Software engineering unit 1

Software Engineering (Dr. A.P.J. Abdul Kalam Technical University)

Scan to open on Studocu

Software is a program or set of programs containing instructions that provide desired functionality. Engineering is the process of designing and building something that serves a particular purpose and finds a cost-effective solution to problems.

What is Software Engineering?

Software Engineering is the process of designing, developing, testing, and maintaining software. It is a systematic and disciplined approach to software development that aims to create high-quality, reliable, and maintainable software.

1. Software engineering includes a variety of techniques, tools, and methodologies, including requirements analysis, design, testing, and maintenance.
2. It is a rapidly evolving field, and new tools and technologies are constantly being developed to improve the software development process.
3. By following the principles of software engineering and using the appropriate tools and methodologies, software developers can create high-quality, reliable, and maintainable software that meets the needs of its users.
4. Software Engineering is mainly used for large projects based on software systems rather than single programs or applications.
5. The main goal of Software Engineering is to develop software applications for improving quality, budget, and time efficiency.
6. Software Engineering ensures that the software that has to be built should be consistent, correct, also on budget, on time, and within the required requirements.

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 in an efficient way.
2. **Reliability:** It provides the assurance 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.

Components of Software :

There are three components of the software: These are : Program, Documentation, and Operating Procedures.

1. **Program –**
   A computer program is a list of instructions that tell a computer what to do.

2. **Documentation –**
   Source information about the product contained in design documents, detailed code comments, etc.

3. **Operating Procedures –**
   Set of step-by-step instructions compiled by an organization to help workers carry out complex routine operations.
4. **Code**: the instructions that a computer executes in order to perform a specific task or set of tasks.
5. **Data**: the information that the software uses or manipulates.
6. **User** interface: the means by which the user interacts with the software, such as buttons, menus, and text fields.
7. **Libraries**: pre-written code that can be reused by the software to perform common tasks.
8. **Documentation**: information that explains how to use and maintain the software, such as user manuals and technical guides.
9. **Test cases**: a set of inputs, execution conditions, and expected outputs that are used to test the software for correctness and reliability.
10. **Configuration files**: files that contain settings and parameters that are used to configure the software to run in a specific environment.
11. **Build and deployment** scripts: scripts or tools that are used to build, package, and deploy the software to different environments.
12. **Metadata**: information about the software, such as version numbers, authors, and copyright information.

All these components are important for software development, testing and deployment.

There are four basic key process activities:

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 users needs.

4. **Software Evolution –**
   It is a process of developing software initially, then timely updating it for various reasons.

Similarities Between Software Engineering Process and Conventional Engineering Process

- Both Software Engineering and Conventional Engineering Processes become automated after some time.
- Both these processes are making our day-to-day place better.
- Both these processes have a fixed working time.
- Both processes must consist of deeper knowledge.

Difference Between Software Engineering Process and Conventional Engineering Process

Software Engineering Process	Conventional Engineering Process
Software Engineering Process is a process that majorly involves computer science, information technology, and discrete mathematics.	The conventional Engineering Process is a process that majorly involves science, mathematics, and empirical knowledge.
It is mainly related to computers, programming, and writing codes for building applications.	It is about building cars, machines, hardware, buildings, etc.
In Software Engineering Process construction and development cost is low.	In Conventional Engineering Process construction and development cost is high.
It can involve the application of new and untested elements in software	It usually applies only known and tested principles to meet product

Software Engineering Process	Conventional Engineering Process
projects.	requirements.
In Software Engineering Process, most development effort goes into building new designs and features.	In Conventional Engineering Process, most development efforts are required to change old designs.
It majorly emphasizes quality.	It majorly emphasizes mass production.
Product development develops intangible products (software).	Product development develops tangible products (e.g. bridges, buildings).
Design requirements may change throughout the development process.	Design Requirements are typically well-defined upfront.
Testing is an integral part of the development process.	Testing occurs mainly after product completion.
Prototyping is common and helps to refine requirements.	Prototyping is less common due to cost and time.
Maintenance and updates are necessary to keep software relevant.	Maintenance is typically scheduled or reactive.
Software development often involves complex logic and algorithms.	Conventional engineering may have more complex physical properties to deal with.
Software development often follows established standards and frameworks.	Conventional engineering may have well-established regulations and standards.

Software Engineering Process	Conventional Engineering Process
Software development is typically less expensive to start, but costs may increase with maintenance and updates.	Conventional engineering may be more expensive to start due to materials and construction but may have lower maintenance costs.
Agile methodologies are commonly used in software development.	Conventional engineering may use more traditional project management approaches.

What is Software Quality?

Software Quality shows how good and reliable a product is. To convey an associate degree example, think about functionally correct software. It performs all functions as laid out in the SRS document. But, it has an associate degree virtually unusable program. even though it should be functionally correct, we tend not to think about it to be a high-quality product.

Another example is also that of a product that will have everything that the users need but has an associate degree virtually incomprehensible and not maintainable code. Therefore, the normal construct of quality as “fitness of purpose” for code merchandise isn’t satisfactory.

Factors of Software Quality

The modern read of high-quality associates with software many quality factors like the following:

1. **Portability:** A software is claimed to be transportable, if it may be simply created to figure in several package environments, in several machines, with alternative code merchandise, etc.
2. **Usability:** A software has smart usability if completely different classes of users (i.e. knowledgeable and novice users) will simply invoke the functions of the merchandise.
3. **Reusability:** A software has smart reusability if completely different modules of the merchandise will simply be reused to develop new merchandise.
4. **Correctness:** Software is correct if completely different needs as laid out in the SRS document are properly enforced.
5. **Maintainability:** A software is reparable, if errors may be simply corrected as and once they show up, new functions may be simply added to

the merchandise, and therefore the functionalities of the merchandise may be simply changed, etc

6. **Reliability.** Software is more reliable if it has fewer failures. Since software engineers do not deliberately plan for their software to fail, reliability depends on the number and type of mistakes they make. Designers can improve reliability by ensuring the software is easy to implement and change, by testing it thoroughly, and also by ensuring that if failures occur, the system can handle them or can recover easily.
7. **Efficiency.** The more efficient software is, the less it uses of CPU-time, memory, disk space, network bandwidth, and other resources. This is important to customers in order to reduce their costs of running the software, although with today's powerful computers, CPU time, memory and disk usage are less of a concern than in years gone by.

Software Crisis is a term used in computer science for the difficulty of writing useful and efficient computer programs in the required time. The software crisis was due to using the same workforce, same methods, and same tools even though rapidly increasing software demand, the complexity of software, and software challenges. With the increase in software complexity, many software problems arise because existing methods were insufficient.

If we use the same workforce, same methods, and same tools after the fast increase in software demand, software complexity, and software challenges, then there arise some issues like software budget problems, software efficiency problems, software quality problems, software management, and delivery problems, etc. This condition is called a S**oftware Crisis.**

Software Crisis

Causes of Software Crisis:

- The cost of owning and maintaining software was as expensive as developing the software.
- At that time Projects were running overtime.
- At that time Software was very inefficient.
- The quality of the software was low quality.
- Software often did not meet user requirements.
- The average software project overshoots its schedule by half.
- At that time Software was never delivered.
- Non-optimal resource utilization.
- Challenging to alter, debug, and enhance.
- The software complexity is harder to change.

Factors Contributing to Software Crisis:

- Poor project management.
- Lack of adequate training in software engineering.
- Less skilled project members.
- Low productivity improvements.

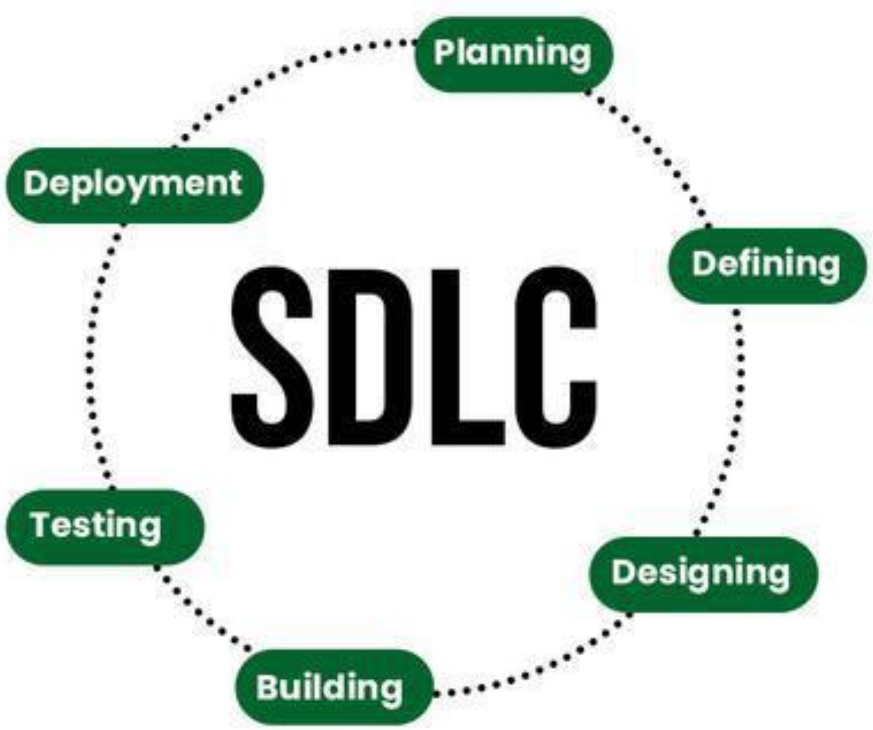
Solution of Software Crisis:

There is no single solution to the crisis. One possible solution to a software crisis is *Software Engineering* because software engineering is a systematic, disciplined, and quantifiable approach. For preventing software crises, there are some guidelines:

- Reduction in software over budget.
- The quality of the software must be high.
- Less time is needed for a software project.
- Experienced and skilled people working on the software project.
- Software must be delivered.
- Software must meet user requirements.

Software development Life cycle

Software development life cycle (SDLC) is a structured process that is used to design, develop, and test good-quality software. SDLC, or software development life cycle, is a methodology that defines the entire procedure of software development step-by-step.



Software Development Life Cycle (SDLC)

The goal of the SDLC life cycle model is to deliver high-quality, maintainable software that meets the user's requirements. SDLC in software engineering models outlines the plan for each stage so that each stage of the software

development model can perform its task efficiently to deliver the software at a low cost within a given time frame that meets users' requirements.

SDLC stands for **Software Development Life Cycle.**

What is Software Development Life Cycle (SDLC)?

SDLC is a process followed for software building within a software organization. SDLC consists of a precise plan that describes how to develop, maintain, replace, and enhance specific software. The life cycle defines a method for improving the quality of software and the all-around development process.

> *Note: If you also want to learn Product Life cycle Please refer this. PLC – Product Life Cycle*

Stages of the Software Development Life Cycle

SDLC specifies the task(s) to be performed at various stages by a software engineer or developer. It ensures that the end product is able to meet the customer's expectations and fits within the overall budget. Hence, it's vital for a software developer to have prior knowledge of this software development process.

The SDLC model **involves six phases or stages** while developing any software. SDLC is a collection of these six stages, and the stages of SDLC are as follows:

Stage-1: Planning and Requirement Analysis

Planning is a crucial step in everything, just as in software development. In this same stage, requirement analysis is also performed by the developers of the organization. This is attained from customer inputs, and sales department/market surveys.

The information from this analysis forms the building blocks of a basic project. The quality of the project is a result of planning. Thus, in this stage, the basic project is designed with all the available information.

Stage-2: Defining Requirements

In this stage, all the requirements for the target software are specified. These requirements get approval from customers, market analysts, and stakeholders.
This is fulfilled by utilizing SRS (Software Requirement Specification). This is a sort of document that specifies all those things that need to be defined and created during the entire project cycle.

Stage-3: Designing Architecture

SRS is a reference for software designers to come up with the best architecture for the software. Hence, with the requirements defined in SRS, multiple designs for the product architecture are present in the Design Document Specification (DDS).

This DDS is assessed by market analysts and stakeholders. After evaluating all the possible factors, the most practical and logical design is chosen for development.

Stage-4: Developing Product

At this stage, the fundamental development of the product starts. For this, developers use a specific programming code as per the design in the DDS. Hence, it is important for the coders to follow the protocols set by the association. Conventional programming tools like compilers, interpreters, debuggers, etc. are also put into use at this stage. Some popular languages like C/C++, Python, Java, etc. are put into use as per the software regulations.

Stage-5: Product Testing and Integration

After the development of the product, testing of the software is necessary to ensure its smooth execution. Although, minimal testing is conducted at every stage of SDLC. Therefore, at this stage, all the probable flaws are tracked, fixed, and retested. This ensures that the product confronts the quality requirements of SRS.

Documentation, Training, and Support: Software documentation is an essential part of the software development life cycle. A well-written document acts as a tool and means to information repository necessary to know about software processes, functions, and maintenance. Documentation also provides information about how to use the product. Training in an attempt to improve the current or future employee performance by increasing an employee's ability to work through learning, usually by changing his attitude and developing his skills and understanding.

Stage 6: Deployment and Maintenance of Products

After detailed testing, the conclusive product is released in phases as per the organization's strategy. Then it is tested in a real industrial environment. It is important to ensure its smooth performance. If it performs well, the organization sends out the product as a whole. After retrieving beneficial feedback, the company releases it as it is or with auxiliary improvements to make it further helpful for the customers. However, this alone is not enough. Therefore, along with the deployment, the product's supervision.

Software Development Life Cycle Models

To this day, we have more than 50 recognized SDLC models in use. But None of them is perfect, and each brings its favourable aspects and disadvantages for a specific software development project or a team.

In this article, We've listed the **top five** most popular SDLC models below.

1. Waterfall Model

It is the fundamental model of the software development life cycle. This is a very simple model. The waterfall model is not in practice anymore, but it is the basis for all other SDLC models. Because of its simple structure, the waterfall model is easier to use and provides a tangible output. In the waterfall model, once a phase seems to be completed, it cannot be changed, and due to this less flexible nature, the waterfall model is not in practice anymore.

2. Agile Model

The agile model was mainly designed to adapt to changing requests quickly. The main goal of the Agile model is to facilitate quick project completion. The agile model refers to a group of development processes. These processes have some similar characteristics but also possess certain subtle differences among themselves.

3. Iterative Model

In the iterative model, each cycle results in a semi-developed but deployable version; with each cycle, some requirements are added to the software, and the final cycle results in the software with the complete requirement specification.

4. Spiral Model

The spiral model is one of the most crucial SDLC models that provides support for risk handling. It has various spirals in its diagrammatic representation; the number of spirals depends upon the type of project. Each loop in the spiral structure indicates the *Phases of the Spiral model*.

5. V-Shaped Model

The V-shaped model is executed in a sequential manner in V-shape. Each stage or phase of this model is integrated with a testing phase. After every development phase, a testing phase is associated with it, and the next phase will start once the previous phase is completed, i.e., development & testing. It is also known as the verification or validation model.

6. Big Bang Model

The Big Bang model in SDLC is a term used to describe an informal and unstructured approach to software development, where there is no specific planning, documentation, or well-defined phases.

What is the need for SDLC?

SDLC is a method, approach, or process that is followed by a software development organization while developing any software. SDLC models were

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

Waterfall Model

The waterfall model is a software development model used in the context of large, complex projects, typically in the field of information technology. It is characterized by a structured, sequential approach to project management and software development.

The waterfall model is useful in situations where the project requirements are well-defined and the project goals are clear. It is often used for large-scale projects with long timelines, where there is little room for error and the project stakeholders need to have a high level of confidence in the outcome.

Features of Waterfall Model

1. **Sequential Approach**: The waterfall model involves a sequential approach to software development, where each phase of the project is completed before moving on to the next one.
2. **Document-Driven:** The waterfall model relies heavily on documentation to ensure that the project is well-defined and the project team is working towards a clear set of goals.
3. **Quality Control:** The waterfall model places a high emphasis on quality control and testing at each phase of the project, to ensure that the final product meets the requirements and expectations of the stakeholders.
4. **Rigorous Planning**: The waterfall model involves a rigorous planning process, where the project scope, timelines, and deliverables are carefully defined and monitored throughout the project lifecycle.

Overall, the waterfall model is used in situations where there is a need for a highly structured and systematic approach to software development. It can be effective in ensuring that large, complex projects are completed on time and within budget, with a high level of quality and customer satisfaction.

Phases of Waterfall Model

Waterfall Model is a classical software development methodology that was first introduced by Winston W. Royce in 1970. It is a linear and sequential approach to software development that consists of several phases that must be completed in a specific order.

The Waterfall Model has six phases:

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

2. Design Phase: 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.

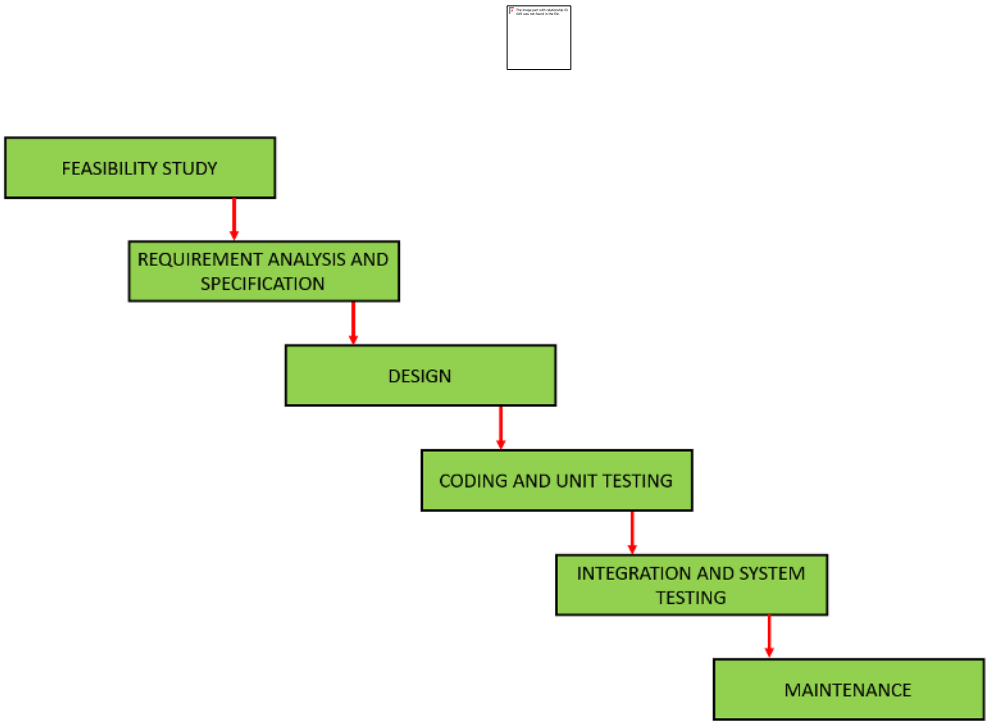
3. Implementation and Unit Testing: The implementation phase involves coding the software based on the design specifications. This phase also includes unit testing to ensure that each component of the software is working as expected.

4. Integration and System 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.

The classical waterfall model divides the life cycle into a set of phases. This model considers that one phase can be started after the completion of the previous phase. That is the output of one phase will be the input to the next phase. Thus the development process can be considered as a sequential flow in the waterfall. Here the phases do not overlap with each other. The different sequential phases of the classical waterfall model are shown in the below figure.



Phases of Classical Waterfall Model

Let us now learn about each of these phases in detail.

1. Feasibility Study

The main goal of this phase is to determine whether it would be financially and technically feasible to develop the software.
The feasibility study involves understanding the problem and then determining the various possible strategies to solve the problem. These different identified solutions are analyzed based on their benefits and drawbacks, The best solution is chosen and all the other phases are carried out as per this solution strategy.

2. Requirements Analysis and Specification

The aim of the requirement analysis and specification phase is to understand the exact requirements of the customer and document them properly. This phase consists of two different activities.

- **Requirement gathering and analysis:** Firstly all the requirements regarding the software are gathered from the customer and then the gathered requirements are analyzed. The goal of the analysis part is to remove incompleteness (an incomplete requirement is one in which some parts of the actual requirements have been omitted) and inconsistencies (an inconsistent requirement is one in which some part of the requirement contradicts some other part).
- **Requirement specification:** These analyzed requirements are documented in a software requirement specification (SRS) document. SRS document serves as a contract between the development team and customers. Any future dispute between the customers and the developers can be settled by examining the SRS document.

3. Design

The goal of this phase is to convert the requirements acquired in the SRS into a format that can be coded in a programming language. It includes high-level and detailed design as well as the overall software architecture. A Software Design Document is used to document all of this effort (SDD).

4. Coding and Unit Testing

In the coding phase software design is translated into source code using any suitable programming language. Thus each designed module is coded. The aim of the unit testing phase is to check whether each module is working properly or not.

5. Integration and System testing

Integration of different modules is undertaken soon after they have been coded and unit tested. Integration of various modules is carried out incrementally over

a number of steps. During each integration step, previously planned modules are added to the partially integrated system and the resultant system is tested. Finally, after all the modules have been successfully integrated and tested, the full working system is obtained and system testing is carried out on this.
System testing consists of three different kinds of testing activities as described below.

- **Alpha testing:** Alpha testing is the system testing performed by the development team.
- **Beta testing:** Beta testing is the system testing performed by a friendly set of customers.
- **Acceptance testing:** After the software has been delivered, the customer performed acceptance testing to determine whether to accept the delivered software or reject it.

6. Maintenance

Maintenance is the most important phase of a software life cycle. The effort spent on maintenance is 60% of the total effort spent to develop a full software. There are basically three types of maintenance.

- **Corrective Maintenance:** This type of maintenance is carried out to correct errors that were not discovered during the product development phase.
- **Perfective Maintenance:** This type of maintenance is carried out to enhance the functionalities of the system based on the customer's request.
- **Adaptive Maintenance:** Adaptive maintenance is usually required for porting the software to work in a new environment such as working on a new computer platform or with a new operating system.

Advantages of Classical 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:** 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:** Classical Waterfall model has very clear and well-understood milestones.
- **Properly Documented:** Processes, actions, and results are very well documented.
- **Reinforces Good Habits:** 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 Classical Waterfall Model

The Classical Waterfall Model suffers from various shortcomings, basically, 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 actually 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.
- **Not Suitable for Complex Projects:** The Waterfall Model is not well-suited for complex projects, as the linear and sequential nature of the model can make it difficult to manage multiple dependencies and interrelated components.

When to Use Waterfall Model?

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

- Only well-defined, unambiguous, and fixed requirements are employed with this paradigm.
- The definition of a product is constant.
- People understand technology.
- There are no unclear prerequisites.
- There are many resources with the necessary knowledge readily available.
- When it's a brief project.

The Waterfall approach involves little client engagement in the product development process. The product can only be shown to end consumers when it is ready.

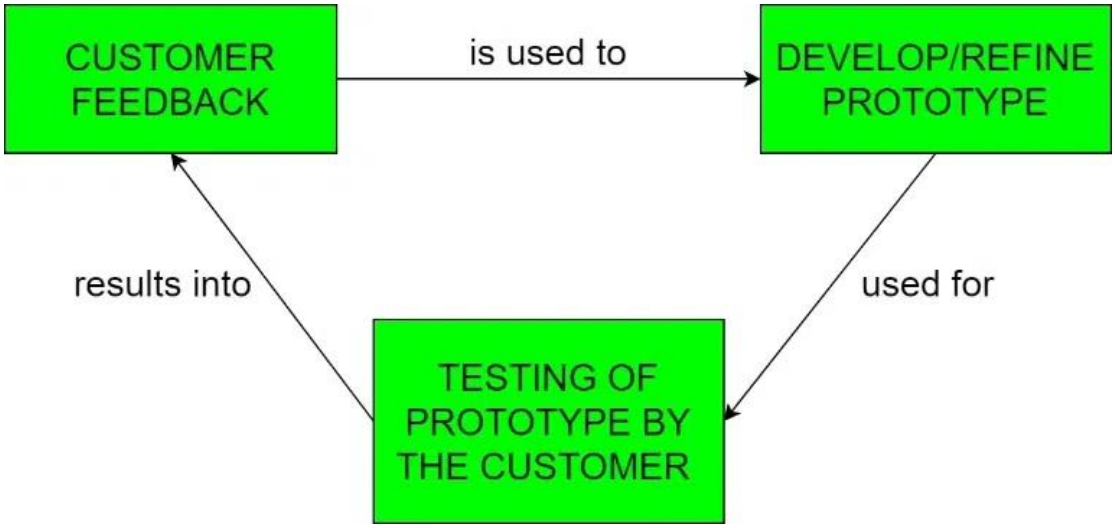
Applications of Classical Waterfall Model

- **Large-scale Software Development Projects:** The Waterfall Model is often used for large-scale software development projects, where a structured and sequential approach is necessary to ensure that the project is completed on time and within budget.
- **Safety-Critical Systems:** The Waterfall Model is often used in the development of safety-critical systems, such as aerospace or medical systems, where the consequences of errors or defects can be severe.
- **Government and Defense Projects:** The Waterfall Model is also commonly used in government and defense projects, where a rigorous and structured approach is necessary to ensure that the project meets all requirements and is delivered on time.
- **Projects with well-defined Requirements:** The Waterfall Model is best suited for projects with well-defined requirements, as the sequential nature of the model requires a clear understanding of the project objectives and scope.
- **Projects with Stable Requirements:** The Waterfall Model is also well-suited for projects with stable requirements, as the linear nature of the model does not allow for changes to be made once a phase has been completed.

Prototype Model

Prototyping is defined as the process of developing a working replication of a product or system that has to be engineered. It offers a small-scale facsimile of the end product and is used for obtaining customer feedback. The Prototyping concept is described below:

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



Prototyping Concept

In this process model, the system is partially implemented before or during the analysis phase thereby giving the customers an opportunity to see the product early in the life cycle. The process starts by interviewing the customers and developing the incomplete high-level paper model. This document is used to build the initial prototype supporting only the basic functionality as desired by the customer. Once the customer figures out the problems, the prototype is further refined to eliminate them. The process continues until the user approves the prototype and finds the working model to be satisfactory.

Steps Prototyping Model

Step 1: Requirement Gathering and Analysis: This is the initial step in designing a prototype model. In this phase, users are asked about what they expect or what they want from the system.

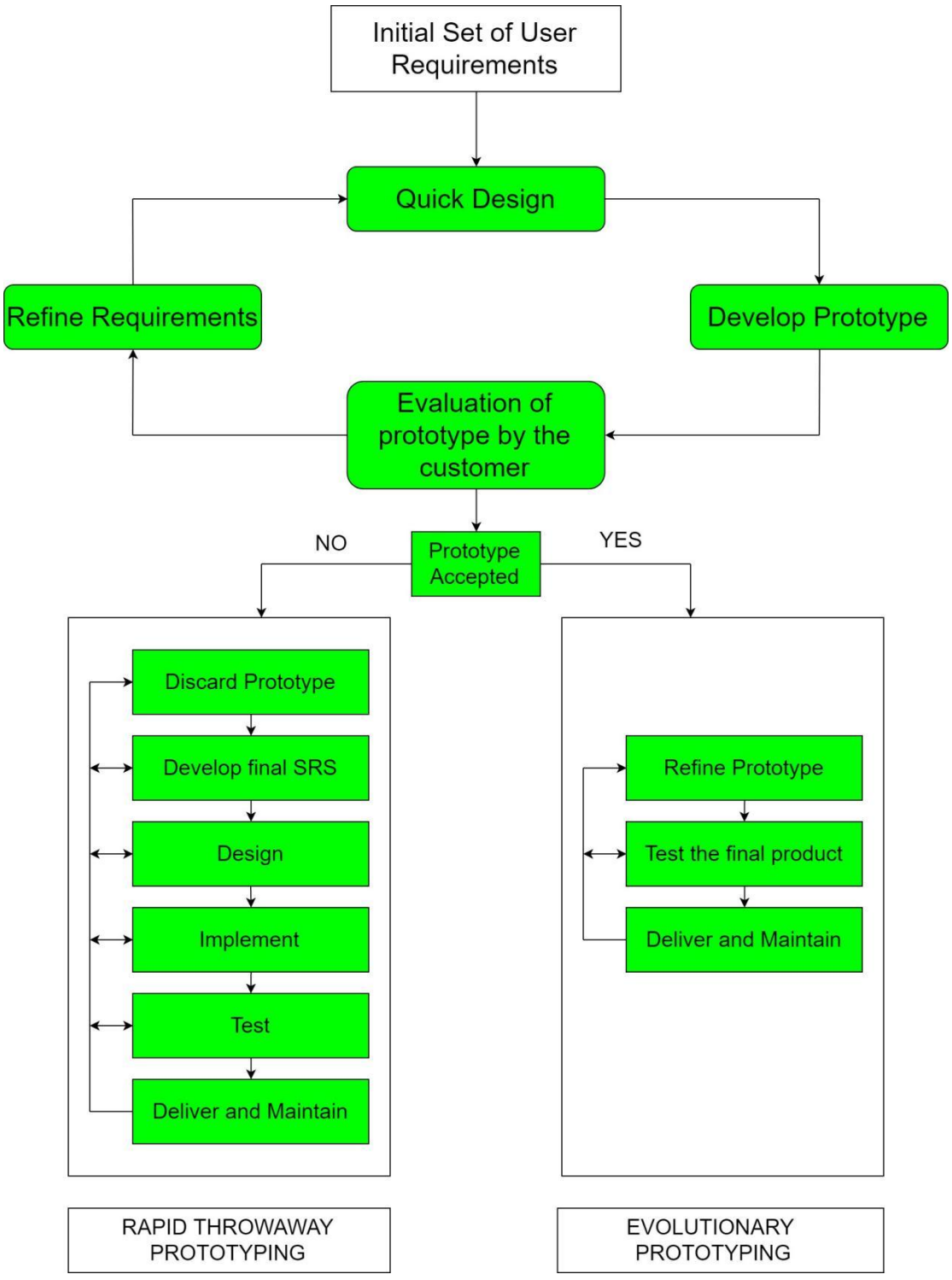
Step 2: Quick Design: This is the second step in Prototyping Model. This model covers the basic design of the requirement through which a quick overview can be easily described.

Step 3: Build a Prototype: This step helps in building an actual prototype from the knowledge gained from prototype design.

Step 4: Initial User Evaluation: This step describes the preliminary testing where the investigation of the performance model occurs, as the customer will tell the strength and weaknesses of the design, which was sent to the developer.

Step 5: Refining Prototype: If any feedback is given by the user, then improving the client's response to feedback and suggestions, the final system is approved.

Step 6: Implement Product and Maintain: This is the final step in the phase of the Prototyping Model where the final system is tested and distributed to production, here the program is run regularly to prevent failures.



Prototyping Models

Types of Prototyping Models

There are four types of Prototyping Models, which are described below.

- Rapid Throwaway Prototyping
- Evolutionary Prototyping
- Incremental Prototyping
- Extreme Prototyping

1. Rapid Throwaway Prototyping

This technique offers a useful method of exploring ideas and getting customer feedback for each of them. In this method, a developed prototype need not necessarily be a part of the ultimately accepted prototype. Customer feedback helps in preventing unnecessary design faults and hence, the final prototype developed is of better quality.

2. Evolutionary Prototyping

In this method, the prototype developed initially is incrementally refined on the basis of customer feedback till it finally gets accepted. In comparison to Rapid Throwaway Prototyping, it offers a better approach that saves time as well as effort. This is because developing a prototype from scratch for every iteration of the process can sometimes be very frustrating for the developers.

3. Incremental Prototyping

In this type of incremental prototyping, the final expected product is broken into different small pieces of prototypes and developed individually. In the end, when all individual pieces are properly developed, then the different prototypes are collectively merged into a single final product in their predefined order. It's a very efficient approach that reduces the complexity of the development process, where the goal is divided into sub-parts and each sub-part is developed individually. The time interval between the project's beginning and final delivery is substantially reduced because all parts of the system are prototyped and tested simultaneously. Of course, there might be the possibility that the pieces just do not fit together due to some lack of ness in the development phase – this can only be fixed by careful and complete plotting of the entire system before prototyping starts.

4. Extreme Prototyping

This method is mainly used for web development. It consists of three sequential independent phases:

1. In this phase, a basic prototype with all the existing static pages is presented in HTML format.
2. In the 2nd phase, Functional screens are made with a simulated data process using a prototype services layer.
3. This is the final step where all the services are implemented and associated with the final prototype.

This Extreme Prototyping method makes the project cycling and delivery robust and fast and keeps the entire developer team focused and centralized on product deliveries rather than discovering all possible needs and specifications and adding necessitated features.

Advantages of Prototyping Model

- The customers get to see the partial product early in the life cycle. This ensures a greater level of customer satisfaction and comfort.
- New requirements can be easily accommodated as there is scope for refinement.
- Missing functionalities can be easily figured out.
- Errors can be detected much earlier thereby saving a lot of effort and cost, besides enhancing the quality of the software.
- The developed prototype can be reused by the developer for more complicated projects in the future.
- Flexibility in design.
- Early feedback from customers and stakeholders can help guide the development process and ensure that the final product meets their needs and expectations.
- Prototyping can be used to test and validate design decisions, allowing for adjustments to be made before significant resources are invested in development.
- Prototyping can help reduce the risk of project failure by identifying potential issues and addressing them early in the process.
- Prototyping can facilitate communication and collaboration among team members and stakeholders, improving overall project efficiency and effectiveness.
- Prototyping can help bridge the gap between technical and non-technical stakeholders by providing a tangible representation of the product.

Disadvantages of the Prototyping Model

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

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

Applications of Prototyping Model

- The Prototyping Model should be used when the requirements of the product are not clearly understood or are unstable.
- The prototyping model can also be used if requirements are changing quickly.
- This model can be successfully used for developing user interfaces, high-technology software-intensive systems, and systems with complex algorithms and interfaces.
- The prototyping Model is also a very good choice to demonstrate the technical feasibility of the product.

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.

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.

What Are the Phases of 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

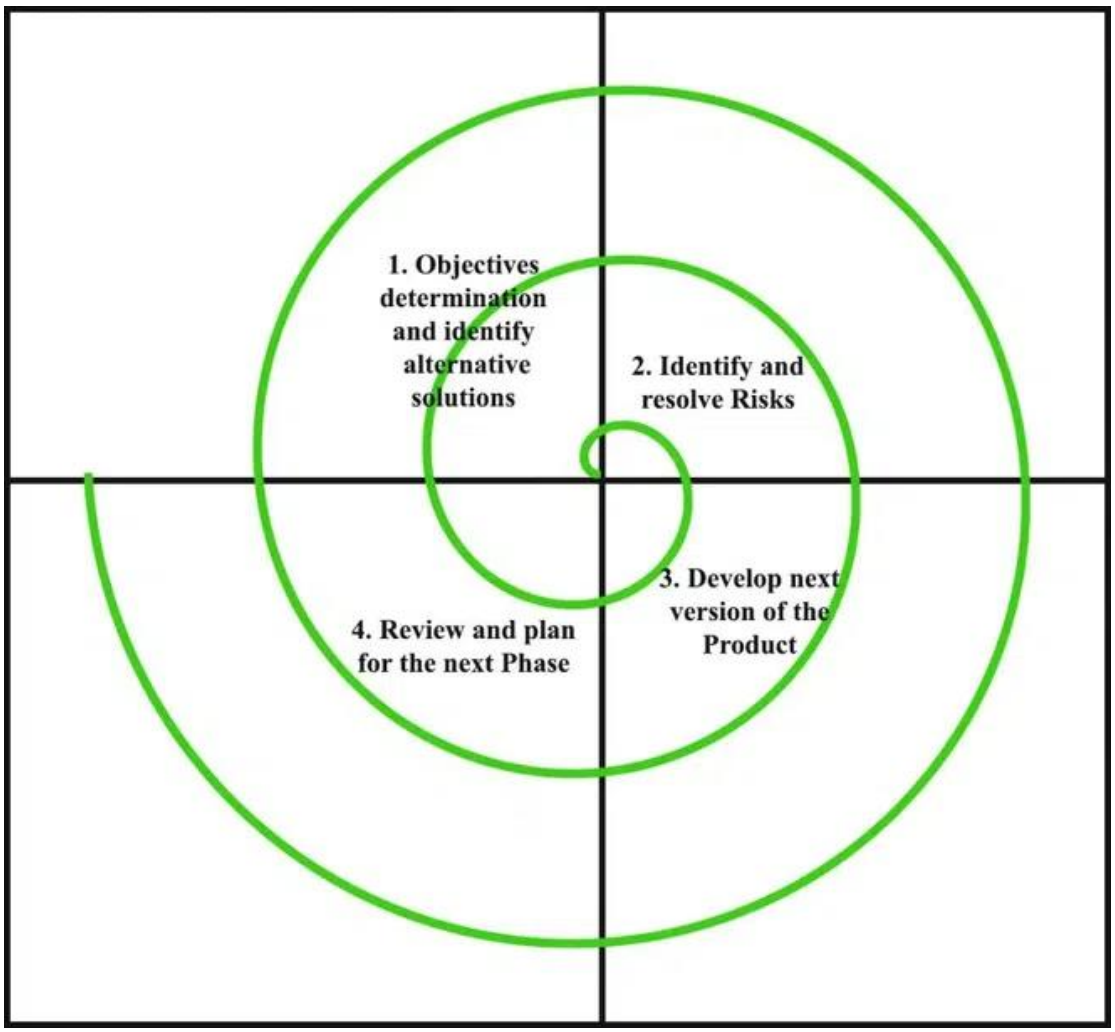
In the evaluation phase, the software is evaluated to determine if it meets the customer's requirements and if it is of high quality.

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.

Why Spiral Model is called Meta Model?

The Spiral model is called a Meta-Model because it subsumes all the other SDLC models. For example, a single loop spiral actually represents the Iterative Waterfall Model.

1. The spiral model incorporates the stepwise approach of the Classical Waterfall Model.
2. The spiral model uses the approach of the Prototyping Model by building a prototype at the start of each phase as a risk-handling technique.
3. Also, the spiral model can be considered as supporting the Evolutionary model – the iterations along the spiral can be considered as evolutionary levels through which the complete system is built.

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.

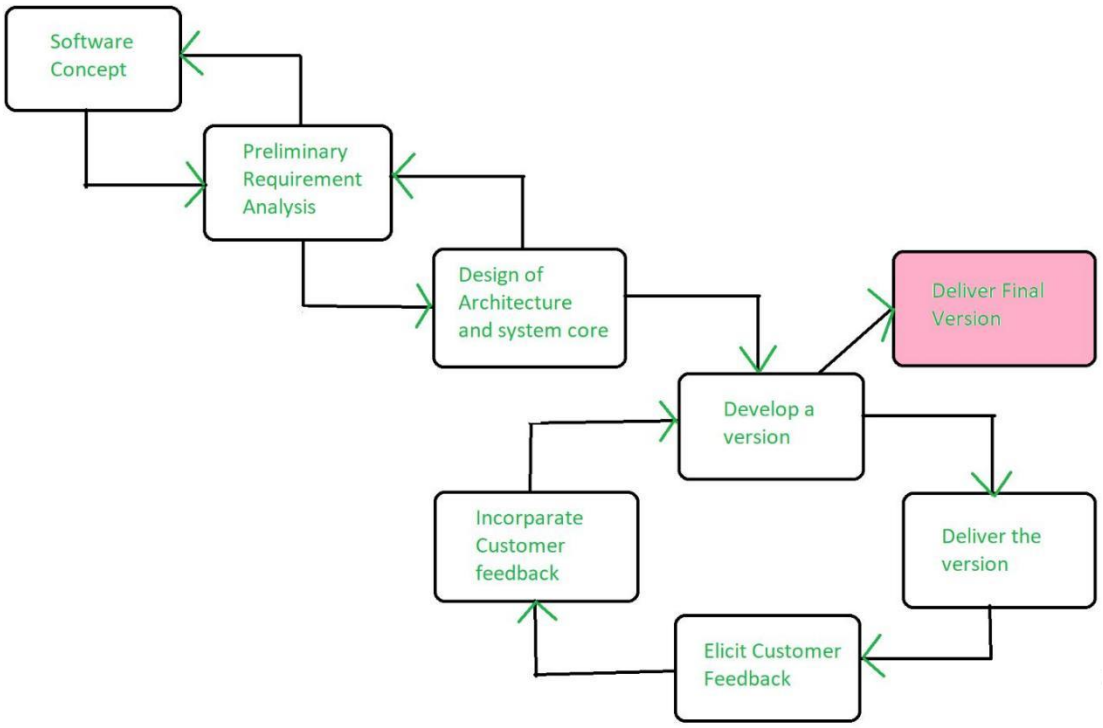
1. **Complex:** The Spiral Model is much more complex than other SDLC models.
2. **Expensive:** Spiral Model is not suitable for small projects as it is expensive.
3. **Too much dependability on Risk Analysis:** The successful completion of the project is very much dependent on Risk Analysis. Without 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.

Evolutionary Development Lifecycle

The Evolutionary development model divides the development cycle into smaller, incremental waterfall models in which users can get access to the product at the end of each cycle.

1. Feedback is provided by the users on the product for the planning stage of the next cycle and the development team responds, often by changing the product, plan, or process.
2. Therefore, the software product evolves with time.
3. All the models have the disadvantage that the duration of time from the start of the project to the delivery time of a solution is very high.
4. The evolutionary model solves this problem with a different approach.
5. The evolutionary model suggests breaking down work into smaller chunks, prioritizing them, and then delivering those chunks to the customer one by one.
6. The number of chunks is huge and is the number of deliveries made to the customer.
7. The main advantage is that the customer's confidence increases as he constantly gets quantifiable goods or services from the beginning of the project to verify and validate his requirements.

8. The model allows for changing requirements as well as all work is broken down into maintainable work chunks.



Evolutionary Model

Application of Evolutionary Model

1. It is used in large projects where you can easily find modules for incremental implementation. Evolutionary model is commonly used when the customer wants to start using the core features instead of waiting for the full software.
2. Evolutionary model is also used in object oriented software development because the system can be easily portioned into units in terms of objects.

Necessary Conditions for Implementing this Model

1. Customer needs are clear and been explained in deep to the developer team.
2. There might be small changes required in separate parts but not a major change.
3. As it requires time, so there must be some time left for the market constraints.
4. Risk is high and continuous targets to achieve and report to customer repeatedly.
5. It is used when working on a technology is new and requires time to learn.

Advantages Evolutionary Model

1. **Adaptability to Changing Requirements:** Evolutionary models work effectively in projects when the requirements are ambiguous or change often. They support adjustments and flexibility along the course of development.

2. **Early and Gradual Distribution:** Functional components or prototypes can be delivered early thanks to incremental development. Faster user satisfaction and feedback may result from this.
3. **User Commentary and Involvement:** Evolutionary models place a strong emphasis on ongoing user input and participation. This guarantees that the software offered closely matches the needs and expectations of the user.
4. **Improved Handling of Difficult Projects:** Big, complex tasks can be effectively managed with the help of evolutionary models. The development process is made simpler by segmenting the project into smaller, easier-to-manage portions.

Disadvantages Evolutionary Model

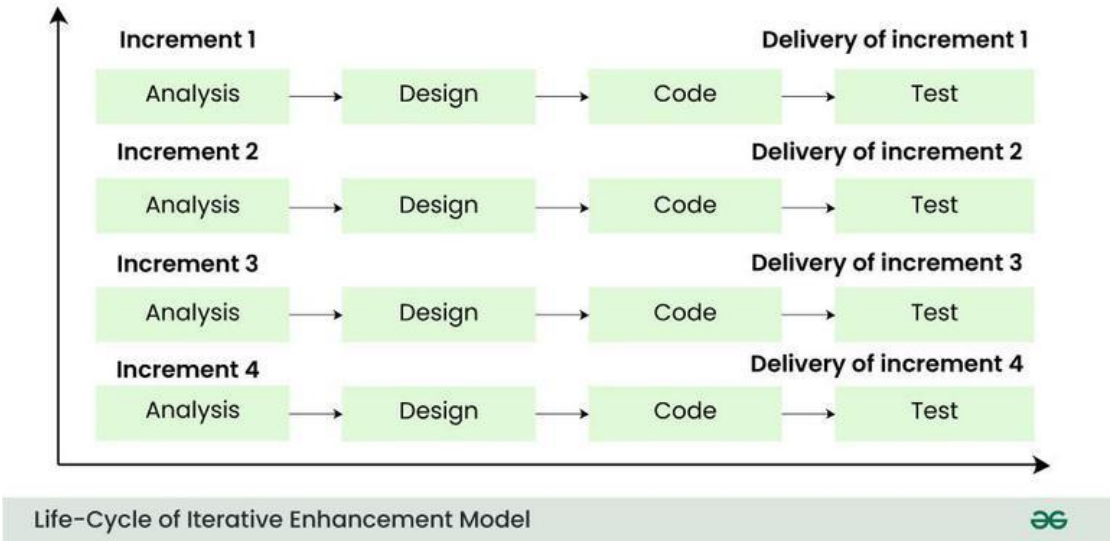
1. **Communication Difficulties:** Evolutionary models require constant cooperation and communication. The strategy may be less effective if there are gaps in communication or if team members are spread out geographically.
2. **Dependence on an Expert Group:** A knowledgeable and experienced group that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

Iterative Enhancement model:

In software development, the Iterative Enhancement Model stands out due to its incremental and iterative nature, it is also known as an **incremental model**. This approach focuses on incremental development and improvement rather than trying to complete a software product in a single phase. This model is based on segmenting the project into smaller units, or iterations, with a set of tasks completed.

What is the Iterative Enhancement Model?

Software development uses a dynamic and adaptable method called the iterative enhancement Model. The iterative enhancement model encourages a software product's ongoing **evolution and improvement**. This methodology is noticeable due to its concentration on **adaptability, flexibility and change responsiveness**. It makes it easier for a product to evolve because it gives developers the freedom to progressively enhance the software, making sure that it complies with evolving specifications, user demands, and market demands. This helps products evolve more easily.



Life-Cycle of Iterative Enhancement Model

The Iterative Enhancement Model creates an environment where development teams can more effectively adjust to changing requirements by segmenting the software development process into smaller, more manageable parts. Every iteration improves on the one before it, adding new features and fixing problems found in earlier stages. Members of the team, stakeholders and end users are encouraged to collaborate and communicate continuously to make sure the software meets changing needs and expectations. Until the software is finished being built, the iteration process is carried out, which involves giving the user the increments.

Advantages of Iterative Enhancement Model

- Adaptation to changing requirements is made possible by its flexibility in accomodating modifications and improvement during each iteration.
- Early software iterations provide clients with functional portions of the product, facilitating prompt feedback and validation.
- Problems and risks can be identified and addressed early in the developement process, reduces chances of issue for future stages.
- Feedback and constant client involvement are encouraged to make sure the finished product lives up to user expectations.
- Every iteration is put through testing and improvement, leading to higher quality product.

Disadvantages of Iterative Enhancement Model

- Especially in larger projects, managing several iterations at once can add complexity.
- Higher cost
- Due to constant changes, there may be delays in documentation, making it more difficult to maintain comprehensive documentation.
- Continuous customer engagement may not be possible in all scenarios, which impacts the effectiveness of the model.