Syllabus

1. Introduction
2. Software development lifecycle
3. Requirement Analysis (software requirement system document)
4. Software Project management
5. Design
6. Software testing
7. Maintenance
8. Quality, management, reuse

Introduction

Software systems are abstract and intangible. Software engineering is essential for the functioning of government, society, and national and international businesses and institutions.

1) **What is software?** Computer programs and associated documentation. Software products may be developed for a particular customer or may be developed for a general market.
2) **What is software engineering?** Software engineering is an engineering discipline that is concerned with all aspects of software production from initial conception to operation and maintenance.
3) **What are the fundamental software engineering activities?** Software specification, software development, software validation and software evolution.
4) **What are software engineering activities?** Software specification, software development, software validation and software evolution.
5) **What is the difference between software engineering and computer science?** Computer science focuses on theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software.
6) **What is the difference between software engineering and system engineering?** System engineering is concerned with all aspects of computer-based systems development including hardware, software and process engineering. Software engineering is part of this more general process.
7) **What are the key challenges facing software engineering?** Coping with increasing diversity, demands for reduced delivery times and developing trustworthy software.
8) **What are the costs of software engineering?** Roughly 60% of software costs are development costs, 40% are testing costs. For custom software, evolution costs often exceed development costs.
9) **What are the best software engineering techniques and methods?** While all software projects must be professionally managed and

developed, different techniques are appropriate for different types of system. For example, games should always be developed using a series of prototypes whereas safety critical control systems require a complete and analyzable specification to be developed. There are no methods and techniques that are good for everything.

10) **What differences has the Internet made to software engineering?** Not only has the Internet led to the development of massive, highly distributed, service-based systems, it has also supported the creation of an “app” industry for mobile devices which has changed the economics of software.

Software engineers are concerned with developing software products, that is, software that can be sold to a customer. There are two kinds of software product:

1. Generic products : These are stand-alone systems that are produced by a development organization and sold on the open market to any customer who can buy them. Examples of this type of product include apps for mobile devices, software for PCs such as databases, word processors, drawing packages, and project management tools.

2. Customized (or bespoke) software: These are systems that are commissioned by and developed for a particular customer. A software contractor designs and implements the software especially for that customer. Examples of this type of software include control systems for electronic devices, systems written to support a particular business process, and air traffic control systems.

Software Ethics

1. Confidentiality You should normally respect the confidentiality of your employers or clients regardless of whether a formal confidentiality agreement has been signed.

2. Competence You should not misrepresent your level of competence. You should not knowingly accept work that is outside your competence.

3. Intellectual property rights You should be aware of local laws governing the use of intellectual property such as patents and copyright. You should be

careful to ensure that the intellectual property of employers and clients is protected.
4. Computer misuse You should not use your technical skills to misuse other people's computers. Computer misuse ranges from relatively trivial (game playing on an employer's machine) to extremely serious (dissemination of viruses or other malware).

Software Engineering Code of Ethics and Professional Practice

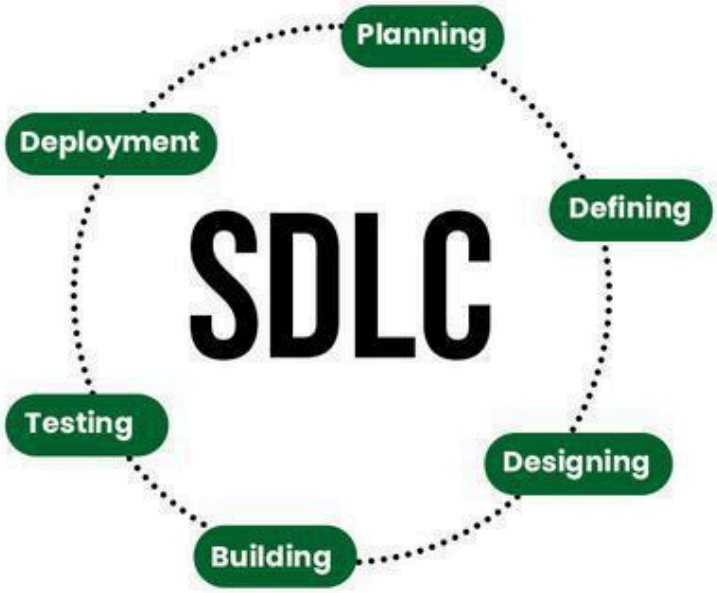
ACM/IEEE-CS Joint Task Force on Software Engineering Ethics and Professional Practices

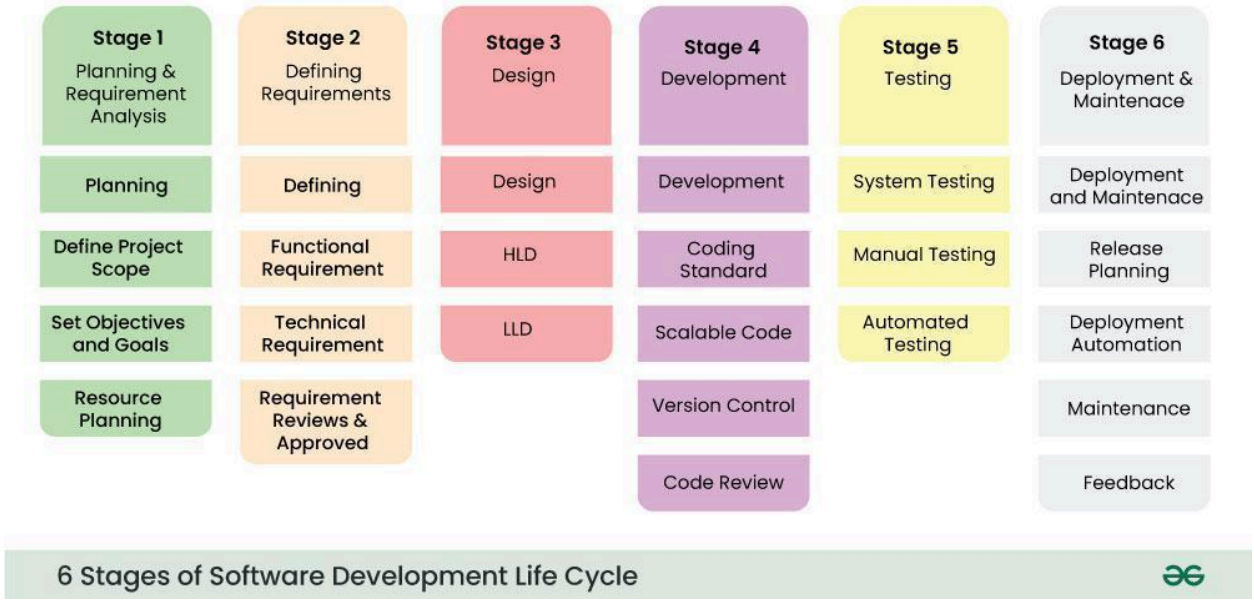
PREAMBLE

Software engineers shall commit themselves to making the analysis, specification, design, development, testing, and maintenance of software a beneficial and respected profession. In accordance with their commitment to the health, safety, and welfare of the public, software engineers shall adhere to the following Eight Principles:

1. PUBLIC — Software engineers shall act consistently with the public interest.
2. CLIENT AND EMPLOYER — Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.
3. PRODUCT — Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.
4. JUDGMENT — Software engineers shall maintain integrity and independence in them professional judgment.
5. MANAGEMENT — Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
6. PROFESSION — Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.
7. COLLEAGUES — Software engineers shall be fair to and supportive of them colleagues.
8. SELF — Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

Software Development Life Cycle: S**oftware development life cycle (SDLC) is a structured process that is used to design, develop, and test good-quality software.** SDLC, or software development life cycle, is a methodology that defines the entire procedure of software development step-by-step. The **goal of the SDLC life cycle model** is to deliver high-quality, maintainable software that meets the user's requirements. SDLC in software engineering models outlines the plan for each stage so that each stage of the software development model can perform its task efficiently to deliver the software at a low cost within a given time frame that meets users requirements.





6 Stages of Software Development Life Cycle

What is the need for SDLC?

SDLC is a method, approach, or process that is followed by a software development organization while developing any software. SDLC models were introduced to follow a disciplined and systematic method while designing software. With the software development life cycle, the process of software design is divided into small parts, which makes the problem more understandable and easier to solve. SDLC comprises a detailed description or step-by-step plan for designing, developing, testing, and maintaining the software.

How to Choose an SDLC Model?

Choosing the right SDLC (Software Development Life Cycle) model is essential for project success. Here are the key factors to consider:

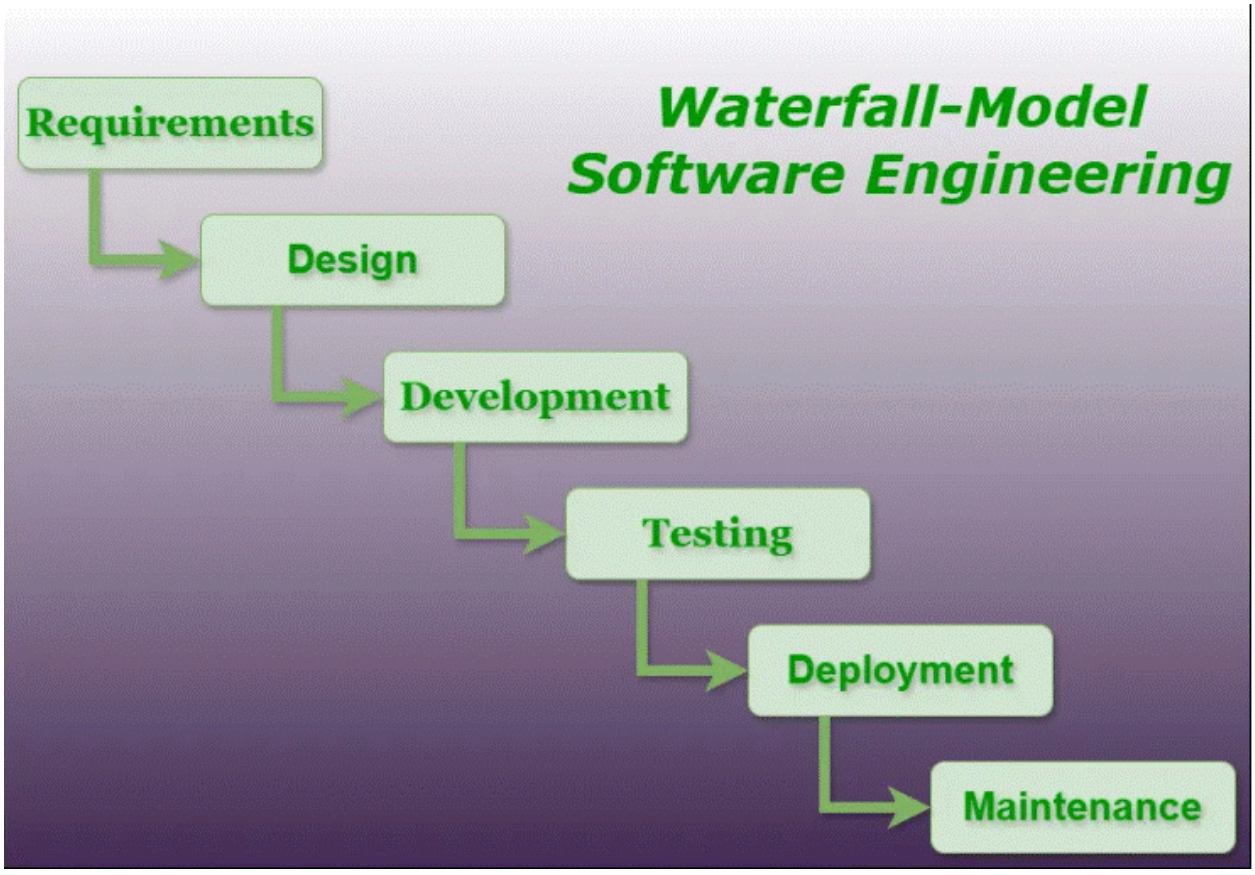
1. **Project Requirements:**
    - **Clear Requirements:** Use **Waterfall** or **V-Model** if requirements are well-defined and unlikely to change.
    - **Changing Requirements:** Use **Agile** or **Iterative** models if requirements are unclear or likely to evolve.

2. **Project Size and Complexity:**
    - **Small Projects:** Use **Waterfall** or **RAD** for small, simple projects.
    - **Large Projects:** Use **Agile**, **Spiral**, or **DevOps** for large, complex projects that need flexibility.
3. **Team Expertise:**
    - **Experienced Teams:** Use **Agile** or **Scrum** if the team is familiar with iterative development.
    - **Less Experienced Teams:** Use **Waterfall** or **V-Model** for teams needing structured guidance.
4. **Client Involvement:**
    - **Frequent Client Feedback:** Use **Agile**, **Scrum**, or **RAD** if regular client interaction is needed.
    - **Minimal Client Involvement:** Use **Waterfall** or **V-Model** if client involvement is low after initial planning.
5. **Time and Budget Constraints:**
    - **Fixed Time and Budget:** Use **Waterfall** or **V-Model** if you have strict time and budget limits.
    - **Flexible Time and Budget:** Use **Agile** or **Spiral** if you can adjust time and budget as needed.
6. **Risk Management:**
    - **High-Risk Projects:** Use **Spiral** for projects with significant risks and uncertainties.
    - **Low-Risk Projects:** Use **Waterfall** for projects with minimal risks.
7. **Product Release Timeline:**
    - **Quick Release Needed:** Use **Agile** or **RAD** to deliver products quickly.
    - **Longer Development Time:** Use **Waterfall** or **V-Model** for projects with no urgent deadlines.
8. **Maintenance and Support:**
    - **Long-Term Maintenance:** Use **Agile** or **DevOps** for projects needing continuous updates and support.
    - **Minimal Maintenance:** Use **Waterfall** or **V-Model** if little future maintenance is expected.

9. **Stakeholder Expectations:**
    - **High Stakeholder Engagement:** Use **Agile** or **Scrum** if stakeholders want ongoing involvement.
    - **Low Stakeholder Engagement:** Use **Waterfall** or **V-Model** if stakeholders prefer involvement only at major milestones.

Models

1) **Waterfall Model:** The **Waterfall Model** is a **Traditional Software Development Methodology**. It was first introduced by **Winston W. Royce** in 1970. It is a linear and sequential approach to software development that consists of several phases. This classical waterfall model is simple and idealistic. It is important because most other **Types of Software Development Life Cycle Models** are a derivative from this. this



Advantages of Waterfall Model

The classical waterfall model is an idealistic model for software development. It is very simple, so it can be considered the basis for other software development life cycle models. Below are some of the major advantages of this SDLC model.

- **Easy to Understand:** The Classical Waterfall Model is very simple and easy to understand.
- **Individual Processing:** Phases in the Classical Waterfall model are processed one at a time.
- **Properly Defined:** In the classical waterfall model, each stage in the model is clearly defined.
- **Clear Milestones:** The classical Waterfall model has very clear and well-understood milestones.
- **Properly Documented:** Processes, actions, and results are very well documented.
- **Reinforces Good Habits:** The Classical Waterfall Model reinforces good habits like define-before-design and design-before-code.
- **Working:** Classical Waterfall Model works well for smaller projects and projects where requirements are well understood.

Disadvantages of Waterfall Model

The Classical Waterfall Model suffers from various shortcomings we can't use it in real projects, but we use other software development lifecycle models which are based on the classical waterfall model. Below are some major drawbacks of this model.

- **No Feedback Path:** In the classical waterfall model evolution of software from one phase to another phase is like a waterfall. It assumes that no error is ever committed by developers during any phase. Therefore, it does not incorporate any mechanism for error correction.
- **Difficult to accommodate Change Requests:** This model assumes that all the customer requirements can be completely and correctly defined at the beginning of the project, but the customer's requirements keep on changing with time. It is difficult to

accommodate any change requests after the requirements specification phase is complete.

- **No Overlapping of Phases:** This model recommends that a new phase can start only after the completion of the previous phase. But in real projects, this can't be maintained. To increase efficiency and reduce cost, phases may overlap.
- **Limited Flexibility:** The Waterfall Model is a rigid and linear approach to software development, which means that it is not well-suited for projects with changing or uncertain requirements. Once a phase has been completed, it is difficult to make changes or go back to a previous phase.
- **Limited Stakeholder Involvement:** The Waterfall Model is a structured and sequential approach, which means that stakeholders are typically involved in the early phases of the project (requirements gathering and analysis) but may not be involved in the later phases (implementation, testing, and deployment).
- **Late Defect Detection:** In the Waterfall Model, testing is typically done toward the end of the development process. This means that defects may not be discovered until late in the development process, which can be expensive and time-consuming to fix.
- **Lengthy Development Cycle:** The Waterfall Model can result in a lengthy development cycle, as each phase must be completed before moving on to the next. This can result in delays and increased costs if requirements change or new issues arise.

When to Use Waterfall Model?

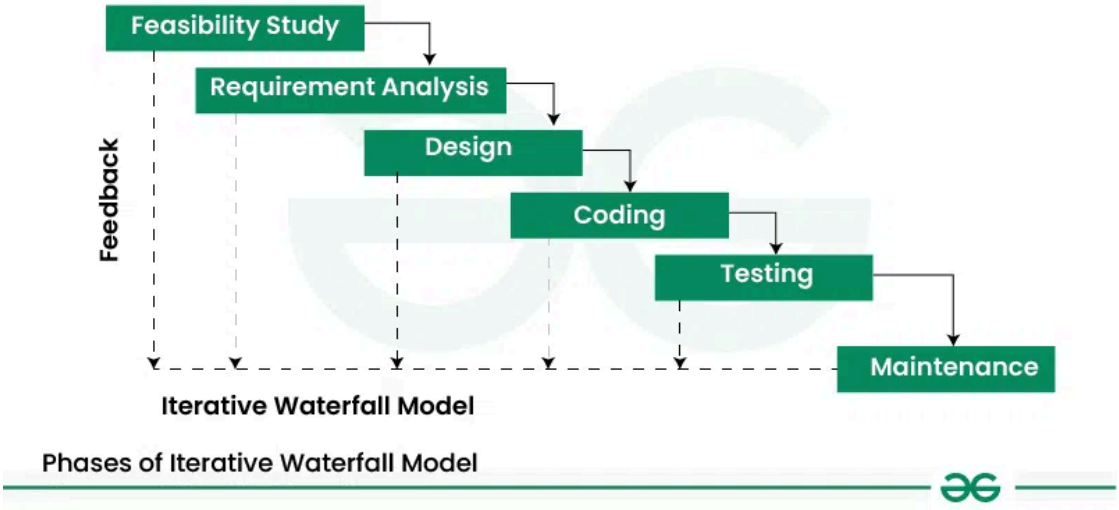
Here are some cases where the use of the Waterfall Model is best suited:

- **Well-understood Requirements:** Before beginning development, there are precise, reliable, and thoroughly documented requirements available.
- **Very Little Changes Expected:** During development, very little adjustments or expansions to the project's scope are anticipated.
- **Small to Medium-Sized Projects**: Ideal for more manageable projects with a clear development path and little complexity.

- **Predictable:** Projects that are predictable, low-risk, and able to be addressed early in the development life cycle are those that have known, controllable risks.
- **Regulatory Compliance is Critical:** Circumstances in which paperwork is of utmost importance and stringent regulatory compliance is required.
- **Client Prefers a Linear and Sequential Approach**: This situation describes the client's preference for a linear and sequential approach to project development.
- **Limited Resources**: Projects with limited resources can benefit from a set-up strategy, which enables targeted resource allocation.

2) **Iterative Waterfall Model:** The **Iterative Waterfall Model** is a **software development** approach that combines the **sequential steps** of the traditional **Waterfall Model** with the flexibility of **iterative design**. It allows for **improvements** and **changes** to be made at each stage of the **development process**, instead of waiting until the end of the project. The **Iterative Waterfall Model** provides **feedback paths** from every phase to its preceding phases, which is the main difference from the classical **Waterfall Model**.
   1. When errors are detected at some later phase, these feedback paths allow for correcting errors committed by programmers during some phase.
   2. The feedback paths allow the phase to be reworked in which errors are committed, and these changes are reflected in the later phases.
   3. Although the feasibility study phase doesn't usually get frequent feedback, the iterative nature of this model allows for revisiting and updating it if new requirements or changes in technology come up later. This helps keep the project aligned with its goals throughout development.
   4. It is good to detect errors in the same phase in which they are committed.
   5. It reduces the effort and time required to correct the errors.

6. A real-life example could be building a new website for a small business.



Phases of Iterative Waterfall Model

Advantages of Iterative Waterfall Model

Following are the advantage of Iterative Waterfall Model:

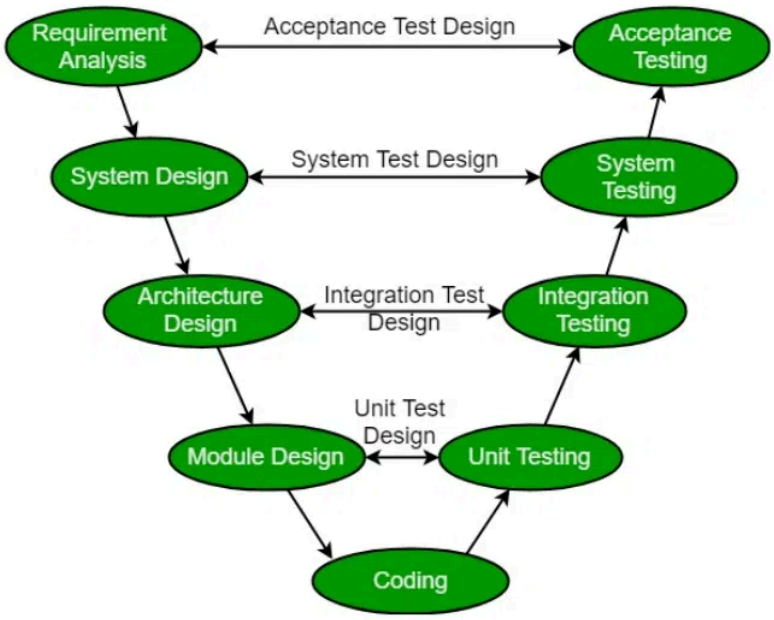
1. **Phase Containment of Errors**: Errors are detected and fixed as close to their source as possible, reducing costly rework and delays.
2. **Collaboration**: Continuous collaboration between business owners and developers ensures the product meets business needs and improves with feedback at each iteration.
3. **Flexibility**: The model allows for easy incorporation of new requirements or features in subsequent iterations, ensuring the product evolves with the business.
4. **Testing and Feedback**: Regular testing and feedback cycles help identify and fix issues early, improving the product's quality and relevance.
5. **Faster Time to Market**: Incremental development allows parts of the product to be delivered sooner, enabling user feedback while further improvements are made.
6. **Risk Reduction**: Continuous feedback and testing help identify risks early, reducing the likelihood of costly errors and delays.

Drawbacks of Iterative Waterfall Model

Following are the disadvantage of Iterative Waterfall Model:

1. **Difficult to incorporate change requests:** The major drawback of the iterative waterfall model is that all the requirements must be clearly stated before starting the development phase. Customers may change requirements after some time but the iterative waterfall model does not leave any scope to incorporate change requests that are made after the development phase starts.
2. **Incremental delivery not supported:** In the iterative waterfall model, the full software is completely developed and tested before delivery to the customer. There is no scope for any intermediate delivery. So, customers have to wait a long for getting the software.
3. **Overlapping of phases not supported:** Iterative waterfall model assumes that one phase can start after completion of the previous phase, But in real projects, phases may overlap to reduce the effort and time needed to complete the project.
4. **Risk handling not supported:** Projects may suffer from various types of risks. But, the Iterative waterfall model has no mechanism for risk handling.
5. **Limited customer interactions:** Customer interaction occurs at the start of the project at the time of requirement gathering and at project completion at the time of software delivery. These fewer interactions with the customers may lead to many problems as the finally developed software may differ from the customers' actual requirements.

3) **V Shaped Model: V-model** in **Software Development Life Cycle (SDLC)** is a method in that include the testing and validation alongside each development phase. It is based on the idea of a "V" shape, with the two legs of the "V" representing the progression of the **Software Development Process** from **Requirements Gathering** and analysis to design, implementation, testing, and maintenance. The V-Model, which includes the **Verification and Validation** it is a structural approach to the software development.



V-Model Verification Phases

This is where the process begins. The first step is to gather and understand the customer's needs for the software. The goal is to define the scope of the project clearly to make sure everyone is on the same page.

It involves a static analysis technique (review) done without executing code. It is the process of evaluation of the product development phase to find whether specified requirements are met.

There are several Verification phases in the V-Model:

1. Business Requirement Analysis

This is the first step of the designation of the development cycle where product requirement needs to be cured from the customer's perspective. in these phases include proper communication with the customer to understand the requirements of the customers. These are the very important activities that need to be handled properly, as most of the time customers do not know exactly what they want, and they are not sure about it at that time then we use an **acceptance test design** planning which is done at the time of business requirement it will be used as an **input** for acceptance testing.

2. System Design

In this phase, the overall structure of the software is planned out. The team develops both the high-level design (how the system will be structured) and detailed design (how the individual components will work). Design of the system will start when the overall we are clear with the product requirements and then need to design the system completely. This understanding will be at the beginning of complete under the product development process. these will be beneficial for the future execution of test cases.

3. Architectural Design

In this stage, architectural specifications are comprehended and designed. Usually, several technical approaches are put out, and the ultimate choice is made after considering both the technical and financial viability. The system architecture is further divided into modules that each handle a distinct function. Another name for this is High-Level Design (HLD). At this point, the exchange of data and communication between the internal modules and external systems are well understood and defined. During this phase, integration tests can be created and documented using the information provided.

4. Module Design

This phase, known as Low-Level Design (LLD), specifies the comprehensive internal design for every system module. Compatibility between the design and other external systems as well as other modules in the system architecture is crucial. Unit tests are a crucial component of any development process since they assist in identifying and eradicating most mistakes and flaws at an early stage. Based on the internal module designs, these unit tests may now be created.

5. Coding Phase

This is where the software is built. Developers write the code based on the design created in the previous phase. The Coding step involves writing the code for the system modules that were created during the Design phase. The

system and architectural requirements are used to determine which programming language is most appropriate. The coding standards and principles are followed when performing the coding. Before the final build is checked into the repository, the code undergoes many code reviews and is optimized for optimal performance.

V-Model Validation Phases

It involves dynamic analysis techniques (functional, and non-functional), and testing done by executing code. Validation is the process of evaluating the software after the completion of the development phase to determine whether the software meets the customer's expectations and requirements.

1. Unit Testing

In **Unit testing**, unit Test Plans are developed during the module design phase. These Unit Test Plans are executed to eliminate bugs in code or unit level.

2. Integration testing

After completion of unit testing **Integration testing** is performed. In integration testing, the modules are integrated and the system is tested. Integration testing is performed in the Architecture design phase. This test verifies the communication of modules among themselves.

3. System Testing

System testing tests the complete application with its functionality, inter-dependency, and communication. It tests the functional and non-functional requirements of the developed application.

4. User Acceptance Testing (UAT)

User Acceptance Testing (UAT) is performed in a user environment that resembles the production environment. UAT verifies that the delivered system meets the user's requirement and the system is ready for use in the real world.

Advantages of V-Model

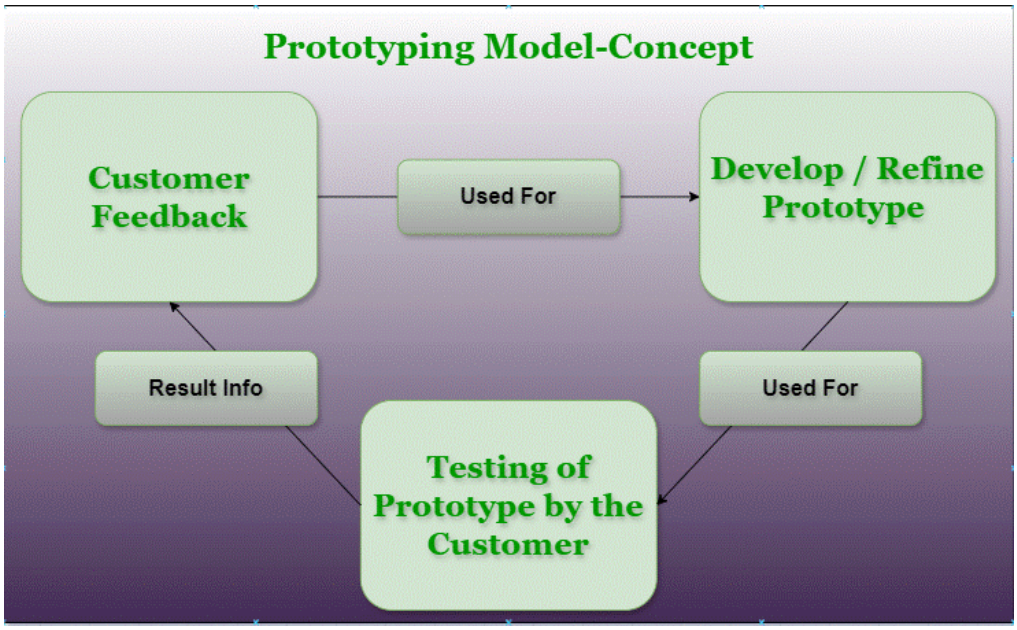
- This is a highly disciplined model and Phases are completed one at a time.
- V-Model is used for small projects where project requirements are clear.
- Simple and easy to understand and use.
- This model focuses on verification and validation activities early in the life cycle thereby enhancing the probability of building an error-free and good quality product.
- It enables project management to track progress accurately.
- Clear and Structured Process: The V-Model provides a clear and structured process for software development, making it easier to understand and follow.
- Emphasis on Testing: The V-Model places a strong emphasis on testing, which helps to ensure the quality and reliability of the software.
- Improved Traceability: The V-Model provides a clear link between the requirements and the final product, making it easier to trace and manage changes to the software.
- Better Communication: The clear structure of the V-Model helps to improve communication between the customer and the development team.

Dis-Advantages of V-Model

- The V-Model is a linear and sequential model, which can make it difficult to adapt to changing requirements or unexpected events.
- The V-Model can be time-consuming, as it requires a lot of documentation and testing.
- High risk and uncertainty.
- It is not good for complex and object-oriented projects.
- It is not suitable for projects where requirements are not clear and contain a high risk of changing.
- This model does not support iteration of phases.

- It does not easily handle concurrent events.
- The V-Model places a strong emphasis on documentation, which can lead to an overreliance on documentation at the expense of actual development work.

4) **Prototype Model** : The Prototyping Model is one of the most popularly used **Software Development Life Cycle Models (SDLC models)**. This model is used when the customers do not know the exact project requirements beforehand. In this model, a prototype of the end product is first developed, tested, and refined as per customer feedback repeatedly till a final acceptable prototype is achieved which forms the basis for developing the final product.



Advantages of Prototyping Model

- The customers get to see the partial product early in the life cycle. This ensures a greater level of customer satisfaction and comfort.
- New requirements can be easily accommodated as there is scope for refinement.
- Missing functionalities can be easily figured out.
- Errors can be detected much earlier thereby saving a lot of effort and cost, besides enhancing the quality of the software.

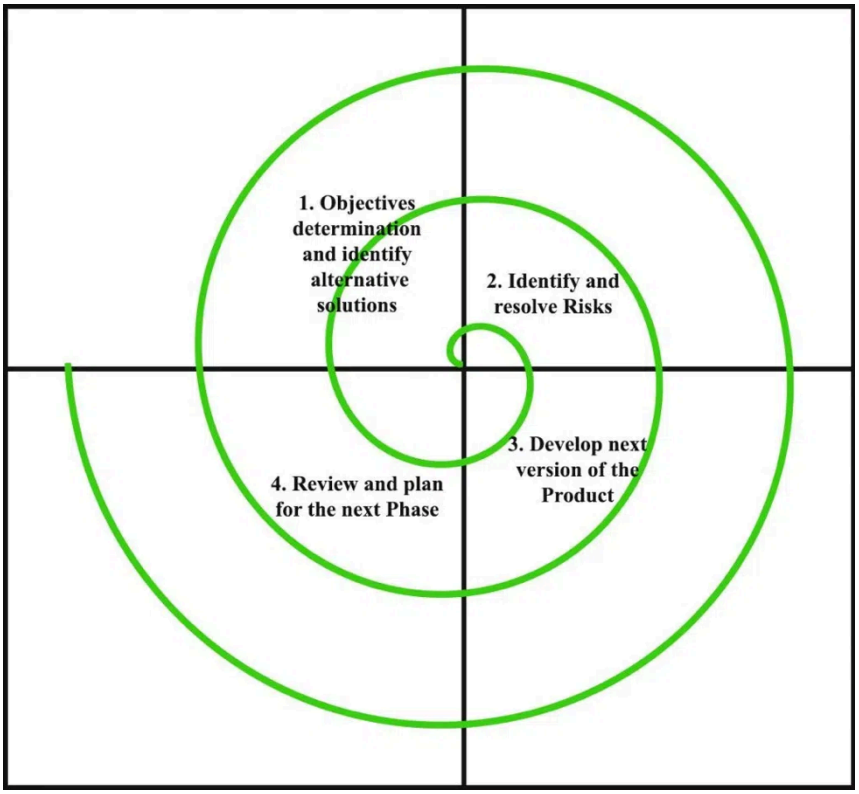
- The developed prototype can be reused by the developer for more complicated projects in the future.
- Flexibility in design.
- Early feedback from customers and stakeholders can help guide the development process and ensure that the final product meets their needs and expectations.
- Prototyping can be used to test and validate design decisions, allowing for adjustments to be made before significant resources are invested in development.
- Prototyping can help reduce the risk of project failure by identifying potential issues and addressing them early in the process.
- Prototyping can facilitate communication and collaboration among team members and stakeholders, improving overall project efficiency and effectiveness.
- Prototyping can help bridge the gap between technical and non-technical stakeholders by providing a tangible representation of the product.

Disadvantages of the Prototyping Model

- Costly concerning time as well as money.
- There may be too much variation in requirements each time the prototype is evaluated by the customer.
- Poor Documentation due to continuously changing customer requirements.
- It is very difficult for developers to accommodate all the changes demanded by the customer.
- There is uncertainty in determining the number of iterations that would be required before the prototype is finally accepted by the customer.
- After seeing an early prototype, the customers sometimes demand the actual product to be delivered soon.
- Developers in a hurry to build prototypes may end up with sub-optimal solutions.
- The customer might lose interest in the product if he/she is not satisfied with the initial prototype.

- The prototype may not be scalable to meet the future needs of the customer.
- The prototype may not accurately represent the final product due to limited functionality or incomplete features.
- The focus on prototype development may shift away from the final product, leading to delays in the development process.
- The prototype may give a false sense of completion, leading to the premature release of the product.
- The prototype may not consider technical feasibility and scalability issues that can arise during the final product development.
- The prototype may be developed using different tools and technologies, leading to additional training and maintenance costs.
- The prototype may not reflect the actual business requirements of the customer, leading to dissatisfaction with the final product.

5) **Spiral Model :** The Spiral Model is a **Software Development Life Cycle (SDLC)** model that provides a systematic and iterative approach to software development. In its diagrammatic representation, looks like a spiral with many loops. The exact number of loops of the spiral is unknown and can vary from project to project. Each loop of the spiral is called a **phase** of the software development process. The Spiral model is called a **Meta-Model** because it subsumes all the other SDLC models



Advantages of the Spiral Model

Below are some advantages of the Spiral Model.

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

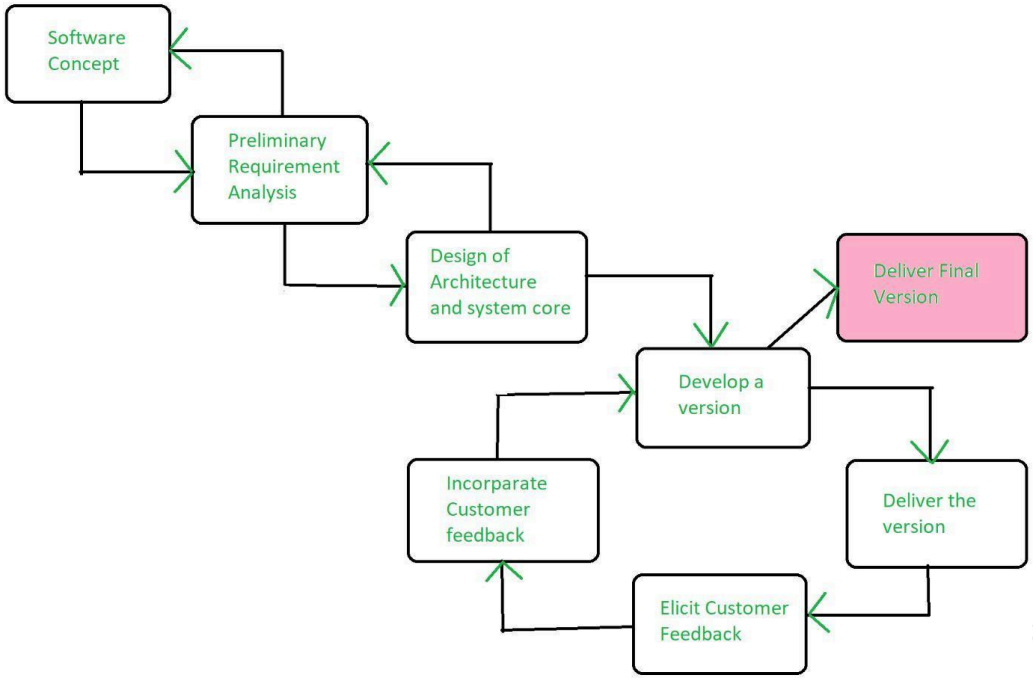
5. **Iterative and Incremental Approach:** The Spiral Model provides an iterative and incremental approach to software development, allowing for flexibility and adaptability in response to changing requirements or unexpected events.
6. **Emphasis on Risk Management:** The Spiral Model places a strong emphasis on risk management, which helps to minimize the impact of uncertainty and risk on the software development process.
7. **Improved Communication:** The Spiral Model provides for regular evaluations and reviews, which can improve communication between the customer and the development team.
8. **Improved Quality:** The Spiral Model allows for multiple iterations of the software development process, which can result in improved software quality and reliability.

Disadvantages of the Spiral Model

Below are some main disadvantages of the spiral model.

1. **Complex:** The Spiral Model is much more complex than other SDLC models.
2. **Expensive:** Spiral Model is not suitable for small projects as it is expensive.
3. **Too much dependability on Risk Analysis:** The successful completion of the project is very much dependent on Risk Analysis. Without very highly experienced experts, it is going to be a failure to develop a project using this model.
4. **Difficulty in time management:** As the number of phases is unknown at the start of the project, time estimation is very difficult.
5. **Complexity:** The Spiral Model can be complex, as it involves multiple iterations of the software development process.
6. **Time-Consuming:** The Spiral Model can be time-consuming, as it requires multiple evaluations and reviews.
7. **Resource Intensive:** The Spiral Model can be resource-intensive, as it requires a significant investment in planning, risk analysis, and evaluations.

6) **Evolutionary Model :** The evolutionary model is a combination of the Iterative and Incremental models of the software development life cycle. Delivering your system in a big bang release, delivering it in incremental process over time is the action done in this model. Some initial requirements and architecture envisioning need to be done. It is better for software products that have their feature sets redefined during development because of user feedback and other factors.



Advantages Evolutionary Model

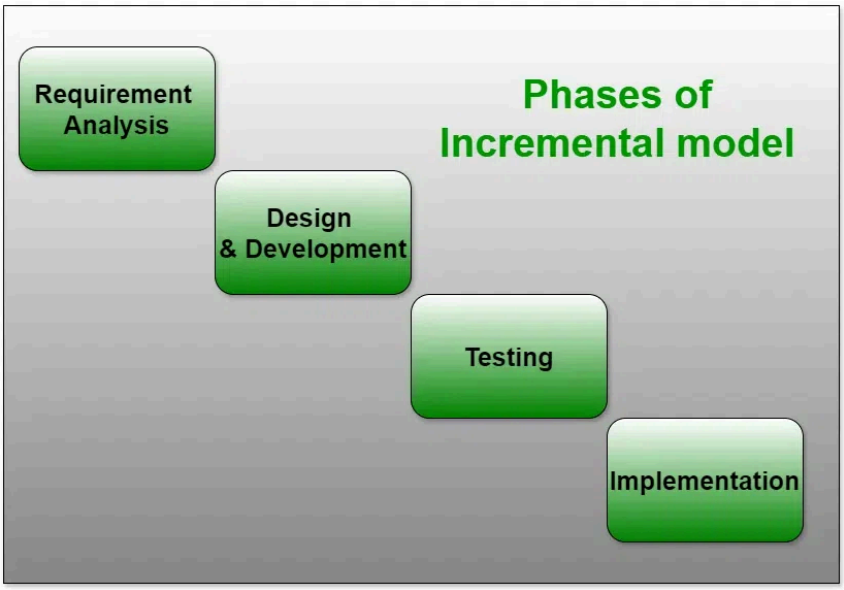
1. **Adaptability to Changing Requirements:** Evolutionary models work effectively in projects when the requirements are ambiguous or change often. They support adjustments and flexibility along the course of development.
2. **Early and Gradual Distribution:** Functional components or prototypes can be delivered early thanks to incremental development. Faster user satisfaction and feedback may result from this.

3. **User Commentary and Involvement:** Evolutionary models place a strong emphasis on ongoing user input and participation. This guarantees that the software offered closely matches the needs and expectations of the user.
4. **Improved Handling of Difficult Projects:** Big, complex tasks can be effectively managed with the help of evolutionary models. The development process is made simpler by segmenting the project into smaller, easier-to-manage portions.

Disadvantages Evolutionary Model

1. **Communication Difficulties:** Evolutionary models require constant cooperation and communication. The strategy may be less effective if there are gaps in communication or if team members are spread out geographically.
2. **Dependence on an Expert Group:** A knowledgeable and experienced group that can quickly adjust to changes is needed for evolutionary models. Teams lacking experience may find it difficult to handle these model's dynamic nature.
3. **Increasing Management Complexity:** Complexity can be introduced by organizing and managing several increments or iterations, particularly in large projects. In order to guarantee integration and synchronization, good project management is needed.
4. **Greater Initial Expenditure:** As evolutionary models necessitate continual testing, user feedback and prototyping, they may come with a greater starting cost. This may be a problem for projects that have limited funding.

7) **Incremental Process Model**: The Incremental model is a software Development approach which is used to breakdown the project into smaller and easily manageable parts. In these each part will go through Requirement, Design, Testing phases and Implementation phase. The overall process continues until we got the complete System.



Advantages of Incremental Process Model

The Incremental Model of software development builds a system in small, manageable sections (increments), making it a good choice for many projects. Below are the key advantages:

- **Faster Software Delivery**
    - Initial working versions of the software can be delivered quickly.
    - Early delivery increases customer satisfaction and feedback opportunities.
- **Clear Understanding for Clients**
    - Clients get to see parts of the system at each stage.
    - This visibility ensures that the final product meets their expectations.
- **Easy to Implement Changes**
    - Requirements can evolve, and changes can be incorporated in subsequent increments.
    - It supports flexibility without heavily disrupting earlier stages.
- **Effective Risk Management**
    - Risks can be identified and handled early due to the staged approach.

    - Each increment allows for testing and validation, reducing the impact of unforeseen issues.
- **Flexible Criteria and Scope**
    - Requirements can be adjusted without a major cost increase.
    - Better scope management helps keep the project aligned with business goals.
- **Cost-Effective**
    - Compared to models like the Waterfall, the incremental model is generally more cost-efficient.
    - Budget is spread across stages, making it easier to manage finances.
- **Simpler Error Identification**
    - Since development is done in parts, it's easier to pinpoint and fix errors within a specific increment.
    - Testing each module separately enhances quality and reliability.

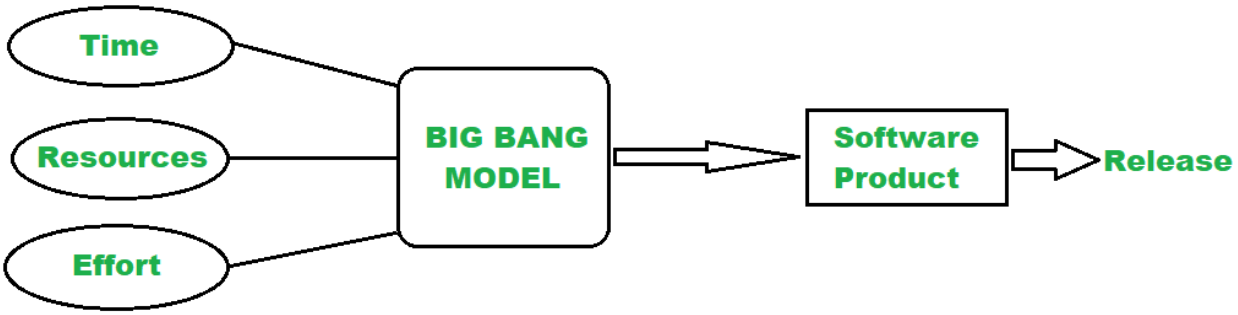
Disadvantages of Incremental Process Model

Incremental Model comes with some limitations and challenges. Below are the key disadvantages:

- **Requires a Skilled Team and Proper Planning**
    - Successful implementation demands an experienced team.
    - Poor planning or coordination can lead to confusion between increments.
- **Cost Can Increase Over Time**
    - Due to repeated testing, redesign, and integration in every cycle, the overall project cost may rise.
    - Continuous iteration involves added overhead
- **Incomplete Requirement Gathering Can Cause Design Issues**
    - If all requirements are not identified early, the system architecture may not support future needs.
    - Inadequate upfront design can lead to rework and architectural mismatches.

- **Lack of Smooth Flow Between Increments**
    - Each iteration may function independently, which can create inconsistencies.
    - There might be integration challenges when combining all the increments into a unified product.
- **High Effort to Fix Repeated Issues**
    - A defect in one increment may exist in others.
    - Fixing the same issue across multiple units can be time-consuming and resource-intensive.

8) **Big Bang Model :** The Big bang model is an SDLC model that starts from nothing. It is the simplest model in SDLC (Software Development Life Cycle) as it requires almost no planning. However, it requires lots of funds and coding and takes more time. The name big bang model was set after the “Great Big Bang” which led to the development of galaxies, stars, planets, etc. Similarly, this SDLC model combines time, efforts, and resources to build a product.



Advantages of Big Bang Model :

- There is no planning required for this.
- Suitable for small projects
- Very few resources are required.
- As there is no proper planning hence it does not require managerial staffs
- Easy to implement
- It develops the skills of the newcomers

- Very much flexible for the developers working on it

Disadvantages of Big Bang Model :

- Not suitable for large projects.
- Highly risky model and uncertain
- Might be expensive if requirements are not clear
- Poor model for ongoing projects

9) **Agile Model** : **The Agile Model** was primarily designed to help a project adapt quickly to change requests. So, the main aim of the Agile model is to facilitate quick project completion. To accomplish this task, it's important that agility is required. Agility is achieved by fitting the process to the project and removing activities that may not be essential for a specific project. Also, anything that is a waste of time and effort is avoided. The Agile Model refers to a group of development processes. These processes share some basic characteristics but do have certain subtle differences among themselves.

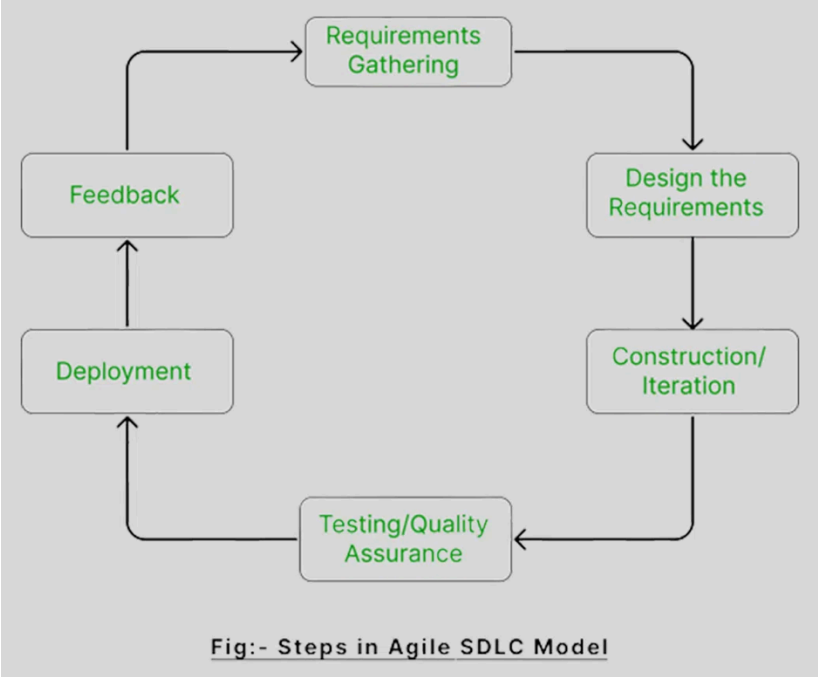


Fig:- Steps in Agile SDLC Model

Advantages of the Agile Model

- Working through Pair programming produces well-written compact programs which have fewer errors as compared to programmers working alone.
- It reduces the total development time of the whole project.
- Agile development emphasizes face-to-face communication among team members, leading to better collaboration and understanding of project goals.
- Customer representatives get the idea of updated software products after each iteration. So, it is easy for him to change any requirement if needed.
- Agile development puts the customer at the center of the development process, ensuring that the product meets their needs.

Disadvantages of the Agile Model

- The lack of formal documents creates confusion and important decisions taken during different phases can be misinterpreted at any time by different team members.
- It is not suitable for handling complex dependencies.
- The agile model depends highly on customer interactions so if the customer is not clear, then the development team can be driven in the wrong direction.
- Agile development models often involve working in short sprints, which can make it difficult to plan and forecast project timelines and deliverables. This can lead to delays in the project and can make it difficult to accurately estimate the costs and resources needed for the project.
- Agile development models require a high degree of expertise from team members, as they need to be able to adapt to changing requirements and work in an iterative environment. This can be challenging for teams that are not experienced in agile development practices and can lead to delays and difficulties in the project.

- Due to the absence of proper documentation, when the project completes and the developers are assigned to another project, maintenance of the developed project can become a problem.