SE Mid 1 QnA

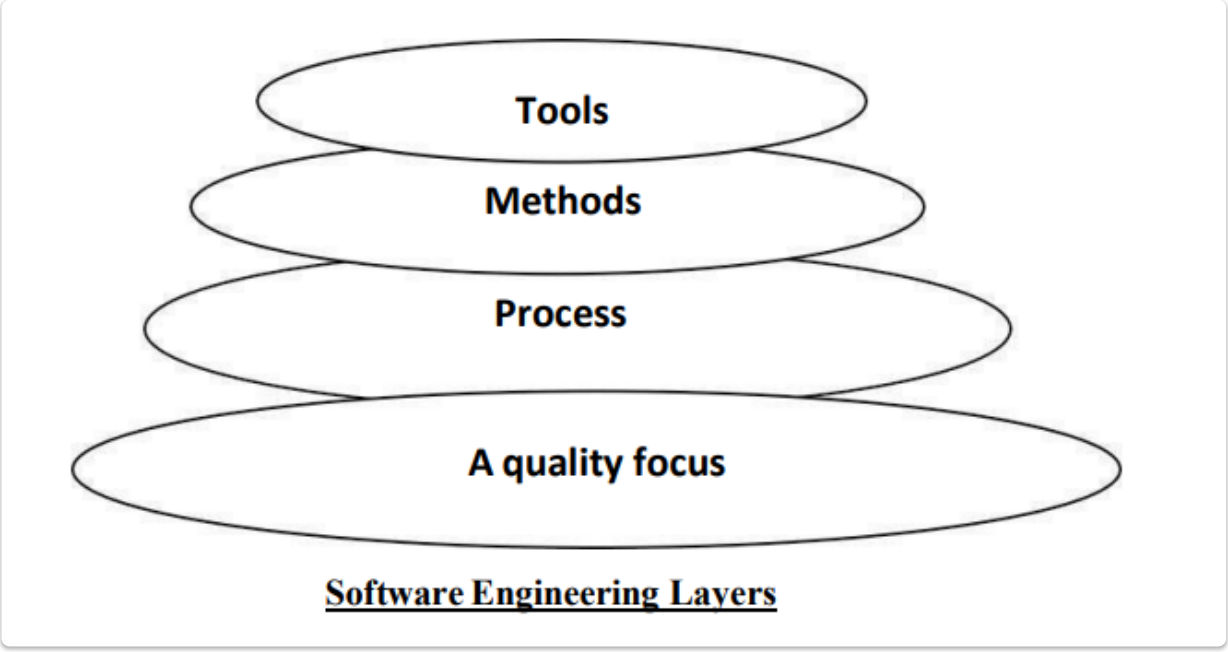
1. Layered Technology.

A GENERIC VIEW OF PROCESS

SOFTWARE ENGINEERING - A LAYERED TECHNOLOGY

Software Engineering Layers

- Tools
- Methods
- Process
- A quality focus



Software engineering is a layered technology. Any engineering approach must rest on an organizational commitment to quality. The bedrock that supports software engineering is a quality focus.

The foundation for software engineering is the process layer. Software engineering process is the glue that holds the technology layers. Process defines a framework that must be established for effective delivery of software engineering technology.

The software forms the basis for management control of software projects and establishes the context in which:

- Technical methods are applied,
- Work products are produced,
- Milestones are established,
- Quality is ensured,
- And change is properly managed.

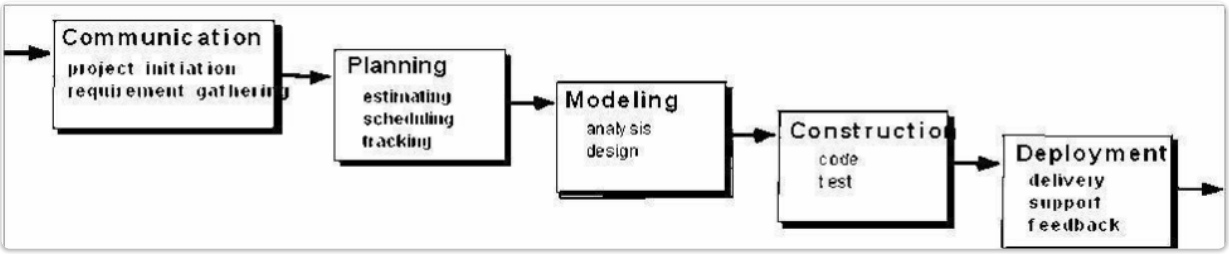
Software engineering methods rely on a set of basic principles that govern the area of the technology and include modeling activities. Methods encompass a broad array of tasks that include communication, requirements analysis, design modeling, program construction, testing, and support.

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.

2. Waterfall model.

THE WATERFALL MODEL

- The waterfall model, sometimes called the classic life cycle, suggests a systematic sequential approach to software development that begins with customer specification of requirements and progresses through planning, modeling, construction, and deployment.



Context:

- Used when requirements are reasonably well understood.

Advantage:

- It can serve as a useful process model in situations where requirements are fixed and work is to proceed to complete in a linear manner.

Problems Encountered:

- Real projects rarely follow the sequential flow that the model proposes. Although the linear model can accommodate iteration, it does so indirectly. As a result, changes can cause confusion as the project team proceeds.
- It is often difficult for the customer to state all requirements explicitly. The waterfall model requires this and has difficulty accommodating the natural uncertainty that exists at the beginning of many projects.
- The customer must have patience. A working version of the programs will not be available until late in the project time-span. If a major blunder is undetected, then it can be disastrous until the program is reviewed.

3. a) Evolving role of software.

EVOLVING ROLE OF SOFTWARE

Software takes on a dual role, functioning both as a product and as a vehicle for delivering a product.

As a Product:

- It delivers the computing potential embodied by computer hardware or by a network of computers.

As a Vehicle:

- It is an information transformer, producing, managing, acquiring, modifying, displaying, or transmitting information that can range from a single bit to as complex as a multimedia presentation.
- Software delivers the most important product of our time - information.
    - It transforms personal data.
    - It manages business information to enhance competitiveness.
    - It provides a gateway to worldwide information networks.
    - It provides the means for acquiring information.

The role of computer software has undergone significant changes over a span of little more than 50 years.

- Dramatic improvements in hardware performance.
- Vast increases in memory and storage capacity.
- A wide variety of exotic input and output options.

Evolution of Software: Key Decades

1970s and 1980s:

- Envisioned "new industrial revolution" and "third wave of change" with microelectronics.
- Predicted shift to an "information society" and recognized information as a power focal point.
- Emphasized the role of the "electronic community" in global knowledge interchange.

1990s:

- Computers brought a "power shift" and "democratization of knowledge."
- Concerns arose about U.S. companies' competitiveness in software.
- Information technologies played a pivotal role in the "reengineering of the corporation."

Mid-1990s:

- Proliferation of computers led to neo-Luddite literature.
- Reevaluated software professionals' prospects and predicted "rise and resurrection" of American programmers.
- Y2K "time bomb" impact felt at century's end.

2000s:

- Discussed emergence's power in interconnections and revisited 9/11's impact on the IT community.
- Presented a "new kind of science" with sophisticated software simulations.
- Explored the evolution of "the semantic web."

Today, a huge software industry has become a dominant factor in the economies of the industrialized world.

3.b) Changing nature of software.

THE CHANGING NATURE OF SOFTWARE

The realm of computer software encompasses seven broad categories, presenting ongoing challenges for software engineers:

1. System Software

System software serves other programs, characterized by:

- Heavy interaction with computer hardware

- Usage by multiple users
- Concurrent operation requiring scheduling, resource sharing, and sophisticated process management
- Complex data structures and multiple external interfaces
  Examples include compilers, editors, and file management utilities.

2. Application Software

Standalone programs addressing specific business needs:

- Facilitates business operations or decision-making
- Controls real-time business functions
  Examples include point-of-sale transaction processing and real-time manufacturing process control.

3. Engineering/Scientific Software

Applications span a wide range from astronomy to automated manufacturing:

- Includes computer-aided design, system simulation, and other interactive applications.

4. Embedded Software

Resides within products or systems, controlling features for end-users and the system itself:

- Performs limited or significant functions.
  Examples include digital functions in automobiles, dashboard displays, and braking systems.

5. Product-line Software

Designed to provide a specific capability for various customers:

- Focuses on a limited market or mass consumer markets Examples include word processing, spreadsheets, multimedia, and database management.

6. Web Applications

Evolving into sophisticated computing environments integrated with corporate databases and business applications.

7. Artificial Intelligence Software

Uses nonnumerical algorithms to solve complex problems:

- Includes robotics, expert systems, pattern recognition, artificial neural networks, theorem proving, and game playing.

New Challenges on the Horizon:

a. Ubiquitous Computing

Developing systems and application software for communication across vast networks among small devices, personal computers, and enterprise systems.

b. Netsourcing

Architecting applications that benefit targeted end-user markets worldwide.

c. Open Source

Building self-descriptive source code and developing techniques for customers and developers to track changes within the software.

d. The "New Economy"

Building applications that facilitate mass communication and mass product distribution.

4.a) Myths.

SOFTWARE MYTHS

Beliefs about software and the development process have persisted since the earliest days of computing, with several myths proving to be particularly challenging.

Management Myths:

Myth: We already have a book full of standards and procedures for building software. Won't that provide my people with everything they need to know?

Reality: The existence of a standards book does not guarantee its use or reflect modern software engineering practices.

Myth: If we get behind schedule, we can add more programmers and catch up.

Reality: Software development is not mechanistic like manufacturing. Adding more people can disrupt productivity without proper planning.

Myth: If I decide to outsource the software project to a third party, I can relax and let that firm build it.

Reality: Outsourcing success relies on a solid understanding of how to manage and control software projects internally.

Customer Myths:

Myth: A general statement of objectives is sufficient to begin writing programs; we can fill in the details later.

Reality: An ambiguous statement of objectives is a recipe for disaster; clear and comprehensive requirements are crucial.

Myth: Project requirements continually change, but change can be easily accommodated because software is flexible.

Reality: While software is adaptable, changes can have varying impacts and may require additional resources and major design modifications.

Practitioner's Myths:

Myth: Once we write the program and get it to work, our jobs are done.

Reality: A significant portion of effort is expended after the software is delivered, indicating the importance of ongoing support and maintenance.

Myth: The only deliverable work product for a successful project is the working program.

Reality: A working program is just one part of a software configuration. Documentation provides guidance for software support.

Myth: Software engineering creates unnecessary documentation and invariably slows down.

Reality: Software engineering focuses on quality, not excessive documentation. Improved quality reduces rework and speeds up delivery times.

4.b) User requirements

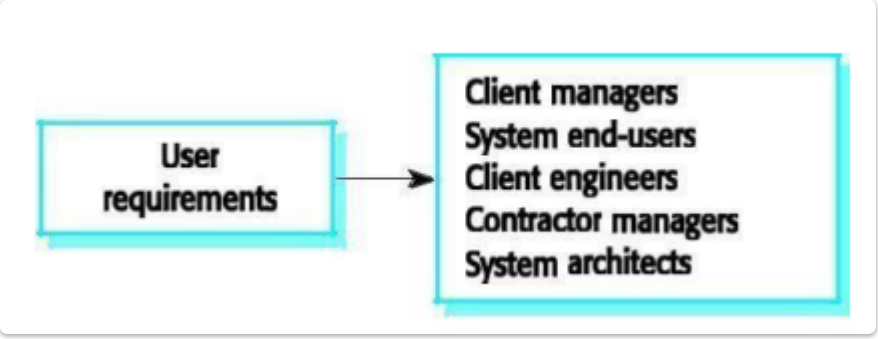
User Requirements

The user requirements outline the functionalities and operational constraints of the system in natural language. These are written for customers to provide a clear understanding of the services the system will provide. Below is an example:

Requirement:

- Functionality:
  - The software must offer a mechanism for representing and accessing external files.
  - It should support files created by other tools.
- Operational Constraints:

- The system should handle external files seamlessly.
- Compatibility with various file formats should be ensured for accessibility.



5 a) Significance of feasibility study

FEASIBILITY STUDIES

A feasibility study decides whether or not the proposed system is worthwhile. The input to the feasibility study is a set of preliminary business requirements, an outline description of the system, and how the system is intended to support business processes. The results of the feasibility study should be a report that recommends whether or not it is worth carrying on with the requirements engineering and system development process.

Key Considerations in Feasibility Study:

- **Alignment with Organizational Objectives:**
  - Assess whether the proposed system contributes to organizational objectives.
- **Technological Feasibility:**
  - Examine if the system can be engineered using current technology and within budget constraints.
- **Integration Feasibility:**
  - Evaluate if the system can be seamlessly integrated with other existing systems.

Feasibility Study Implementation:

A feasibility study involves information assessment, information collection, and report writing.

Questions for People in the Organization:

- What if the system wasn't implemented?
- What are the current process problems?
- How will the proposed system help?
- What will be the integration problems?
- Is new technology needed? What skills?
- What facilities must be supported by the proposed system?

In a feasibility study, you may consult information sources such as the managers of the departments where the system will be used, software engineers who are familiar with the type of system that is proposed, technology experts, and end-users of the system. They should try to complete a feasibility study in two or three weeks.

Once you have the information, you write the feasibility study report. You should make a recommendation about whether or not the system development should continue. In the report, you may propose changes to the scope, budget, and schedule of the system and suggest further high-level requirements for the system.

5.b) Explain Quality Attributes.

Quality Attributes

The FURPS quality attributes serve as a target for all software design, encompassing various aspects:

1. Functionality

- Assessed by evaluating the feature set and capabilities of the program.
- Considers the generality of delivered functions and the security of the overall system.

2. Usability

- Assessed by considering human factors, overall aesthetics, consistency, and documentation.

3. Reliability

- Evaluated by measuring the frequency and severity of failure.
- Considers the accuracy of output results and mean-time-to-failure (MTTF).
- Assess the ability to recover from failure and the predictability of the program.

4. Performance

- Measured by processing speed, response time, resource consumption, throughput, and efficiency.

5. Supportability

- Combines the ability to extend the program (extensibility), adaptability, and serviceability.
- These attributes collectively represent a more common term, maintainability.

Not every software quality attribute is weighted equally as the software design is developed. Each application may stress different aspects based on its unique requirements. For instance:

- One application may emphasize functionality with a special focus on security.

- Another may prioritize performance, placing particular emphasis on processing speed.
- A third might focus primarily on reliability.

6. what are the activities of requirements & analysis ? Explain

1. **Requirements Discovery:**
    - *Definition:* This is the initial stage where technical staff interacts with stakeholders to gather information about the application domain and the services the system should provide.
    - *Process:* Involves interacting with stakeholders through interviews, observations, and possibly using scenarios and prototypes to extract user and system requirements.
    - *Outcome:* Domain requirements are discovered at this stage.
2. **Requirements Classification and Organisation:**
    - *Definition:* This activity involves grouping related requirements into coherent clusters and organizing them.
    - *Process:* Identifying overlapping requirements from different stakeholders and using a model of the system architecture to associate requirements with subsystems.
    - *Outcome:* Coherent clusters of requirements, associated with subsystems, are formed.
3. **Prioritisation and Negotiation:**
    - *Definition:* In this activity, priorities are assigned to requirements, and conflicts between different stakeholders' requirements are resolved.
    - *Process:* Regular stakeholder negotiations are organized to reach compromises when conflicting views arise.

- *Outcome:* A prioritized set of requirements that addresses conflicts among stakeholders.

4. **Requirements Documentation:**
   - *Definition:* The finalized requirements are documented in a way that facilitates further requirements discovery.
   - *Process:* Documenting the requirements in a structured manner to aid in the development of the system.
   - *Outcome:* A documented set of requirements that serves as input for the next round of the requirements spiral.
5. **Viewpoint Identification:**
   - *Definition:* Viewpoints are identified to represent the perspectives of different stakeholders in structuring the requirements.
   - *Process:* Identification based on providers and receivers of system services, systems interacting with the specified system, regulations, standards, sources of business and non-functional requirements, and individuals involved in system development and maintenance.
   - *Outcome:* Different viewpoints representing the varied perspectives of stakeholders.
6. **Types of Viewpoints:**
   - *Interactor Viewpoints:* Detailed system requirements from people or systems directly interacting with the system.
   - *Indirect Viewpoints:* Higher-level organizational requirements and constraints from stakeholders who do not use the system themselves.
   - *Domain Viewpoints:* Characteristics and constraints from the application domain that influence requirements.

7. functional and non- functional requirements.

Functional Requirements:

Functional requirements describe the services a system should provide, how it reacts to inputs, and its behavior in specific situations. They are often dependent on the type of software, expected users, and the system's context. Examples of functional requirements for a library system (LIBSYS) include:

1. Users can search for articles in different libraries.
2. Users can download and print articles for personal study.
3. The system provides a single interface to multiple databases.
4. The user can search all databases or select a subset.
5. The system offers appropriate viewers for reading documents.
6. Each order is assigned a unique identifier (ORDER_ID).

Requirements Imprecision:

- Ambiguous requirements may lead to different interpretations.
- Example: The term "appropriate viewers" can be interpreted differently by users and developers.

Requirements Completeness and Consistency:

- Ideally, requirements should be both complete and consistent.
- Complete: All required facilities are described.
- Consistent: No conflicts or contradictions in the descriptions.
- Practical challenges in achieving complete and consistent requirements.

Non-Functional Requirements:

Non-functional requirements define system properties and constraints, such as reliability, response time, and storage requirements. They may also include process requirements,

specifying tools, languages, or methods. Non-functional requirements can be categorized into types:

1. **Product Requirements:**
    - Specify how the delivered product must behave.
    - Example: The LIBSYS user interface must be implemented as simple HTML without frames or Java applets.
2. **Organisational Requirements:**
    - Arise from organizational policies and procedures.
    - Example: System development processes must conform to the standards defined in XYZCo-SP-STAN-95.
3. **External Requirements:**
    - Arise from factors external to the system and its development process.
    - Example: The system must not disclose personal information about customers, except for their name and reference number, to system operators.

Note:

- Non-functional requirements can be critical, as failure to meet them may render the system useless.
- Examples of non-functional requirements include reliability, response time, and adherence to organizational standards.

8. Spiral Model (Advantages).

The Spiral Model, proposed by Boehm, is an evolutionary software development process that blends the iterative nature of prototyping with the structured aspects of the waterfall model. It is designed to be adaptable throughout the entire life cycle of an application, from conceptualization to maintenance.

Advantages of the Spiral Model:

1. **Rapid Development:** The model allows for the rapid development of increasingly complete versions of the software.
2. **Realism for Large-scale Systems:** Particularly suitable for the development of large-scale systems.
3. **Risk Reduction Mechanism:** Utilizes prototyping as a risk reduction mechanism and permits its application at any stage in product evolution.

Drawbacks of the Spiral Model:

1. **Customer Convincing:** It may be challenging to convince customers that the evolutionary approach is controllable.
2. **Expertise Requirement:** Success relies on considerable risk assessment expertise.
3. **Risk Management:** If a major risk goes undiscovered and unmanaged, problems may occur.

9. Important of s/w engineering.

Importance of Software Engineering:

1. **Structured Approach:**
   - Software engineering provides a structured and organized method for developing, operating, and maintaining software, ensuring a clear path from start to finish.
2. **Discipline and Consistency:**
   - It introduces discipline, adhering to well-defined principles and practices, ensuring consistency and reliability in software development.
3. **Measurable Progress:**

- By incorporating quantifiable measures, it enables the assessment of progress, quality, and success throughout the software development process.

4. **Efficiency and Cost-Effectiveness:**
   - Aims for efficient and cost-effective software development by following engineering principles, including proper planning, design, coding, testing, and maintenance.
5. **Risk Management:**
   - Identifies and addresses potential issues early, minimizing risks and ensuring a more predictable and successful development process.
6. **Quality Assurance:**
   - Emphasizes quality assurance at every stage, ensuring that the final software product meets high standards and performs as intended.
7. **Adaptability to Change:**
   - Provides methodologies that enable software systems to adapt to changing requirements and technologies, ensuring long-term viability.
8. **Documentation and Communication:**
   - Promotes clear documentation, aiding communication among team members and supporting future maintenance efforts.
9. **Effective Project Management:**
   - Incorporates project management principles to ensure effective planning, resource allocation, and on-time, on-budget project delivery.
10. **Customer Satisfaction:**
    - Enhances customer satisfaction by delivering software that meets or exceeds expectations through systematic

development and user requirement adherence.

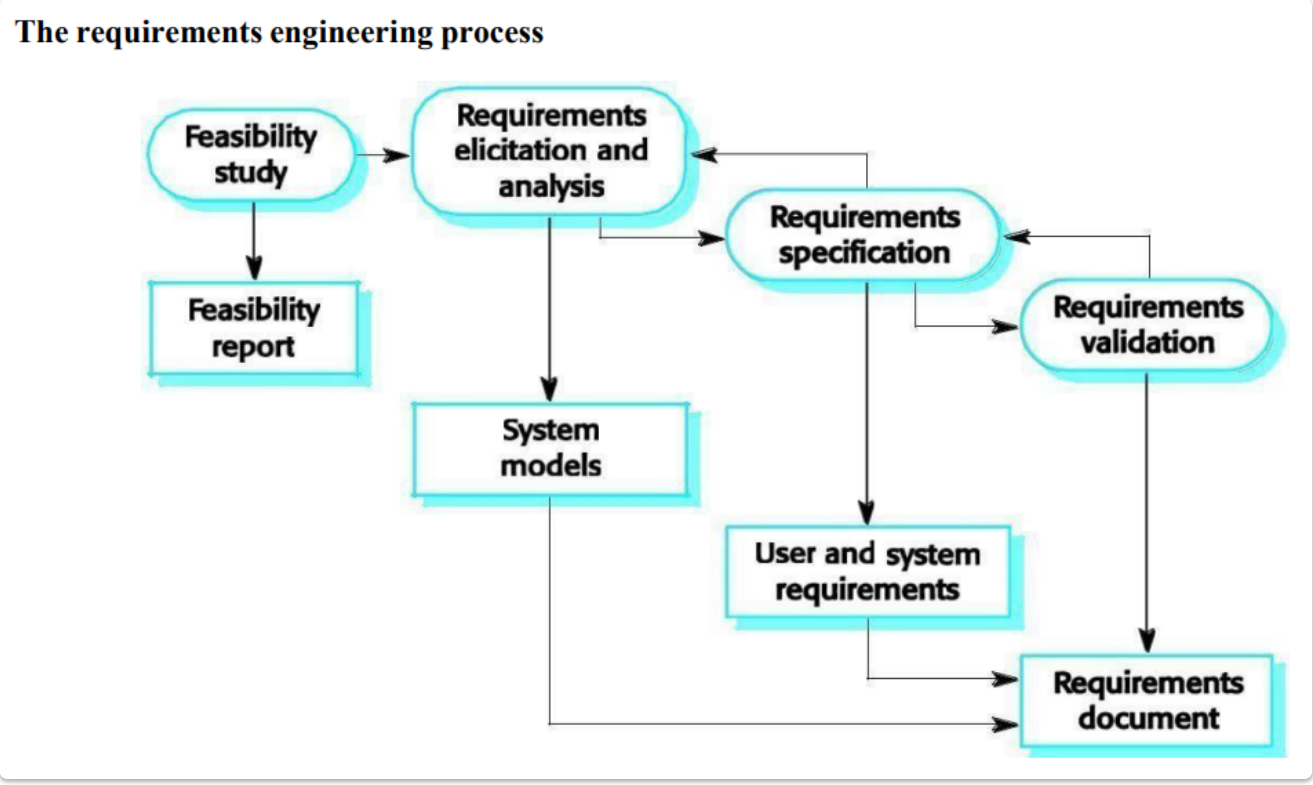
10. Requirement engineering process.

REQUIREMENTS ENGINEERING PROCESSES

The goal of the requirements engineering process is to create and maintain a system requirements document. This process encompasses four high-level requirement engineering sub-processes:

1. **Feasibility Study:**
   - Assessing whether the system is useful to the business.
2. **Elicitation and Analysis:**
   - Discovering requirements through a detailed exploration of user needs and system functionalities.
3. **Specification:**
   - Converting elicited requirements into a standardized form.
4. **Validation:**
   - Checking that the requirements accurately define the system that the customer wants.

Additionally, the process of managing changes in the requirements is termed **Requirement Management.**



Alternative Perspective - Three-Stage Activity

The alternative perspective presents the requirements engineering process as a three-stage activity organized in an iterative process around a spiral. The effort devoted to each activity in each iteration depends on the stage of the overall process and the type of system being developed. Early stages focus on understanding high-level business, non-functional requirements, and user requirements. In the outer rings of the spiral, more effort is dedicated to system requirements engineering and system modeling.

This spiral model accommodates approaches to development in which the requirements are developed to different levels of detail. The number of iterations around the spiral can vary, allowing for exit after eliciting some or all of the user requirements.

Structured Analysis Method

Some people consider requirements engineering as the process of applying a structured analysis method, such as object-oriented analysis. This involves analyzing the system and developing a set of

graphical system models (e.g., use-case models), which then serve as a system specification. The set of models describes the behavior of the system and is annotated with additional information describing factors like required performance or reliability.

11. Agile Methodology.

Agile Process Model

- The meaning of Agile is swift or versatile. "Agile process model" refers to a software development approach based on iterative development.
- Agile methods break tasks into smaller iterations, or parts do not directly involve long-term planning.
- The project scope and requirements are laid down at the beginning of the development process.
- Plans regarding the number of iterations, the duration, and the scope of each iteration are clearly defined in advance.
- Each iteration is considered as a short time "frame" in the Agile process model, which typically lasts from one to four weeks.
- The division of the entire project into smaller parts helps to minimize the project risk and to reduce the overall project delivery time requirements.
- Each iteration involves a team working through a full software development life cycle including planning, requirements analysis, design, coding, and testing before a working product

is demonstrated to the client.

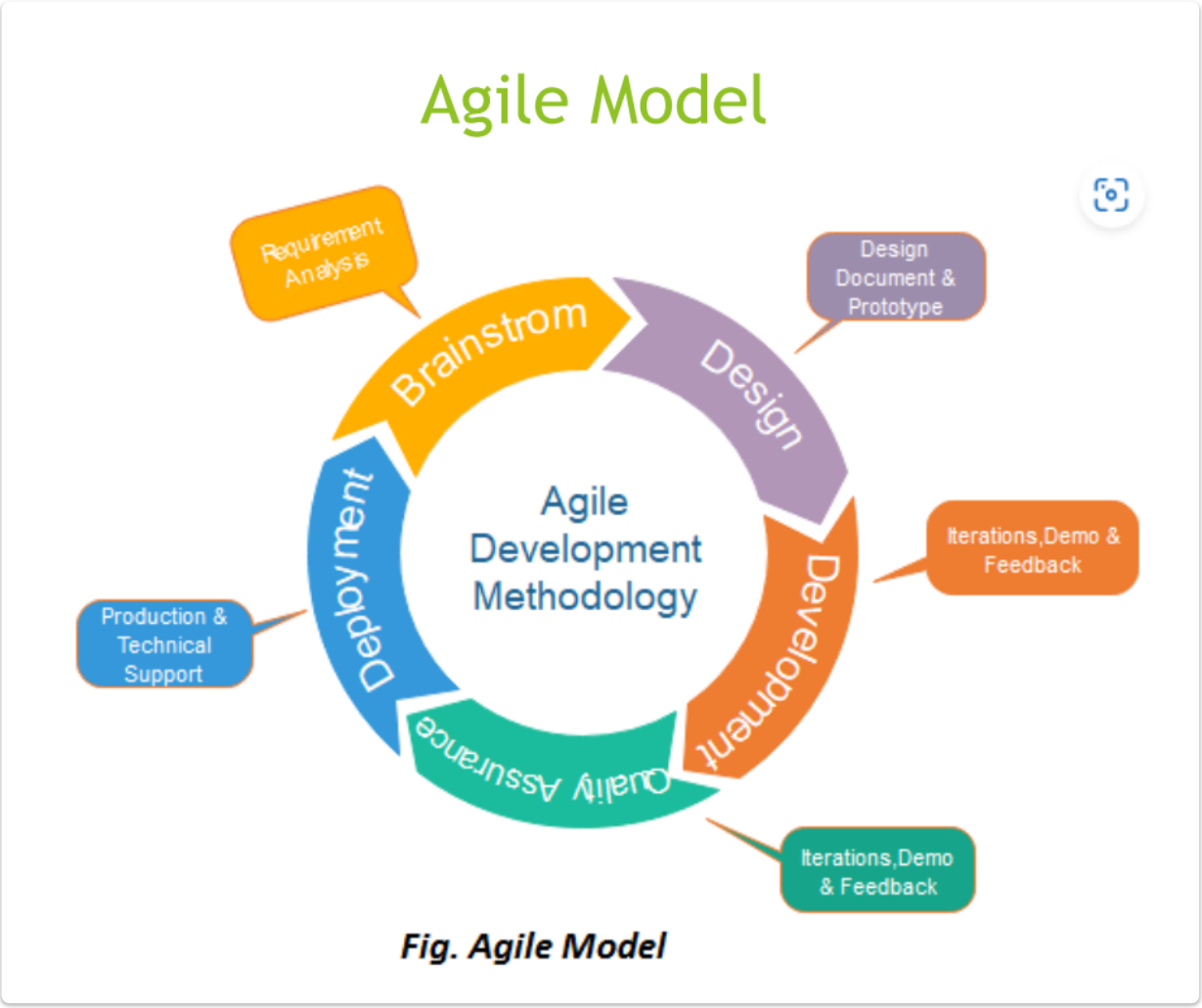


Fig. Agile Model

Phases of Agile Model:

1. Requirements Gathering
2. Design the Requirements
3. Construction/Iteration
4. Testing/Quality Assurance
5. Deployment
6. Feedback

Advantages (Pros) of Agile Method:

- Frequent Delivery
- Face-to-Face Communication with clients.
- Efficient design and fulfills the business requirement.
- Anytime changes are acceptable.

- It reduces total development time.

Disadvantages (Cons) of Agile Model:

- Due to the shortage of formal documents, it creates confusion and crucial decisions taken throughout various phases can be misinterpreted at any time by different team members.
- Due to the lack of proper documentation, once the project completes and the developers allotted to another project, maintenance of the finished project can become a difficulty.

12. Design Concepts

Fundamental software design concepts provide the necessary framework for getting it right.

1. **Abstraction:**
    - Levels: Highest level for broad solutions, lower levels for detailed descriptions.
    - Types: Procedural (limited function sequences), Data (named collection describing data objects).
2. **Architecture:**
    - Definition: Overall structure providing conceptual integrity for a system.
    - Representation: Structured models, framework models, dynamic models, process models.
3. **Patterns:**
    - Definition: Named solutions to recurring problems in a specific context.
    - Intent: Guide for current work, reusability, and development of similar but different patterns.
4. **Modularity:**

- Definition: Software division into separately named and addressable components (modules).
- Importance: Easier development, compartmentalization, simplified interfaces, and easier maintenance.

5. **Information Hiding:**
   - Principle: Modules characterized by design decisions hiding information from others.
   - Benefit: Reduces inadvertent error propagation during modifications.
6. **Functional Independence:**
   - Concept: Modules with single-minded functions, aversion to excessive interaction.
   - Benefit: Easier development, limited code modifications, reduced error propagation.
7. **Refinement:**
   - Strategy: Top-down design, successively refining procedural details.
   - Process: Elaboration, revealing low-level details as design progresses.
8. **Refactoring:**
   - Definition: Reorganizing to simplify design without changing function/behavior.
   - Purpose: Improve internal structure for easier integration, testing, and maintenance.
9. **Design Classes:**
   - Types: User interface, business domain, process, persistent, system classes.
   - Characteristics: Complete and sufficient, primitiveness, high cohesion, low coupling.

- Review: Ensure well-formed attributes and operations, following the law of Demeter.