1. **Explain Agility in Software Engineering**

Rapid delivery of operational software and ignoring the importance of intermediate work products (not always a good thing); it adopts the customer as a part of the development team and works to eliminate the “us and them” attitude that continues to pervade many software projects; it recognizes that planning in an uncertain world has its limits and that a project plan must be flexible.

Agility can be applied to any software process. However, to accomplish this, it is essential that the process be designed in a way that allows the project team to adapt tasks and to streamline them, conduct planning in a way that understands the fluidity of an agile development approach, eliminate all but the most essential work products and keep them lean, and emphasize an incremental delivery strategy that gets working software to the customer as rapidly as feasible for the product type and operational environment.

2. **Explain the cost of change phenomenon in Agility**

The conventional wisdom in software development is that the cost of change increases nonlinearly as a project progresses.

It is relatively easy to accommodate a change when a software team is gathering requirements

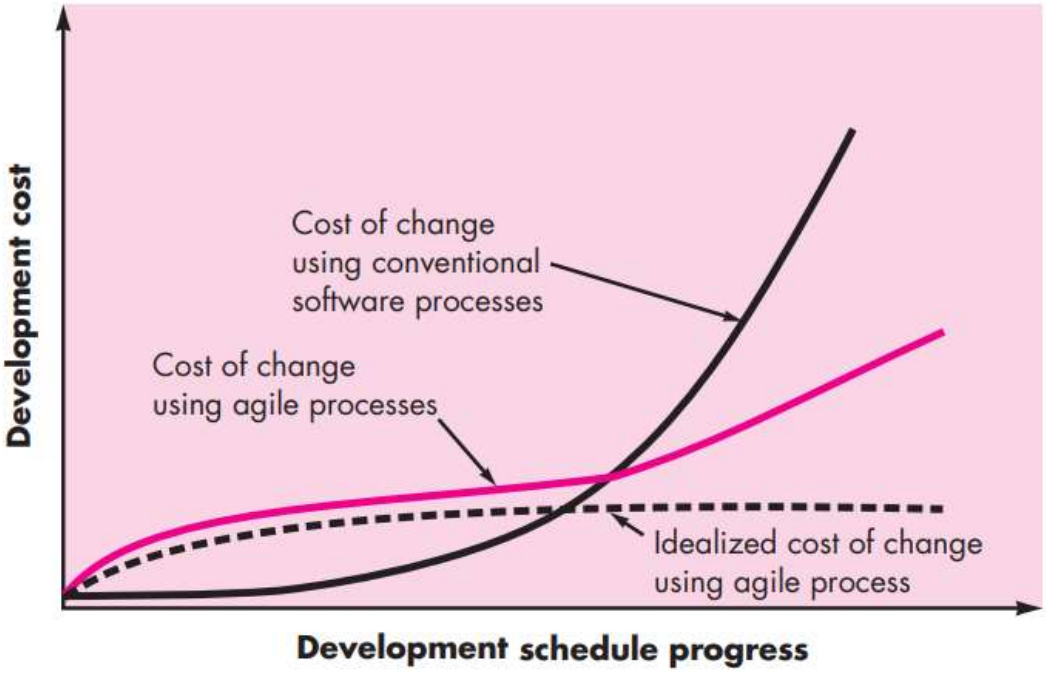
A usage scenario might have to be modified, a list of functions may be extended, or a written specification can be edited. The costs of doing this work are minimal, and the time required will not adversely affect the outcome of the project.

The team is in the middle of validation testing and an important stakeholder is requesting a major functional change.

The change requires a modification to the architectural design of the software, the design and construction of three new components, modifications to another five components, the design of new tests, and so on.

Costs escalate quickly, and the time and cost required to ensure that the change is made without unintended side effects is nontrivial.

When incremental delivery is coupled with other agile practices such as continuous unit testing and pair programming, the cost of making a change is attenuated.



3. Explain the assumptions addressed by the Agile Process.

1. It is difficult to predict in advance which software requirements will persist and which will change. It is equally difficult to predict how customer priorities will change as the project proceeds.

2. For many types of software, design and construction are interleaved. That is, both activities should be performed in tandem so that design models are proven as they are created. It is difficult to predict how much design is necessary before construction is used to prove the design.

3. Analysis, design, construction, and testing are not as predictable as we might like

4. **Explain the principles of Agility.**

The Agile Alliance (see [Agi03], [Fow01]) defines 12 agility principles for those who want to achieve agility:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

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

5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

7. Working software is the primary measure of progress.

8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

9. Continuous attention to technical excellence and good design enhances agility.

10. Simplicity—the art of maximizing the amount of work not done—is essential.

11. The best architectures, requirements, and designs emerge from self– organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly

5. **Paraphrase the Politics of Agile Development**

The feeling of the pro-agility camp ("agilists") when considered, gives us the feeling of Traditional methodologists.

"Traditional methodologists are a bunch of stick-in-the-muds who'd rather produce flawless documentation than a working system that meets business needs."

The position of the traditional software engineering camp can be stated as:

"Light-weight, er, 'agile' methodologists are a bunch of glorified hackers who are going to be in for a heck of a surprise when they try to scale up their toys into enterprise-wide software."

Like all software technology arguments, this methodology debate risks degenerating into a religious war.

The real question is: What is the best way to achieve it? As important, how do you build software that meets customers' needs today and exhibits the quality characteristics that will enable it to be extended and scaled to meet customers' needs over the long term?

There are no absolute answers to either of these questions.

Bottom line: there is much that can be gained by considering the best of both schools and virtually nothing to be gained by denigrating either approach.

6**. Describe the human factors considered in Agile Software Development**

Competence. "competence" encompasses innate talent, specific software-related skills, and overall knowledge of the process that the team has chosen to apply. Skill and knowledge of process can and should be taught to all people who serve as agile team members.

Common focus. Although members of the agile team may perform differ ent 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. To achieve this goal, the team will also focus on continual adaptations (small and large) that will make the process fit the needs of the team.

Collaboration. Software engineering (regardless of process) 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.

Decision-making ability. Any good software team (including agile teams) must be allowed the freedom to control its own destiny. This implies that the team is given autonomy—decision-making authority for both technical and project issues.

Fuzzy problem-solving ability. Software managers must recognize that the agile team will continually have to deal with ambiguity and will continually be buffeted by change. In some cases, the team must accept the fact that the problem they are solving today may not be the problem that needs to be solved tomorrow.

Mutual trust and respect. The agile team must become what DeMarco and Lister [DeM98] call a "jelled" team. 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."

Self-organization. In the context of agile development, self-organization implies three things:

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

(2) the team organizes the process to best accommodate its local environment,

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

7. **Describe the Extreme Programming values**

A set of five values that establish a foundation for all work performed as part of XP—communication, simplicity, feedback, courage, and respect.

In order to achieve effective communication between software engineers and other stakeholders, XP emphasizes close, yet informal (verbal) collaboration between customers and developers, the establishment of effective metaphors for communicating important concepts, continuous feedback, and the avoidance of voluminous documentation as a communication medium.

To achieve simplicity, XP restricts developers to design only for immediate needs, rather than consider future needs.

The intent is to create a simple design that can be easily implemented in code.

If the design must be improved, it can be refactored at a later time.

Feedback is derived from three sources:

- the implemented software itself,
- the customer, and
- other software team members.

By designing and implementing an effective testing strategy, the software (via test results) provides the agile team with feedback. XP makes use of the unit test as its primary testing tactic.

As each class is developed, the team develops a unit test to exercise each operation according to its specified functionality.

As an increment is delivered to a customer, the user stories or use cases that are implemented by the increment are used as a basis for acceptance tests.

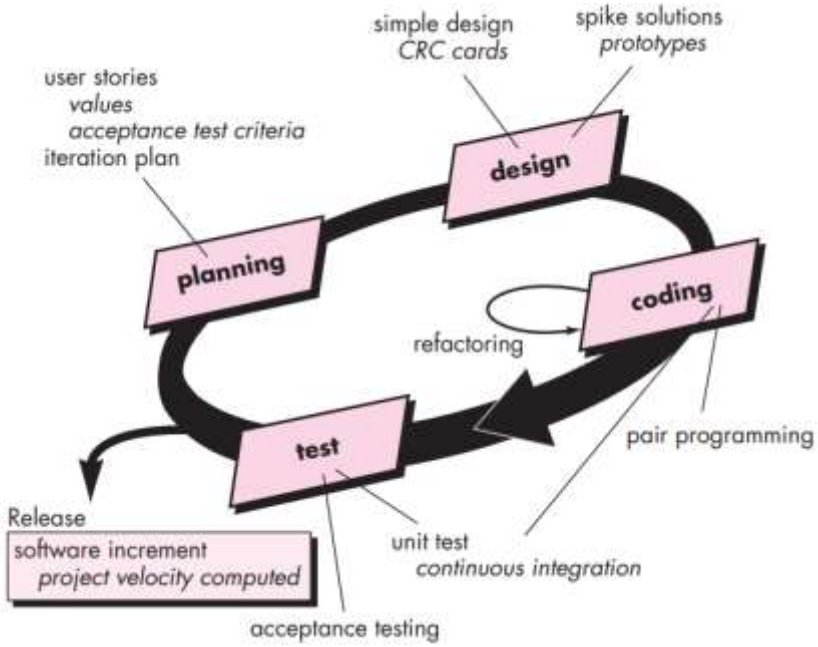
The degree to which the software implements the output, function, and behaviour of the use case is a form of feedback.

Finally, as new requirements are derived as part of iterative planning, the team provides the customer with rapid feedback regarding cost and schedule impact.

8. **Illustrate the XP Process with a neat diagram**

Key XP activities are summarized in the paragraphs that follow.

Planning. The planning activity (also called the planning game) begins with listening—a requirements gathering activity that enables the technical members of the XP team to understand the business context for the software and to get a broad feel for required output



and major features and functionality.

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 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.

If the story is estimated to require more than three development weeks, the customer is asked to split the story into smaller stories and the assignment of value and cost occurs again.

It is important to note that new stories can be written at any time.

Customers and developers work together to decide how to group stories into the next release (the next software increment) to be developed by the XP team.

Once a basic commitment is made for a release, the XP team orders the stories that will be developed in one of three ways:

(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.

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

(1) help estimate delivery dates and schedule for subsequent releases and

(2) determine whether an overcommitment has been made for all stories across the entire development project.

If an overcommitment occurs, the content of releases is modified or end delivery dates are changed.

As development work proceeds, the customer can add stories, change the value of an existing story, split stories, or eliminate them.

The XP team then reconsiders all remaining releases and modifies its plans accordingly.

Design. XP design rigorously follows the KIS (keep it simple) principle.

A simple design is always preferred over a more complex representation.

In addition, the design provides implementation guidance for a story as it is written—nothing less, nothing more.

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.

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.

Called a spike solution, the design prototype is implemented and evaluated.

The intent is to lower risk when true implementation starts and to validate the original estimates for the story containing the design problem.

The intent of refactoring is to control these modifications by suggesting small design changes that "can radically improve the design" .

A central notion in XP is that design occurs both before and after coding commences. Refactoring means that design occurs continuously as the system is constructed.

Coding. After 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).

Once the unit test has been created, the developer is better able to focus on what must be implemented to pass the test.

Once the code is complete, it can be unit-tested immediately, thereby providing instantaneous feedback to the developers.

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

This provides a mechanism for real time problem solving (two heads are often better than one) and real-time quality assurance (the code is reviewed as it is created).

It also keeps the developers focused on the problem at hand.

This "continuous integration" strategy helps to avoid compatibility and interfacing problems and provides a "smoke testing" environment that helps to uncover 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).

As the individual unit tests are organized into a "universal testing suite", integration and validation testing of the system can occur on a daily basis.

This provides the XP team with a continual indication of progress and also can raise warning flags early if things go awry.

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.

Acceptance tests are derived from user stories that have been imple mented as part of a software release.

9. **Explain the concept of Industrial XP**

IXP incorporates six new practices that are designed to help ensure that an XP project works successfully for significant projects within a large organization

Readiness assessment. Prior to the initiation of an IXP project, the organization should conduct a readiness assessment. The assessment ascertains whether

(1) an appropriate development environment exists to support IXP,

(2) the team will be populated by the proper set of stakeholders,

(3) the or ganization has a distinct quality program and supports continuous improvement,

(4) the organizational culture will support the new values of an agile team, and

(5) the broader project community will be populated appropriately.

Project community. Classic XP suggests that the right people be used to populate the agile team to ensure success.

The implication is that people on the team must be well-trained, adaptable and skilled, and have the proper temperament to contribute to a self-organizing team.

When XP is to be applied for a significant project in a large organization, the concept of the "team" should morph into that of a community.

In IXP, the community members and their roles should be explicitly defined and mechanisms for communication and coordination between community members should be established.

Project chartering. The IXP team assesses the project itself to determine whether an appropriate business justification for the project exists and whether the project will further the overall goals and objectives of the organization.

Chartering also examines the context of the project to determine how it complements, extends, or replaces existing systems or processes.

Test-driven management. An IXP project requires measurable criteria for assessing the state of the project and the progress that has been made to date.

Test-driven management establishes a series of measurable "destinations" and then defines mechanisms for determining whether or not these destinations have been reached.

Retrospectives. An IXP team conducts a specialized technical review after a software increment is delivered. Called a retrospective, the review examines "issues, events, and lessons-learned" across a software increment and/or the entire software release.

The intent is to improve the IXP process.

Continuous learning. Because learning is a vital part of continuous process improvement, members of the XP team are encouraged to learn new methods and techniques that can lead to a higher quality product.

10. **Paraphrase on the XP debate with relevant key factors**

The codependent nature of XP practices are both its strength and its weakness.

Because many organizations adopt only a subset of XP practices, they weaken the efficacy of the entire process.

Proponents counter that XP is continuously evolving and that many of the issues raised by critics have been addressed as XP practice matures.

Some critics of XP are:

• **Requirements volatility**. Because the customer is an active member of the XP team, changes to requirements are requested informally.

As a consequence, the scope of the project can change and earlier work may have to be modified to accommodate current needs.

Proponents argue that this happens regardless of the process that is applied and that XP provides mechanisms for controlling scope creep.

• **Conflicting customer needs**. Many projects have multiple customers, each with his own set of needs. In XP, the team itself is tasked with assimilating the needs of different customers, a job that may be beyond their scope of authority.

• **Requirements are expressed informally**. User stories and acceptance tests are the only explicit manifestation of requirements in XP.

Critics argue that a more formal model or specification is often needed to ensure that omissions, inconsistencies, and errors are uncovered before the system is built.

Proponents counter that the changing nature of requirements makes such models and specification obsolete almost as soon as they are developed.

• **Lack of formal design.** XP deemphasizes the need for architectural design and in many instances, suggests that design of all kinds should be relatively informal.

Critics argue that when complex systems are built, design must be emphasized to ensure that the overall structure of the software will exhibit quality and maintainability. XP proponents suggest that the incremental nature of the XP process limits complexity and therefore reduces the need for extensive design.

11. **Explain Adaptive Software Development method with a neat diagram**

Adaptive Software Development (ASD) has been proposed by Jim Highsmith as a technique for building complex software and systems.

The philosophical under pinning of ASD focus on human collaboration and team self-organization.

An agile, adaptive development approach based on collaboration is “as much a source of order in our complex interactions as discipline and engineering.”

He defines an ASD “life cycle” that incorporates three phases, speculation, collaboration, and learning During speculation, the project is initiated and adaptive cycle planning is conducted.

Adaptive cycle planning uses project initiation information—the customer’s mission statement, project constraints (e.g., delivery dates or user descriptions), and basic requirements—to define the set of release cycles (software increments) that will be required for the project.

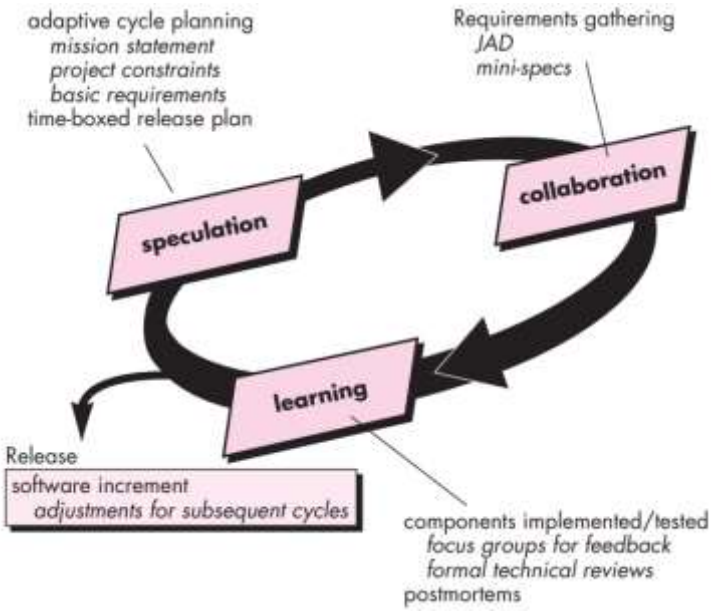
No matter how complete and farsighted the cycle plan, it will invariably change.

Based on information obtained at the completion of the first cycle, the plan is reviewed and adjusted so that planned work better fits the reality in which an ASD team is working.

Motivated people use collaboration in a way that multiplies their talent and creative output beyond their absolute numbers.

This approach is a recurring theme in all agile methods. But collaboration is not easy.

It encompasses communication and teamwork, but it also emphasizes individualism, because individual creativity plays an important role in collaborative thinking.



People working together must trust one another to

(1) criticize without animosity,

(2) assist without resentment,

(3) work as hard as or harder than they do,

(4) have the skill set to contribute to the work at hand, and

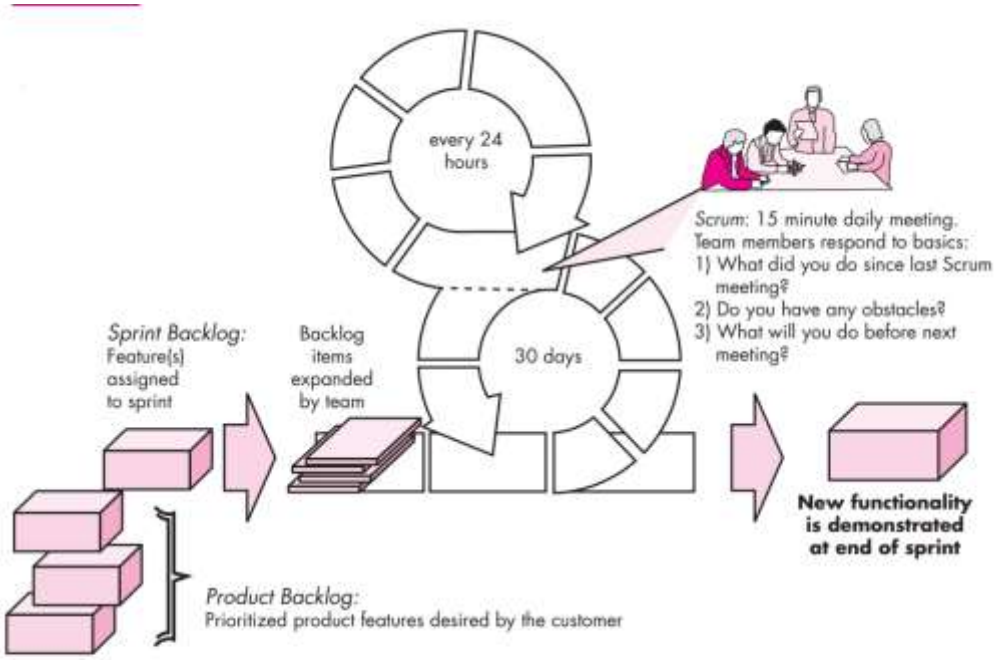
(5) communicate problems or concerns in a way that leads to effective action.

The ASD philosophy has merit regardless of the process model that is used. ASD's overall emphasis on the dynamics of self-organizing teams, interpersonal collaboration, and individual and team learning yield software project teams that have a much higher likelihood of success.

12. **Explain Scrum – agile development method with relevant diagram**

Scrum is an agile software development method that was conceived by Jeff Sutherland and his development team in the early 1990s.

Scrum principles are consistent with the agile manifesto and are used to guide development activities within a process that incorporates the following framework activities: requirements, analysis, design, evolution, and delivery.



Within each framework activity, work tasks occur within a process pattern called a sprint.

The work conducted within a sprint is adapted to the problem at hand and is defined and often modified in real time by the Scrum team.

The overall flow of the Scrum process is illustrated in Figure shown above.

Scrum emphasizes the use of a set of software process patterns that have proven effective for projects with tight timelines, changing requirements, and business criticality.

Each of these process patterns defines a set of development actions:

Backlog—a prioritized list of project requirements or features that provide business value for the customer. Items can be added to the backlog at any time. The product manager assesses the backlog and updates priorities as required.

Sprints—consist of work units that are required to achieve a requirement de fined in the backlog that must be fit into a predefined time-box14 (typically 30 days)

Scrum meetings—are short (typically 15 minutes) meetings held daily by the Scrum team. Three key questions are asked and answered by all team members:

• What did you do since the last team meeting?

• What obstacles are you encountering?

• What do you plan to accomplish by the next team meeting? A

team leader, called a Scrum master, leads the meeting and assesses the responses from each person. The Scrum meeting helps the team to uncover potential problems as early as possible. Also, these daily meetings lead to "knowledge socialization" [Bee99] and thereby promote a self-organizing team structure.

Demos—deliver the software increment to the customer so that functionality that has been implemented can be demonstrated and evaluated by the customer.

It is important to note that the demo may not contain all planned functionality, but rather those functions that can be delivered within the time-box that was established.

13. **Explain the agile based Dynamic Systems Development Method (DSDM)**

The Dynamic Systems Development Method (DSDM) is an agile software development approach that "provides a framework for building and maintaining systems which meet tight time constraints through the use of incremental prototyping in a controlled project environment".

The DSDM philosophy is borrowed from a modified version of the Pareto principle—80 percent of an application can be delivered in 20 percent of the time it would take to deliver the complete (100 percent) application.

DSDM is an iterative software process in which each iteration follows the 80 percent rule.

That is, only enough work is required for each increment to facilitate movement to the next increment. The remaining detail can be completed later when more business requirements are known or changes have been requested and accommodated.

The consortium has defined an agile process model, called the DSDM life cycle that defines three different iterative cycles, preceded by two additional life cycle activities:

Feasibility study—establishes the basic business requirements and constraints associated with the application to be built and then assesses whether the application is a viable candidate for the DSDM process.

Business study—establishes the functional and information requirements that will allow the application to provide business value; also, defines the basic application architecture and identifies the maintainability requirements for the application.

Functional model iteration—produces a set of incremental prototypes that demonstrate functionality for the customer. The intent during this iterative cycle is to gather additional requirements by eliciting feedback from users as they exercise the prototype.

Design and build iteration—revisits prototypes built during functional model iteration to ensure that each has been engineered in a manner that will enable it to provide operational business value for end users. In some cases, functional model iteration and design and build iteration occur concurrently.

Implementation—places the latest software increment (an "operationalized" pro□totype) into the operational environment.

It should be noted that

(1) the increment may not be 100 percent complete or

(2) changes may be requested as the increment is put into place. In either case, DSDM development work continues by returning to the functional model iteration activity

14. **Paraphrase on Crystal , an agile development method**

A resource - limited, cooperative game of invention and communication, with a primary goal of delivering useful, working software and a secondary goal of setting up for the next game.

To achieve maneuverability, a set of methodologies have been defined, each with core elements that are common to all, and roles, process patterns, work products, and practice that are unique to each.

The Crystal family is actually a set of example agile processes that have been proven effective for different types of projects.

The intent is to allow agile teams to select the member of the crystal family that is most appropriate for their project and environment.

15. **Explain the Feature Driven Development (FDD)**

FDD adopts a philosophy that

(1) emphasizes collaboration among people on an FDD team;

(2) manages problem and project complexity using feature-based decomposition followed by the integration of software increments, and

(3) communication of technical detail using verbal, graphical, and text-based means.

FDD emphasizes software quality assurance activities by encouraging an incremental development strategy, the use of design and code inspections, the application of software quality assurance audits the collection of metrics, and the use of patterns.

In the context of FDD, a feature "is a client-valued function that can be implemented in two weeks or less".

The emphasis on the definition of features provides the following benefits:

• Because features are small blocks of deliverable functionality, users can describe them more easily; understand how they relate to one another more readily; and better review them for ambiguity, error, or omissions.

• Features can be organized into a hierarchical business-related grouping. • Since a feature is the FDD deliverable software increment, the team develops operational features every two weeks.

• Because features are small, their design and code representations are easier to inspect effectively.

• Project planning, scheduling, and tracking are driven by the feature hierarchy, rather than an arbitrarily adopted software engineering task set.

Template for defining a feature:

<action> the **<result>** **<by | for | of | to>** a(n) **<object>**

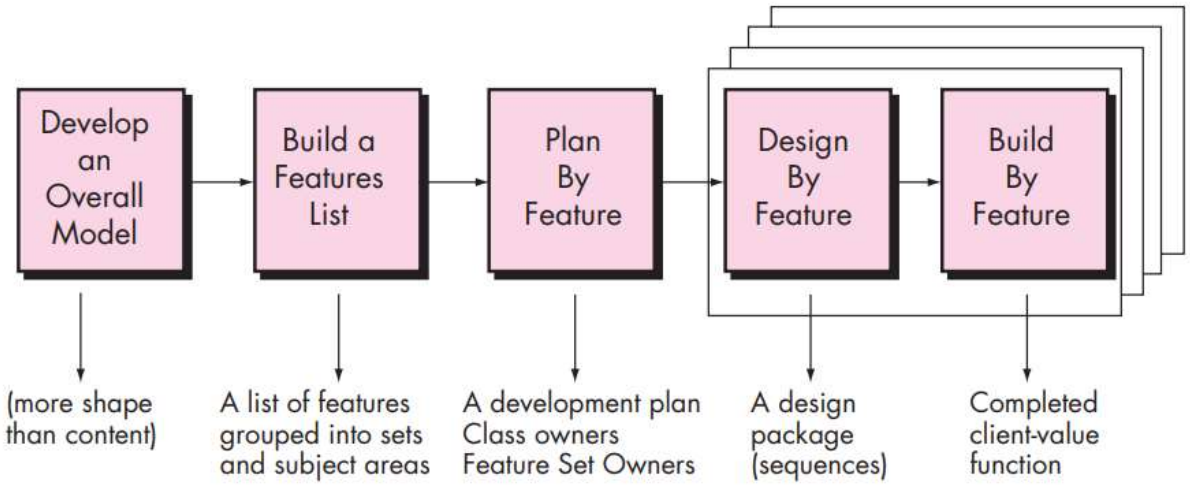
where an **<object>** is "a person, place, or thing (including roles, moments in time or intervals of time, or catalog-entry-like descriptions)."

Examples of features for an e-commerce application might be:

Add the product to shopping cart

Display the technical-specifications of the product

Store the shipping-information for the customer



A feature set groups related features into business-related categories and is defined as:

<action><-ing> a(n) **<object>**

For example: *Making a product sale* is a feature set that would encompass the features noted earlier and other

16. **Explain the Lean Software Development (LSD) method**

Lean Software Development (LSD) has adapted the principles of lean manufacturing to the world of software engineering.

The lean principles that inspire the LSD process can be summarized as eliminate waste, build quality in, create knowledge, defer commitment, deliver fast, respect people, and optimize the whole. Each of these principles can be adapted to the software process.

For example, eliminate waste within the context of an agile software project can be interpreted to mean:

(1) adding no extraneous features or functions,

(2) assessing the cost and schedule impact of any newly requested requirement,

(3) removing any superfluous process steps,

(4) establishing mechanisms to improve the way team members find information,

(5) ensuring the testing finds as many errors as possible

(6) reducing the time required to request and get a decision that affects the software or the process that is applied to create it, and

(7) streamlining the manner in which information is transmitted to all stakeholders involved in the process.

17. **Explain the concept of Agile Modeling (AM)**

Agile modeling adopts all of the values that are consistent with the agile manifesto.

The agile modeling philosophy recognizes that an agile team must have the courage to make decisions that may cause it to reject a design and refactor.

The team must also have the humility to recognize that technologists do not have all the answers and that business experts and other stakeholders should be respected and embraced.

Although AM suggests a wide array of "core" and "supplementary" modeling principles, those that make AM unique are:

Model with a purpose. A developer who uses AM should have a specific goal in mind before creating the model. Once the goal for the model is identified, the type of notation to be used and level of detail required will be more obvious

Use multiple models. There are many different models and notations that can be used to describe software. Only a small subset is essential for most projects. AM suggests that to

provide needed insight, each model should present a different aspect of the system and only those models that provide value to their intended audience should be used.

Travel light. As software engineering work proceeds, keep only those mod els that will provide long-term value and jettison the rest. Every work product that is kept must be maintained as changes occur.

Content is more important than representation. Modeling should impart information to its intended audience. A syntactically perfect model that imparts little useful content is not as valuable as a model with flawed notation that nevertheless provides valuable content for its audience.

Know the models and the tools you use to create them. Understand the strengths and weaknesses of each model and the tools that are used to create it.

Adapt locally. The modeling approach should be adapted to the needs of the agile team

18. **Explain the Agile Unified Process (AUP)**

The Agile Unified Process (AUP) adopts a "serial in the large" and "iterative in the small" philosophy for building computer-based systems.

By adopting the classic UP phased activities—inception, elaboration, construction, and transition—AUP provides a serial overlay that enables a team to visualize the overall process flow for a software project.

Each AUP iteration addresses the following activities:

• *Modeling.* UML representations of the business and problem domains are created. However, to stay agile, these models should be "just barely good enough" [Amb06] to allow the team to proceed.

• *Implementation*. Models are translated into source code.

• *Testing*. Like XP, the team designs and executes a series of tests to uncover errors and ensure that the source code meets its requirements.

• *Deployment*. Like the generic process activity, deployment in this context focuses on the delivery of a software increment and the acquisition of feedback from end users.

• *Configuration and project management*. In the context of AUP, configuration management addresses change management, risk management, and the control of any persistent work

products that are produced by the team. Project management tracks and controls the progress of the team and coordinates team activities.

• *Environment management*. Environment management coordinates a process infrastructure that includes standards, tools, and other support technology available to the team