INTRODUCTION TO SOFTWARE ENGINEERING

The term software engineering is composed of two words, software and engineering.

Software is more than just a program code. A program is an executable code, which serves some computational purpose. Software is considered to be a collection of executable programming code, associated libraries and documentations. Software, when made for a specific requirement is called software product.

Engineering on the other hand, is all about developing products, using well-defined, scientific principles and methods.

So, we can define software engineering as an engineering branch associated with the development of software product using well-defined scientific principles, methods and procedures. The outcome of software engineering is an efficient and reliable software product.

IEEE defines software engineering as:

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

We can alternatively view it as a systematic collection of past experience. The experience is arranged in the form of methodologies and guidelines. A small program can be written without using software engineering principles. But if one wants to develop a large software product, then software engineering principles are absolutely necessary to achieve a good quality software cost effectively.

Without using software engineering principles it would be difficult to develop large programs. In industry it is usually needed to develop large programs to accommodate multiple functions. A

problem with developing such large commercial programs is that the complexity and difficulty levels of the programs increase exponentially with their sizes. Software engineering helps to reduce this programming complexity. Software engineering principles use two important techniques to reduce problem complexity: abstraction and decomposition. The principle of abstraction implies that a problem can be simplified by omitting irrelevant details. In other words, the main purpose of abstraction is to consider only those aspects of the problem that are relevant for certain purpose and suppress other aspects that are not relevant for the given purpose. Once the simpler problem is solved, then the omitted details can be taken into consideration to solve the next lower level abstraction, and so on. Abstraction is a powerful way of reducing the complexity of the problem. The other approach to tackle problem complexity is

decomposition. In this technique, a complex problem is divided into several smaller problems and then the smaller problems are solved one by one. However, in this technique any random decomposition of a problem into smaller parts will not help. The problem has to be decomposed such that each component of the decomposed problem can be solved independently and then the solution of the different components can be combined to get the full solution. A good decomposition of a problem should minimize interactions among various components. If the different subcomponents are interrelated, then the different components cannot be solved separately and the desired reduction in complexity will not be realized.

NEED OF SOFTWARE ENGINEERING

The need of software engineering arises because of higher rate of change in user requirements and environment on which the software is working.

Large software - It is easier to build a wall than to a house or building, likewise, as the size of software become large engineering has to step to give it a scientific process.

Scalability- If the software process were not based on scientific and engineering concepts, it would be easier to re-create new software than to scale an existing one.

Cost- As hardware industry has shown its skills and huge manufacturing has lower down the price of computer and electronic hardware. But the cost of software remains high if proper process is not adapted.

Dynamic Nature- The always growing and adapting nature of software hugely depends upon the environment in which the user works. If the nature of software is always changing, new enhancements need to be done in the existing one. This is where software engineering plays a good role.

Quality Management- Better process of software development provides better and quality software product.

CHARACTERESTICS OF GOOD SOFTWARE

A software product can be judged by what it offers and how well it can be used. This software must satisfy on the following grounds:

Operational
Transitional
Maintenance

Well-engineered and crafted software is expected to have the following characteristics:

Operational

This tells us how well software works in operations. It can be measured on:

Budget
Usability
Efficiency
Correctness
Functionality

- Dependability
- Security
- Safety

Transitional

This aspect is important when the software is moved from one platform to another:

- Portability
- Interoperability
- Reusability
- Adaptability

Maintenance

This aspect briefs about how well a software has the capabilities to maintain itself in the ever-changing environment:

- Modularity
- Maintainability
- Flexibility
- Scalability

In short, Software engineering is a branch of computer science, which uses well-defined engineering concepts required to produce efficient, durable, scalable, in-budget and on-time software products

LIFE CYCLE MODEL

A software life cycle model (also called process model) is a descriptive and diagrammatic representation of the software life cycle. A life cycle model represents all the activities required to make a software product transit through its life cycle phases. It also captures the order in which these activities are to be undertaken. In other words, a life cycle model maps the different activities performed on a software product from its inception to retirement. Different life cycle models may map the basic development activities to phases in different ways. Thus, no matter which life cycle model is followed, the basic activities are included in all life cycle models

though the activities may be carried out in different orders in different life cycle models. During
any life cycle phase, more than one activity may also be carried out.

THE NEED FOR A SOFTWARE LIFE CYCLE MODEL

The development team must identify a suitable life cycle model for the particular project and
then adhere to it. Without using of a particular life cycle model the development of a software
product would not be in a systematic and disciplined manner. When a software product is being
developed by a team there must be a clear understanding among team members about when and
what to do. Otherwise it would lead to chaos and project failure. This problem can be illustrated
by using an example. Suppose a software development problem is divided into several parts and
the parts are assigned to the team members. From then on, suppose the team members are
allowed the freedom to develop the parts assigned to them in whatever way they like. It is
possible that one member might start writing the code for his part, another might decide to
prepare the test documents first, and some other engineer might begin with the design phase of
the parts assigned to him. This would be one of the perfect recipes for project failure. A software
life cycle model defines entry and exit criteria for every phase. A phase can start only if its
phase-entry criteria have been satisfied. So without software life cycle model the entry and exit
criteria for a phase cannot be recognized. Without software life cycle models it becomes difficult
for software project managers to monitor the progress of the project.

Different software life cycle models

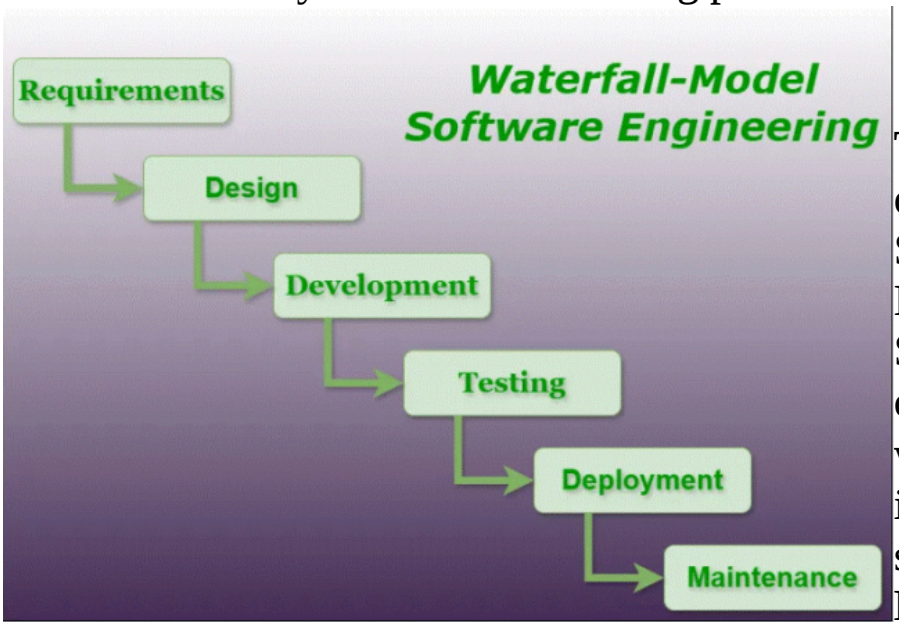
Many life cycle models have been proposed so far. Each of them has some advantages as well as
some disadvantages. A few important and commonly used life cycle models are as follows:

- Waterfall Model
- Iterative Waterfall Model

Prototyping Model
Spiral Model

1. WATERFALL MODEL

The classical waterfall model is intuitively the most obvious way to develop software. Though
the classical waterfall model is elegant and intuitively obvious, it is not a practical model in the
sense that it cannot be used in actual software development projects. Thus, this model can be
considered to be a theoretical way of developing software. But all other life cycle models are
essentially derived from the classical waterfall model. So, in order to be able to appreciate other
life cycle models it is necessary to learn the classical waterfall model.
Classical waterfall model
divides the life cycle into the following phases



The Spiral Model is one of the most important Software Development Life Cycle models. The Spiral Model is a combination of the waterfall model and the iterative model. It provides support for **Risk Handling**. The Spiral Model was first proposed by **Barry Boehm**. This article focuses on discussing the Spiral Model in detail.

What is the Spiral Model?

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

Some Key Points regarding the phase of a Spiral Model:

1. The exact number of phases needed to develop the product can be varied by the project manager depending upon the project risks.
2. As the project manager dynamically determines the number of phases, the project manager has an important role in developing a product using the spiral model.
3. It is based on the idea of a spiral, with each iteration of the spiral representing a complete software development cycle, from requirements gathering and analysis to design, implementation, testing, and maintenance.

Phases of the Spiral Model?

The Spiral Model is a risk-driven model, meaning that the focus is on managing risk through multiple iterations of the software development process. It consists of the following phases:

1. **Planning:** The first phase of the Spiral Model is the planning phase, where the scope of the project is determined and a plan is created for the next iteration of the spiral.
2. **Risk Analysis:** In the risk analysis phase, the risks associated with the project are identified and evaluated.
3. **Engineering:** In the engineering phase, the software is developed based on the requirements gathered in the previous iteration.
4. **Evaluation:** In the evaluation phase, the software is evaluated to determine if it meets the customer's requirements and if it is of high quality.
5. **Planning:** The next iteration of the spiral begins with a new planning phase, based on the results of the evaluation.

The Spiral Model is often used for complex and large software development projects, as it allows for a more flexible and adaptable approach to software development. It is also well-suited to projects with significant uncertainty or high levels of risk.

The Radius of the spiral at any point represents the expenses (cost) of the project so far, and the angular dimension represents the progress

made so far in the current phase.

Spiral Model

Each phase of the Spiral Model is divided into four quadrants as shown in the above figure. The functions of these four quadrants are discussed below:

1. **Objectives determination and identify alternative solutions:** Requirements are gathered from the customers and the objectives are identified, elaborated, and analyzed at the start of every phase. Then alternative solutions possible for the phase are proposed in this quadrant.
2. **Identify and resolve Risks:** During the second quadrant, all the possible solutions are evaluated to select the best possible solution. Then the risks associated with that solution are identified and the risks are resolved using the best possible strategy. At the end of this quadrant, the Prototype is built for the best possible solution.
3. **Develop the next version of the Product:** During the third quadrant, the identified features are developed and verified through testing. At the end of the third quadrant, the next version of the software is available.
4. **Review and plan for the next Phase:** In the fourth quadrant, the Customers evaluate the so-far developed version of the software. In the end, planning for the next phase is started.

Risk Handling in Spiral Model

A risk is any adverse situation that might affect the successful completion of a software project. The most important feature of the spiral model is handling these unknown risks after the project has started. Such risk resolutions are easier done by developing a prototype.

1. The spiral model supports coping with risks by providing the scope to build a prototype at every phase of software development.
2. The **Prototyping Model** also supports risk handling, but the risks must be identified completely before the start of the development work of the project.
3. But in real life, project risk may occur after the development work starts, in that case, we cannot use the Prototyping Model.
4. In each phase of the Spiral Model, the features of the product dated and analyzed, and the risks at that point in time are identified and are resolved through prototyping.
5. Thus, this model is much more flexible compared to other SDLC models.

Advantages of the Spiral Model

Below are some advantages of the Spiral Model.

1. **Risk Handling:** The projects with many unknown risks that occur as the development proceeds, in that case, Spiral Model is the best development model to follow due to the risk analysis and risk handling at every phase.
2. **Good for large projects:** It is recommended to use the Spiral Model in large and complex projects.
3. **Flexibility in Requirements:** Change requests in the Requirements at a later phase can be incorporated accurately by using this model.
4. **Customer Satisfaction:** Customers can see the development of the product at the early phase of the software development and thus, they habituated with the system by using it before completion of the total product.
5. **Iterative and Incremental Approach:** The Spiral Model provides an iterative and incremental approach to software development, allowing for flexibility and adaptability in response to changing requirements or unexpected events.
6. **Emphasis on Risk Management:** The Spiral Model places a strong emphasis on risk management, which helps to minimize the impact of uncertainty and risk on the software development process.
7. **Improved Communication:** The Spiral Model provides for regular evaluations and reviews, which can improve communication between the customer and the development team.
8. **Improved Quality:** The Spiral Model allows for multiple iterations of the software development process, which can result in improved software quality and reliability.

Disadvantages of the Spiral Model

Below are some main disadvantages of the spiral model.

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

The most serious issue we face in the cascade model is that taking a long length to finish the item, and the product became obsolete. To tackle this issue, we have another methodology, which is known as the Winding model or spiral model. The winding model is otherwise called the cyclic model.

When To Use the Spiral Model?

1. When a project is vast in software engineering, a spiral model is utilized.
2. A spiral approach is utilized when frequent releases are necessary.
3. When it is appropriate to create a prototype
4. When evaluating risks and costs is crucial
5. The spiral approach is beneficial for projects with moderate to high risk.
6. The SDLC's spiral model is helpful when requirements are complicated and ambiguous.
7. If modifications are possible at any moment
8. When committing to a long-term project is impractical owing to shifting economic priorities.

Conclusion

Spiral Model is a valuable choice for software development projects where risk management is on high priority. Spiral Model deliver high-quality software by promoting risk identification, iterative development and continuous client feedback. When a project is vast in software engineering, a spiral model is utilized.

The Rapid Application Development Model was first proposed by IBM in the 1980s. The RAD model is a type of incremental process model in which there is an extremely short development cycle. When the requirements are fully understood and the component-based construction approach is adopted then the RAD model is used. Various phases in RAD are Requirements Gathering, Analysis and Planning, Design, Build or Construction, and finally Deployment.

The use of powerful developer tools such as JAVA, C++, Visual BASIC, XML, etc. is also an integral part of the projects. This model consists of 4 basic phases:

1. **Requirements Planning –** This involves the use of various techniques used in requirements elicitation like brainstorming, task analysis, form analysis, user scenarios, FAST (Facilitated Application Development Technique), etc. It also consists of the entire structured plan describing the critical data, methods to obtain it, and then processing it to form a final refined model.
2. **User Description –** This phase consists of taking user feedback and building the prototype using developer tools. In other words, it includes re-examination and validation of the data collected in the first phase. The dataset attributes are also identified and elucidated in this phase.
3. **Construction –** In this phase, refinement of the prototype and delivery takes place. It includes the actual use of powerful automated tools to transform processes and data models into the final working product. All

the required modifications and enhancements are to be done in this phase.
4. **Cutover –** All the interfaces between the independent modules developed by separate teams have to be tested properly. The use of powerfully automated tools and subparts makes testing easier. This is followed by acceptance testing by the user.

The process involves building a rapid prototype, delivering it to the customer, and taking feedback. After validation by the customer, the SRS document is developed and the design is finalized.

When to use the RAD Model?

1. **Well-understood Requirements:** When project requirements are stable and transparent, RAD is appropriate.
2. **Time-sensitive Projects:** Suitable for projects that need to be developed and delivered quickly due to tight deadlines.
3. **Small to Medium-Sized Projects:** Better suited for smaller initiatives requiring a controllable number of team members.
4. **High User Involvement:** Fits where ongoing input and interaction from users are essential.
5. **Innovation and Creativity**: Helpful for tasks requiring creative inquiry and innovation.
6. **Prototyping:** It is necessary when developing and improving prototypes is a key component of the development process.
7. **Low technological Complexity:** Suitable for tasks using comparatively straightforward technological specifications.

Objectives of Rapid Application Development Model (RAD)

1. Speedy Development

Accelerating the software development process is RAD's main goal. RAD prioritizes rapid prototyping and iterations to produce a working system as soon as possible. This is especially helpful for projects when deadlines must be met.

2. Adaptability and Flexibility

RAD places a strong emphasis on adapting quickly to changing needs. Due to the model's flexibility, stakeholders can modify and improve the system in response to changing requirements and user input.

3. Stakeholder Participation

Throughout the development cycle, RAD promotes end users and stakeholders' active participation. Collaboration and frequent feedback make it possible to make sure that the changing system satisfies both user and corporate needs.

4. Improved Interaction

Development teams and stakeholders may collaborate and communicate more effectively thanks to RAD. Frequent communication and feedback loops guarantee that all project participants are in agreement, which lowers the possibility of misunderstandings.

5. Improved Quality via Prototyping

Prototypes enable early system component testing and visualization in Rapid Application Development (RAD). This aids in spotting any problems, confirming design choices, and guaranteeing that the finished product lives up to consumer expectations.

6. Customer Satisfaction

Delivering a system that closely satisfies user expectations and needs is the goal of RAD. Through rapid delivery of functioning prototypes and user involvement throughout the development process, Rapid Application Development (RAD) enhances the probability of customer satisfaction with the final product.

Advantages of Rapid Application Development Model (RAD)

- The use of reusable components helps to reduce the cycle time of the project.
- Feedback from the customer is available at the initial stages.
- Reduced costs as fewer developers are required.
- The use of powerful development tools results in better quality products in comparatively shorter periods.
- The progress and development of the project can be measured through the various stages.
- It is easier to accommodate changing requirements due to the short iteration time spans.
- Productivity may be quickly boosted with a lower number of employees.

Disadvantages of Rapid application development model (RAD)

- The use of powerful and efficient tools requires highly skilled professionals.
- The absence of reusable components can lead to the failure of the project.
- The team leader must work closely with the developers and customers to close the project on time.

- The systems which cannot be modularized suitably cannot use this model.
- Customer involvement is required throughout the life cycle.
- It is not meant for small-scale projects as in such cases, the cost of using automated tools and techniques may exceed the entire budget of the project.
- Not every application can be used with RAD.

Applications of Rapid Application Development Model (RAD)

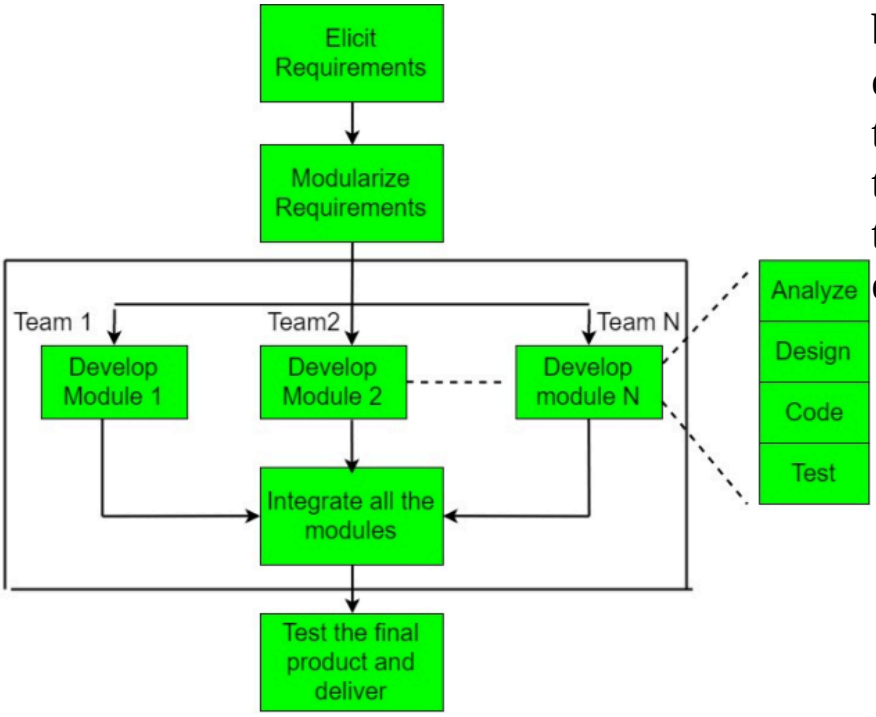
1. This model should be used for a system with known requirements and requiring a short development time.
2. It is also suitable for projects where requirements can be modularized and reusable components are also available for development.
3. The model can also be used when already existing system components can be used in developing a new system with minimum changes.
4. This model can only be used if the teams consist of domain experts. This is because relevant knowledge and the ability to use powerful techniques are a necessity.
5. The model should be chosen when the budget permits the use of automated tools and techniques required.

Drawbacks of Rapid Application Development

- It requires multiple teams or a large number of people to work on scalable projects.
- This model requires heavily committed developers and customers. If commitment is lacking then RAD projects will fail.
- The projects using the RAD model require heavy resources.
- If there is no appropriate modularization then RAD projects fail. Performance can be a problem for such projects.

Agile Software Development is a software development methodology that values flexibility, collaboration, and customer satisfaction. It is based on the Agile Manifesto, a set of principles for software development that prioritize individuals and interactions, working software, customer collaboration, and responding to change.

Agile Software Development is an iterative and incremental approach to software development that emphasizes the importance of delivering a working product quickly and frequently. It involves close collaboration



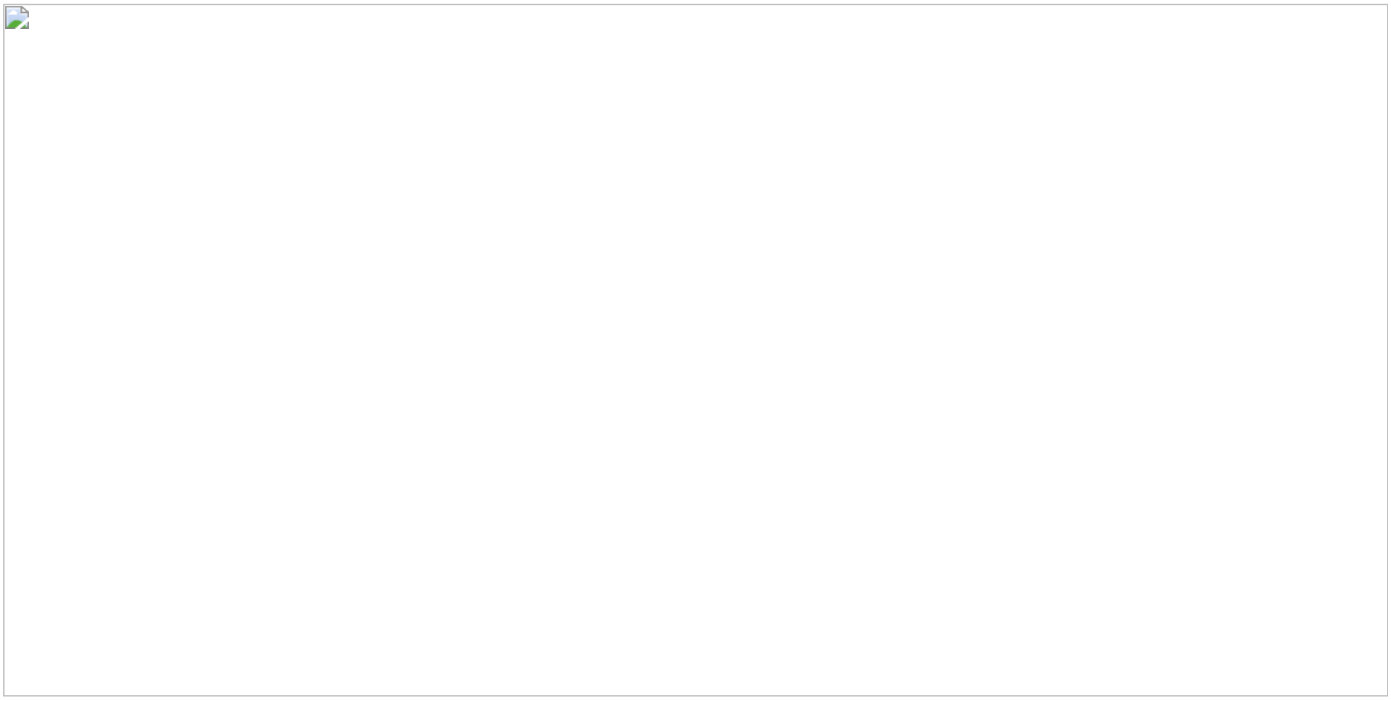
between the development team and the customer to ensure that the product meets their needs and expectations.

Why Agile is Used?

1. **Creating Tangible Value:** Agile places a high priority on creating tangible value as soon as possible in a project. Customers can benefit from early delivery of promised advantages and opportunity for prompt feedback and modifications.
2. **Concentrate on Value-Added Work**: Agile methodology promotes teams to concentrate on producing functional and value-added product increments, hence reducing the amount of time and energy allocated to non-essential tasks.
3. **Agile as a Mindset:** Agile represents a shift in culture that values adaptability, collaboration, and client happiness. It gives team members more authority and promotes a cooperative and upbeat work atmosphere.
4. **Quick Response to Change:** Agile fosters a culture that allows teams to respond swiftly to constantly shifting priorities and requirements. This adaptability is particularly useful in sectors of the economy or technology that experience fast changes.
5. **Regular Demonstrations:** Agile techniques place a strong emphasis on regular demonstrations of project progress. Stakeholders may clearly see the project's status, upcoming problems, and upcoming new features due to this transparency.
6. **Cross-Functional Teams:** Agile fosters self-organizing, cross-functional teams that share information effectively, communicate more effectively and feel more like a unit.

4 Core Values of Agile Software Development

The Agile Software Development Methodology Manifesto describe four core values of Agile in software development.

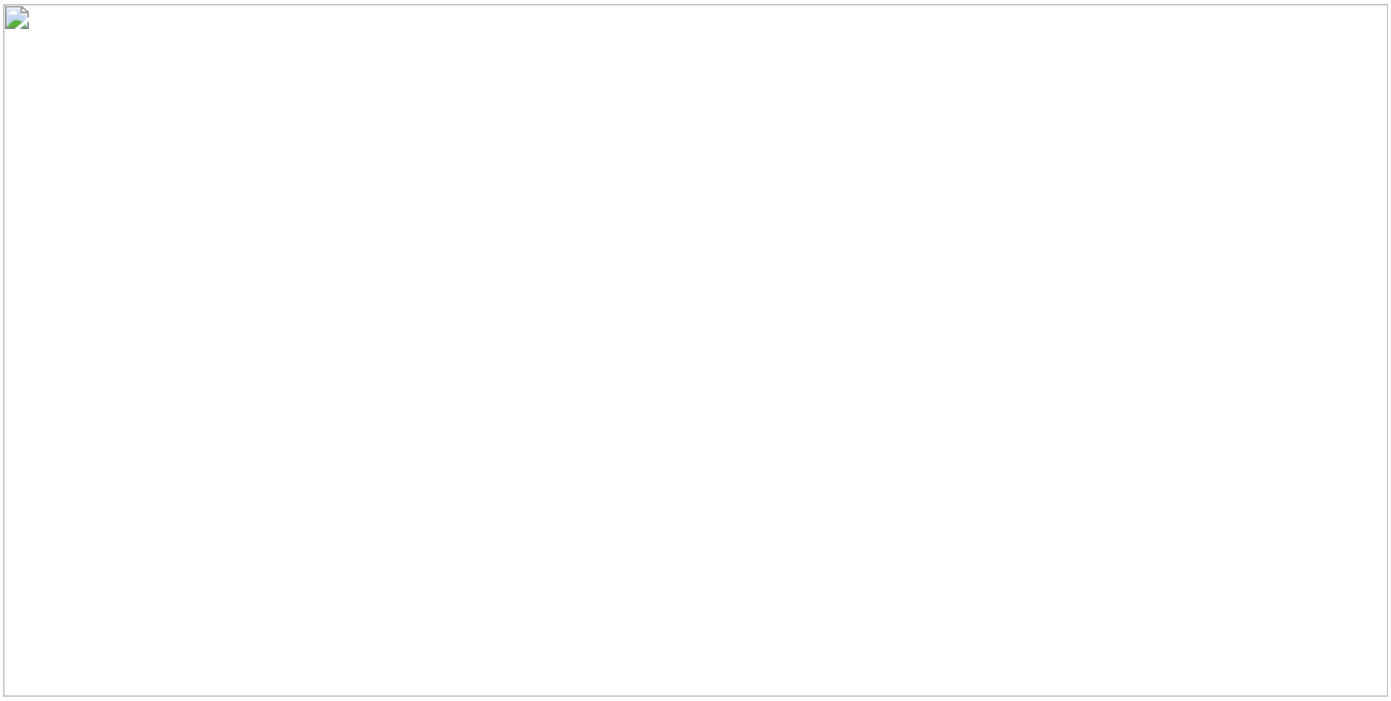


4 Values of Agile

1. Individuals and Interactions over Processes and Tools
2. Working Software over Comprehensive Documentation
3. Customer Collaboration over Contract Negotiation
4. Responding to Change over Following a Plan

12 Principles of Agile Software Development

The Agile Manifesto is based on four values and twelve principles that form the basis, for methodologies.

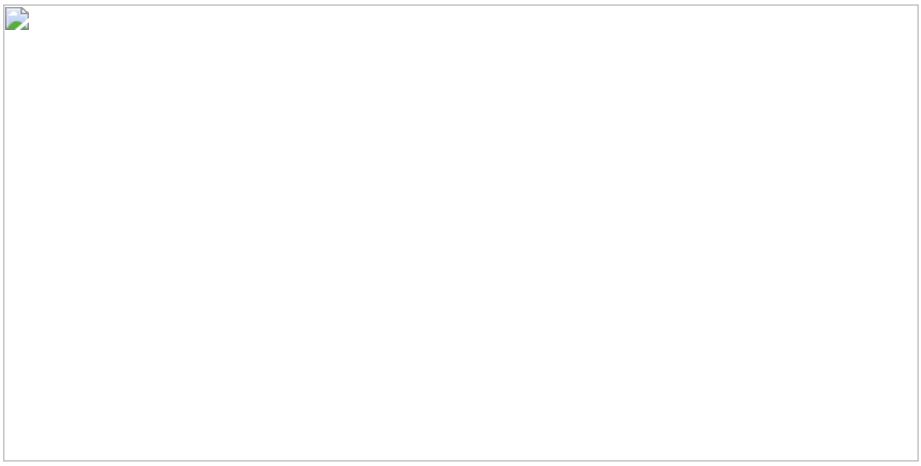


12 Principles of Agile Methodology

These principles include:

1. Ensuring customer satisfaction through the early delivery of software.
2. Being open to changing requirements in the stages of the development.
3. Frequently delivering working software with a main focus on preference for timeframes.
4. Promoting collaboration between business stakeholders and developers as an element.
5. Structuring the projects around individuals. Providing them with the necessary environment and support.
6. Prioritizing face to face communication whenever needed.
7. Considering working software as the measure of the progress.
8. Fostering development by allowing teams to maintain a pace indefinitely.
9. Placing attention on excellence and good design practices.
10. Recognizing the simplicity as crucial factor aiming to maximize productivity by minimizing the work.
11. Encouraging self organizing teams as the approach to design and build systems.
12. Regularly reflecting on how to enhance effectiveness and to make adjustments accordingly.

The Agile Software Development Process



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

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.

Design Process of Agile software Development

- 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.

Example of Agile Software Development

Let's go through an example to understand clearly how agile works. A Software company named **ABC** wants to make a new web browser for the latest release of its operating system. The deadline for the task is 10 months. The company's head assigned two teams named **Team A** and **Team B** for this task. To motivate the teams, the company head says that the first team to develop the browser would be given a salary hike and a one-week full-sponsored travel plan. With the dreams of their wild travel fantasies, the two teams set out on the journey of the web browser. Team A decided to play by the book and decided to choose the Waterfall model for the development. Team B after a heavy discussion decided to take a leap of faith and choose Agile as their development model. The Development Plan of the Team A is as follows:

- Requirement analysis and Gathering – 1.5 Months
- Design of System – 2 Months
- Coding phase – 4 Months
- System Integration and Testing – 2 Months
- User Acceptance Testing – 5 Weeks

The Development Plan for the Team B is as follows:

- Since this was an Agile, the project was broken up into several iterations.
- The iterations are all of the same time duration.
- At the end of each iteration, a working product with a new feature has to be delivered.
- Instead of Spending 1.5 months on requirements gathering, they will decide the core features that are required in the product and decide which of these features can be developed in the first iteration.
- Any remaining features that cannot be delivered in the first iteration will be delivered in the next subsequent iteration, based on the priority.
- At the end of the first iterations, the team will deliver working software with the core basic features.

The team has put their best efforts into getting the product to a complete stage. But then out of the blue due to the rapidly changing environment, the company's head came up with an entirely new set of features that wanted to be implemented as quickly as possible and wanted to push out a working model in 2 days. Team A was now in a fix, they were still in their design phase and had not yet started coding and they had no working model to display. Moreover, it was practically impossible for them to implement new

features since the waterfall model there is not revert to the old phase once you proceed to the next stage, which means they would have to start from square one again. That would incur heavy costs and a lot of overtime. Team B was ahead of Team A in a lot of aspects, all thanks to Agile Development. They also had a working product with most of the core requirements since the first increment. And it was a piece of cake for them to add the new requirements. All they had to do was schedule these requirements for the next increment and then implement them.

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.

Agile is a framework that defines how software development needs to be carried on. Agile is not a single method, it represents the various collection of methods and practices that follow the value statements provided in the manifesto. Agile methods and practices do not promise to solve every problem present in the software industry (No Software model ever can). But

they sure help to establish a culture and environment where solutions emerge.

Agile software development is an iterative and incremental approach to software development. It emphasizes collaboration between the development team and the customer, flexibility, and adaptability in the face of changing requirements, and the delivery of working software in short iterations.

The Agile Manifesto, which outlines the principles of agile development, values individuals and interactions, working software, customer collaboration, and response to change.

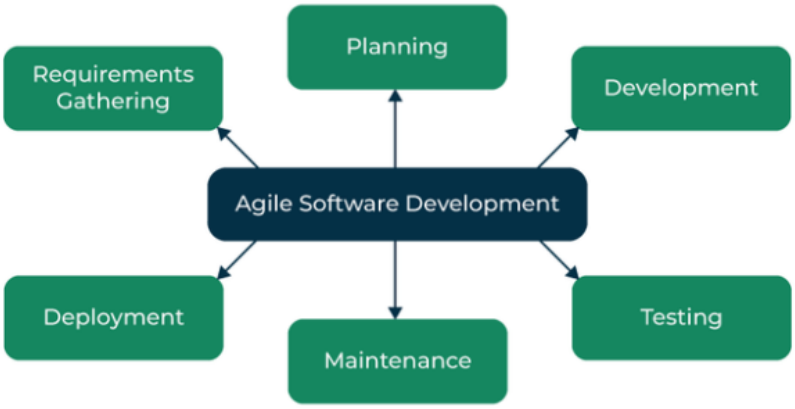
Practices of Agile Software Development

- **Scrum:** Scrum is a framework for agile software development that involves iterative cycles called sprints, daily stand-up meetings, and a product backlog that is prioritized by the customer.
- **Kanban:** Kanban is a visual system that helps teams manage their work and improve their processes. It involves using a board with columns to represent different stages of the development process, and cards or sticky notes to represent work items.
- **Continuous Integration:** Continuous Integration is the practice of frequently merging code changes into a shared repository, which helps to identify and resolve conflicts early in the development process.
- **Test-Driven Development:** Test-Driven Development (TDD) is a development practice that involves writing automated tests before writing the code. This helps to ensure that the code meets the requirements and reduces the likelihood of defects.
- P**air Programming:** Pair programming involves two developers working together on the same code. This helps to improve code quality, share knowledge, and reduce the likelihood of defects.

Advantages of Agile Software Development over traditional software development approaches

1. **Increased customer satisfaction:** Agile development involves close collaboration with the customer, which helps to ensure that the software meets their needs and expectations.
2. **Faster time-to-market:** Agile development emphasizes the delivery of working software in short iterations, which helps to get the software to market faster.
3. **Reduced risk:** Agile development involves frequent testing and feedback, which helps to identify and resolve issues early in the development process.
4. **Improved team collaboration:** Agile development emphasizes collaboration and communication between team members, which helps to improve productivity and morale.

5. **Adaptability to change:** Agile Development is designed to be flexible and adaptable, which means that changes to the project scope, requirements, and timeline can be accommodated easily. This can help the team to respond quickly to changing business needs and market demands.
6. **Better quality software:** Agile Development emphasizes continuous testing and feedback, which helps to identify and resolve issues early in the development process. This can lead to higher-quality software that is more reliable and less prone to errors.
7. **Increased transparency:** Agile Development involves frequent communication and collaboration between the team and the customer, which helps to improve transparency and visibility into the project status and progress. This can help to build trust and confidence with the customer and other stakeholders.
8. **Higher productivity:** Agile Development emphasizes teamwork and collaboration, which helps to improve productivity and reduce waste. This can lead to faster delivery of working software with fewer defects and rework.
9. **Improved project control:** Agile Development emphasizes continuous monitoring and measurement of project metrics, which helps to improve project control and decision-making. This can help the team to stay on track and make data-driven decisions throughout the development process.



In summary, Agile software development is a popular approach to software development that emphasizes collaboration, flexibility, and the delivery of working software in short iterations. It has several advantages over traditional software development approaches, including increased customer satisfaction, faster time-to-market, and reduced risk.

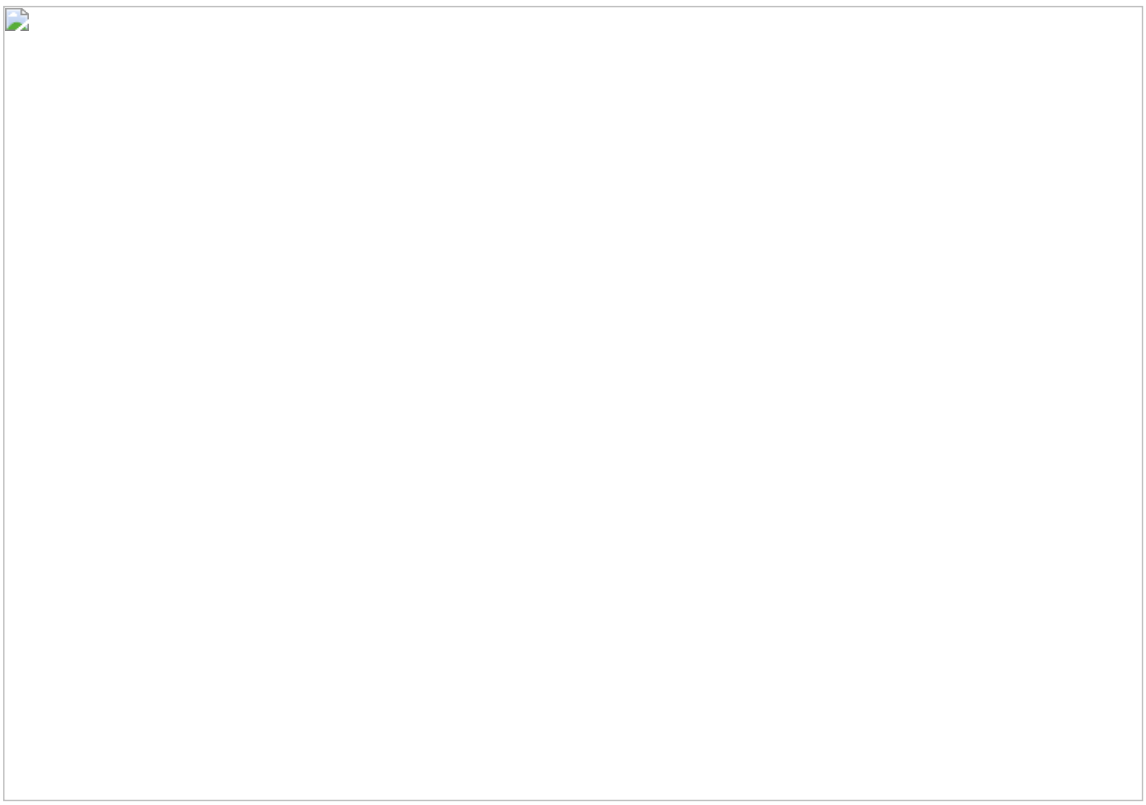
Incremental Process Model is also known as the Successive version model. This article focuses on discussing the Incremental Process Model in detail.

First, a simple working system implementing only a few basic features is built and then that is delivered to the customer. Then thereafter many successive iterations/ versions are implemented and delivered to the customer until the desired system is released.

A, B, and C are modules of Software Products that are incrementally developed and delivered.

Phases of incremental model

Requirements of Software are first broken down into several modules that can be incrementally constructed and delivered.



Phases of incremental model

1. **Requirement analysis:** In Requirement Analysis At any time, the plan is made just for the next increment and not for any kind of long-term plan. Therefore, it is easier to modify the version as per the needs of the customer.
2. **Design & Development:** At any time, the plan is made just for the next increment and not for any kind of long-term plan. Therefore, it is easier to modify the version as per the needs of the customer. The Development Team first undertakes to develop core features (these do not need services from other features) of the system. Once the core features are fully developed, then these are refined to increase levels of capabilities by

adding new functions in Successive versions. Each incremental version is usually developed using an iterative waterfall model of development.

3. **Deployment and Testing:** After Requirements gathering and specification, requirements are then split into several different versions starting with version 1, in each successive increment, the next version is constructed and then deployed at the customer site. in development and Testing the product is checked and tested for the actual process of the model.
4. **Implementation:** In implementation After the last version (version n), it is now deployed at the client site.

Requirement Process Model

Requirement Process Model

Types of Incremental Model

1. Staged Delivery Model
2. Parallel Development Model

1. Staged Delivery Model

Construction of only one part of the project at a time.

Staged Delivery Model

2. Parallel Development Model

Different subsystems are developed at the same time. It can decrease the calendar time needed for the development, i.e. TTM (Time to Market) if enough resources are available.

Parallel Development Model

When to use the Incremental Process Model

1. Funding Schedule, Risk, Program Complexity, or need for early realization of benefits.
2. When Requirements are known up-front.
3. When Projects have lengthy development schedules.
4. Projects with new Technology.
   - Error Reduction (core modules are used by the customer from the beginning of the phase and then these are tested thoroughly).
   - Uses divide and conquer for a breakdown of tasks.
   - Lowers initial delivery cost.
   - Incremental Resource Deployment.
5. Requires good planning and design.
6. The total cost is not lower.
7. Well-defined module interfaces are required.

Characteristics of Incremental Process Model

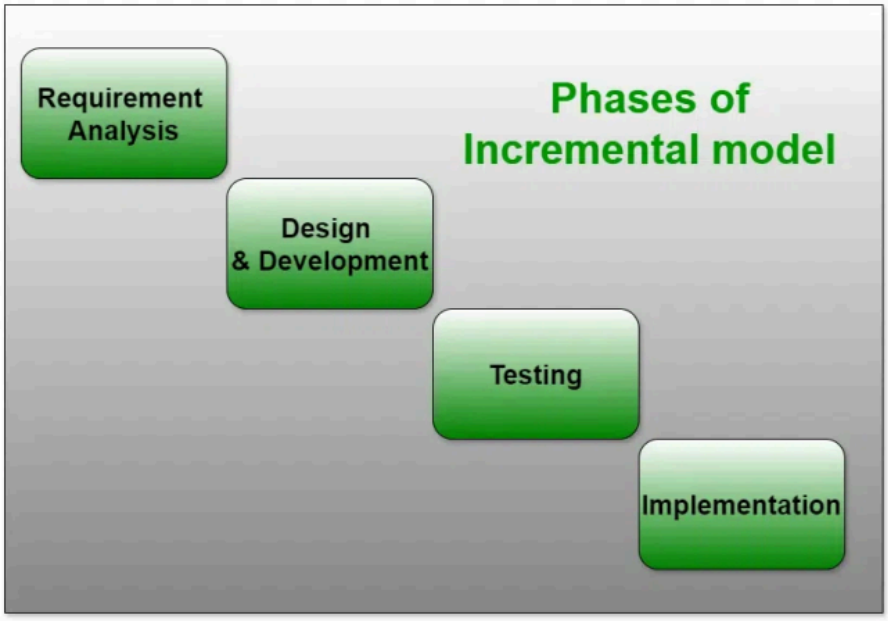
1. System development is divided into several smaller projects.
2. To create a final complete system, partial systems are constructed one after the other.
3. Priority requirements are addressed first.
4. The requirements for that increment are frozen once they are created.

Advantages of the Incremental Process Model

1. Prepares the software fast.
2. Clients have a clear idea of the project.
3. Changes are easy to implement.
4. Provides risk handling support, because of its iterations.
5. Adjusting the criteria and scope is flexible and less costly.
6. Comparing this model to others, it is less expensive.
7. The identification of errors is simple.

Disadvantages of the Incremental Process Model

1. A good team and proper planned execution are required.
2. Because of its continuous iterations the cost increases.
3. Issues may arise from the system design if all needs are not gathered upfront throughout the program lifecycle.
4. Every iteration step is distinct and does not flow into the next.
5. It takes a lot of time and effort to fix an issue in one unit if it needs to be corrected in all the units.



V-Model

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

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.

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.

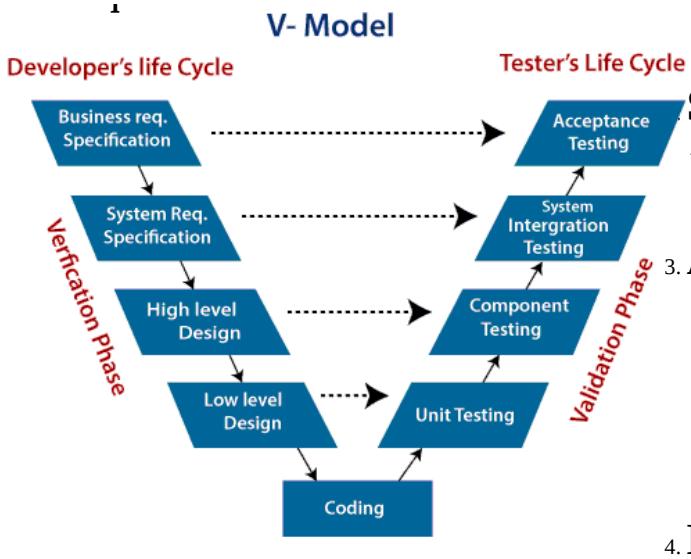
Disadvantage (Cons) of V-Model:

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.

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.

Specialised Process Models

Specialized process models take on many of the characteristics of one or more of the traditional models

However, these models tend to be applied when a specialized or narrowly defined software engineering

approach is chosen.

There are 3 types of specialized process models:

1. Component Based Development

2. Formal Methods Model

3. Aspect Oriented Software development

1. Component Based Development : Commercial off-the-shelf (COTS) software

components, developed by vendors who offer them as products, provide targeted

functionality with well-defined interfaces that enable the component to be integrated into

the software that is to be built. The component-based development model incorporates

many of the characteristics of the spiral model. It is evolutionary in nature, demanding an

iterative approach to the creation of software. However, the component-based

development model constructs applications from prepackaged software component.

Modeling and construction activities begin with the identification of candidate components.

These components can be designed as either conventional software modules or object-

oriented classes or packages of classes. Regardless of the technology that is used to create

the components, the component-based development model incorporates the following

steps:

1. Available component-based products are researched and evaluated for the application

domain in question.

2. Component integration issues are considered.

3. A software architecture is designed to accommodate the components.

4. Components are integrated into the architecture.

5. Comprehensive testing is conducted to ensure proper functionality

The component-based development model leads to software reuse, and reusability provides

software engineers with a number of measurable benefits. software engineering team can

achieve a reduction in development cycle time as well as a reduction in project cost if

component reuse becomes part of your culture.

2. Formal Methods Model : The formal methods model encompasses a set of

activities that leads to formal mathematical specification of computer software. Formal

methods enable to specify, develop, and verify a computer-based system by applying a

rigorous, mathematical notation. A variation on this approach, called cleanroom software

engineering is currently applied by some software development organizations. When formal

methods are used during development, they provide a mechanism for eliminating many of

the problems that are difficult to overcome using other software engineering paradigms.

Ambiguity, incompleteness, and inconsistency can be discovered and corrected more easily,

through the application of mathematical analysis. When formal methods are used during

design, they serve as a basis for program verification and therefore enable you to discover

and correct errors that might otherwise go undetected. The formal methods model offers

the promise of defect-free software.There are some of the disadvantages too:

• The development of formal models is currently quite time consuming and expensive.

• Because few software developers have the necessary background to apply formal

methods, extensive training is required.

• It is difficult to use the models as a communication mechanism for technically

unsophisticated customers

3. Aspect Oriented Software Development : Regardless of the software process

that is chosen, the builders of complex software invariably implement a set of localized

features, functions, and information content. These localized software characteristics are

modeled as components and then constructed within the context of a system architecture.

As modern computer-based systems become more sophisticated certain concerns span the

entire architecture. Some concerns are high-level properties of a system, Other concerns

affect functions, while others are systemic.

When concerns cut across multiple system functions, features, and information, they are

often referred to as crosscutting concerns. Aspectual requirements define those

crosscutting concerns that have an impact across the software architecture. Aspect-oriented

software development (AOSD), often referred to as aspect-oriented programming (AOP), is

a relatively new software engineering paradigm that provides a process and methodological

approach for defining, specifying, designing, and constructing aspects.

A distinct aspect-oriented process has not yet matured. However, it is likely that such a

process will adopt characteristics of both evolutionary and concurrent process models. The

evolutionary model is appropriate as aspects are identified and then constructed. The

parallel nature of concurrent development is essential because aspects are engineered

independently of localized software components and yet, aspects have a direct impact on

these components. It is essential to instantiate asynchronous communication between the

software process activities applied to the engineering and construction of aspects and

components.