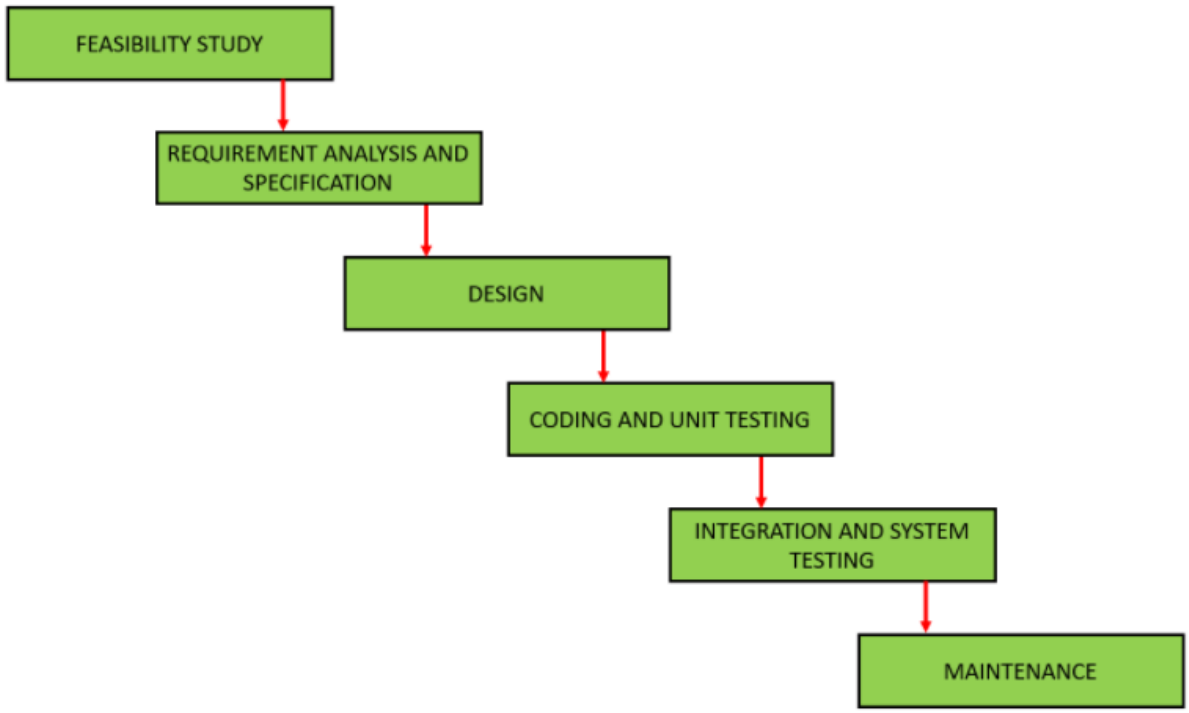
Classical Waterfall Model

Classical waterfall model is the basic **software development life cycle** model. It is very simple but idealistic. Earlier this model was very popular but nowadays it is not used. But it is very important because all the other software development life cycle models are based on the classical waterfall model.

Classical waterfall model divides the life cycle into a set of phases. This model considers that one phase can be started after 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:



Let us now learn about each of these phases in brief details:

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 determine 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 (inconsistent requirement is one in which some part of the requirement contradicts with some other part).
   - **Requirement specification:** These analyzed requirements are documented in a software requirement specification (SRS) document. SRS document serves as a contract between development team and customers. Any future dispute between the customers and the developers can be settled by examining the SRS document.
3. **Design**: The aim of the design phase is to transform the requirements specified in the SRS document into a structure that is suitable for implementation in some programming language.
4. **Coding and Unit testing**: In 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 are 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 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 the acceptance testing to determine whether to accept the delivered software or to reject it.
6. **Maintenance:** Maintenance is the most important phase of a software life cycle. The effort spent on maintenance is the 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 work on a new computer platform or with a new operating system.

Advantages of Classical Waterfall Model

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

- This model is very simple and is easy to understand.
- Phases in this model are processed one at a time.
- Each stage in the model is clearly defined.
- This model has very clear and well understood milestones.
- Process, actions and results are very well documented.
- Reinforces good habits: define-before- design, design-before-code.
- This model works well for smaller projects and projects where requirements are well understood.

Drawbacks of Classical Waterfall Model

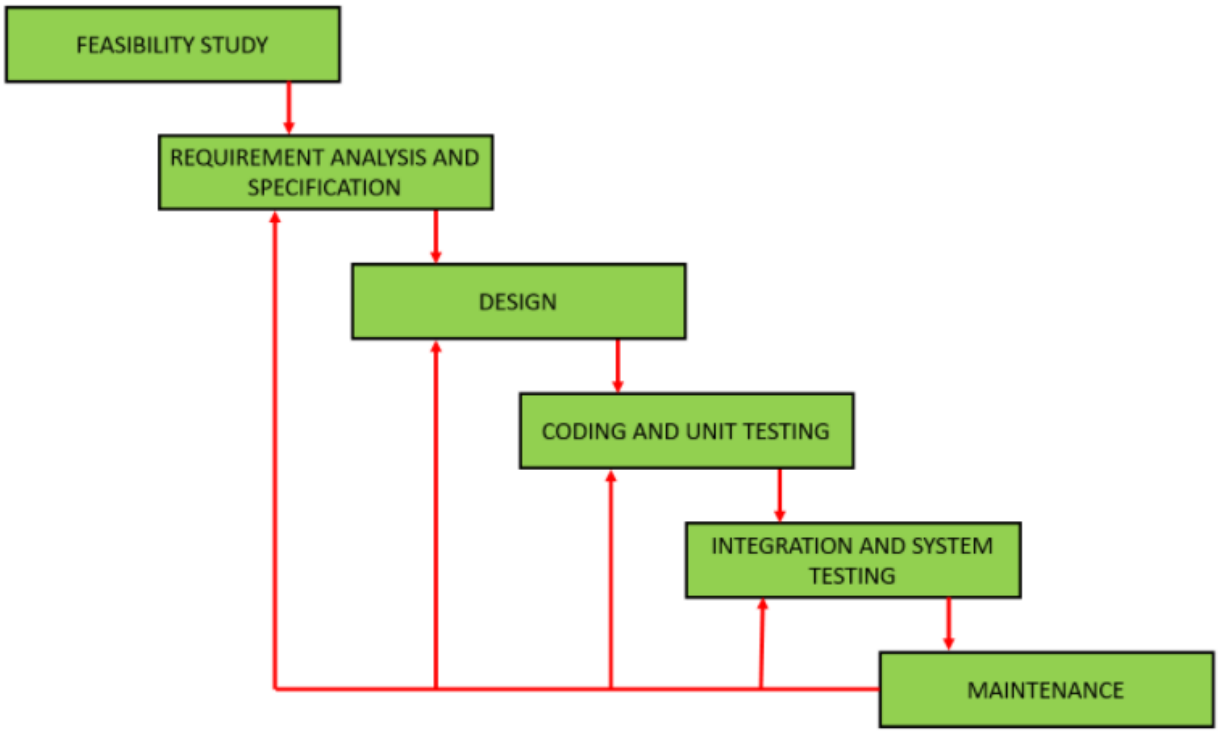
Classical waterfall model suffers from various shortcomings, basically we can't use it in real projects, but we use other software development lifecycle models which are based on the classical waterfall model. Below are some major drawbacks of this model:

- **No feedback path:** In 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 phases. 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 customers' 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 new phase can start only after the completion of the previous phase. But in real projects, this can't be maintained. To increase the efficiency and reduce the cost, phases may overlap.

Iterative Waterfall Model

In a practical software development project, the classical waterfall model is hard to use. So, the Iterative waterfall model can be thought of as incorporating the necessary changes to the classical waterfall model to make it usable in practical software development projects. It is almost the same as the classical waterfall model except some changes are made to increase the efficiency of the software development.
The iterative waterfall model provides feedback paths from every phase to its preceding phases, which is the main difference from the classical waterfall model.

Feedback paths introduced by the iterative waterfall model are shown in the figure below.



When errors are detected at some later phase, these feedback paths allow correcting errors committed by programmers during some phase. The feedback paths allow the phase to be reworked in which errors are committed and these changes are reflected in the later phases. But, there is no feedback path to the stage – feasibility study, because once a project has been taken, does not give up the project easily.

It is good to detect errors in the same phase in which they are committed. It reduces the effort and time required to correct the errors.

Phase Containment of Errors :

The principle of detecting errors as close to their points of commitment as possible is known as Phase containment of errors.

Advantages of Iterative Waterfall Model :

- **Feedback Path –**
  In the classical waterfall model, there are no feedback paths, so there is no mechanism for error correction. But in the iterative waterfall model feedback path from one phase to its preceding phase allows correcting the errors that are committed and these changes are reflected in the later phases.
- **Simple –**
  Iterative waterfall model is very simple to understand and use. That's why it is one of the most widely used software development models.
- **Cost-Effective –**
  It is highly cost-effective to change the plan or requirements in the model. Moreover, it is best suited for agile organizations.
- **Well-organized –**
  In this model, less time is consumed on documenting and the team can spend more time on development and designing.

Drawbacks of Iterative Waterfall Model :

- **Difficult to incorporate change requests –**
  The major drawback of the iterative waterfall model is that all the requirements must be clearly stated before starting the development phase. Customers may change requirements after some time but the iterative waterfall model does not leave any scope to incorporate change requests that are made after the development phase starts.

- **Incremental delivery not supported –**
  In the iterative waterfall model, the full software is completely developed and tested before delivery to the customer. There is no scope for any intermediate delivery. So, customers have to wait a long for getting the software.

- **Overlapping of phases not supported –**
  Iterative waterfall model assumes that one phase can start after completion of the previous phase, But in real projects, phases may overlap to reduce the effort and time needed to complete the project.

- **Risk handling not supported –**
  Projects may suffer from various types of risks. But, the Iterative waterfall model has no mechanism for risk handling.

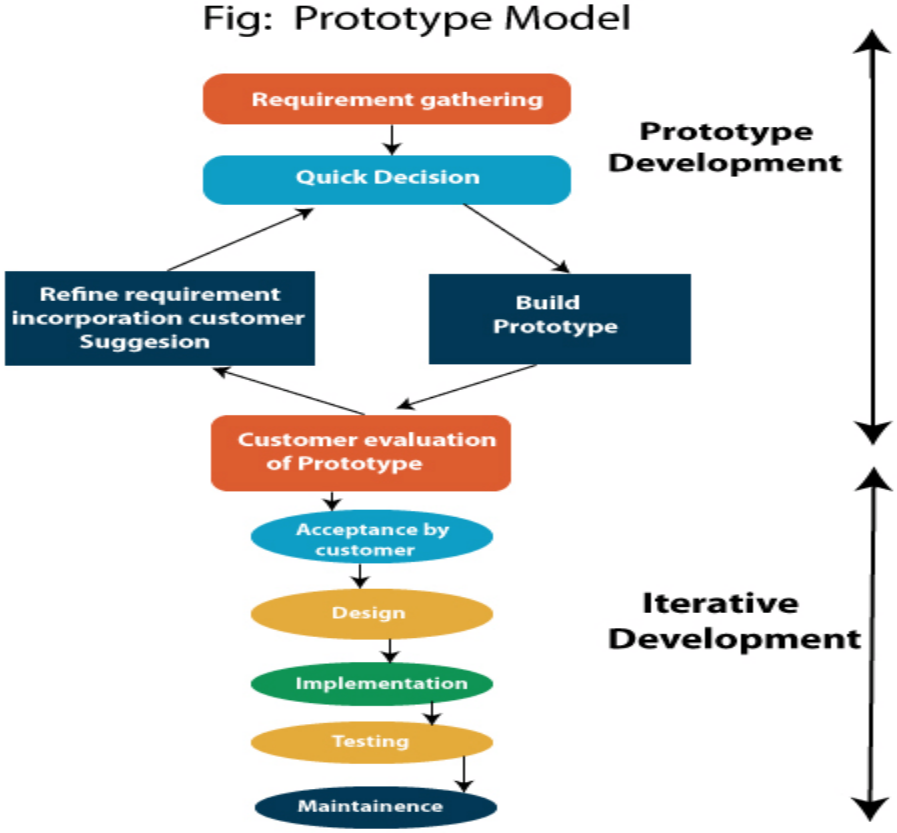
- **Limited customer interactions –**

Customer interaction occurs at the start of the project at the time of requirement gathering and at project completion at the time of software delivery. These fewer interactions with the customers may lead to many problems as the finally developed software may differ from the customers' actual requirements.

Prototype Model

The prototype model requires that before carrying out the development of actual software, a working prototype of the system should be built. A prototype is a toy implementation of the system. A prototype usually turns out to be a very crude version of the actual system, possible exhibiting limited functional capabilities, low reliability, and inefficient performance as compared to actual software. In many instances, the client only has a general view of what is expected from the software product. In such a scenario where there is an absence of detailed information regarding the input to the system, the processing needs, and the output requirement, the prototyping model may be employed.

Fig: Prototype Model



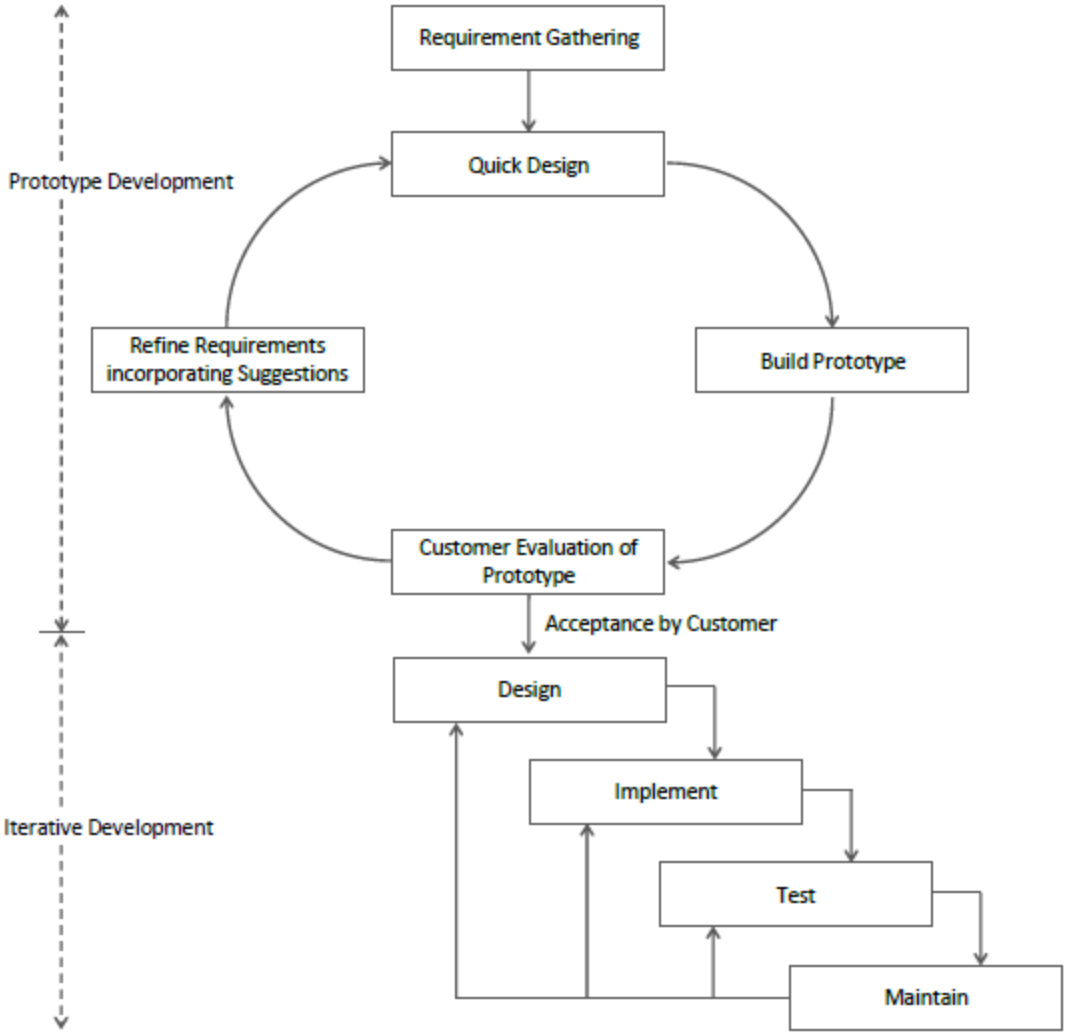


Figure 3: Prototype Model of Software Development

Steps of Prototype Model

1. Requirement Gathering and Analyst
2. Quick Decision
3. Build a Prototype
4. Assessment or User Evaluation
5. Prototype Refinement
6. Engineer Product

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.

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.

Use –
The Prototyping Model should be used when the requirements of the product are not clearly understood or are unstable. It 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. It is also a very good choice to demonstrate the technical feasibility of the product.

Advantage of Prototype Model

1. Reduce the risk of incorrect user requirement
2. Good where requirement are changing/uncommitted
3. Regular visible process aids management
4. Support early product marketing
5. Reduce Maintenance cost.
6. Errors can be detected much earlier as the system is made side by side.
7. The customers get to see the partial product early in the life cycle. This ensures a greater level of customer satisfaction and comfort.
8. The developed prototype can be reused by the developer for more complicated projects in the future.
9. Flexibility in design.
10. Missing functionalities can be easily figured out.

Disadvantage of Prototype Model

1. An unstable/badly implemented prototype often becomes the final product.

2. Require extensive customer collaboration
    - Costs customer money
    - Needs committed customer
    - Difficult to finish if customer withdraw
    - May be too customer specific, no broad market
3. Difficult to know how long the project will last.
4. Easy to fall back into the code and fix without proper requirement analysis, design, customer evaluation, and feedback.
5. Prototyping tools are expensive.
6. Special tools & techniques are required to build a prototype.
7. It is a time-consuming process.
8. Developers in a hurry to build prototypes may end up with sub-optimal solutions.
9. The customer might lose interest in the product if he/she is not satisfied with the initial prototype.
10. After seeing an early prototype, the customers sometimes demand the actual product to be delivered soon.
11. There may be too much variation in requirements each time the prototype is evaluated by the customer.
12. Poor Documentation due to continuously changing customer requirements.

What is Spiral Model?

Spiral Model is a risk-driven software development process model. It is a combination of waterfall model and iterative model. Spiral Model helps to adopt software development elements of multiple process models for the software project based on unique risk patterns ensuring efficient development process.
Each phase of spiral model in software engineering begins with a design goal and ends with the client reviewing the progress. The spiral model in software engineering was first mentioned by Barry Boehm in his 1986 paper.

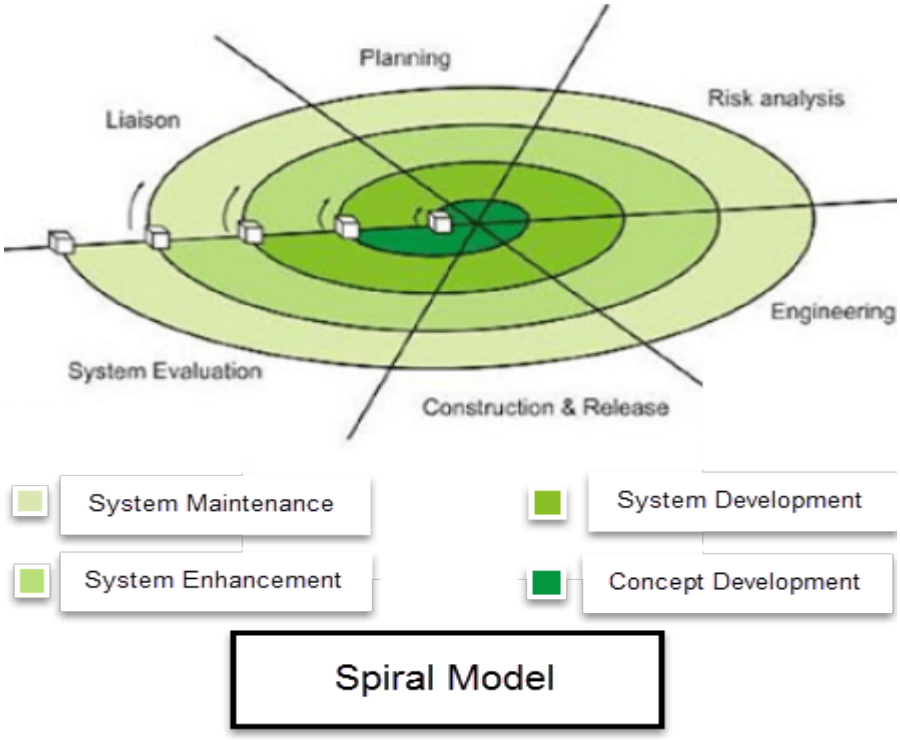
The development process in Spiral model in SDLC, starts with a small set of requirement and goes through each development phase for those set of requirements. The software engineering team adds functionality for the additional requirement in every-increasing spirals until the application is ready for the production phase.

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. The spiral model incorporates the stepwise approach of the Classical Waterfall Model. 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. 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.

The spiral model is called a meta model since **it encompasses all other life cycle models**. Risk handling is inherently built into this model. The spiral model is suitable for development of technically challenging software products that are prone to several kinds of risks.

The below figure very well explain Spiral Model:



Spiral Model Diagram

Spiral Model Phases

Spiral Model Phases	Activities performed during phase
Planning	• It includes estimating the cost, schedule and resources for the iteration. It also involves understanding the system requirements for continuous communication between the system analyst and the customer

Risk Analysis	• Identification of potential risk is done while risk mitigation strategy is planned and finalized
Engineering	• It includes testing, coding and deploying software at the customer site
Evaluation	• Evaluation of software by the customer. Also, includes identifying and monitoring risks such as schedule slippage and cost overrun

When to use Spiral Model?

- A Spiral model in software engineering is used when project is large
- When releases are required to be frequent, spiral methodology is used
- When creation of a prototype is applicable
- When risk and costs evaluation is important
- Spiral methodology is useful for medium to high-risk projects
- When requirements are unclear and complex, Spiral model in SDLC is useful
- When changes may require at any time
- When long term project commitment is not feasible due to changes in economic priorities

Spiral Model Advantages and Disadvantages

Advantages	Disadvantages
• Additional functionality or changes can be done at a later stage	• Risk of not meeting the schedule or budget
• Cost estimation becomes easy as the prototype building is done in small fragments	• Spiral development works best for large projects only also demands risk assessment expertise
• Continuous or repeated development helps in risk management	• For its smooth operation spiral model protocol needs to be followed strictly
• Development is fast and features are added in a systematic way in Spiral development	• Documentation is more as it has intermediate phases
• There is always a space for customer feedback	• Spiral software development is not advisable for smaller project, it might cost them a lot

Agile Development Models

In earlier days Iterative Waterfall model was very popular to complete a project. But nowadays developers face various problems while using it to develop software. The main difficulties included handling change requests from customers during project development and the high cost and time required to incorporate these changes. To overcome these drawbacks of Waterfall model, in the mid-1990s the Agile Software Development model was proposed.

The Agile model was primarily designed to help a project to adapt to change requests quickly. So, the main aim of the Agile model is to facilitate quick project completion. To accomplish this task agility is required. Agility is achieved by fitting the process to the project, removing activities that may not be essential for a specific project. Also, anything that is wastage of time and effort is avoided.

Actually Agile model refers to a group of development processes. These processes share some basic characteristics but do have certain subtle differences among themselves. A few Agile SDLC models are given below:

- Crystal
- Atern
- Feature-driven development
- Scrum
- Extreme programming (XP)
- Lean development
- Unified process

In the Agile model, the requirements are decomposed into many small parts that can be incrementally developed. The Agile model adopts Iterative development. Each incremental part is developed over an iteration. Each iteration is intended to be small and easily manageable and that can be completed within a couple of weeks only. At a time one iteration is planned, developed and deployed to the customers. Long-term plans are not made.

Agile model is the combination of iterative and incremental process models. Steps involve in agile SDLC models are:

- Requirement gathering
- Requirement Analysis
- Design
- Coding
- Unit testing
- Acceptance testing

The time to complete an iteration is known as a Time Box. Time-box refers to the maximum amount of time needed to deliver an iteration to customers. So, the end date for an iteration does not change. Though the development team can decide to reduce the delivered functionality during a Time-box if necessary to deliver it on time. The central principle of the Agile model is the delivery of an increment to the customer after each Time-box.

Principles of Agile model:

- To establish close contact with the customer during development and to gain a clear understanding of various requirements, each Agile project usually includes a customer

representative on the team. At the end of each iteration stakeholders and the customer representative review, the progress made and re-evaluate the requirements.
- Agile model relies on working software deployment rather than comprehensive documentation.
- Frequent delivery of incremental versions of the software to the customer representative in intervals of few weeks.
- Requirement change requests from the customer are encouraged and efficiently incorporated.
- It emphasizes on having efficient team members and enhancing communications among them is given more importance. It is realized that enhanced communication among the development team members can be achieved through face-to-face communication rather than through the exchange of formal documents.
- It is recommended that the development team size should be kept small (5 to 9 people) to help the team members meaningfully engage in face-to-face communication and have collaborative work environment.
- Agile development process usually deploy Pair Programming. In Pair programming, two programmers work together at one work-station. One does coding while the other reviews the code as it is typed in. The two programmers switch their roles every hour or so.

Advantages:
- Working through Pair programming produce well written compact programs which has fewer errors as compared to programmers working alone.
- It reduces total development time of the whole project.
- Customer representatives get the idea of updated software products after each iteration. So, it is easy for him to change any requirement if needed.

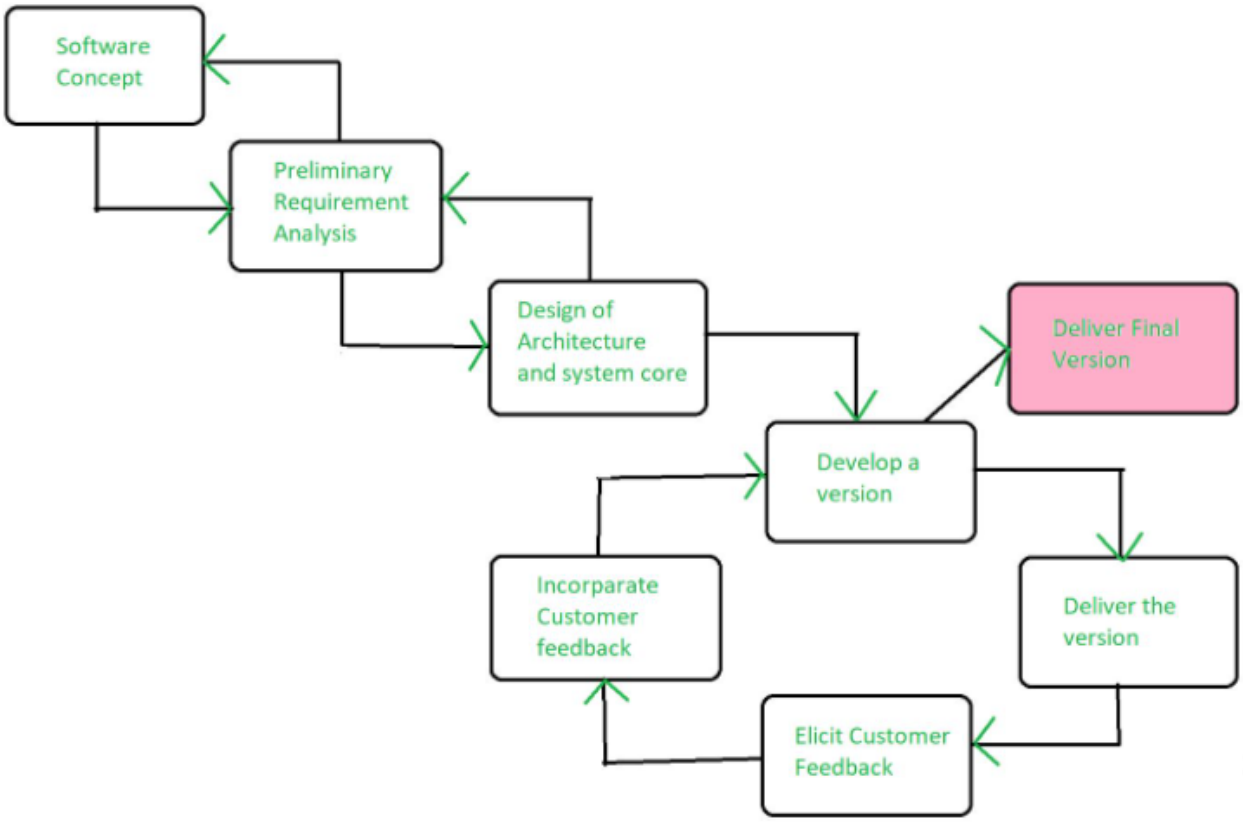
Disadvantages:
- Due to lack of formal documents, it creates confusion and important decisions taken during different phases can be misinterpreted at any time by different team members.
- Due to the absence of proper documentation, when the project completes and the developers are assigned to another project, maintenance of the developed project can become a problem.

Software Engineering | Evolutionary Model

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

development cycle into smaller, incremental waterfall models in which users are able to get access to the product at the end of each cycle.

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. Therefore, the software product evolves with time.
All the models have the disadvantage that the duration of time from start of the project to the delivery time of a solution is very high. Evolutionary model solves this problem in a different approach.



Evolutionary model suggests breaking down of work into smaller chunks, prioritizing them and then delivering those chunks to the customer one by one. The number of chunks is huge and is the number of deliveries made to the customer. 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. The model allows for changing requirements as well as all work in broken down into maintainable work chunks.

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.

Advantages:

- In evolutionary model, a user gets a chance to experiment partially developed system.
- It reduces the error because the core modules get tested thoroughly.

Disadvantages:

- Sometimes it is hard to divide the problem into several versions that would be acceptable to the customer which can be incrementally implemented and delivered.

Software Requirement Specifications

The production of the requirements stage of the software development process is **Software Requirements Specifications (SRS)** (also called a **requirements document**). This report lays a foundation for software engineering activities and is constructing when entire requirements are elicited and analyzed. **SRS** is a formal report, which acts as a representation of software that enables the customers to review whether it (SRS) is according to their requirements. Also, it comprises user requirements for a system as well as detailed specifications of the system requirements.

The SRS is a specification for a specific software product, program, or set of applications that perform particular functions in a specific environment. It serves several goals depending on who is writing it. First, the SRS could be written by the client of a system. Second, the SRS could be written by a developer of the system. The two methods create entirely various situations and establish different purposes for the document altogether. The first case, SRS, is used to define the needs and expectation of the users. The second case, SRS, is written for various purposes and serves as a contract document between customer and developer.

Following are the characteristics of a good SRS document:

1. **Correctness:**
   User review is used to ensure the correctness of requirements stated in the SRS. SRS is said to be correct if it covers all the requirements that are actually expected from the system.

2. **Completeness:**
   Completeness of SRS indicates every sense of completion including the numbering of all the pages, resolving the to be determined parts to as much extent as possible as well as covering all the functional and non-functional requirements properly.

3. **Consistency:**

Requirements in SRS are said to be consistent if there are no conflicts between any set of requirements. Examples of conflict include differences in terminologies used at separate places, logical conflicts like time period of report generation, etc.

4. **Unambiguousness:**
   A SRS is said to be unambiguous if all the requirements stated have only 1 interpretation. Some of the ways to prevent unambiguousness include the use of modelling techniques like ER diagrams, proper reviews and buddy checks, etc.

5. **Ranking for importance and stability:**
   There should a criterion to classify the requirements as less or more important or more specifically as desirable or essential. An identifier mark can be used with every requirement to indicate its rank or stability.

6. **Modifiability:**
   SRS should be made as modifiable as possible and should be capable of easily accepting changes to the system to some extent. Modifications should be properly indexed and cross-referenced.

7. **Verifiability:**
   A SRS is verifiable if there exists a specific technique to quantifiably measure the extent to which every requirement is met by the system. For example, a requirement starting that the system must be user-friendly is not verifiable and listing such requirements should be avoided.

8. **Traceability:**
   One should be able to trace a requirement to design component and then to code segment in the program. Similarly, one should be able to trace a requirement to the corresponding test cases.

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

10. **Testability:**

A SRS should be written in such a way that it is easy to generate test cases and test plans from the document.

11. **Understandable by the customer:**
An end user maybe an expert in his/her specific domain but might not be an expert in computer science. Hence, the use of formal notations and symbols should be avoided to as much extent as possible. The language should be kept easy and clear.

12. **Right level of abstraction:**
If the SRS is written for the requirements phase, the details should be explained explicitly. Whereas, for a feasibility study, fewer details can be used. Hence, the level of abstraction varies according to the purpose of the SRS.

Properties of a good SRS document

The essential properties of a good SRS document are the following:

Concise: The SRS report should be concise and at the same time, unambiguous, consistent, and complete. Verbose and irrelevant descriptions decrease readability and also increase error possibilities.

Structured: It should be well-structured. A well-structured document is simple to understand and modify. In practice, the SRS document undergoes several revisions to cope up with the user requirements. Often, user requirements evolve over a period of time. Therefore, to make the modifications to the SRS document easy, it is vital to make the report well-structured.

Black-box view: It should only define what the system should do and refrain from stating how to do these. This means that the SRS document should define the external behavior of the system and not discuss the implementation issues. The SRS report should view the system to be developed as a black box and should define the externally visible behavior of the system. For this reason, the SRS report is also known as the black-box specification of a system.

Conceptual integrity: It should show conceptual integrity so that the reader can merely understand it. Response to undesired events: It should characterize acceptable responses to unwanted events. These are called system response to exceptional conditions.

Verifiable: All requirements of the system, as documented in the SRS document, should be correct. This means that it should be possible to decide whether or not requirements have been met in an implementation.