Unit-1

Introduction to Software Engineering:

What is Software Engineering?

- The term **software engineering** is the product of two words, **software** and **engineering**.
- The **software** is a collection of integrated programs.
- Software subsists of carefully-organized instructions and code written by developers on any of various particular computer languages.
- Computer programs and related documentation such as requirements, design models and user manuals.
- **Engineering** is the application of **scientific** and **practical** knowledge to **invent, design, build, maintain** and **improve frameworks, processes, etc**.

Software Engineering is an engineering branch related to the evolution of software product using well-defined scientific principles, techniques and procedures. The result of software engineering is an effective and reliable software product.

Why is Software Engineering required?

Software Engineering is required due to the following reasons:

- To manage large software
- For more Scalability
- Cost Management
- To manage the dynamic nature of software
- For better quality Management

Need of Software Engineering:

The necessity of software engineering appears because of a higher rate of progress in user requirements and the environment on which the program is working.

- **Huge Programming:** It is simpler to manufacture a wall than to a house or building, similarly, as the measure of programming become extensive engineering has to step to give it a scientific process.
- **Adaptability:** If the software procedure were not based on scientific and engineering ideas, it would be simpler to re-create new software than to scale an existing one.
- **Cost:** As the hardware industry has demonstrated its skills and huge manufacturing has let down the cost of computer and electronic hardware. But the cost of programming remains high if the proper process is not adapted.

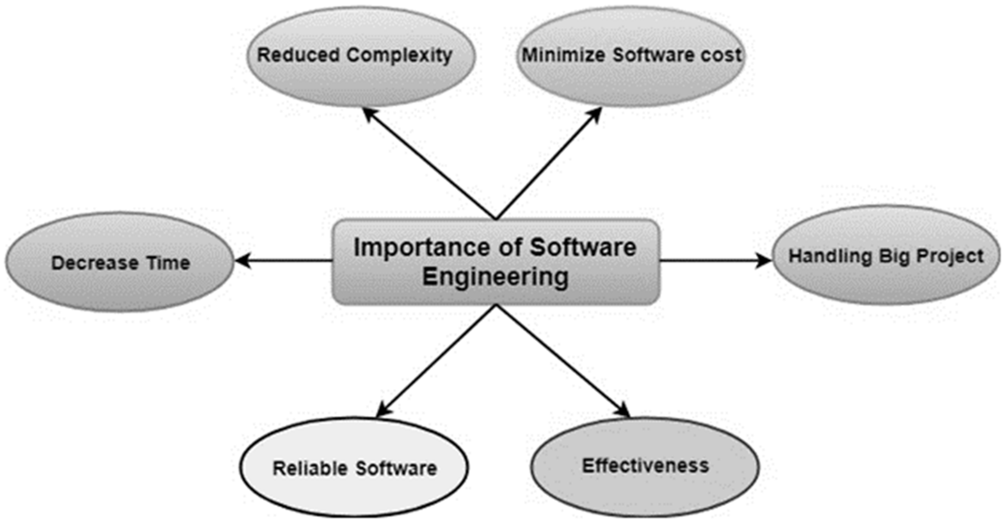
- **Dynamic Nature:** The continually growing and adapting nature of programming hugely depends upon the environment in which the client works. If the quality of the software is continually changing, new upgrades need to be done in the existing one.
- **Quality Management:** Better procedure of software development provides a better and quality software product.

Characteristics of a good software engineer:

The features that good software engineers should possess are as follows:

- Exposure to systematic methods, i.e., familiarity with software engineering principles.
- Good technical knowledge of the project range (Domain knowledge).
- Good programming abilities.
- Good communication skills. These skills comprise of oral, written and interpersonal skills.
- High motivation.
- Sound knowledge of fundamentals of computer science.
- Intelligence.
- Ability to work in a team
- Discipline, etc.

Importance of Software Engineering:



The importance of Software engineering is as follows:

1. **Reduces complexity:** Big software is always complicated and challenging to progress. Software engineering has a great solution to reduce the complication of any project. Software engineering divides big problems into various small issues. And then start solving each small issue one by one. All these small problems are solved independently to each other.
2. **To minimize software cost:** Software needs a lot of hard work and software engineers are highly paid experts. A lot of manpower is required to develop software with a large number of codes. But

in software engineering, programmers project everything and decrease all those things that are not needed. In turn, the cost for software productions becomes less as compared to any software that does not use software engineering method.

3. **To decrease time:** Anything that is not made according to the project always wastes time. And if you are making great software, then you may need to run many codes to get the definitive running code. This is a very time-consuming procedure, and if it is not well handled, then this can take a lot of time. So, if you are making your software according to the software engineering method, then it will decrease a lot of time.
4. **Handling big projects:** Big projects are not done in a couple of days, and they need lots of patience, planning and management. And to invest six and seven months of any company, it requires heaps of planning, direction, testing, and maintenance. No one can say that he has given four months of a company to the task, and the project is still in its first stage. Because the company has provided many resources to the plan and it should be completed. So, to handle a big project without any problem, the company has to go for a software engineering method.
5. **Reliable software:** Software should be secure, means if you have delivered the software, then it should work for at least its given time or subscription. And if any bugs come in the software, the company is responsible for solving all these bugs. Because in software engineering, testing and maintenance are given, so there is no worry of its reliability.
6. **Effectiveness:** Effectiveness comes if anything has made according to the standards. Software standards are the big target of companies to make it more effective. So, Software becomes more effective in the act with the help of software engineering.

Key Principles of Software Engineering:

1. **Modularity**: Breaking the software into smaller, reusable components that can be developed and tested independently.
2. **Abstraction**: Hiding the implementation details of a component and exposing only the necessary functionality to other parts of the software.
3. **Encapsulation**: Wrapping up the data and functions of an object into a single unit, and protecting the internal state of an object from external modifications.
4. **Reusability**: Creating components that can be used in multiple projects, which can save time and resources.
5. **Maintenance**: Regularly updating and improving the software to fix bugs, add new features, and address security vulnerabilities.
6. **Testing**: Verifying that the software meets its requirements and is free of bugs.

7. **Design Patterns**: Solving recurring problems in software design by providing templates for solving them.
8. **Agile methodologies:** Using iterative and incremental development processes that focus on customer satisfaction, rapid delivery and flexibility.
9. **Continuous Integration & Deployment:** Continuously integrating the code changes and deploying them into the production environment.

Main Attributes of Software Engineering:

Software Engineering is a systematic, disciplined, quantifiable study and approach to the design, development, operation and maintenance of a software system. There are four main Attributes of Software Engineering.

1. **Efficiency:** It provides a measure of the resource requirement of a software product efficiently.
2. **Reliability:** It assures that the product will deliver the same results when used in similar working environment.
3. **Reusability:** This attribute makes sure that the module can be used in multiple applications.
4. **Maintainability:** It is the ability of the software to be modified, repaired or enhanced easily with changing requirements.

Dual Role of Software:

There is a dual role of software in the industry. The first one is as a product and the other one is as a vehicle for delivering the product. We will discuss both of them.

1. **As a Product**

- It delivers computing potential across networks of Hardware.
- It enables the Hardware to deliver the expected functionality.
- It acts as an information transformer because it produces, manages, acquires, modifies, displays or transmits information.

2. **As a Vehicle for Delivering a Product**

- It provides system functionality (e.g., payroll system).
- It controls other software (e.g., an operating system).
- It helps build other software (e.g., software tools).

Objectives of Software Engineering:

1. **Maintainability:** It should be feasible for the software to evolve to meet changing requirements.
2. **Efficiency:** The software should not make wasteful use of computing devices such as memory, processor cycles, etc.

3. **Correctness:** A software product is correct if the different requirements specified in the SRS Document have been correctly implemented.
4. **Reusability:** A software product has good reusability if the different modules of the product can easily be reused to develop new products.
5. **Testability:** Here software facilitates both the establishment of test criteria and the evaluation of the software concerning those criteria.
6. **Reliability:** It is an attribute of software quality. The extent to which a program can be expected to perform its desired function, over an arbitrary time period.
7. **Portability:** In this case, the software can be transferred from one computer system or environment to another.
8. **Adaptability:** In this case, the software allows differing system constraints and the user needs to be satisfied by making changes to the software.
9. **Interoperability:** Capability of 2 or more functional units to process data cooperatively.

Advantages of Software Engineering:

There are several advantages to using a systematic and disciplined approach to software development, such as:

1. **Improved Quality:** By following established software engineering principles and techniques, the software can be developed with fewer bugs and higher reliability.
2. **Increased Productivity:** Using modern tools and methodologies can streamline the development process, allowing developers to be more productive and complete projects faster.
3. **Better Maintainability:** Software that is designed and developed using sound software engineering practices is easier to maintain and update over time.
4. **Reduced Costs:** By identifying and addressing potential problems early in the development process, software engineering can help to reduce the cost of fixing bugs and adding new features later on.
5. I**ncreased Customer Satisfaction:** By involving customers in the development process and developing software that meets their needs, software engineering can help to increase customer satisfaction.
6. **Better Team Collaboration:** By using Agile methodologies and continuous integration, software engineering allows for better collaboration among development teams.
7. **Better Scalability**: By designing software with scalability in mind, software engineering can help to ensure that software can handle an increasing number of users and transactions.
8. **Better Security:** By following the Software Development Life Cycle (SDLC) and performing security testing, software engineering can help to prevent security breaches and protect sensitive data.

Disadvantages of Software Engineering:

While Software Engineering offers many advantages, there are also some potential disadvantages to consider:

1. **High upfront costs:** Implementing a systematic and disciplined approach to software development can be resource-intensive and require a significant investment in tools and training.
2. **Limited flexibility:** Following established software engineering principles and methodologies can be rigid and may limit the ability to quickly adapt to changing requirements.
3. **Bureaucratic**: Software Engineering can create an environment that is bureaucratic, with a lot of processes and paperwork, which may slow down the development process.
4. **Complexity**: With the increase in the number of tools and methodologies, software engineering can be complex and difficult to navigate.
5. **Limited creativity:** The focus on structure and process can stifle creativity and innovation among developers.
6. **High learning curve:** The development process can be complex, and it requires a lot of learning and training, which can be challenging for new developers.
7. **High dependence on tools:** Software engineering heavily depends on the tools, and if the tools are not properly configured or are not compatible with the software, it can cause issues.
8. **High maintenance**: The software engineering process requires regular maintenance to ensure that the software is running efficiently, which can be costly and time-consuming.

The Essential Software Engineering Code of Ethics:

Ethics in software engineering is about making decisions that respect user privacy, promote fairness, and benefit society. It involves adhering to professional codes, navigating ethical dilemmas, and understanding the impact of unethical practices.

In software engineering, ethics play a crucial role in guiding the actions and decisions of professionals in this field.

The software engineering code of ethics is fundamental to ensuring that software engineers adhere to standards and promote responsible behavior in their work.

Understanding the importance of a software engineering code of ethics:

- Software engineering is constantly evolving, with new technologies emerging at a rapid pace.
- Software engineers must understand and adhere to ethical standards in this fast-paced environment.
- A software engineering code of ethics is a moral compass, guiding professionals to make responsible decisions and ensuring that the products and services they develop align with societal values and expectations.
- Software engineering is about writing code and creating solutions that positively impact society.

- Ethical considerations are crucial in this process, as they help software engineers navigate complex issues and make choices that priorities the well-being of users and the broader community.

The role of ethics in software development:

- Ethics play a significant role in software development as they guide engineers in making choices that align with ethical principles.
- Software development involves various stages, including planning, designing, coding, testing and maintenance.
- At each stage, ethical considerations should be considered to ensure that the software created is reliable secure and respects user privacy.

The Code of Ethics: during the planning phase:

- During the planning phase, software engineers must consider the potential impact of their work on different stakeholders.
- They need to assess the ethical implications of the software's purpose and functionality, ensuring that it does not harm individuals or perpetuate discrimination.
- This requires a deep understanding of societal values and anticipating potential risks.

The Code of Ethics: during the design phase:

- When designing the software, ethical considerations come into play regarding user experience and accessibility.
- Engineers must strive to create intuitive, inclusive interfaces and respect users' privacy.
- They should also consider the environmental impact of their designs, aiming for energy efficiency and sustainability.
- The coding phase is where software engineers translate their designs into actual code.
- Ethical coding involves writing clean, well-documented code that is easy to maintain and understand.
- Engineers should also prioritize security measures, implementing robust authentication and encryption mechanisms to protect user data.

The Code of Ethics: during the testing phase:

- Testing is another critical stage where ethics are essential. Engineers must thoroughly test the software for reliability, accuracy and security.
- They should also consider the potential biases that may arise from the algorithms used in the software and take steps to mitigate them.

The Code of Ethics: when maintaining software:

- Maintenance is an ongoing process in software development, and ethical considerations should continue to be prioritized.
- Engineers must respond to user feedback and promptly address any issues or vulnerabilities.
- They should also stay current with emerging ethical standards and adapt their software accordingly.

Key principles of the software engineering code of ethics:

- The software engineering code of ethics establishes guiding principles to help professionals make ethical decisions. These principles encompass responsibility, accountability, fairness and justice.
- Responsibility and accountability in software engineering
- Software engineers must ensure their software is reliable, safe and meets users' needs.
- They must also be accountable for the consequences of their actions and decisions. This includes addressing and rectifying any issues or vulnerabilities that may arise in the software.
- Fairness and justice in software engineering practices
- Software engineers should strive to create inclusive and accessible software that does not discriminate against individuals based on race, gender, age or disability.
- They should also ensure that the software they develop adheres to legal and ethical standards, promoting fairness and justice in their work.

Implementing the software engineering code of ethics:

- While ethical principles are a foundation for responsible behavior, implementation can often present challenges.
- However, software engineering organizations must prioritize ethical practices and create an environment that promotes ethical behavior.
- Promoting ethical behavior in the workplace
- Organisations should foster a culture of ethics and integrity, where ethical standards are communicated and upheld throughout all company levels.
- This involves training and education on ethical principles, encouraging open discussions on ethical dilemmas, and establishing mechanisms for reporting unethical behavior.
- Overcoming challenges in ethical implementation
- Implementing a software engineering code of ethics can pose challenges, particularly in complex projects and tight deadlines.
- However, it is essential to prioritize ethical considerations and integrate them into the development process.
- This may require allocating sufficient resources and time for ethical evaluations, conducting regular honest reviews, and involving stakeholders in ethical decision-making.

Software Process Model:

Software Processes is a coherent set of activities for specifying, designing, implementing and testing software systems. A software process model is an abstract representation of a process that presents a description of a process from some particular perspective. There are many different software processes but all involve:

1. Specification – defining what the system should do;
2. Design and implementation – defining the organization of the system and implementing the system;
3. Validation – checking that it does what the customer wants;
4. Evolution – changing the system in response to changing customer needs.

Types of Software Process Model:

Software processes, methodologies and frameworks range from specific prescriptive steps that can be used directly by an organization in day-to-day work, to flexible frameworks that an organization uses to generate a custom set of steps tailored to the needs of a specific project or group. In some cases, a "sponsor" or "maintenance" organization distributes an official set of documents that describe the process.

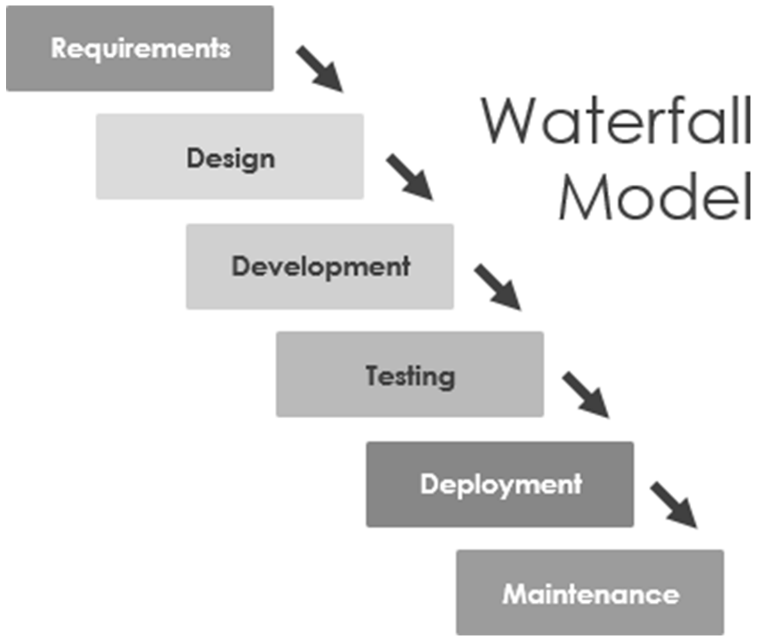
Software Process and Software Development Lifecycle Model:

One of the basic notions of the software development process is SDLC models which stands for Software Development Life Cycle models. There are many development life cycle models that have been developed in order to achieve different required objectives. The models specify the various stages of the process and the order in which they are carried out. The most used, popular and important SDLC models are given below:

1. Waterfall model
2. V model
3. Incremental model
4. RAD model
5. Agile model
6. Iterative model
7. Spiral model

Waterfall Model:

The waterfall model is a breakdown of project activities into linear sequential phases, where each phase depends on the deliverables of the previous one and corresponds to a specialization of tasks. The approach is typical for certain areas of engineering design.



The Waterfall Model has six phases which are:

1. Requirements: The first phase involves gathering requirements from stakeholders and analyzing them to understand the scope and objectives of the project.

2. Design: Once the requirements are understood, the design phase begins. This involves creating a detailed design document that outlines the software architecture, user interface, and system components.

3. Development: The Development phase include implementation involves coding the software based on the design specifications. This phase also includes unit testing to ensure that each component of the software is working as expected.

4. Testing: In the testing phase, the software is tested as a whole to ensure that it meets the requirements and is free from defects.

5. Deployment: Once the software has been tested and approved, it is deployed to the production environment.

6. Maintenance: The final phase of the Waterfall Model is maintenance, which involves fixing any issues that arise after the software has been deployed and ensuring that it continues to meet the requirements over time.

Advantages of the SDLC 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 the SDLC 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 the SDLC **Waterfall Model?**

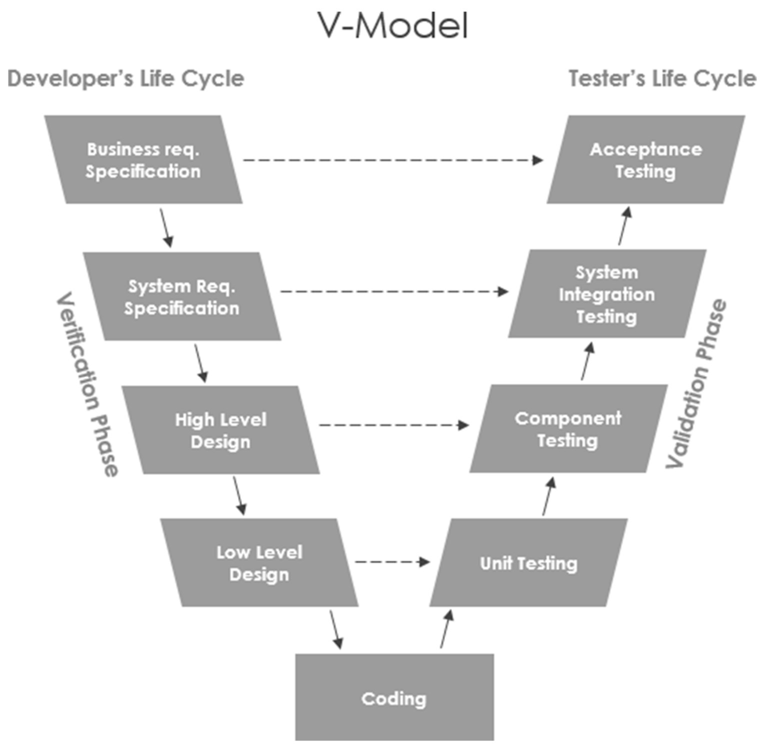
Here are some cases where the use of the Waterfall Model is best suited:

- **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.

V Model:

The V-model represents a development process that may be considered an extension of the waterfall model and is an example of the more general V-model. Instead of moving down in a linear way, the process steps are bent upwards after the coding phase, to form the typical V shape. The V-Model demonstrates the relationships between each phase of the development life cycle and its associated phase of testing. The horizontal and vertical axes represent time or project completeness (left-to-right) and level of abstraction (coarsest-grain abstraction uppermost), respectively.



Verification: It involves a static analysis method (review) done without executing code. It is the process of evaluation of the product development process to find whether specified requirements meet.

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

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

There are the various phases of Verification Phase of V-model:

1. **Business requirement analysis:** This is the first step where product requirements understood from the customer's side. This phase contains detailed communication to understand customer's expectations and exact requirements.
2. **System Design:** In this stage system engineers analyze and interpret the business of the proposed system by studying the user requirements document.
3. **Architecture Design:** The baseline in selecting the architecture is that it should understand all which typically consists of the list of modules, brief functionality of each module, their interface relationships, dependencies, database tables, architecture diagrams, technology detail, etc. The integration testing model is carried out in a particular phase.
4. **Module Design:** In the module design phase, the system breaks down into small modules. The detailed design of the modules is specified, which is known as Low-Level Design
5. **Coding Phase:** After designing, the coding phase is started. Based on the requirements, a suitable programming language is decided. There are some guidelines and standards for coding. Before checking in the repository, the final build is optimized for better performance, and the code goes through many code reviews to check the performance.

There are the various phases of Validation Phase of V-model:

1. **Unit Testing:** In the V-Model, Unit Test Plans (UTPs) are developed during the module design phase. These UTPs are executed to eliminate errors at code level or unit level. A unit is the smallest entity which can independently exist, e.g., a program module. Unit testing verifies that the smallest entity can function correctly when isolated from the rest of the codes/ units.
2. **Integration Testing:** Integration Test Plans are developed during the Architectural Design Phase. These tests verify that groups created and tested independently can coexist and communicate among themselves.
3. **System Testing:** System Tests Plans are developed during System Design Phase. Unlike Unit and Integration Test Plans, System Tests Plans are composed by the client’s business team. System Test ensures that expectations from an application developer are met.
4. **Acceptance Testing:** Acceptance testing is related to the business requirement analysis part. It includes testing the software product in user atmosphere. Acceptance tests reveal the compatibility problems with the different systems, which is available within the user atmosphere. It conjointly discovers the non-functional problems like load and performance defects within the real user atmosphere.

When to use V-Model?

- When the requirement is well defined and not ambiguous.

- The V-shaped model should be used for small to medium-sized projects where requirements are clearly defined and fixed.
- The V-shaped model should be chosen when sample technical resources are available with essential technical expertise.

Advantage (Pros) of V-Model:

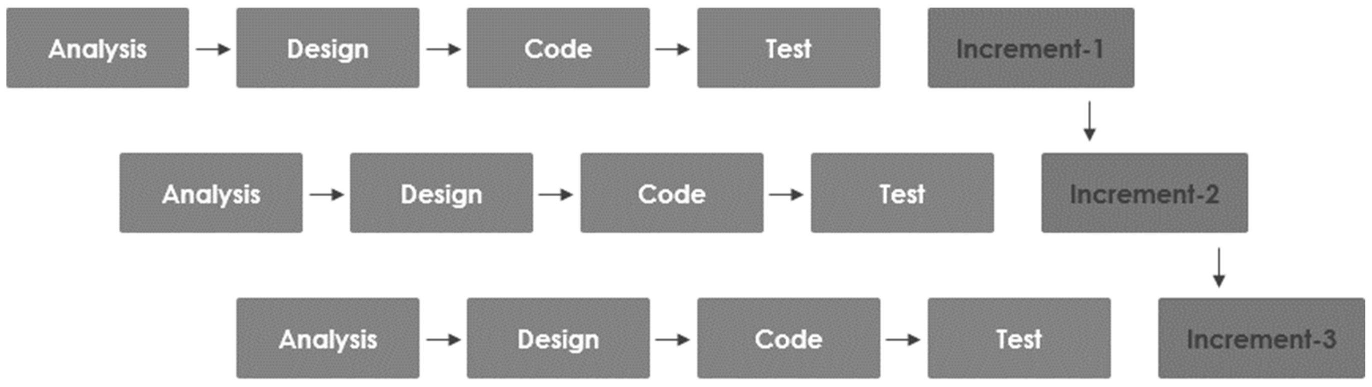
1. Easy to Understand.
2. Testing Methods like planning, test designing happens well before coding.
3. This saves a lot of time. Hence a higher chance of success over the waterfall model.
4. Avoids the downward flow of the defects.
5. Works well for small plans where requirements are easily understood.

Disadvantage (Cons) of V-Model:

1. Very rigid and least flexible.
2. Not a good for a complex project.
3. Software is developed during the implementation stage, so no early prototypes of the software are produced.
4. If any changes happen in the midway, then the test documents along with the required documents, has to be updated.

Incremental model:

The incremental build model is a method of software development where the model is designed, implemented and tested incrementally (a little more is added each time) until the product is finished. It involves both development and maintenance. The product is defined as finished when it satisfies all of its requirements. Each iteration passes through the requirements, design, coding and testing phases. And each subsequent release of the system adds function to the previous release until all designed functionally has been implemented. This model combines the elements of the waterfall model with the iterative philosophy of prototyping.



Incremental Model

The various phases of incremental model are as follows:

1. Requirement analysis: In the first phase of the incremental model, the product analysis expertise identifies the requirements. And the system functional requirements are understood by the requirement analysis team. To develop the software under the incremental model, this phase performs a crucial role.

2. Design & Development: In this phase of the Incremental model of SDLC, the design of the system functionality and the development method are finished with success. When software develops new practicality, the incremental model uses style and development phase.

3. Testing: In the incremental model, the testing phase checks the performance of each existing function as well as additional functionality. In the testing phase, the various methods are used to test the behavior of each task.

4. Implementation: Implementation phase enables the coding phase of the development system. It involves the final coding that design in the designing and development phase and tests the functionality in the testing phase. After completion of this phase, the number of the product working is enhanced and upgraded up to the final system product

When we use the Incremental Model?

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

Advantage of Incremental Model:

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

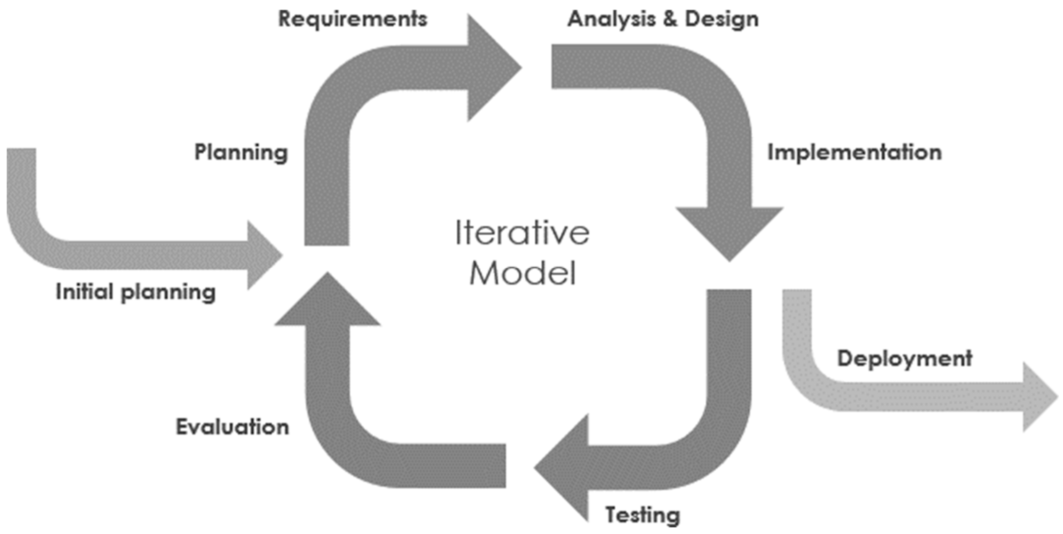
Disadvantage of Incremental Model:

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

Iterative Model:

An iterative life cycle model does not attempt to start with a full specification of requirements by first focusing on an initial, simplified set user feature, which then progressively gains more complexity and a broader set of features until the targeted system is complete. When adopting the iterative approach, the philosophy of incremental development will also often be used liberally and interchangeably.

In other words, the iterative approach begins by specifying and implementing just part of the software, which can then be reviewed and prioritized in order to identify further requirements. This iterative process is then repeated by delivering a new version of the software for each iteration. In a light-weight iterative project the code may represent the major source of documentation of the system; however, in a critical iterative project a formal software specification may also be required.



Phases of iterative model:

1. **Requirement gathering & analysis:** In this phase, all the software requirements of the customer are collected and it is analyzed whether those requirements can be meet or not. Besides, it is also checked whether this project will not go beyond our budget.
2. **Design**: In this phase the design of software is prepared. For this, various diagrams like Data Flow diagram, class diagram, activity diagram, state transition diagram, etc. are used.
3. **Implementation**: Now the design of software is implemented in coding through various programming languages. We also call this coding phase.
4. **Testing**: After the coding of the software is done, it is now tested so that the bugs and errors present in it can be identified. To do this, various testing techniques like performance testing, security testing, requirement testing, stress testing, etc. are done.
5. **Deployment**: Finally, the software is given to the customer. After this the customer starts using that software in his work environment.

6. **Review**: After the software is deployed in its work environment, it is reviewed. If any error/bug is found or any new requirements come in front of developer, then again these phases are repeated with new iteration and a new version is developed.
7. **Maintenance**: In this phase we look at customer feedback, solve problems, fix errors, update software, etc.

Advantage of Iterative model:

- In iterative models, bugs and errors can be identified quickly.
- Under this model, software is prepared quickly with some specifications.
- Testing and debugging the software becomes easier during each iteration.
- We get reliable feedback from users along with blueprints.
- This model is easily adaptable to constantly changing needs.
- During the software development process, additional time is devoted to development and limited time to documentation.
- Risks are identified and resolved during iteration.

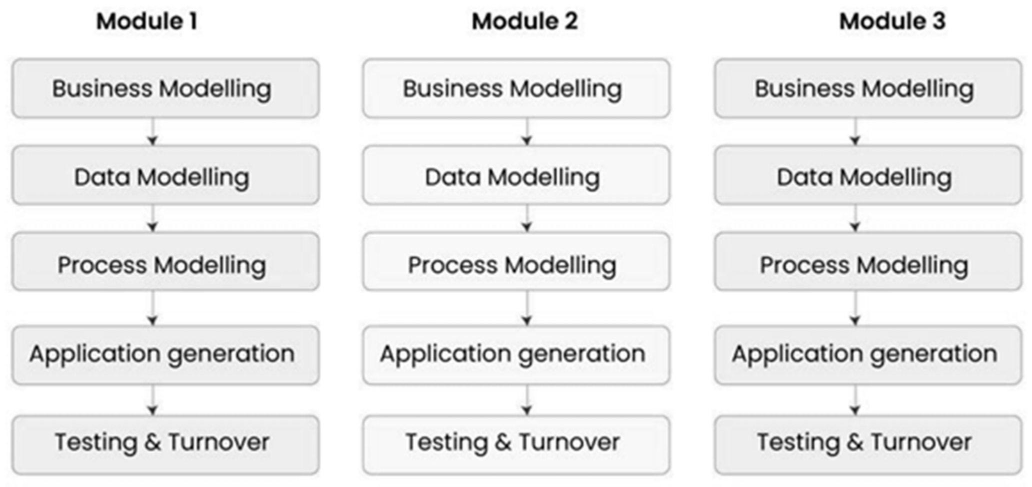
Disadvantage of Iterative model:

- Iterative model is not suitable for small projects.
- Since we have to repeat iterations many times in the software development process due to which we require more resources.
- Since the requirements are constantly changing, we have to make frequent changes in the software.
- Due to constantly changing requirements, the budget of the project also increases and it takes more time to complete it.
- In this model, it is complicated to control the entire process of software development.
- It is very difficult to tell by what date the complete software will be ready.

RAD model:

Rapid application development was a response to plan-driven waterfall processes, developed in the 1970s and 1980s, such as the Structured Systems Analysis and Design Method (SSADM). Rapid application development (RAD) is often referred as the adaptive software development. RAD is an incremental prototyping approach to software development that end users can produce better feedback when examining a live system, as opposed to working strictly with documentation. It puts less emphasis on planning and more emphasis on an adaptive process.

RAD may resulted in a lower level of rejection when the application is placed into production, but this success most often comes at the expense of a dramatic overruns in project costs and schedule. RAD approach is especially well suited for developing software that is driven by user interface requirements. Thus, some GUI builders are often called rapid application development tools.



Phases RAD model:

1. **Business modeling:** In this phase, the business model is designed on the basis of whatever functions the business has. If we speak in a little technical language, then we design the business model for the product on the basis of flow of information and distribution of information between different business channels. Here information flow means what type of information drives the business, where the information comes from and where it goes, who generates it, etc. This means that a complete business analysis is done in this phase.
2. **Data modeling:** Using the business model we had prepared, the data objects required for the business are defined.
3. **Process modeling:** The data objects that we defined in the data modeling phase are converted to establish the business information flow. It is necessary to achieve specific business objectives.
4. **Application generation:** In this phase we start building the software based on the output of the above three phases. For this we take the help of automation tools. However, in this phase we do not develop the actual software but make a working prototype.
5. **Testing and turnover:** Whatever prototype we have prepared or whatever components and interfaces we have, they are tested in this phase. Since prototypes are tested separately during each iteration, the overall testing time in rapid application development is reduced.

Advantage of RAD Model:

- It reduces the time taken in development.
- In this the components are reused.
- It is flexible and it is easy to make any changes in it.
- It is easy to transfer like scripts because high level abstraction and intermediate codes are used in it.
- There are very few defects in it because it is a prototype by nature.

- In this, productivity can be increased in less time with less people.
- It is cost effective.
- It is suitable for small projects.

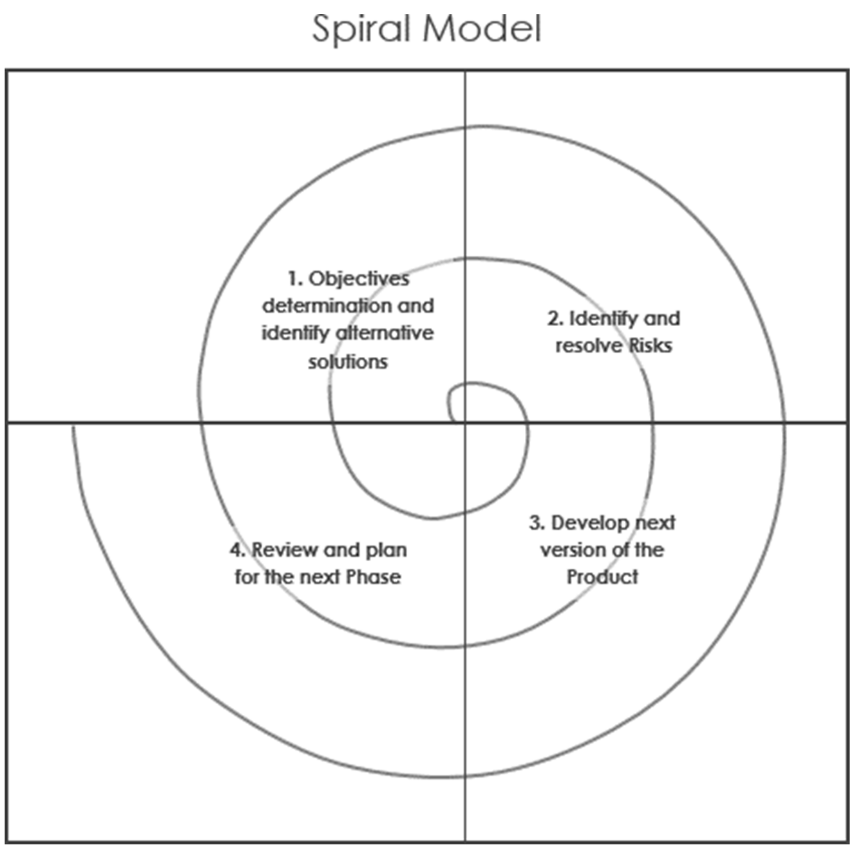
Disadvantages of RAD Model:

- In this we need highly skilled developers and designers.
- It is very difficult to manage.
- It is not suitable for project that are complex and takes long time.
- In this, feedback from the client is required for the development of each phase.
- Automated code generation is very expensive.
- This model is suitable only for component based and scalable systems.

Spiral model:

The spiral model, first described by Barry Boehm in 1986, is a risk-driven software development process model which was introduced for dealing with the shortcomings in the traditional waterfall model. A spiral model looks like a spiral with many loops. The exact number of loops of the spiral is unknown and can vary from project to project. This model supports risk handling, and the project is delivered in loops. Each loop of the spiral is called a Phase of the software development process.

The initial phase of the spiral model in the early stages of Waterfall Life Cycle that is needed to develop a software product. The exact number of phases needed to develop the product can be varied by the project manager depending upon the project risks. As the project manager dynamically determines the number of phases, so the project manager has an important role to develop a product using a spiral model.


Spiral Model
1. Objectives determination and identify alternative solutions
2. Identify and resolve Risks
3. Develop next version of the Product
4. Review and plan for the next Phase

Those phases are as follows:

- **Determining objectives and alternate solutions:** In the first phase, whatever requirements the customer has related to the software are collected. On the basis of which objectives are identified and analyzed and various alternative solutions are proposed.
- **Identifying and resolving risks:** In this phase, all the proposed solutions are assessed and the best solution is selected. Now that solution is analyzed and the risks related to it are identified. Now the identified risks are resolved through some best strategy.
- **Develop and test:** Now the development of software is started. In this phase various features are implemented, that is, their coding is done. Then those features are verified through testing.
- **Review and plan for the next phase:** In this phase the developed version of the software is given to the customer and he evaluates it. Gives his feedback and tells new requirements. Finally planning for the next phase (next spiral) is started.

Advantages of Spiral Model:

- If we have to add additional functionality or make any changes to the software, then through this model we can do so in the later stages also.
- Spiral model is suitable for large and complex projects.
- It is easy to estimate how much the project will cost.
- Risk analysis is done in each phase of this model.
- The customer can see the look of his software only in the early stages of the development process.
- Since continuous feedback is taken from the customer during the development process, the chances of customer satisfaction increases.

Disadvantage of Spiral Model:

- This is the most complex model of SDLC, due to which it is quite difficult to manage.
- This model is not suitable for small projects.
- The cost of this model is quite high.
- It requires more documentation than other models.
- Experienced experts are required to evaluate and review the project from time to time.
- Using this model, the success of the project depends greatly on the risk analysis phase.

Software process activities:

Real software processes are inter-leaved sequences of technical, collaborative and managerial activities with the overall goal of specifying, designing, implementing and testing a software system.

The four basic process activities of specification, development, validation and evolution are organized differently in different development processes. In the waterfall model, they are organized in sequence, whereas in incremental development they are interleaved.

Software specification:

The process of establishing what services are required and the constraints on the system's operation and development.

- Requirements engineering process
- **Feasibility study**: is it technically and financially feasible to build the system?
- Requirements **elicitation and analysis**: what do the system stakeholders require or expect from the system?
- Requirements **specification**: defining the requirements in detail
- Requirements **validation**: checking the validity of the requirements

Software design and implementation:

The process of converting the system specification into an executable system.

- **Software design**: design a software structure that realizes the specification;
- **Implementation**: translate this structure into an executable program;

The activities of design and implementation are closely related and may be interleaved.

Design activities include:

- **Architectural design**: identify the overall structure of the system, the principal components (sometimes called sub-systems or modules), their relationships and how they are distributed.
- **Interface design**: define the interfaces between system components.
- **Component design**: take each system component and design how it will operate.
- **Database design**: design the system data structures and how these are to be represented in a database.

Software validation:

Verification and validation (V & V) are intended to show that a system conforms to its specification and meets the requirements of the system customer.

- **Validation**: are we building the right product (what the customer wants)?
- **Verification**: are we building the product right?

V & V involves checking and review processes and system testing. System testing involves executing the system with test cases that are derived from the specification of the real data to be processed by the system.

Testing is the most commonly used V & V activity and includes the following stages:

- **Development or component testing**: individual components are tested independently; components may be functions or objects or coherent groupings of these entities.
- **System testing**: testing of the system as a whole, testing of emergent properties is particularly important.
- **Acceptance testing**: testing with customer data to check that the system meets the customer's needs.

Software evolution:

Software is inherently flexible and can change. As requirements change through changing business circumstances, the software that supports the business must also evolve and change. Although there has been a demarcation between development and evolution (maintenance) this is increasingly irrelevant as fewer and fewer systems are completely new.

Coping with change:

Change is inevitable in all large software projects. Business changes lead to new and changed system requirements new technologies open up new possibilities for improving implementations. Changing platforms require application changes. Change leads to rework so the costs of change include both rework (e.g. re-analyzing requirements) as well as the costs of implementing new functionality.

Two strategies to **reduce the costs** of rework:

1. **Change avoidance**
   a. The software process includes activities that can anticipate possible changes before significant rework is required. For example, a prototype system may be developed to show some key features of the system to customers.
2. **Change tolerance**

   The process is designed so that changes can be accommodated at relatively low cost. This normally involves some form of incremental development. Proposed changes may be implemented in increments that have not yet been developed. If this is impossible, then only a single increment (a small part of the system) may have been altered to incorporate the change.

Software prototyping:

A prototype is an initial version of a system used to demonstrate concepts and try out design options. A prototype can be used in:

- The requirements engineering process to help with requirements elicitation and validation;
- In design processes to explore options and develop a UI design;
- In the testing process to run back-to-back tests.

Benefits of prototyping:

- Improved system usability.
- A closer match to users' real needs.
- Improved design quality.
- Improved maintainability.
- Reduced development effort.

Prototypes may be based on rapid prototyping languages or tools. They may involve **leaving out functionality**:

- Prototype should focus on areas of the product that are not well-understood;
- Error checking and recovery may not be included in the prototype;
- Focus on functional rather than non-functional requirements such as reliability and security.

Prototypes should be **discarded** after development as they are not a good basis for a production system:

- It may be impossible to tune the system to meet non-functional requirements;
- Prototypes are normally undocumented;
- The prototype structure is usually degraded through rapid change;
- The prototype probably will not meet normal organizational quality standards.

Key Process Activities:

There four basic key process activities are:

1. **Software Specifications:** In this process, detailed description of a software system to be developed with its functional and non-functional requirements.
2. **Software Development:** In this process, designing, programming, documenting, testing and bug fixing is done.
3. **Software Validation:** In this process, evaluation software product is done to ensure that the software meets the business requirements as well as the end user’s needs.
4. **Software Evolution:** It is a process of developing software initially, then timely updating it for various reasons.

Agile Software Development:

The Agile Software Development Process:

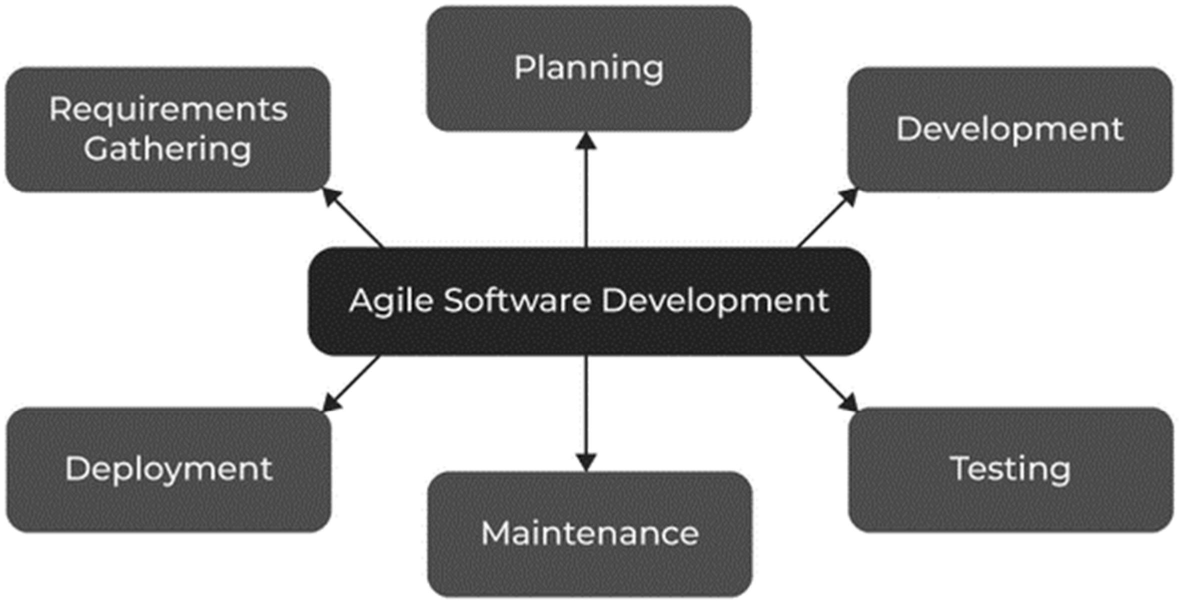


Fig: Agile Software Development

1. **Requirements Gathering:** The customer's requirements for the software are gathered and prioritized.
2. **Planning:** The development team creates a plan for delivering the software, including the features that will be delivered in each iteration.
3. **Development:** The development team works to build the software, using frequent and rapid iterations.
4. **Testing:** The software is thoroughly tested to ensure that it meets the customer's requirements and is of high quality.
5. **Deployment:** The software is deployed and put into use.
6. **Maintenance:** The software is maintained to ensure that it continues to meet the customer's needs and expectations.

Agile Software Development is widely used by software development teams and is considered to be a flexible and adaptable approach to software development that is well-suited to changing requirements and the fast pace of software development.

Agile is a time-bound, iterative approach to software delivery that builds software incrementally from the start of the project, instead of trying to deliver all at once.

Agile Software development cycle:

Let's see a brief overview of how development occurs in Agile philosophy.

1. concept

2. inception
3. iteration/construction
4. release
5. production
6. retirement

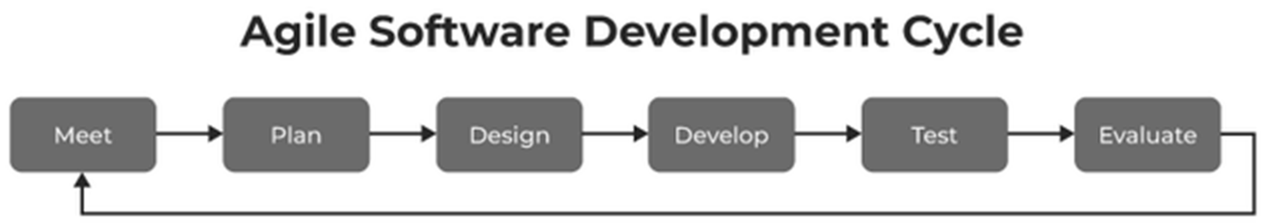


Fig: Agile software development cycle:

- **Step** 1: In the first step, concept and business opportunities in each possible project are identified and the amount of time and work needed to complete the project is estimated. Based on their technical and financial viability, projects can then be prioritized and determined which ones are worthwhile pursuing.
- **Step** 2: In the second phase, known as inception, the customer is consulted regarding the initial requirements, team members are selected, and funding is secured. Additionally, a schedule outlining each team's responsibilities and the precise time at which each sprint's work is expected to be finished should be developed.
- **Step** 3: Teams begin building functional software in the third step, iteration/construction, based on requirements and ongoing feedback. Iterations, also known as single development cycles, are the foundation of the Agile software development cycle.

<u>Design Process of Agile software Development:</u>

- In Agile development, Design and Implementation are considered to be the central activities in the software process.
- The design and Implementation phase also incorporates other activities such as requirements elicitation and testing.
- In an agile approach, iteration occurs across activities. Therefore, the requirements and the design are developed together, rather than separately.
- The allocation of requirements and the design planning and development as executed in a series of increments. In contrast with the conventional model, where requirements gathering needs to be completed to proceed to the design and development phase, it gives Agile development an extra level of flexibility.
- An agile process focuses more on code development rather than documentation.

Advantages Agile Software Development:

- Deployment of software is quicker and thus helps in increasing the trust of the customer.
- Can better adapt to rapidly changing requirements and respond faster.
- Helps in getting immediate feedback which can be used to improve the software in the next increment.
- People – Not Process. People and interactions are given a higher priority than processes and tools.
- Continuous attention to technical excellence and good design.
- **Increased collaboration and communication:** Agile Software Development Methodology emphasize collaboration and communication among team members, stakeholders, and customers. This leads to improved understanding, better alignment, and increased buy-in from everyone involved.
- **Flexibility and adaptability:** Agile methodologies are designed to be flexible and adaptable, making it easier to respond to changes in requirements, priorities, or market conditions. This allows teams to quickly adjust their approach and stay focused on delivering value.
- **Improved quality and reliability:** Agile methodologies place a strong emphasis on testing, quality assurance, and continuous improvement. This helps to ensure that software is delivered with high quality and reliability, reducing the risk of defects or issues that can impact the user experience.
- **Enhanced customer satisfaction:** Agile methodologies prioritize customer satisfaction and focus on delivering value to the customer. By involving customers throughout the development process, teams can ensure that the software meets their needs and expectations.
- **Increased team morale and motivation:** Agile methodologies promote a collaborative, supportive, and positive work environment. This can lead to increased team morale, motivation, and engagement, which can in turn lead to better productivity, higher quality work and improved outcomes.

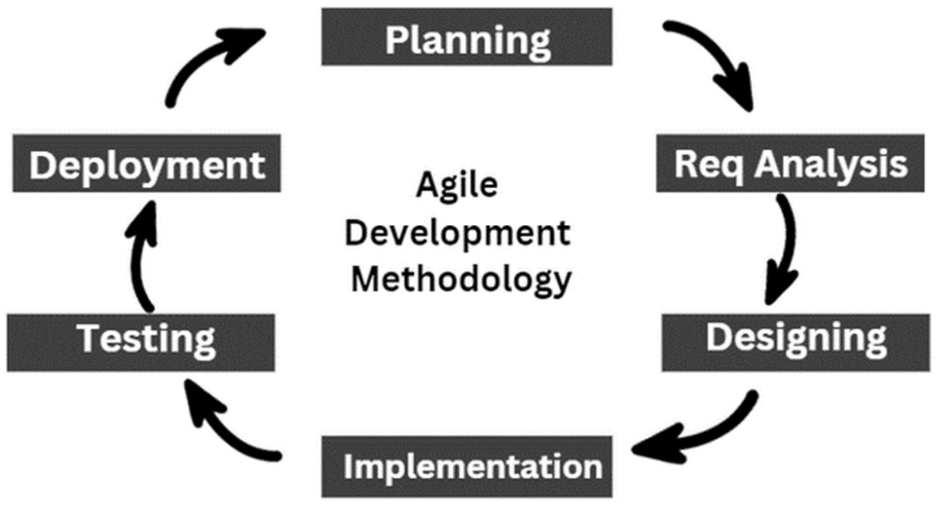
Disadvantages Agile Software Development:

- In the case of large software projects, it is difficult to assess the effort required at the initial stages of the software development life cycle.
- Agile Development is more code-focused and produces less documentation.
- Agile development is heavily dependent on the inputs of the customer. If the customer has ambiguity in his vision of the outcome, it is highly likely that the project to get off track.
- Face-to-face communication is harder in large-scale organizations.

- Only senior programmers are capable of making the kind of decisions required during the development process. Hence, it’s a difficult situation for new programmers to adapt to the environment.
- **Lack of predictability:** Agile Development relies heavily on customer feedback and continuous iteration, which can make it difficult to predict project outcomes, timelines and budgets.
- Limited scope control: Agile Development is designed to be flexible and adaptable, which means that scope changes can be easily accommodated. However, this can also lead to scope creep and a lack of control over the project scope.
- **Lack of emphasis on testing:** Agile Development places a greater emphasis on delivering working code quickly, which can lead to a lack of focus on testing and quality assurance. This can result in bugs and other issues that may go undetected until later stages of the project.
- **Risk of team burnout:** Agile Development can be intense and fast-paced, with frequent sprints and deadlines. This can put a lot of pressure on team members and lead to burnout, especially if the team is not given adequate time for rest and recovery.
- **Lack of structure and governance:** Agile Development is often less formal and structured than other development methodologies, which can lead to a lack of governance and oversight. This can result in inconsistent processes and practices, which can impact project quality and outcomes.

Life cycle of Agile Methodology:

The Agile software development life cycle helps you break down each project you take on into six simple stages:



1. Requirement Gathering:

- In this stage, the project team identifies and documents the needs and expectations of various stakeholders, including clients, users, and subject matter experts.
- It involves defining the project's scope, objectives and requirements.
- Establishing a budget and schedule.
- Creating a project plan and allocating resources.

2. Design:

- Developing a high-level system architecture.
- Creating detailed specifications, which include data structures, algorithms, and interfaces.
- Planning for the software's user interface.

3. Development (Coding):

Writing the actual code for the software. Conducting unit testing to verify the functionality of individual components.

4. Testing:

This phase involves several types of testing:

1. **Integration Testing:** Ensuring that different components work together.
2. **System Testing:** Testing the entire system as a whole.
3. **User Acceptance Testing:** Confirming that the software meets user requirements.
4. **Performance Testing:** Assessing the system's speed, scalability and stability.

5. Deployment:

1. Deploying the software to a production environment.
2. Put the software into the real world where people can use it.
3. Make sure it works smoothly in the real world.
4. Providing training and support for end-users.

6. Review (Maintenance):

1. Addressing and resolving any issues that may arise after deployment.
2. Releasing updates and patches to enhance the software and address problems.

When to use the Agile Methodology?

If you want to know when to use the Agile methodology, then it is particularly well-suited for projects and organizations where the following conditions or needs are present:

1. **Unclear or Changing Requirements:** Agile is great for projects with requirements that aren't well-defined or might change.
2. **Complex Projects:** It's good for big, complex projects by breaking them into smaller pieces.
3. **Customer Focus:** Use Agile when making customers happy is a priority and you want to involve them throughout.

4. **Quick Time-to-Market:** If you need to get your product out fast, Agile can help.
5. **Small to Medium Teams:** Agile works well for teams of a few to a few dozen people.
6. **Team Skills:** It's best when you have a mix of skills in your team, like development, testing, design and more.
7. **Collaboration:** Agile promotes working together and open communication.
8. **Regular Updates:** If you want to check progress often and make changes as needed.
9. **Transparency:** Agile emphasizes being open and clear with everyone involved in the project.
10. **Risk Control:** It helps manage risks by tackling issues as they come up.
11. **Innovation:** If you encourage trying new things and learning from experience, Agile supports that.
12. **Continuous Improvement:** Agile fosters a culture of always getting better over time.

Benefits of Agile Methodology:

The advantages of the agile model are as follows:

1. **Immediate Feedback:** It allows immediate feedback, which aids software improvement in the next increment.
2. **Adapts to Changing Requirements:** It is a highly adaptable methodology in which rapidly changing requirements, allowing responsive adjustments.
3. **Face-to-Face Communication:** Agile methodology encourages effective face-to-face communication.
4. **Time-Efficient:** It is well-suited for its time-efficient practices, which help in delivering software quickly and reducing time-to-market.
5. **Frequent Changes:** It effectively manages and accommodates frequent changes in project requirements according to stakeholder convenience.
6. **Customer Satisfaction:** It prioritizes customer satisfaction.
7. **Flexibility and Adaptability:** Agile methodologies are known for their flexibility and adaptability.

Limitations of Agile Methodology:

The disadvantages of the agile model are as follows:

1. **Less Documentation:** Agile methodologies focus on less documentation; it prioritizes working on projects rather than paperwork.
2. **Challenges in Large Organizations:** Busy schedule of clients can make daily meetup and face-to-face communication difficult.
3. **Need for Senior Programmers:** It may require experienced programmers to make critical decisions during the development of software.

4. **Limited Scope Control:** It has less rigid scope control, which may not be suitable in certain situations.
5. **Predictability**: Compared to more structured project management methods, it may lack predictability.

Different Types of Agile Methods or Frameworks:

1. Kanban:

Kanban methodology is about day-to-day workflows and processes. It is a simple, visual means of managing projects that enables teams to see the progress so far and what's coming up next. Kanban projects are primarily managed through a Kanban board, which segments tasks into three columns: "To Do" "Doing" and "Done."

Many project management software solutions have Kanban board integration, which teams can use to plot out a task's progress from start to finish.

The main benefit of this methodology is the increased transparency, allowing team leaders to clearly see which tasks are assigned to which team members and what is yet to be completed to make meaningful progress.

2.Scrum:

Scrum is one of the most popular Agile methodologies, as it can bring teams together with a sharp focus and an efficient, collaborative approach to task execution.

It is similar to Kanban in many ways. Scrum typically uses a Scrum board, similar to a Kanban board, and groups tasks into columns based on progress. Unlike Kanban, Scrum focuses on breaking a project down into sprints and only planning and managing one sprint at a time.

Under this methodology, you would assign a Scrum leader and product owner, who have a direct influence over the rest of the team. This assigned leadership can help teams to spearhead projects and complete them over a short space of time through a series of sprints.

Sprints are the Scrum way of breaking projects down into iterations that can last anywhere between one and four weeks each. Bringing team members together from different departments, these sprints help you channel a collective focus to your projects.

Scrum also features a robust set of principles and activities that dictate how you work. These include:

- **Sprint planning:** Planning sessions to identify the purpose behind your sprints
- **Roles:** Key roles in the Scrum project management process
- **Product backlog:** A list of tasks arranged according to priority level

3.Extreme Programming (XP):

Extreme Programming (XP) was designed for Agile software development projects. It focuses on continuous development and customer delivery and uses intervals or sprints, similar to a Scrum methodology. However, XP also has 12 supporting processes specific to the world of software development:

- Planning game
- Small releases
- Customer acceptance tests
- Simple design
- Pair programming
- Test-driven development
- Refactoring
- Continuous integration
- Collective code ownership
- Coding standards
- Metaphor
- Sustainable pace

Extreme Programming puts the customer at the center of everything the team does. In XP, teams can involve the end users in a meaningful way and use the feedback they get to deliver the best possible product.

4.Feature-driven development (FDD):

Feature-driven development is another software-specific Agile framework. This methodology involves creating software models every two weeks and requires a development and design plan for every model feature. It has more rigorous documentation requirements than XP, so it's better for teams with advanced design and planning abilities. FDD breaks projects down into five basic activities:

- Develop an overall model
- Build a feature list
- Plan by feature
- Design by feature
- Build by feature

5.Dynamic Systems Development Method (DSDM):

The Dynamic Systems Development Method (DSDM) was born of the need for a common industry framework for rapid software delivery. Rework is to be expected, and any development changes that occur must be reversible. Like Scrum, XP and FDD, DSDM uses sprints. This framework is based on eight fundamental principles:

- Focus on the business need
- Deliver on time
- Collaborate

- Never compromise quality
- Build incrementally from firm foundations
- Develop iteratively
- Communicate continuously and clearly
- Demonstrate control

6.Crystal:

Crystal is a family of Agile methodologies that includes Crystal Clear, Crystal Yellow, Crystal Orange, Crystal Red, etc. Each has a unique framework. Your choice depends on several project factors, such as your team size, priorities and project criticality.

7.Lean:

Lean development is often grouped with Agile, but it’s an entirely different methodology that happens to share many of the same values. The main principles of the Lean methodology include:

- Eliminating waste
- Build quality in
- Create knowledge
- Defer commitment
- Deliver fast
- Respect people
- Optimize the whole

How is plan-driven development different from agile development?

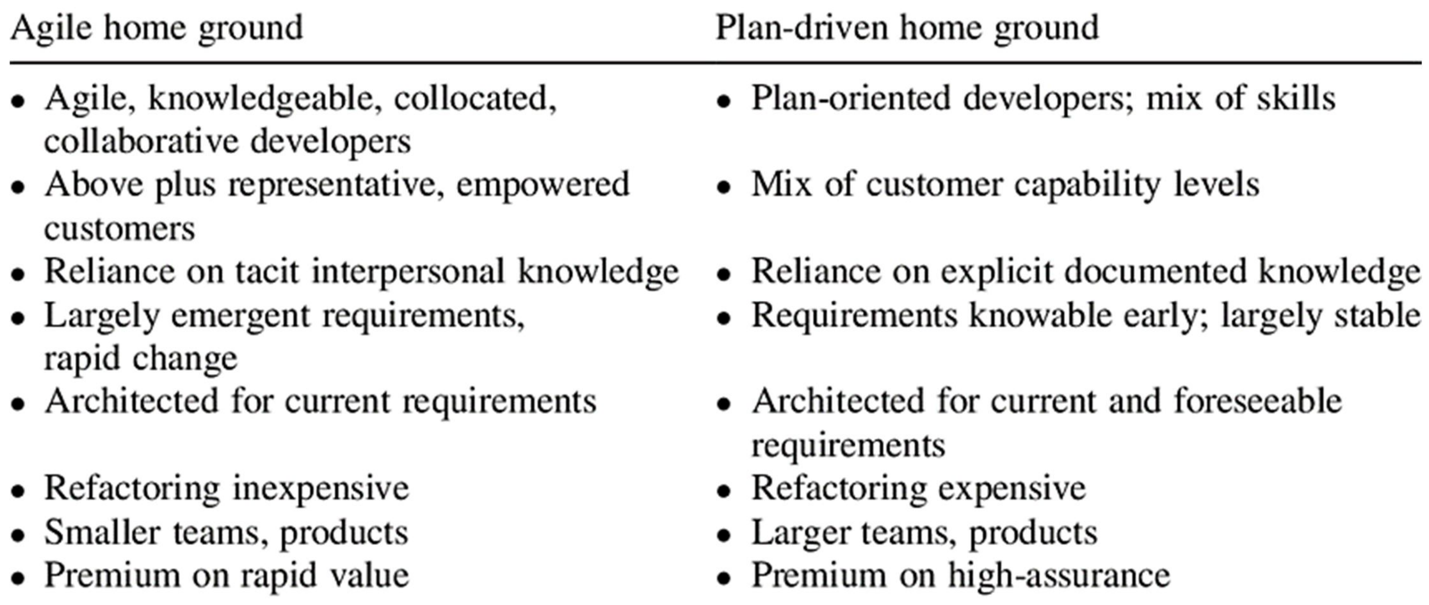
A **plan-driven development is** a process where all the activities are planned in advance, and we measure the progress in advance against that plan. So, as it appears from the term ‘plan-driven development, it is always planned and the results are to be shown at the end of the product.

While in **agile development,** planning is always incremental, where you can change the plan according to your customer requirements. So, as the term ‘agile development’ suggests, it is the process-based method where one can alter the way things and planning and replace them with the more suitable ones.

Which one is the best?

Well, both have different approaches. They are used with the feasibility and requirements of the project, and implementation is based on what the project needs.

To uncover the more detailed differences between plan-driven development and agile development, let’s study their definitions and more details with different approaches.



Agile home ground	Plan-driven home ground
• Agile, knowledgeable, collocated, collaborative developers	• Plan-oriented developers; mix of skills
• Above plus representative, empowered customers	• Mix of customer capability levels
• Reliance on tacit interpersonal knowledge	• Reliance on explicit documented knowledge
• Largely emergent requirements, rapid change	• Requirements knowable early; largely stable
• Architected for current requirements	• Architected for current and foreseeable requirements
• Refactoring inexpensive	• Refactoring expensive
• Smaller teams, products	• Larger teams, products
• Premium on rapid value	• Premium on high-assurance

Plan-Driven Development:

It is a software development method. When a user wants to develop all the features that should be developed in a final product, this plan plays its role. It also determines how all those features should be developed. So, you can say that this is an ACTION plan based on executing an arranged set of task-specified levels.

It is basically an attitude to software engineering where it locates how the developmental procedure will occur. This process handles mega project engineering and administers the old style of methodology to tackle the software development ventures. Also, management planning is unfolded to measure progress and make project decisions.

Planning of Projects:

In software engineering, you can say that it is an attitude of software engineering to feature the development process and where it is taking place.

So, a plan-driven development project focus on setting out a program for any project that includes the available resources, work completion scheduling, work breakdowns, etc. Below are the more detailed checklists.

- To organize the team and the roles or team members.
- Probabilities and analysis of risk, strategies of arising and prospects
- Hardware and software requirements of the resources.
- To decide the milestones, deliveries, and activities
- To allow the location to people
- To monitor and report the system

Characteristics of Plan Driven Development:

- You can focus on the repeatability and predictability

- It is defined, as an incremental improving process, and a standardized way
- It is done through a proper documentation process
- There is a software system architecture that has been defined up-front
- You can find detailed plans, roles, work-product descriptions, workflow, and responsibilities
- It contains resources for specialists: process monitoring, controlling, and educating
- It Focuses on verification and validation

Following are the methodologies of the plan-driven method:

1. Personal Software Process (PSP)
2. Team Software Process (TSP, TSP)
3. Rational Unified Process (RUP)

To understand the term more, let’s uncover the phases of the plan-driven method:

Inception Phase –In this phase, we decide what to do and the scope of the project. An initial project plan has been made with the requirement of rough estimations of time and resources. Risks are defined and handled in the elaboration phase.

Elaboration phase – This phase analyzes the problem domain and defines a technically reasonable architecture. Lessen the most significant risks to the projects. Make a comprehensive project plan with highlighted activities.

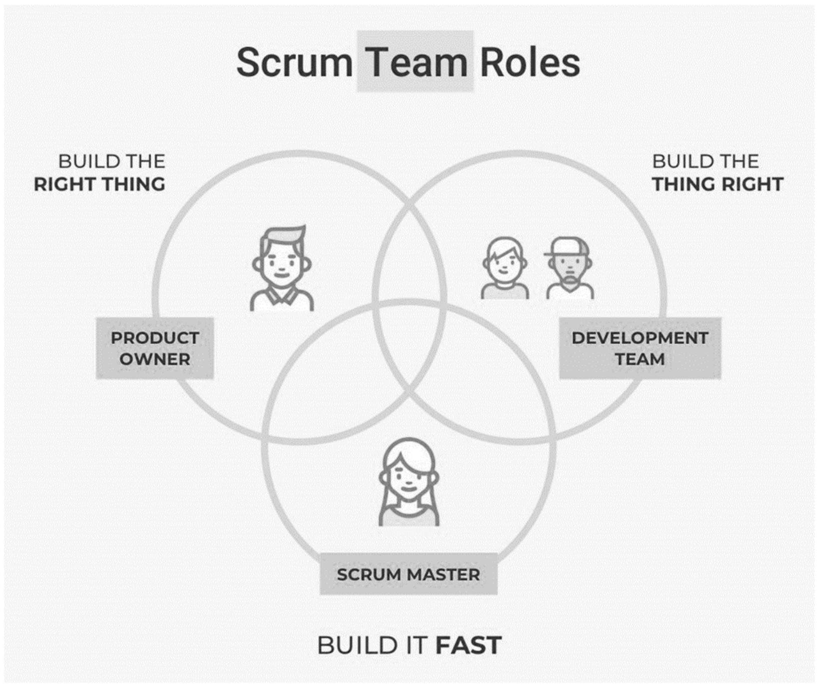
Construction phase – In this phase, the product’s development, integration, and testation are defined. It also optimizes the resources so that they can work out equally and recycle each other’s work.

Transition phase – Deliver the product to the customers and sustain it.

Agile Software Development:

Now, let’s study the agile software in detail so that you can comprehend both terminologies well.

In agile software development, software development methodologies centers around the iterative developmental idea, in which needs and solutions revolve through an alliance between self-organizing cross-functional teams. It is also crucial to note that agile development enables teams to deliver value faster with excellent quality, prediction, and aptitude to respond to change. There are two most widely used methods called Scrum and Kanban.



Below you can find the most frequently asked questions about Agile and Scrum.

What is Agile?

- Agile software development belongs to a group of software development methodologies centered on iterative aspect development, in which requirements and solutions progress through cooperation between self-organizing cross-functional teams.
- Agile processes usually promote a well-disciplined project management process. It also boosts frequent checks and variation, a leadership viewpoint that inspires teamwork, self-organization, and liability.
- Agile development denotes any development method associated with the ideas of the Agile Manifesto. Fourteen leading figures in the software industry developed this Manifesto and revealed their experience of what approaches should and should not work for software development.

What is Scrum?

Scrum is a subgroup of Agile. It is a trivial process framework for agile development and the most widely-used one.

What are the benefits of agile development?

By reading about the benefits of agile, you can completely comprehend the difference between the two approaches.

- More responsive to requests
- High-value features
- Delivered more quickly with short cycles

Development team

- ✓ Enjoy development work is valued and used
- ✓ Reduced non-productive work

Scrum Master

- ▪ Planning tracking in daily meetings
- ▪ Tremendous awareness of the project state
- ▪ Catching and addressing issues quickly

Vendor

- Focused development on high-value features
- Increased efficiently
- Reduce wastage and decrease overhead

Product Owner

- ❖ Development work aligns with customer needs
- ❖ Frequent opportunities to re-prioritize work
- ❖ Maximum delivery value

Requirements Engineering:

Requirements Engineering is the process of identifying, eliciting, analyzing, specifying, validating, and managing the needs and expectations of stakeholders for a software system.

A systematic and strict approach to the definition, creation and verification of requirements for a software system is known as requirements engineering. To guarantee the effective creation of a software product, the requirements engineering process entails several tasks that help in understanding, recording and managing the demands of stakeholders.

Functional vs Non-Functional Requirements:

Requirements analysis is a very critical process that enables the success of a system or software project to be assessed. Requirements are generally split into two types: Functional and Non-functional requirements.

Functional Requirements:

These are the requirements that the end user specifically demands as basic facilities that the system should offer. All these functionalities need to be necessarily incorporated into the system as a part of the contract.

These are represented or stated in the form of input to be given to the system, the operation performed and the output expected. They are the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements.

Example:

- What are the features that we need to design for this system?
- What are the edge cases we need to consider, if any, in our design?

Non-Functional Requirements:

These are the quality constraints that the system must satisfy according to the project contract. The priority or extent to which these factors are implemented varies from one project to another. They are also called non-behavioral requirements. They deal with issues like:

1. Portability
2. Security
3. Maintainability
4. Reliability
5. Scalability
6. Performance
7. Reusability
8. Flexibility

Example:

- Each request should be processed with the minimum latency?
- System should be highly valuable.

Extended Requirements:

These are basically “nice to have” requirements that might be out of the scope of the System.

Example:

- ✓ Our system should record metrices and analytics.
- ✓ Service heath and performance monitoring.

Difference between Functional Requirements and Non-Functional Requirements:

Functional Requirements	**Non-Functional Requirements**
A functional requirement defines a system or its component.	A non-functional requirement defines the quality attribute of a software system.
It specifies “What should the software system do?”	It places constraints on “How should the software system fulfill the functional requirements?”
Functional requirement is specified by User.	Non-functional requirement is specified by technical peoples e.g. Architect, Technical leaders and software developers.
It is mandatory.	It is not mandatory.
It is captured in use case.	It is captured as a quality attribute.

Defined at a component level.	Applied to a system as a whole.
Helps you verify the functionality of the software.	Helps you to verify the performance of the software.
Functional Testing like System, Integration, End to End, API testing, etc are done.	Non-Functional Testing like Performance, Stress, Usability, Security testing, etc are done.
Usually easy to define.	Usually more difficult to define.
Example: 1) Authentication of user whenever he/she logs into the system. 2) System shutdown in case of a cyber-attack. 3) A Verification email is sent to user whenever he/she registers for the first time on some software system.	**Example**: 1) Emails should be sent with a latency of no greater than 12 hours from such an activity. 2) The processing of each request should be done within 10 seconds 3) The site should load in 3 seconds when the number of simultaneous users are > 10000

Software Requirement Specification (SRS) Format or Document:

Software Requirement Specification (SRS) Format as the name suggests, is a complete specification and description of requirements of the software that need to be fulfilled for the successful development of the software system. These requirements can be functional as well as non-functional depending upon the type of requirement. The interaction between different customers and contractors is done because it is necessary to fully understand the needs of customers.

Depending upon information gathered after interaction, SRS is developed which describes requirements of software that may include changes and modifications that is needed to be done to increase quality of product and to satisfy customer's demand.

Introduction:

- **Purpose of this Document –** At first, main aim of why this document is necessary and what's purpose of document is explained and described.
- **Scope of this document –** In this, overall working and main objective of document and what value it will provide to customer is described and explained. It also includes a description of development cost and time required.
- **Overview –** In this, description of product is explained. It's simply summary or overall review of product.

General description:

In this, general functions of product which includes objective of user, a user characteristic, features, benefits, about why its importance is mentioned. It also describes features of user community.

Functional Requirements:

In this, possible outcome of software system which includes effects due to operation of program is fully explained. All functional requirements which may include calculations, data processing, etc. are placed in a ranked order. Functional requirements specify the expected behavior of the system-which outputs should be produced from the given inputs. They describe the relationship between the input and output of the system. For each functional requirement, detailed description all the data inputs and their source, the units of measure and the range of valid inputs must be specified.

Interface Requirements:

In this, software interfaces which mean how software program communicates with each other or users either in form of any language, code, or message are fully described and explained. Examples can be shared memory, data streams, etc.

Performance Requirements:

In this, how a software system performs desired functions under specific condition is explained. It also explains required time, required memory, maximum error rate, etc. The performance requirements part of an SRS specifies the performance constraints on the software system. All the requirements relating to the performance characteristics of the system must be clearly specified. There are two types of performance requirements: static and dynamic. Static requirements are those that do not impose constraint on the execution characteristics of the system. Dynamic requirements specify constraints on the execution behaviour of the system.

Design Constraints:

In this, constraints which simply means limitation or restriction are specified and explained for design team. Examples may include use of a particular algorithm, hardware and software limitations, etc. There are a number of factors in the client’s environment that may restrict the choices of a designer leading to design constraints such factors include standards that must be followed resource limits, operating environment, reliability and security requirements and policies that may have an impact on the design of the system. An SRS should identify and specify all such constraints.

Non-Functional Attributes:

In this, non-functional attributes are explained that are required by software system for better performance. An example may include Security, Portability, Reliability, Reusability, Application compatibility, Data integrity, Scalability capacity, etc.

Preliminary Schedule and Budget:

In this, initial version and budget of project plan are explained which include overall time duration required and overall cost required for development of project.

Appendices:

In this, additional information like references from where information is gathered, definitions of some specific terms, acronyms, abbreviations, etc. are given and explained.

Uses of SRS document:

- Development team require it for developing product according to the need.
- Test plans are generated by testing group based on the describe external behaviour.
- Maintenance and support staff need it to understand what the software product is supposed to do.
- Project manager base their plans and estimates of schedule, effort and resources on it.
- customer rely on it to know that product they can expect.
- As a contract between developer and customer.
- in documentation purpose.

Characteristics of good SRS:

Following are the features of a good SRS document:

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

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

(1). All essential requirements, whether relating to functionality, performance, design, constraints, attributes or external interfaces.

(2). Definition of their responses of the software to all realizable classes of input data in all available categories of situations.

(3). Full labels and references to all figures, tables and diagrams in the SRS and definitions of all terms and units of measure.

3. Consistency: The SRS is consistent if, and only if, no subset of individual requirements described in its conflict. There are three types of possible conflict in the SRS:

(1). The specified characteristics of real-world objects may conflict. For example,

(a) The format of an output report may be described in one requirement as tabular but in another as textual.

(b) One condition may state that all lights shall be green while another states that all lights shall be blue.

(2). There may be a reasonable or temporal conflict between the two specified actions. For example,

(a) One requirement may determine that the program will add two inputs and another may determine that the program will multiply them.

(b) One condition may state that "A" must always follow "B," while other requires that "A and B" co-occurs.

(3). Two or more requirements may define the same real-world object but use different terms for that object. For example, a program's request for user input may be called a "prompt" in one requirement and a "cue" in another. The use of standard terminology and descriptions promotes consistency.

4. Unambiguousness: SRS is unambiguous when every fixed requirement has only one interpretation. This suggests that each element is uniquely interpreted. In case there is a method used with multiple definitions, the requirements report should determine the implications in the SRS so that it is clear and simple to understand.

5. Ranking for importance and stability: The SRS is ranked for importance and stability if each requirement in it has an identifier to indicate either the significance or stability of that particular requirement.

Typically, all requirements are not equally important. Some prerequisites may be essential, especially for life-critical applications, while others may be desirable. Each element should be identified to make these differences clear and explicit. Another way to rank requirements is to distinguish classes of items as essential, conditional and optional.

6. Modifiability: SRS should be made as modifiable as likely and should be capable of quickly obtain changes to the system to some extent. Modifications should be perfectly indexed and cross-referenced.

7. Verifiability: SRS is correct when the specified requirements can be verified with a cost-effective system to check whether the final software meets those requirements. The requirements are verified with the help of reviews.

8. Traceability: The SRS is traceable if the origin of each of the requirements is clear and if it facilitates the referencing of each condition in future development or enhancement documentation.

There are two types of Traceability:

1. Backward Traceability: This depends upon each requirement explicitly referencing its source in earlier documents.

2. Forward Traceability: This depends upon each element in the SRS having a unique name or reference number.

The forward traceability of the SRS is especially crucial when the software product enters the operation and maintenance phase. As code and design document is modified, it is necessary to be able to ascertain the complete set of requirements that may be concerned by those modifications.

9. Design Independence: There should be an option to select from multiple design alternatives for the final system. More specifically, the SRS should not contain any implementation details.

10. Testability: An SRS should be written in such a method that it is simple to generate test cases and test plans from the report.

11. Understandable by the customer: An end user may be an expert in his/her explicit domain but might not be trained in computer science. Hence, the purpose of formal notations and symbols should be avoided too as much extent as possible. The language should be kept simple and clear.

12. The right level of abstraction: If the SRS is written for the requirements stage, the details should be explained explicitly. Whereas, for a feasibility study, fewer analysis can be used. Hence, the level of abstraction modifies according to the objective of the SRS.

<u>Requirements Specification:</u>

Requirements specification is a critical part of the Requirements Engineering process. It is the third phase, after Requirements Capture and Analysis. The goal is to create a document, or Requirements Specification, with the corresponding level of detail. This document will contain all requirements that are to be imposed on the design and verification of the product. It will also contain other related information necessary for the design, verification, and maintenance of the product.

What is Requirements Specification?

Requirement specification, also known as documentation, is a process of jotting down all the system and user requirements in the form of a document. These requirements must be clear, complete, comprehensive and consistent.

During the capturing activity, we gather all the requirements from various sources. During the analysis and negotiation activities, we analyze and understand those requirements. Now, we must prepare a formal document explaining those requirements. That is what the requirement specification is. To be precise, it is the process of documenting all the user and system needs and constraints in a clear and accurate manner.

What is a System Requirement?

System requirements can be called the expanded version of the user requirements. System requirements act as the commencement point for any new system design. These requirements are a detailed description of the user requirements the system must satisfy.

What is a User Requirement?

User requirement is a combination of functional and non-functional requirements. These user requirements must be designed in such a way that they are easily understandable by users who do not have any sort of technical knowledge. Hence, they must be written in natural language using simple tables, forms, and diagrams. Also, make sure the document does not have details on system design, software, or formal notations.

What are Functional and Non-Functional Requirements?

Functional requirements, as the name suggests, describe the functions of the system to be designed. It is a description of what the system will be and how it will function to satisfy user needs. They provide a clear description of how the system is supposed to respond to a particular command, the features, and what the users expect.

Non-functional requirements explain the limitations and constraints of the system to be designed. These requirements do not have any impact on the functionality of the application. Furthermore, there is a common practice of sub-classifying the non-functional requirements into various categories like

- User Interface
- Reliability
- Security
- Performance
- Maintenance
- Standards

Sub-classifying the non-functional requirements is a good practice. It helps when creating a checklist of the requirements that are to be met in the system to be designed.

Non-functional requirements are as important as functional requirements are. If functional requirements specify what a system should do, non-functional requirements describe how the system will do it. For example, the new application shall provide us with the final list of all connected users. That is a part of functional requirements. If the requirement says that the system would only work on a Windows and a Linux system, that would be a part of non-functional requirements.

The only difference between the two is that the system cannot function without satisfying all the functional requirements. On the other hand, the system will give you the desired outcome even when it does not satisfy the non-functional requirements.

What are The Benefits of having a Requirements Specification?

There are many benefits of having a requirements specification. Some of them are listed below:

- Helps to ensure that all stakeholders have a common understanding of the system that is to be developed. This avoids any misunderstanding during later stages of development.
- Serves as a reference point for all stakeholders during the development process.
- Helps to identify any gaps in the requirements at an early stage.
- Reduces the overall cost and time of development as it avoids rework due to changes in requirements.

Standards for writing requirements?

EARS would be an effective methodology here. It stands for Easy Approach to Requirements Syntax. In this method, we write clear, concise, and understandable language. This improves the whole requirements engineering workflow and simplifies the work by making things pretty easy to understand.

To achieve this, here are some principles that must be kept in mind while writing the requirements. They involve:

- Each requirement must be in the form of a complete sentence. No bullet, acronyms, abbreviations, or buzzwords should be used. Try to make short, direct, and complete sentences.
- Make sure that each requirement has a proper subject, predicate, and verb. The subject would be the user type or the system that we are talking about. The predicate would be the conditions or actions or desired results we expect. We must use words like ‘shall’, ‘will’, and ‘must’ to express some kind of necessity, and words like ‘may’ to express optionality in the requirement.
- Each requirement must efficiently explain the end result we desire from the system.
- Also, the requirement must describe the quality we expect from the system. It helps when we measure the end result and see if the requirement is properly implemented or not.

Types of Requirements Specifications:

There are numerous sorts of requirements specifications. They include Functional Requirement Specifications (FRS), Performance Requirement Specification (PRS), Configurations Requirement Specification (CRF), Business Requirement Specification (BRS), Reliability Requirement Specification (RRF), Compatibility Requirement Specification (CRF) and Software Requirement Specification (SRS).

Functional Requirement Specifications: A functional requirement specification (FRS) is a document that captures the functions that a system must perform. It includes all functionalities, premises, security measures, and other relevant information. Simply put, an FRS is a document that contains everything that a particular system should do.

Performance Requirement Specifications: A performance requirement specification (PRS) is a document that captures all the performance-related aspects of a system. This includes response time, data throughput, efficiency, scalability, etc. Basically, anything that can be quantified and improved upon falls under the PRS category.

Configurations Requirement Specification: A configuration requirement specification (CRS) is a document that captures all information related to the configuration of a system. This includes details such as supported platforms, software/hardware dependencies, minimum system requirements, etc.

Business Requirement Specifications: A business requirement specification (BRS) is a document that captures all business-related aspects of a system. This includes features such as user management, security, data integrity, etc. Basically, anything that affects the business operations of a system falls under the BRS category.

Reliability Requirement Specifications: A reliability requirement specification (RRF) is a document that captures all information related to the reliability of a system. This includes aspects such as uptime, recovery time, error rates, etc.

Compatibility Requirement Specifications: A compatibility requirement specification (CRF) is a document that captures all information related to the compatibility of a system. This includes aspects such as supported platforms, software/hardware dependencies, minimum system requirements, etc.

Software Requirement Specifications: A software requirement specification (SRS) is a document that captures all software-related aspects of a system. This includes aspects such as functionality, performance, scalability, etc. Basically, anything that affects the software operations of a system falls under the SRS category.

<u>Software Requirement Specification Vs Business Requirement Specification:</u>

People sometimes mix the concepts of software and business requirement specifications. Actually, they both are quite different.

The main difference between software requirement specification and business requirement specification is that the former captures all information related to the software while the latter captures all information related to the business.

Business Requirements Specification (BRS)	**Software Requirements Specification (SRS)**
It is a formal document that describes the various requirements provided by the client/stakeholders.	It specifies the functional and non-functional requirements that are present in the software.
It is derived from the client's requirements and interactions.	It is derived from the Business Requirements Specification (BRS).
It is created by a business analyst.	It is created by a system analyst or a system architect or a business analyst.
It describes the functional specifications of the software at a very high level.	It describes both the technical and functional specifications of the software also at a high level.
It deals with business requirements.	It deals with the resources that the company provides.
It defines the customer's needs. The document is used from the beginning to the end of the project.	It describes how the business functions when using the software or application.
Tables and use cases are not included.	Tables and use cases are included.

Characteristics of a Software Requirements Specification Document:

- **Precise –** The goal of an SRS document is to be simple to comprehend. Nothing should be unclear, so there are no disputes between stakeholders.
- **Measurable –** The requirements in your SRS document must be measurable, therefore the finished product may be validated and certified against the requirements.
- **Complete –** The SRS document should include enough information for your development team and testers to complete the product and ensure that it fulfills the user's requirements without bugs.
- **Feasible –** The requirements should reflect the actual state of affairs, including cost, timeline and technology. They shouldn't be contingent on future technological advancements.
- **Flexible –** Because circumstances might change in the workplace, your SRS document should be adaptable enough to accommodate changes. Don't include superfluous material in several sections that will have to be updated every time there's a shift.
- **Verifiable –** Everyone on the development team should have access to the document so that they may reference it as often as required. Because requirements must be clear, team members don't want more information. They should all be accessible in the SRS document.
- **Consistent –** The requirements should be compatible. There should be no contradiction between parts of your requirement document. The document should be structured consistently and terminology used in the same way throughout.
- **No Implementation Constraints –** In general, an SRS document should be detailed enough to get the job done but not so specific that it stops development. A lot of development is based on third-party services that developers have no control over.
- **Accurate –** Requirements specified in the documents should be very precise to avoid any sort of confusion. They must not have any loopholes, ambiguities, subjectivity, superlatives or comparisons.

Essential Components of an SRS:

The main sections of a software requirements specification are:

- **Business Drivers –** The reasons why the customer is looking to build a system are described in this section. This section further includes the problems the customer is facing with the current system and the opportunities the new system will be providing.
- **Business Model –** The business model that the system is entailed to support is discussed in this section. It further includes various other details like the organizational and business context, main business functions and process flow diagrams of the system.

- **Functional and System Requirements –** This section typically details requirements that are organized in a hierarchical structure. The functional requirements are at the top level and the detailed system requirements are listed as sub-items.
- **System Use Cases –** This section consists of a Unified Modeling Language (UML) use case diagram explaining all the key external entities that will be interacting with the system and the different use cases they'll have to perform.
- **Technical Requirements –** This section discusses all the non-functional requirements that make up the technical environment and the technical limitations in which the software will be operating.
- **System Qualities –** In this section, the numerous qualities of the system are defined such as reliability, serviceability, security, scalability, availability, and maintainability.
- **Limitations and Assumptions –** This section describes all the limitations imposed on the system design from the customer's point of view. The various assumptions by the engineering team about what to expect during the development are also discussed here.
- **Acceptance Criteria –** Details on all the conditions that are to be met before the system is handed over to the final customers are discussed in this section.

Requirements Engineering Process:

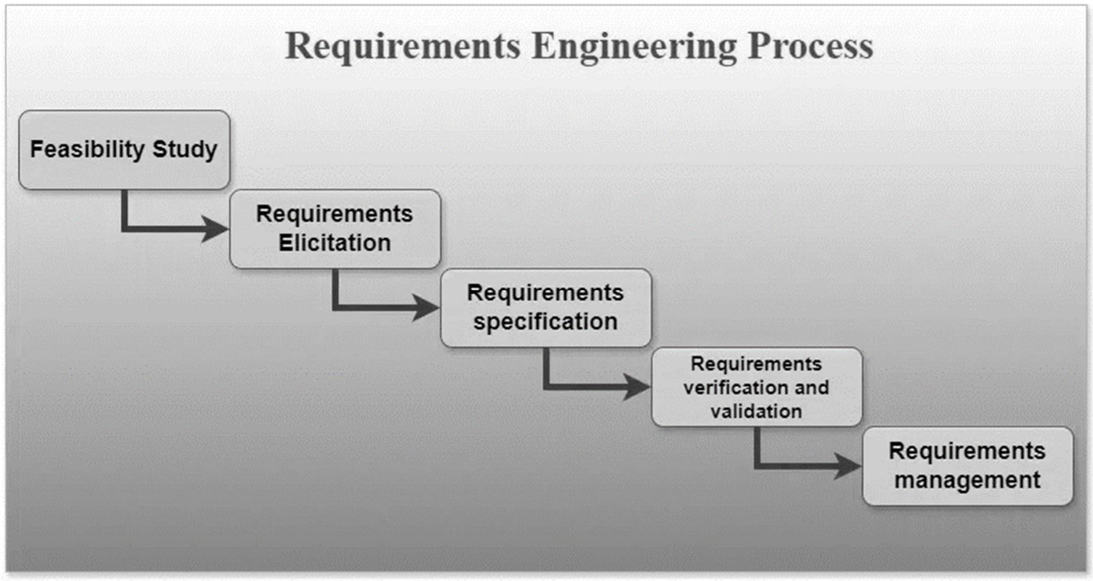

Requirements Engineering Process
Feasibility Study
Requirements Elicitation
Requirements specification
Requirements verification and validation
Requirements management

Fig: Requirements Engineering Process

1. Feasibility Study
2. Requirements elicitation
3. Requirements specification
4. Requirements for verification and validation
5. Requirements management

1. Feasibility Study:

The feasibility study mainly concentrates on below five mentioned areas below. Among these Economic Feasibility Study is the most important part of the feasibility analysis and the Legal Feasibility Study is less considered feasibility analysis.

1. **Technical Feasibility**: In Technical Feasibility current resources both hardware software along required technology are analyzed/assessed to develop the project. This technical feasibility study reports whether there are correct required resources and technologies that will be used for project development. Along with this, the feasibility study also analyzes the technical skills and capabilities of the technical team, whether existing technology can be used or not, whether maintenance and up-gradation are easy or not for the chosen technology, etc.
2. **Operational Feasibility:** In Operational Feasibility degree of providing service to requirements is analyzed along with how easy the product will be to operate and maintain after deployment. Along with this other operational scopes are determining the usability of the product, Determining suggested solution by the software development team is acceptable or not, etc.
3. **Economic Feasibility:** In the Economic Feasibility study cost and benefit of the project are analyzed. This means under this feasibility study a detailed analysis is carried out will be cost of the project for development which includes all required costs for final development hardware and software resources required, design and development costs operational costs, and so on. After that, it is analyzed whether the project will be beneficial in terms of finance for the organization or not.
4. **Legal Feasibility:** In legal feasibility, the project is ensured to comply with all relevant laws, regulations, and standards. It identifies any legal constraints that could impact the project and reviews existing contracts and agreements to assess their effect on the project's execution. Additionally, legal feasibility considers issues related to intellectual property, such as patents and copyrights, to safeguard the project's innovation and originality.
5. **Schedule Feasibility:** In schedule feasibility, the project timeline is evaluated to determine if it is realistic and achievable. Significant milestones are identified, and deadlines are established to track progress effectively. Resource availability is assessed to ensure that the necessary resources are accessible to meet the project schedule. Furthermore, any time constraints that might affect project delivery are considered to ensure timely completion. This focus on schedule feasibility is crucial for the successful planning and execution of a project.

2. Requirements Elicitation:

It is related to the various ways used to gain knowledge about the project domain and requirements. The various sources of domain knowledge include customers, business manuals, the existing software of the same type, standards, and other stakeholders of the project. The techniques used for requirements elicitation include interviews, brainstorming, task analysis, Delphi technique, prototyping, etc. Some of these are discussed here. Elicitation does not produce formal models of the requirements understood. Instead, it widens the domain knowledge of the analyst and thus helps in providing input to the next stage.

Requirements elicitation is the process of gathering information about the needs and expectations of stakeholders for a software system. This is the first step in the requirements engineering process and it is critical to the success of the software development project. The goal of this step is to understand the problem that the software system is intended to solve and the needs and expectations of the stakeholders who will use the system.

Several techniques can be used to elicit requirements, including:

- **Interviews**: These are one-on-one conversations with stakeholders to gather information about their needs and expectations.
- **Surveys**: These are questionnaires that are distributed to stakeholders to gather information about their needs and expectations.
- **Focus Groups**: These are small groups of stakeholders who are brought together to discuss their needs and expectations for the software system.
- **Observation**: This technique involves observing the stakeholders in their work environment to gather information about their needs and expectations.
- **Prototyping**: This technique involves creating a working model of the software system, which can be used to gather feedback from stakeholders and to validate requirements.

It's important to document, organize, and prioritize the requirements obtained from all these techniques to ensure that they are complete, consistent and accurate.

3. Requirements Specification:

This activity is used to produce formal software requirement models. All the requirements including the functional as well as the non-functional requirements and the constraints are specified by these models in totality. During specification, more knowledge about the problem may be required which can again trigger the elicitation process. The models used at this stage include ER diagrams, data flow diagrams (DFDs), function decomposition diagrams (FDDs), data dictionaries, etc.

Requirements specification is the process of documenting the requirements identified in the analysis step in a clear, consistent and unambiguous manner. This step also involves prioritizing and grouping the requirements into manageable chunks.

The goal of this step is to create a clear and comprehensive document that describes the requirements for the software system. This document should be understandable by both the development team and the stakeholders.

Several types of requirements are commonly specified in this step, including

1. **Functional Requirements:** These describe what the software system should do. They specify the functionality that the system must provide, such as input validation, data storage and user interface.
2. **Non-Functional Requirements**: These describe how well the software system should do it. They specify the quality attributes of the system, such as performance, reliability, usability and security.
3. **Constraints:** These describe any limitations or restrictions that must be considered when developing the software system.
4. **Acceptance Criteria**: These describe the conditions that must be met for the software system to be considered complete and ready for release.

To make the requirements specification clear, the requirements should be written in a natural language and use simple terms, avoiding technical jargon, and using a consistent format throughout the document. It is also important to use diagrams, models and other visual aids to help communicate the requirements effectively.

Once the requirements are specified, they must be reviewed and validated by the stakeholders and development team to ensure that they are complete, consistent and accurate.

4. Requirements Verification and Validation

Verification: It refers to the set of tasks that ensures that the software correctly implements a specific function.

Validation: It refers to a different set of tasks that ensures that the software that has been built is traceable to customer requirements. If requirements are not validated, errors in the requirement definitions would propagate to the successive stages resulting in a lot of modification and rework. The main steps for this process include:

1. The requirements should be consistent with all the other requirements i.e. no two requirements should conflict with each other.
2. The requirements should be complete in every sense.
3. The requirements should be practically achievable.

Reviews, buddy checks, making test cases, etc. are some of the methods used for this.

Requirements verification and validation (V&V) is the process of checking that the requirements for a software system are complete, consistent, and accurate and that they meet the needs and expectations of the stakeholders. The goal of V&V is to ensure that the software system being developed meets the requirements and that it is developed on time, within budget, and to the required quality.

1. Verification is checking that the requirements are complete, consistent and accurate. It involves reviewing the requirements to ensure that they are clear, testable, and free of errors

and inconsistencies. This can include reviewing the requirements document, models, and diagrams, and holding meetings and walkthroughs with stakeholders.

2. Validation is the process of checking that the requirements meet the needs and expectations of the stakeholders. It involves testing the requirements to ensure that they are valid and that the software system being developed will meet the needs of the stakeholders. This can include testing the software system through simulation, testing with prototypes and testing with the final version of the software.
3. Verification and Validation is an iterative process that occurs throughout the software development life cycle. It is important to involve stakeholders and the development team in the V&V process to ensure that the requirements are thoroughly reviewed and tested.

It's important to note that V&V is not a one-time process, but it should be integrated and continue throughout the software development process and even in the maintenance stage.

5. Requirements Management:

Requirement management is the process of analyzing, documenting, tracking, prioritizing, and agreeing on the requirement and controlling the communication with relevant stakeholders. This stage takes care of the changing nature of requirements. It should be ensured that the SRS is as modifiable as possible to incorporate changes in requirements specified by the end users at later stages too. Modifying the software as per requirements in a systematic and controlled manner is an extremely important part of the requirements engineering process.

Requirements management is the process of managing the requirements throughout the software development life cycle, including tracking and controlling changes, and ensuring that the requirements are still valid and relevant. The goal of requirements management is to ensure that the software system being developed meets the needs and expectations of the stakeholders and that it is developed on time, within budget, and to the required quality.

Several key activities are involved in requirements management, including:

1. **Tracking and controlling changes:** This involves monitoring and controlling changes to the requirements throughout the development process, including identifying the source of the change, assessing the impact of the change and approving or rejecting the change.
2. **Version control**: This involves keeping track of different versions of the requirements document and other related artifacts.
3. **Traceability**: This involves linking the requirements to other elements of the development process, such as design, testing and validation.
4. **Communication:** This involves ensuring that the requirements are communicated effectively to all stakeholders and that any changes or issues are addressed promptly.

5. **Monitoring and reporting**: This involves monitoring the progress of the development process and reporting on the status of the requirements.

Requirements management is a critical step in the software development life cycle as it helps to ensure that the software system being developed meets the needs and expectations of stakeholders and that it is developed on time, within budget, and to the required quality. It also helps to prevent scope creep and to ensure that the requirements are aligned with the project goals.

Stages in Software Engineering Process:

Requirements engineering is a critical process in software engineering that involves identifying, analyzing, documenting, and managing the requirements of a software system. The requirements engineering process consists of the following stages:

- **Elicitation:** In this stage, the requirements are gathered from various stakeholders such as customers, users, and domain experts. The aim is to identify the features and functionalities that the software system should provide.
- **Analysis:** In this stage, the requirements are analyzed to determine their feasibility, consistency, and completeness. The aim is to identify any conflicts or contradictions in the requirements and resolve them.
- **Specification:** In this stage, the requirements are documented in a clear, concise, and unambiguous manner. The aim is to provide a detailed description of the requirements that can be understood by all stakeholders.
- **Validation:** In this stage, the requirements are reviewed and validated to ensure that they meet the needs of all stakeholders. The aim is to ensure that the requirements are accurate, complete and consistent.
- **Management:** In this stage, the requirements are managed throughout the software development lifecycle. The aim is to ensure that any changes or updates to the requirements are properly documented and communicated to all stakeholders.
- Effective requirements engineering is crucial to the success of software development projects. It helps ensure that the software system meets the needs of all stakeholders and is delivered on time, within budget and to the required quality standards.

<u>Challenges and Risks Involved in Requirement Engineering:</u>

- **Incomplete Requirements:**
    - Sometimes stakeholders may not fully articulate their needs or omit crucial details, leading to incomplete requirements.
    - Incomplete requirements can result in systems that do not meet user expectations or fail to address critical functionalities.

- **Ambiguous Requirements:**
    - Vague or ambiguous language can lead to different interpretations of requirements by different team members.
    - Ambiguities can result in miscommunication, design flaws and delays in development.
- **Changing Requirements:**
    - Stakeholder needs and market conditions can evolve, causing requirements to change during the project lifecycle.
    - Frequent changes can lead to scope creep, budget overruns and development delays.
- **Conflicting Requirements:**
    - Different stakeholders may have conflicting needs or priorities that are challenging to reconcile.
    - Resolving conflicts requires careful negotiation and trade-offs to ensure a balanced solution.
- **Poor Requirement Traceability:**
    - Ensuring that each requirement is appropriately tracked throughout the development process can be complex.
    - Poor traceability can make it difficult to verify that all requirements have been addressed and tested, leading to quality issues.

Tools Involved in Requirement Engineering:

Requirement Engineering (RE) is a crucial phase in software development that focuses on understanding, documenting, and managing the needs and expectations of stakeholders for a software system. To effectively perform requirement engineering, various tools are employed to facilitate the process and ensure that the project's goals are met. These tools aid in capturing, analyzing, validating and managing requirements throughout the software development lifecycle. Here, I'll detail some of the key tools involved in requirement engineering:

1. **Document Editors and Word Processors:** Traditional document editors like Microsoft Word, Google Docs, and LaTeX are often used to create textual requirement documents. These tools help write and format textual descriptions of requirements, use cases, user stories and other related documentation.
2. **Spreadsheet Tools:** Tools like Microsoft Excel or Google Sheets can be used to create requirement lists, matrices, and traceability tables. These tools are useful for managing and organizing requirements and they allow for easy sorting, filtering and categorization.
3. **Diagramming Tools:** Diagrams are powerful visual aids for understanding and representing requirements. Tools like Microsoft Visio, draw.io and Lucidchart enable the creation of diagrams

such as use case diagrams, data flow diagrams, activity diagrams and entity-relationship diagrams to illustrate system behavior, interactions, and data flow.

4. **Requirements Management Tools:** These tools are specifically designed to capture, organize, track changes, and manage requirements throughout the project lifecycle. Examples include IBM Engineering Requirements Management DOORS, Jama Connect and Helix RM. These tools often support features like version control, traceability, and collaboration.
5. **Prototyping Tools:** Prototyping tools like Balsamiq, Axure and Sketch are useful for creating interactive mockups and prototypes of the user interface. Prototypes allow stakeholders to visualize how the software will look and behave, providing a tangible representation of requirements.
6. **Modeling Languages:** Modeling languages like Unified Modeling Language (UML) provide a standardized way to represent system requirements and designs. UML diagrams such as use case diagrams, class diagrams, sequence diagrams and state diagrams help in communicating complex requirements and system structures.
7. **Collaboration Tools:** Collaboration tools like Confluence, SharePoint and Microsoft Teams facilitate communication and collaboration among distributed teams. They allow team members to share and review requirement documents, provide feedback and discuss changes.
8. **Requirement Elicitation Tools:** Tools like surveys, interviews, questionnaires and workshops are essential for gathering requirements directly from stakeholders. While these tools are not software applications, they are crucial techniques used to extract requirements information.
9. **Validation and Verification Tools:** Tools such as formal methods, static analysis tools and simulation tools help in validating and verifying requirements for correctness and consistency. These tools aid in identifying potential issues early in the process.
10. **Traceability Tools:** Traceability tools help establish and manage the relationships between different levels of requirements, ensuring that all requirements are covered by design, implementation and testing. These tools assist in maintaining a clear link between the various phases of software development.

Advantages of Requirements Engineering in Software Engineering:

- **Customer Satisfaction:** Well-defined requirements lead to software that meets user needs, enhancing customer satisfaction.
- **Early Issue Detection:** Detecting issues in requirements early prevents costly errors and misunderstandings later in development.
- **Cost and Time Savings:** Detailed requirements reduce scope creep, saving time and resources by avoiding rework.

- **Effective Planning:** Clear requirements aid in accurate project planning, resource allocation and effort estimation.
- **Traceability:** Requirements traceability links software components to specific requirements, aiding in impact analysis during changes.
- **Validation and Verification:** Accurate requirements serve as a basis for validating and verifying software correctness.
- **Collaboration:** Engaging stakeholders fosters collaboration, shared understanding and effective teamwork.
- **Scope Management:** Formalized requirements prevent uncontrolled scope expansion, managing project scope effectively.
- **Risk Mitigation:** Thorough requirements facilitate risk identification and mitigation planning.
- **Quality Assurance:** Rigorous requirements lead to a higher-quality end product and informed testing efforts.
- **Regulatory Compliance:** Clear requirements assist in meeting industry regulations and standards.
- **Documentation and Maintenance:** Accurate requirements serve as valuable documentation for ongoing maintenance and enhancements.
- **User Involvement:** Engaging users ensures their needs are considered, resulting in a product aligned with expectations.
- **Change Management:** Requirements provide a structured approach to managing change requests.
- **Long-Term Benefits:** Investing in requirements engineering yields lasting benefits throughout the software's lifecycle.

Disadvantages of Requirements Engineering in Software Engineering:

- **Ambiguity: Poorly defined requirements can lead to confusion** and misinterpretation among stakeholders and developers.
- **Scope Creep:** Uncontrolled additions to requirements can expand project scope, causing delays and increased costs.
- **Changing Requirements:** Shifting requirements during development can lead to delays, rework and potential conflicts.
- **User Involvement Challenges:** Engaging users effectively can be difficult, leading to missed requirements and dissatisfaction.
- **Overemphasis on Documentation:** Excessive focus on documentation can slow down development and limit agility.

- **Miscommunication:** Inadequate communication can result in gaps between stakeholder expectations and developer understanding.
- **Time-Consuming Process:** Elaborate requirement gathering and analysis can prolong project initiation.
- **Resistance to Change:** Stakeholders might resist necessary changes due to emotional attachment to initial requirements.
- **Dependency on Tools:** Overreliance on requirement management tools can lead to issues if tools fail or become obsolete.
- **Neglecting Non-Functional Requirements:** Focusing solely on functional aspects might lead to overlooking critical non-functional aspects like performance and security.