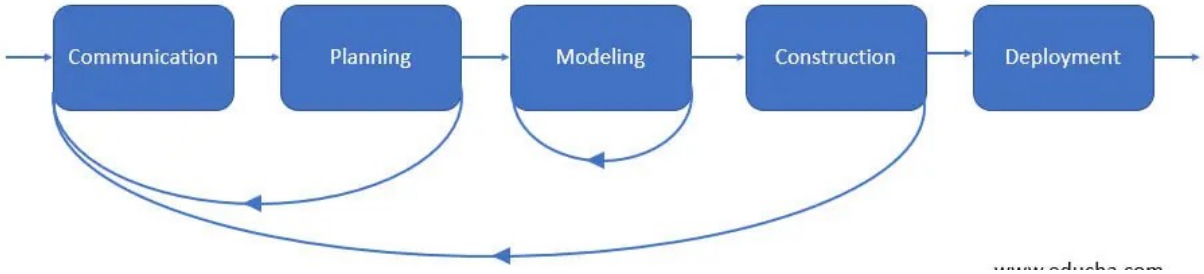
Testing checks if the program or the code provides expected output, and also checks whether the flow of the code is correct or not.

- **Deployment:** This step consists of delivering the final product to the customer and take feedback. If there are any suggestions or addition of other capabilities, then this change is required for improvement in software quality.

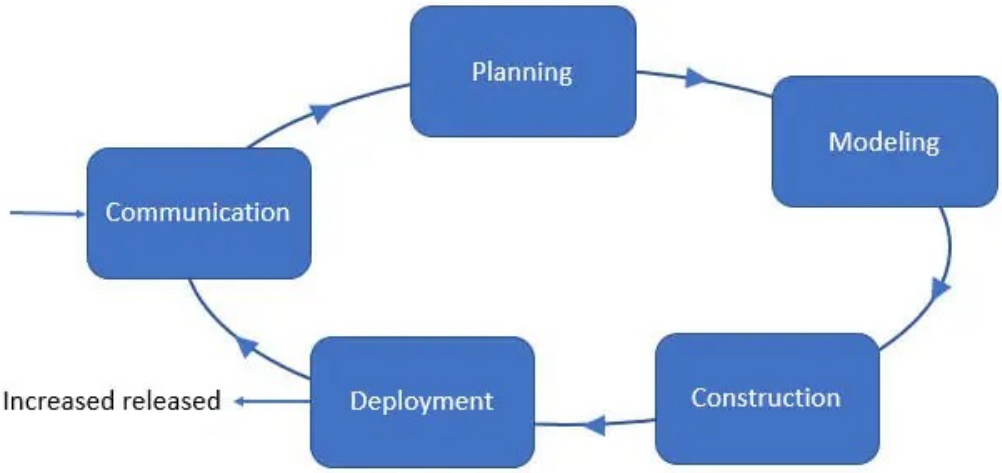
These five framework activities are used for all software developments independent of the application domain, the size of the project, complexity or the efforts, etc.

- **Linear process flow:** It executes each activity listed above in sequence form.

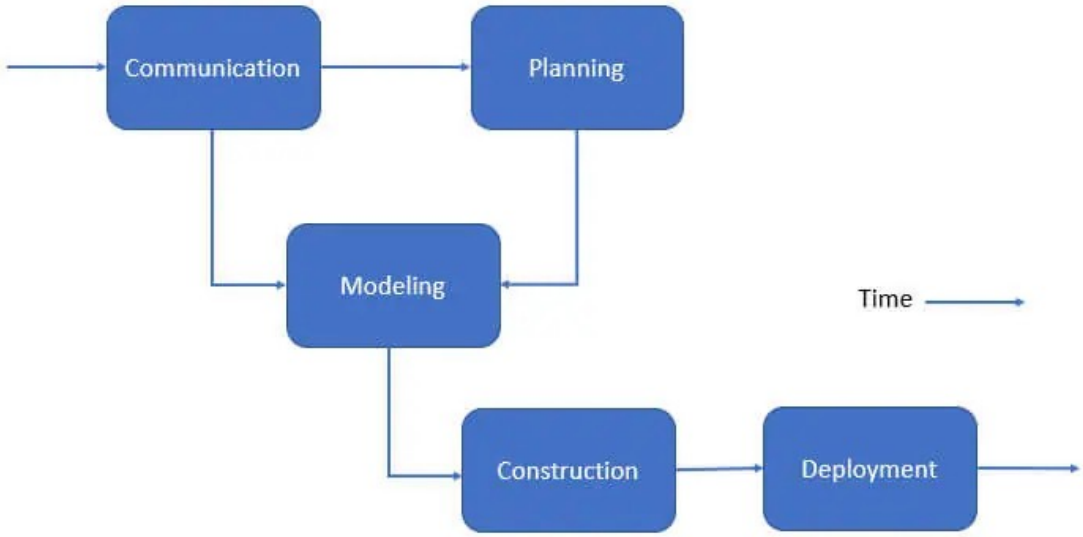
- **Iterative Process flow:** This flow repeats one or more activities that are listed above before starting the next activity.



- **Evolutionary Process flow:** This flow carries out the activities listed above in a circular manner. Each circle leads to the more complete version of the software.



- **Parallel process flow:** This flow executes the activities listed above in parallel with each other i.e. modeling for one aspect of the software in parallel to another aspect of the software.



5. Is it possible to combine different process models? If so, provide an example.

Ans: Yes, it is possible to combine different process models.

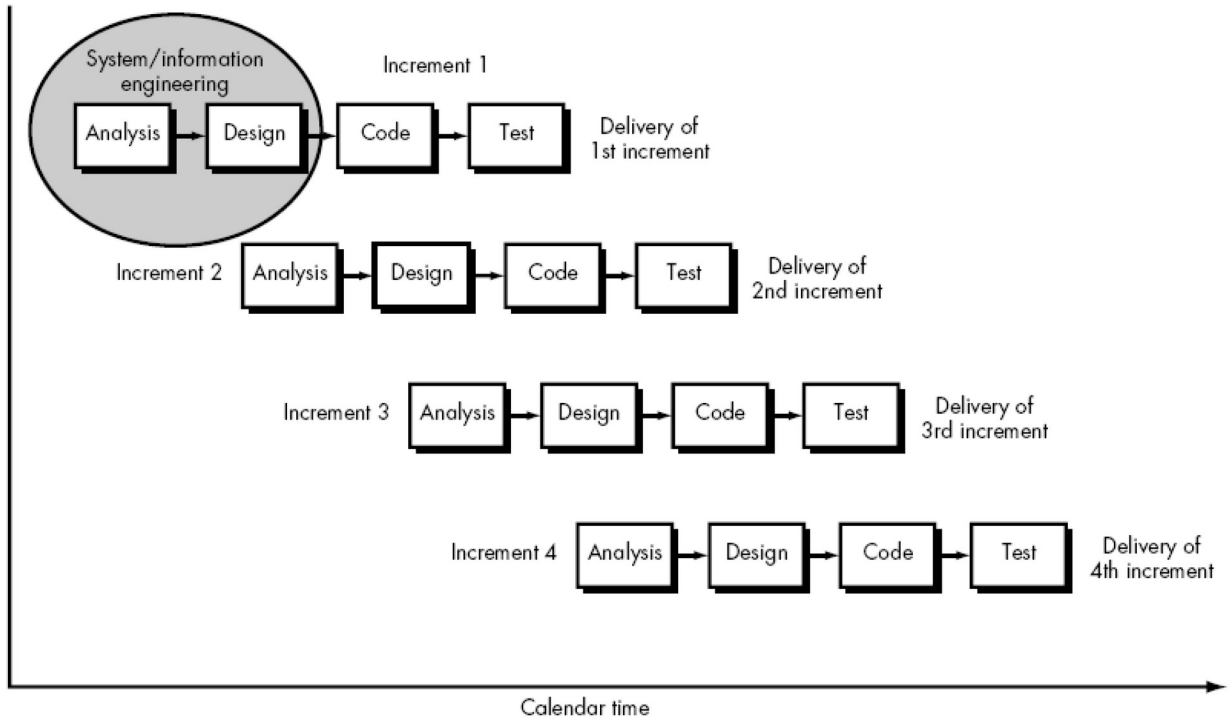
Incremental Process Models

The incremental model combines elements of linear and parallel process flows. The incremental model applies linear sequences in a staggered fashion as calendar time progresses. Each linear sequence produces deliverable “increments” of the software.

When an incremental model is used, the first increment is often a core product. That is, basic requirements are addressed but many supplementary features (some known, others unknown) remain undelivered. The core product is used by the customer (or undergoes detailed evaluation).

As a result of use and/or evaluation, a plan is developed for the next increment. The plan addresses the modification of the core product to better meet the needs of the customer and the delivery of additional features and functionality. This process is repeated following the delivery of each increment, until the complete product is produced.

The incremental process model focuses on the delivery of an operational product with each increment. Early increments are stripped-down versions of the final product, but they do provide capability that serves the user and also provide a platform for evaluation by the user.



6.What is Agility? Why do we need agile process model? What is an agile process? What are the human factors needed to build an agile team?

Ans: Agility means characteristics of being dynamic, content specific, aggressively change embracing and growth oriented. Agility in software development refers to a development team's capacity to quickly respond to evolving requirements and consistently deliver top-notch products within defined timelines.

We need agile process model for -

- **Increased flexibility**: Agile development is more flexible than other project management methodologies. Development teams can make changes on the fly more easily.
- **Improved communication**: Agile development helps to improve communication between the development team and the product owner. Because of this, there is a greater focus on collaboration and feedback.

- **Reduced risks**: Agile development can help to reduce the risks associated with complex projects. By breaking down complex projects into smaller sprints, project managers can dissect them and achieve shareholder demands.
- **Increased customer satisfaction:** Agile development environments often lead to increased customer satisfaction. This is because the customer is involved in the development process and provides feedback at each stage of the project.

The Agile process is a project management approach that involves breaking the project into phases and emphasizes continuous collaboration and improvement. Teams follow a cycle of planning, executing, and evaluating.

Agile team characteristics/Human factors of an agile team :

1. Competence. In an agile development context, “competence encompasses innate, specific software-related skill, and overall knowledge of the process that the team has chosen to apply. Skills and knowledge of process can and should be taught to all people who serves as agile team members.

2. Common focus. Although members of the agile team may perform different tasks and bring different skills to the project, all should be focused on one goal – to deliver a working software increment to the customer within the time promised.

3. Collaboration. Software engineering is about assessing, analyzing, and using information that is communicated to the software team; creating information that will help all stakeholders understand the work of the team; and building information (computer software and relevant databases) that provides business value for the customer. To accomplish these tasks, team members must collaborate – with one another and all other stakeholders.

4. Decision-making ability. Any good software team must be allowed the freedom to control its own destiny. This implies that

the team is given autonomy – decision-making for both technical and project issues.

5. Fuzzy problem-solving ability. Software managers must recognize that agile team will continually have to deal with ambiguity and will continually be buffeted by change and some problems have to solved with imprecise data and incomplete information.

6. Mutual trust and respect. A jelled team exhibits the trust and respect that are necessary to make them "so strongly knit that the whole is greater than the sum of the parts."

7. Self-organization. It implies three things:

(1) the agile team organizes itself for the work to be done

(2) The team selects how much work it believes it can perform within the iteration, and the team commits to the work.

(3) the team organizes the work schedule to best achieve delivery of the software increment.

Self-organization servers to improve collaboration and boost team morale. In essence, the team serves as its own management. Nothing motivates a team as much as accepting the responsibility for fulfilling commitments that it made itself.

7. Illustrate the Scrum agile process model with an appropriate figure.

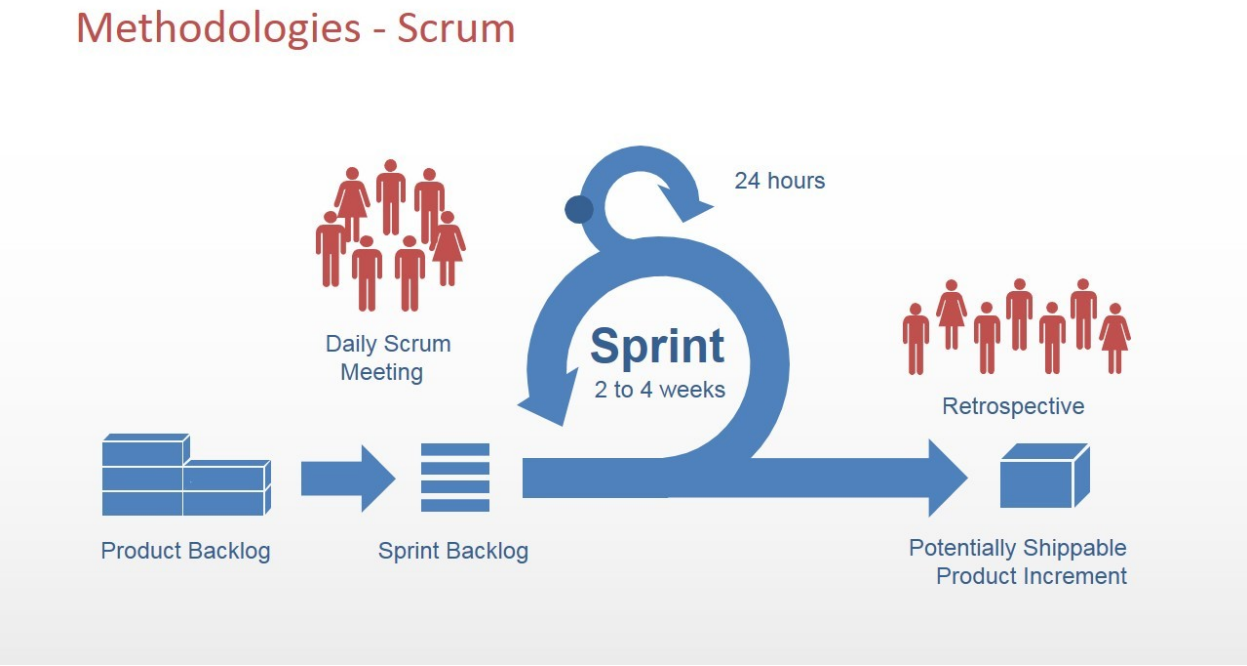
Answer: Scrum is a framework for effective collaborations among teams working on complex products. Scrum is a type of agile technology that consists of meetings, roles, and tools to help teams working on complex projects collaborate and better structure and manage their workload.

Scrum is one of the many types of agile methodology, known for breaking projects down into sizable chunks called "sprints." Agile scrum methodology is good for businesses that need to finish specific projects quickly.

Agile scrum methodology is a project management system that relies on incremental development. Each iteration consists of two- to four-week sprints, where the goal of each sprint is to build the most important features first and come out with a potentially deliverable product. More features are built into the product in subsequent sprints and are adjusted based on stakeholder and customer feedback between sprints.

Whereas other project management methods emphasize building an entire product in one operation from start to finish, agile scrum methodology focuses on delivering several iterations of a product to provide stakeholders with the highest business value in the least amount of time.

Agile scrum methodology has several benefits. First, it encourages products to be built faster, since each set of goals must be completed within each sprint's time frame. It also requires frequent planning and goal setting, which helps the scrum team focus on the current sprint's objectives and increase productivity.



8. Draw the XP Process model and illustrate the XP values.

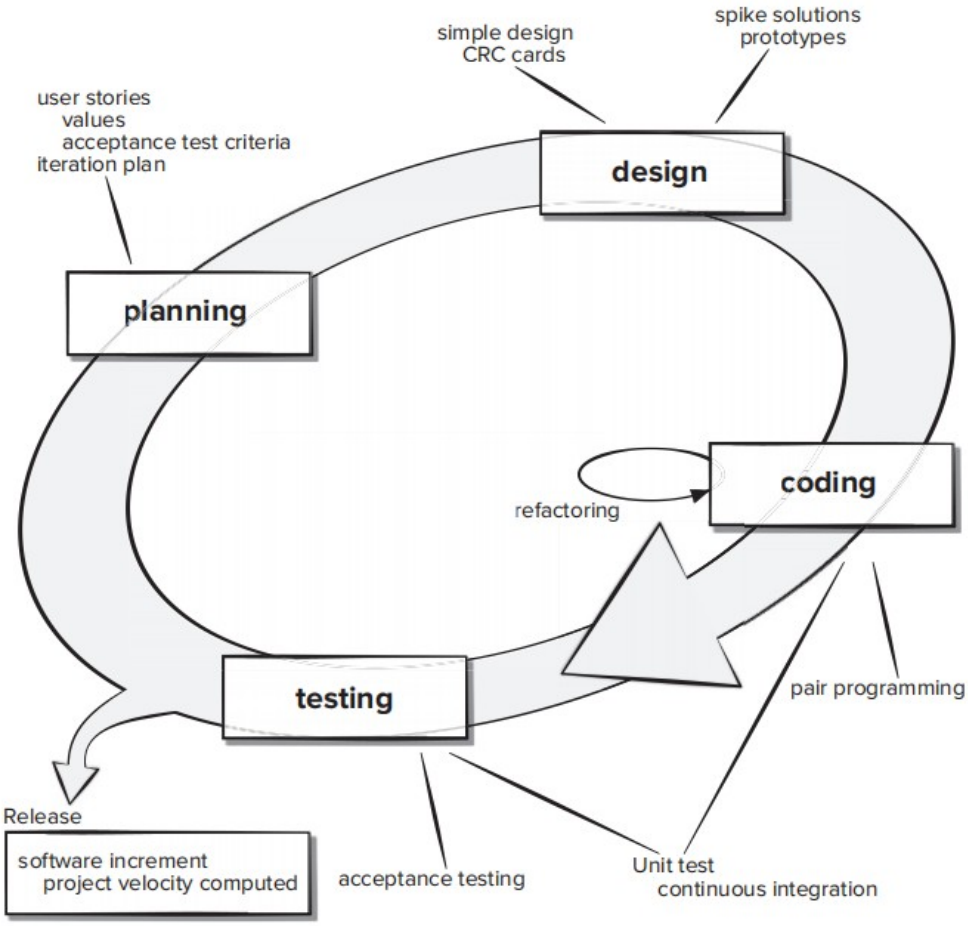
Ans: Extreme Programming encompasses a set of rules and practices that occur within the context of four framework activities: planning, design, coding, and testing.

Planning. The planning activity (also called the *planning game*) begins with a requirements activity called *listening*. Listening leads to the creation of a set of "stories" (also called *user stories*) that describe required output, features, and functionality for software to be built. Each *user story* is written by the customer and is placed on an index card. The customer assigns a *value* (i.e., a priority) to the story based on the overall business value of the feature or function. Members of the XP team then assess each story and assign a *cost*—measured in development weeks—to it. It is important to note that new stories can be written at any time.

(1) all stories will be implemented immediately (within a few weeks)

(2) the stories with highest value will be moved up in the schedule and implemented first, or

(3) the riskiest stories will be moved up in the schedule and implemented first.



Design. XP design rigorously follows the KIS (keep it simple) principle. The design of extra functionality is discouraged.

XP encourages the use of CRC cards as an effective mechanism for thinking about the software in an object-oriented context. CRC (class-responsibility collaborator) cards identify and organize the object-oriented classes that are relevant to the current software increment. CRC cards are the only design work product produced as part of the XP process.

If a difficult design problem is encountered as part of the design of a story, XP recommends the immediate creation of an operational prototype of that portion of the design.

Coding. After user stories are developed and preliminary design work is done, the team does *not* move to code, but rather

develops a series of unit tests that will exercise each of the stories that is to be included in the current release (software increment). After the creation of unit test, developer focuses on coding.

A key concept during the coding activity is *pair programming*. XP recommends that two people work together at one computer to create code for a story.

As pair programmers complete their work, the code they develop is integrated with the work of others. This "continuous integration" strategy helps uncover compatibility and interfacing errors early.

Testing. The unit tests that are created should be implemented using a framework that enables them to be automated (hence, they can be executed easily and repeatedly). This encourages implementing a regression testing strategy when ever code is modified . XP *acceptance tests,* also called *customer tests,* are specified by the customer and focus on overall system features and functionality that are visible and reviewable by the customer.

After the first project release (also called a software increment) has been delivered, the XP team computes project velocity. Stated simply, *project velocity* is the number of customer stories implemented during the first release. Project velocity can then be used to help estimate delivery dates and schedule for subsequent releases. The XP team modifies its plans accordingly.

Values of extreme programming

XP has simple rules that are based on 5 values to guide the teamwork:

1. **Communication.** Everyone on a team works jointly at every stage of the project.
2. **Simplicity.** Developers strive to write simple code bringing more value to a product, as it saves time and effort.

3. **Feedback.** Team members deliver software frequently, get feedback about it, and improve a product according to the new requirements.
4. **Respect.** Every person assigned to a project contributes to a common goal.
5. **Courage.** Programmers objectively evaluate their own results without making excuses and are always ready to respond to changes.

These values represent a specific mindset of motivated team players who do their best on the way to achieving a common goal.

9. Mention the debating factors over XP.

Answer: The XP (Extreme Programming) process model is an agile software development methodology that emphasizes collaboration, flexibility, and rapid feedback. When discussing debating factors related to the XP process model, several key points may arise:

1. **Customer Involvement:** One debating factor is the level of customer involvement in the XP process. Supporters of XP argue that close collaboration with customers leads to better understanding of requirements and quicker feedback, while critics may debate the feasibility of maintaining constant customer involvement throughout the project.

2. **Iterative Development:** Iterative development is a core principle of XP, with frequent releases and continuous improvement. Advocates of XP argue that this approach allows for early detection of issues and adaptation to changing requirements, while critics may question the sustainability and scalability of such rapid iterations.

3. **Pair Programming:** Pair programming, where two programmers work together at one workstation, is a key practice in XP. Supporters argue that it improves code quality, knowledge sharing, and problem-solving, while opponents may debate its efficiency, particularly in terms of resource utilization.

4. **Test-Driven Development (TDD):** Test-driven development involves writing tests before writing code. Advocates assert that TDD leads to more robust and maintainable code, while skeptics may question its practicality and overhead, especially in certain contexts.

5. **Simplicity:** XP emphasizes simplicity in design and implementation. Supporters argue that simpler solutions are easier to understand, maintain, and extend, while opponents may debate whether simplicity can adequately address complex requirements and scalability.

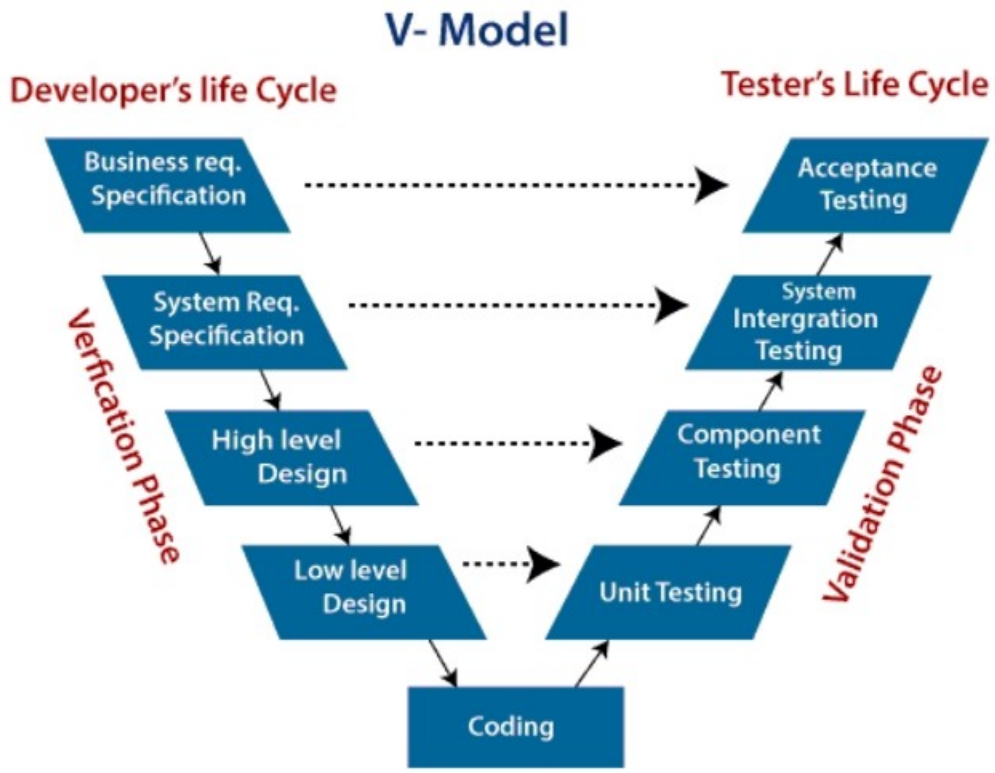
6. **Continuous Integration:** Continuous integration involves integrating code changes frequently and running automated tests. Advocates claim that it reduces integration issues and ensures a more stable codebase, while critics may question the feasibility of maintaining continuous integration in large or distributed teams.

7. **Collective Code Ownership:** XP promotes collective code ownership, where any team member can modify any part of the codebase. Supporters argue that it fosters collaboration and knowledge sharing, while opponents may debate potential challenges related to accountability and code quality.

10. Illustrate the V-Model and compare it with the waterfall model.

Ans: **V-Model** also referred to as the Verification and Validation Model. In this, each phase of SDLC must complete before the next phase starts. It follows a sequential design process same as the waterfall model. Testing of the device is planned in parallel with a corresponding stage of development.

V-Model contains Verification phases on one side of the Validation phases on the other side. Verification and Validation process is joined by coding phase in V-shape. Thus it is known as V-Model.



Verification Phase of V-model:

1. **Business requirement analysis:** Detailed communication to understand customer's expectations and exact product requirements.

2. **System Design:** In this stage system engineers analyze and interpret the business of the proposed system by studying the user requirements document.
3. **Architecture Design:** A process is followed for identifying all the modules making up a system and the framework for module control and communication,brief functionality of each module, their interface relationships, dependencies, database tables etc.
4. **Module Design:** In the module design phase, the system breaks down into small modules. The detailed design of the modules is specified, which is known as Low-Level Design.
5. **Coding Phase:** After designing, the coding phase is started. Based on the requirements, a suitable programming language is decided. Before checking in the repository, the final build is optimized for better performance, and the code goes through many code reviews to check the performance.

Validation Phase of V-model:

1. **Unit Testing:** Unit Test Plans (UTPs) are developed during the module design phase and executed to eliminate errors at code level or unit level. A unit is the smallest entity which can independently exist, e.g., a program module. Unit testing verifies that the smallest entity can function correctly when isolated from the rest of the codes/ units.
2. **Integration Testing:** Integration Test Plans are developed during the Architectural Design Phase. These tests verify that the groups created and tested independently can coexist and communicate among themselves.
3. **System Testing:** System Tests Plans are developed during System Design Phase. System Test ensures that expectations from an application developer are met and are composed by the clients business team.
4. **Acceptance Testing:** Acceptance testing is related to the business requirement analysis part. It includes testing the

software product in user atmosphere. Acceptance tests reveal the compatibility problems with the different systems. It conjointly discovers the non-functional problems like load and performance defects within the real user atmosphere.

S. No.	Waterfall model	V-model
1.	The cost of Waterfall model is low.	V-model is expensive.
2.	Flexibility of Waterfall model is Rigid.	Flexibility of V-model is Little flexible.
3.	There is no way to return to the earlier phase.	There is no such constraint in V-model.
4.	Waterfall model is a sequential execution process.	It is also a sequential execution process.
5.	Waterfall model’s steps move in a linear way.	V-model’s steps don’t move in linear way.
6.	User involvement in Waterfall model is only in beginning.	User involvement in V-model is also only in beginning.
7.	Guarantee of success through Waterfall model is low.	Guarantee of success through V-model is high.
8.	It is not possible to test a software during its development.	There is possibility to test a software during its development.
9.	Debugging is done after the last phase.	Debugging can be done in between phases.

11. Illustrate the evolutionary process flow models and identify their weaknesses.

Answer: **Evolutionary Process Model**

The evolutionary model is based on the concept of making an initial product and then evolving the software product over time with iterative and incremental approaches with proper feedback. In this type of model, the product will go through several iterations and come up when the final product is built through multiple iterations.

Following are the evolutionary process models.

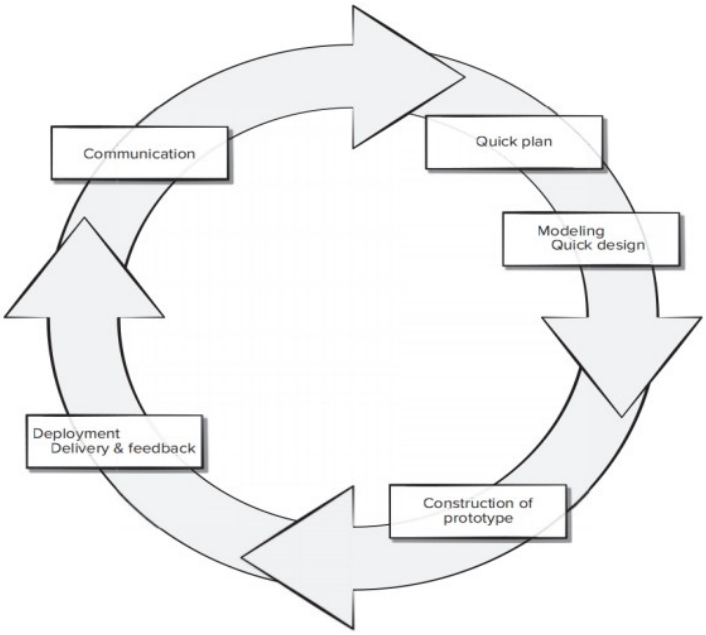
1. The prototyping model
2. The spiral model

Prototyping Process Model

Often, a customer defines a set of general objectives for software but does not identify detailed requirements for functions and features. In other cases, the developer may be unsure of the efficiency of an algorithm, the adaptability of an operating system, or the form that human-machine interaction should take. In these, and many other situations, a *prototyping paradigm* may offer the best approach.

-can be implemented within the context of any one of the process models

-assists you and other stakeholders to better understand what is to be built when requirements are fuzzy.



The prototyping paradigm begins with communication. You meet with other stakeholders to define the overall objectives for the software, identify whatever requirements are known, and outline areas where further definition is mandatory. A prototyping iteration is planned quickly, and modeling (in the form of a “quick design”) occurs. A quick design focuses on a representation of those aspects of the software that will be visible to end users (e.g., human interface layout or output display formats).

The quick design leads to the construction of a prototype. The prototype is deployed and evaluated by stakeholders, who provide feedback that is used to further refine requirements. Iteration occurs as the prototype is tuned to satisfy the needs of various stakeholders, while at the same time enabling you to better understand what needs to be done.

Ideally, the prototype serves as a mechanism for identifying software requirements. If a working prototype is to be built, you can make use of existing program fragments or apply tools that enable working programs to be generated quickly.

Both stakeholders and software engineers like the prototyping paradigm. Users get a feel for the actual system, and developers

get to build something immediately. Yet, prototyping can be problematic for the following reasons:

1. Stakeholders see what appears to be a working version of the software. They may be unaware that the prototype architecture (program structure) is also evolving. This means that the developers may not have considered the overall software quality or long-term maintainability.

2. As a software engineer, you may be tempted to make implementation compromises to get a prototype working quickly. If you are not careful, these less than-ideal choices have now become an integral part of the evolving system.

3. Work is lost in a throwaway prototype.

The key is to define the rules of the game at the beginning; that is, all stakeholders should agree that the prototype is built in part to serve as a mechanism for defining requirements. It is often desirable to design a prototype so it can be evolved into the final product. The reality is developers may need to discard (at least in part) a prototype to better meet the customer's evolving needs.

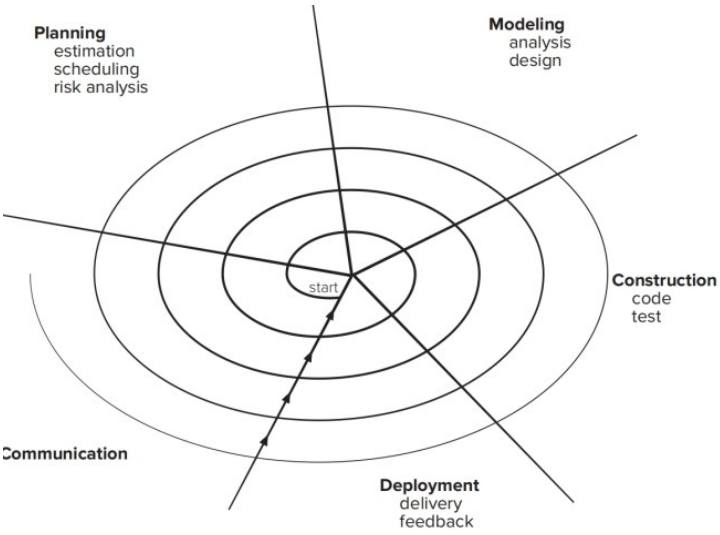
Spiral Model

The *spiral model* is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model.

Using the spiral model, software is developed in a series of evolutionary releases. During early iterations, the release might be a model or prototype. During later iterations, increasingly more complete versions of the engineered system are produced. A spiral model is divided into a set of framework activities defined by the software engineering team.

As this evolutionary process begins, the software team performs activities that are implied by a circuit around the spiral in a clockwise direction, beginning at the center. Risk is considered as each revolution is made. Anchor point milestones—a

combination of work products and conditions that are attained along the path of the spiral—are noted for each evolutionary pass. The first circuit around the spiral might result in the development of a product specification; subsequent passes around the spiral might be used to develop a prototype and then progressively more sophisticated versions of the software. Each pass through the planning region results in adjustments to the project plan. Cost and schedule are adjusted based on feedback derived from the customer after delivery. In addition, the project manager adjusts the planned number of iterations required to complete the software.



Spiral Cons:

i. Risk analysis failures can doom the project.
ii. The project may be hard to manage.
iii. It requires an expert development team.

12. Define requirements engineering? What are the generic steps for Requirements Engineering?

Requirements engineering is the term for the broad spectrum of tasks and techniques that lead to an understanding of requirements. From a software process perspective, requirements engineering is a major software engineering action

that begins during the communication activity and continues into the modeling activity.

Requirements engineering establishes a solid base for design and construction. Without it, the resulting software has a high probability of not meeting customer's needs. It must be adapted to the needs of the process, the project, the product, and the people doing the work. It is important to realize that each of these tasks is done iteratively as the project team and the stakeholders continue to share information about their respective concerns.

Requirements engineering builds a bridge to design and construction. But where does the bridge originate? One could argue that it begins with the project stakeholders (e.g., managers, customers, and end users), where business needs are defined, user scenarios are described, functions and features are delineated, and project constraints are identified. Others might suggest that it begins with a broader system definition, where software is but one component of the larger system domain. But regardless of the starting point, the journey across the bridge takes you high above the project, allowing you to examine the context of the software work to be performed; the specific needs that design and construction must address; the priorities that guide the order in which work is to be completed; and the information, functions, and behaviors that will have a profound impact on the resulting design.

Requirements engineering encompasses seven tasks with sometimes muddy boundaries: *inception, elicitation, elaboration, negotiation, specification, validation,* and *management.*

It is important to note that some of these tasks occur in parallel and all are adapted to the needs of the project. Expect to do a bit of design during requirements work and a bit of requirements work during design.

13. You have been given a responsibility to elicit requirements from a customer who tells you he is too busy to meet with you. What should you do?

Ans: When a customer is too busy to meet in person, there are still several effective ways to elicit requirements from them:

1. **Schedule a Phone Call:** Arrange a time for a phone call where you can discuss their requirements in detail. This allows for a more personal interaction while accommodating their busy schedule.
2. **Send a Detailed Questionnaire:** Prepare a detailed questionnaire or survey that the customer can fill out at their convenience. This can help gather specific requirements and preferences.
3. **Request Email Feedback:** Ask the customer to provide written feedback via email. This allows them to respond when they have a moment and provides a record of their requirements.
4. **Utilize Collaboration Tools:** Use online collaboration tools or project management platforms where the customer can contribute their requirements and feedback at their own pace.
5. **Engage Stakeholders:** If the customer is unavailable, consider engaging with other stakeholders who have insight into the requirements and can provide valuable input.

By being flexible and accommodating, you can still effectively elicit requirements from a busy customer without requiring them to meet in person

14. Discuss some of the problems that occur when requirements must be elicited from three or four different customers.

Answer:

Diverse Expectations: Different customers may have different expectations and requirements. This can lead to conflicts and disagreements, making it difficult to reach a consensus on what the final product should look like.

Time and Resource Constraints: Eliciting requirements from multiple customers can be time-consuming and resource-intensive. It may also lead to delays in the project timeline. Public organizations are under intense pressure to satisfy stakeholders' needs despite limited funds, people, and equipment.

Confidentiality Issues: There may be confidentiality issues if the requirements of one customer are shared with another customer.

Identifying critical requirements: Identifying the set of requirements that have to be implemented at any cost is very important. The requirements should be prioritized so that crucial ones can be implemented first with the highest priority.

Competing priorities between stakeholders: Stakeholders carry their own expectations and goals into the project. For example, you may have a stakeholder who's own personal goal is to wrap-up the project before they leave on vacation. They may be highly committed to seeing the project's completion happen as soon as possible, even if this timeline is shorter than what other stakeholders might prefer.

15. Develop at least four context free questions that you might ask a stakeholder during inception?

Answer: Questions asked at the inception of the project should be "context free". The first set of context-free questions focuses on the customer and other stakeholders and the overall project goals and benefits. For example, you might ask:

- Who is behind the request for this work?
- Who will use the solution?
- What will be the economic benefit of a successful solution?

• Is there another source for the solution that you need?

These questions help to identify all stakeholders who will have interest in the software to be built. In addition, the questions identify the measurable benefit of a successful implementation and possible alternatives to custom software development.

16. What does "win-win" mean in the context of negotiation during the requirements engineering activity?

Answer: In an ideal world, the requirements engineering tasks (inception, elicitation, and elaboration) determine customer requirements in sufficient detail to proceed to subsequent software engineering activities. Unfortunately, this rarely happens. You may have to enter into *negotiations* with one or more stakeholders. In most cases, stakeholders are asked to balance functionality, performance, and other product or system characteristics against cost and time to market. The intent of these negotiations is to develop a project plan that meets stakeholder needs while at the same time reflecting the real-world constraints (e.g., time, people, budget) that have been placed on the software team.

The best negotiations strive for a “win-win” result. That is, stakeholders win by getting the system or product that satisfies most their needs, and you (as a member of the software team) win by working to realistic and achievable budgets and deadlines.

The **Win Win** strategy entails engaging the key stakeholders in a system's success in a negotiation process, so they may come to an agreement on a set of needs that will **benefit** both parties.

17. Let's assume that you have convinced the customer to agree to every demand that you have as a developer. Does that make you a master negotiator? Why?

Answer: Convincing a customer to agree to every demand as a developer does not necessarily make you a master negotiator. While it may demonstrate effective communication skills and

persuasion techniques, true mastery in negotiation involves more than just getting your demands met.

A master negotiator understands the needs and interests of both parties involved and works towards a mutually beneficial solution. This often requires active listening, empathy, creativity, and problem-solving skills. It's about finding common ground, exploring alternatives, and building relationships based on trust and respect.

Additionally, a master negotiator knows when to compromise and when to stand firm, recognizing that not every demand can or should be met. They also understand the importance of long-term relationships and the potential consequences of pushing too hard or being overly aggressive in negotiations.

In summary, while convincing a customer to agree to every demand can be a sign of negotiation skills, true mastery in negotiation goes beyond just getting what you want and focuses on achieving outcomes that satisfy all parties involved.

High probability

5.1 Why is it that many software developers don’t pay enough attention to requirements engineering? Are there ever circumstances where you can skip it?

Designing and building computer software is challenging, creative, and just plain fun. In fact, building software is so compelling that many software developers want to jump right in before they have a clear understanding of what is needed.

They argue that things will become clear as they build, that project stakeholders will be able to understand need only after examining early iterations of the software, that things change so rapidly that any attempt to understand requirements in detail is

a waste of time, that the bottom line is producing a working program and all else is secondary.What makes these arguments

seductive is that they contain elements of truth if just for small project (less than one month) and smaller. But as software grows in size and comp

lexity, these arguments begin to break down.

The broad spectrum of tasks and techniques that lead to an understanding of requirements is called Requirement engineering. Requirement engineering very important to understand what the customer wants, analyzing needs, assessing feasibility, negotiating a reasonable solution, specifying the solution unambiguously, validating the specification, and managing the requirements as they are transformed into an operational system.

Requirement engineering divide into : inception, elicitation, elaboration, negotiation, specification, validation, and management. I think the step that can we skip is management, if we create a big software that means complex and big size, maybe management is very important, but if we just make a small software for example for your reset or just for fun, I think management not really important. So the step that we can skip is management, but it’s depend on the software that we creating.

5.4 Why do we say that the requirements model represents a snapshot of a system in time?

The intent of the analysis model is to provide a description of the required informational, functional, and behavioral domains for a computer-based system. The model changes dynamically as you learn more about the system to be built, and other stakeholders understand more about what they really require. For that reason, the analysis model is a snapshot of requirements at any given time.

If we want make a software that stakeholder wants, we must requirements models of the software that we want to create. If we create models, this models represent how the software works in time from example who will use this system, how this system work, etc. That’s why requirements model represents a snapshot of a system in time.