SARDAR PATEL COLLEGE OF ENGINEERING.BAKROL-388315

# CHAPTER 1: INTRODUCTION TO SOFTWARE AND SOFTWARE ENGINEERING

PREPARED BY: Prof. Tarun A.Saluja
I.T Department
(SPCE-124)

SOFTWARE ENGINEERING:2160701

# TOPICS TO BE COVERED:

1. WHAT IS SOFTWARE, WHY WE NEED SOFTWARE.
2. WHAT IS SOFTWARE ENGINEERING.
3. ADVANTAGES OF SOFTWARE ENGINEERING.
4. SOFTWARE MYTHS.
5. LAYERED TECHNOLOGY
6. PROCESS MODELS.
7. TYPES OF PROCESS MODELS.
   - 7.1 PROTOTYPE MODEL
   - 7.2 RAD MODEL
8. EVOLUTIONARY PROCESS MODELS.
9. AGILE PROCESS
10. COMPONENT BASED DEVELOPMENT
11. PROCESS, PRODUCT AND PROCESS

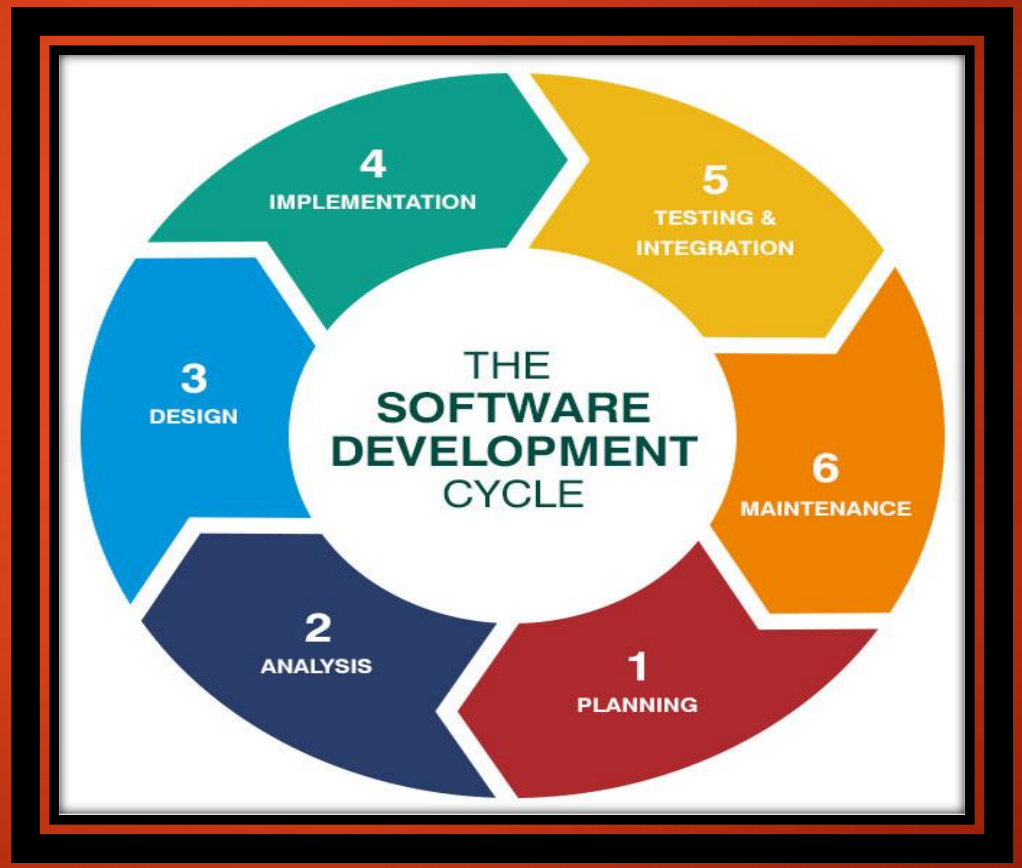
- **What is software? Why we need software**

- **WHAT IS SOFTWARE ?**
- **software** is a collection of instructions that enable the user to interact with a computer, its hardware, or perform tasks. without software, most computers would be useless. for example, without your internet browser software, you could not surf the internet and without an operating system, the browser could not run on your computer

- **WHY WE NEED SOFTWARE?**
- **software** testing is really required to point out the defects and errors that were made during the development phases. it's essential since it makes sure of the customer's reliability and their satisfaction in the application.

**There are following six phases in every Software development life cycle model:**

1. Requirement gathering and analysis.
2. Design.
3. Implementation or coding.
4. Testing.
5. Deployment.
6. Maintenance.



**1.Planning**: Without the perfect plan, calculating the strengths and weaknesses of the project, development of software is meaningless. Planning kicks off a project flawlessly and affects its progress positively.

2. **Analysis**: This step is about analyzing the performance of the software at various stages and making notes on additional requirements. Analysis is very important to proceed further to the next step.

3. **Design**: Once the analysis is complete, the step of designing takes over, which is basically building the architecture of the project. This step helps remove possible flaws by setting a standard and attempting to stick to it.

4. **Development & Implementation**: The actual task of developing the software starts here with data recording going on in the background. Once the software is developed, the stage of implementation comes in where the product goes through a pilot study to see if it's functioning properly.

5. **Testing**: The testing stage assesses the software for errors and documents bugs if there are any.

6. **Maintenance**: Once the software passes through all the stages without any issues, it is to undergo a maintenance process wherein it will be maintained and upgraded from time to time to adapt to changes. Almost every software development Indian company follows all the six steps, leading to the reputation that the country enjoys in the software market today.

## What is Software 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 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.


**Software engineering** is an engineering branch associated with development of software product using well-defined scientific principles, methods and procedures. The outcome of software engineering is an efficient and reliable software product

## Software Evolution

- The process of developing a software product using software engineering principles and methods is referred to as **software evolution.** This includes the initial development of software and its maintenance and updates, till desired software product is developed, which satisfies the expected requirements.

- Evolution starts from the requirement gathering process. After which developers create a prototype of the intended software and show it to the users to get their feedback at the early stage of software product development. The users suggest changes, on which several consecutive updates and maintenance keep on changing too. This process changes to the original software, till the desired software is accomplished.

- Even after the user has desired software in hand, the advancing technology and the changing requirements force the software product to change accordingly. Re-creating software from scratch and to go one-on-one with requirement is not feasible. The only feasible and economical solution is to update the existing software so that it matches the latest requirements.

- **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 he 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 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.

- **Characteristics 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.

**SOFTWARE MYTHS:**

**1. Management myths:** The managers are often grasps at a belief in a software myth, same as a drowning person who grasps at a straw.
**Members acquires all the information:** Generally, there is a myth that the members of the organization acquire all the information containing procedures, principles and standards. In reality, the developers don't have information about all the established standards because they are often outdated, incomplete and unadaptable. Plus, there is a rare chance that the developer will follow all the standards.

**Adding more people can reduce the time gap:** Another myth in the people is that more the number of programmers, lesser will be the time gap. If a project is behind the schedule, adding more manpower will further delay it because new workers will take more time to learn about the ongoing project.

**The management can relax themselves by outsourcing its project**: If an organization outsource its software to a third party, it does not relieve the management of its duties. When the organization outsources the software project, they suffer invariably.

**2. Customers Myths:** The customers are encouraged by some marketing people in underestimating the difficulty of developing software.

- **Software is malleable as a result of which changes are easy to accommodate:** There is an enormous amount of labor required to have a change in the software after the release. It is not so easy to accommodate these changes.
- **To start coding, a general statement of need is enough:** The developers can't read the customer's mind and requires detailed descriptions of the requirements, in order to start coding.

**3. Developer Myths:** The software development art is becoming an engineering discipline, but there are lots of myths.

- **Once the code is delivered, the software can be called complete:** In reality, more than 60% of the efforts are expended after the delivery of the software to the user.
- **The software's success depends on the product's produced quality:** The project does not become successful on the quality of the programs because both the software configuration and documentation also play an important role in its success.
- **The unnecessary documentation is required in software engineering, which further slows down the growth rate of a project:** This myth is a no brainer because in reality, the proper documentation enhances the project's quality and results in reduction of the rework. Also, this field is just about creating quality at all the level of the project.
- **The assessment of the software quality can be addressed after the execution of the program:** During any phase of the development process, the software's quality can be measured just by applying the mechanism of quality assurance.
- **The product, which is delivered after the project's completion can be called working program:** The deliverables of a successful project don't only consist working program, but also the documentation which can guide the users about how to use the software.

## SOFTWARE ENGINEERING: A LAYERED TECHNOLOGY

Software development is totally a layered technology. That means, to develop software one will have to go from one layer to another. The layers are related and each layer demands the fulfillment of the previous layer. Figure below is the upward flowchart of the layers of software development.

- The **process layer** allows the development of software on time. It defines an outline for a set of key process areas that must be acclaimed for effective delivery of software engineering technology.

- The **method layer** provides technical knowledge for developing software. This layer covers a broad array of tasks that include requirements analysis, design, coding, testing, and maintenance phase of the software development.

- The **tools layer** provides computerized or semi-computerized support for the process and the method layer. Sometimes tools are integrated in such a way that other tools can use information created by one tool. This multi-usage is commonly referred to as **Computer-Aided Software Engineering (CASE).**CASE combines software, hardware, and software engineering database to create software engineering analogous to **Computer-Aided Design (CAD) for** hardware. CASE helps in application development including analysis, design, code generation, and debugging and testing. This is possible by using CASE tools, which provide automated methods for designing and documenting traditional-structure programming techniques. For example, two prominent technologies using CASE tools are PC-based workstations and application generators that provide graphics-based interfaces to automate the development process.

**Software Process Model**

A software process (also knows as software methodology) is a set of related activities that leads to the production of the software. These activities may involve the development of the software from the scratch, or, modifying an existing system.

Any software process must include the following four activities:

1. Software specification (or requirements engineering): Define the main functionalities of the software and the constrains around them.
2. Software design and implementation: The software is to be designed and programmed.
3. Software verification and validation: The software must conforms to it's specification and meets the customer needs.
4. Software evolution (software maintenance): The software is being modified to meet customer and market requirements changes.

**What is Process Models**

- A Process Model describes the sequence of phases for the entire lifetime of a product. Therefore it is sometimes also called Product Life Cycle. This covers everything from the initial commercial idea until the final de-installation or disassembling of the product after its use

- **Usually there are three main phases:**

  - concept phase
  - implementation phase
  - maintenance phase

- Each of these main phases usually has some sub-phases, like a requirements engineering phase, a design phase, a build phase and a testing phase. The sub-phases may occur in more than one main phase each of them with a specific peculiarity depending on the main phase.

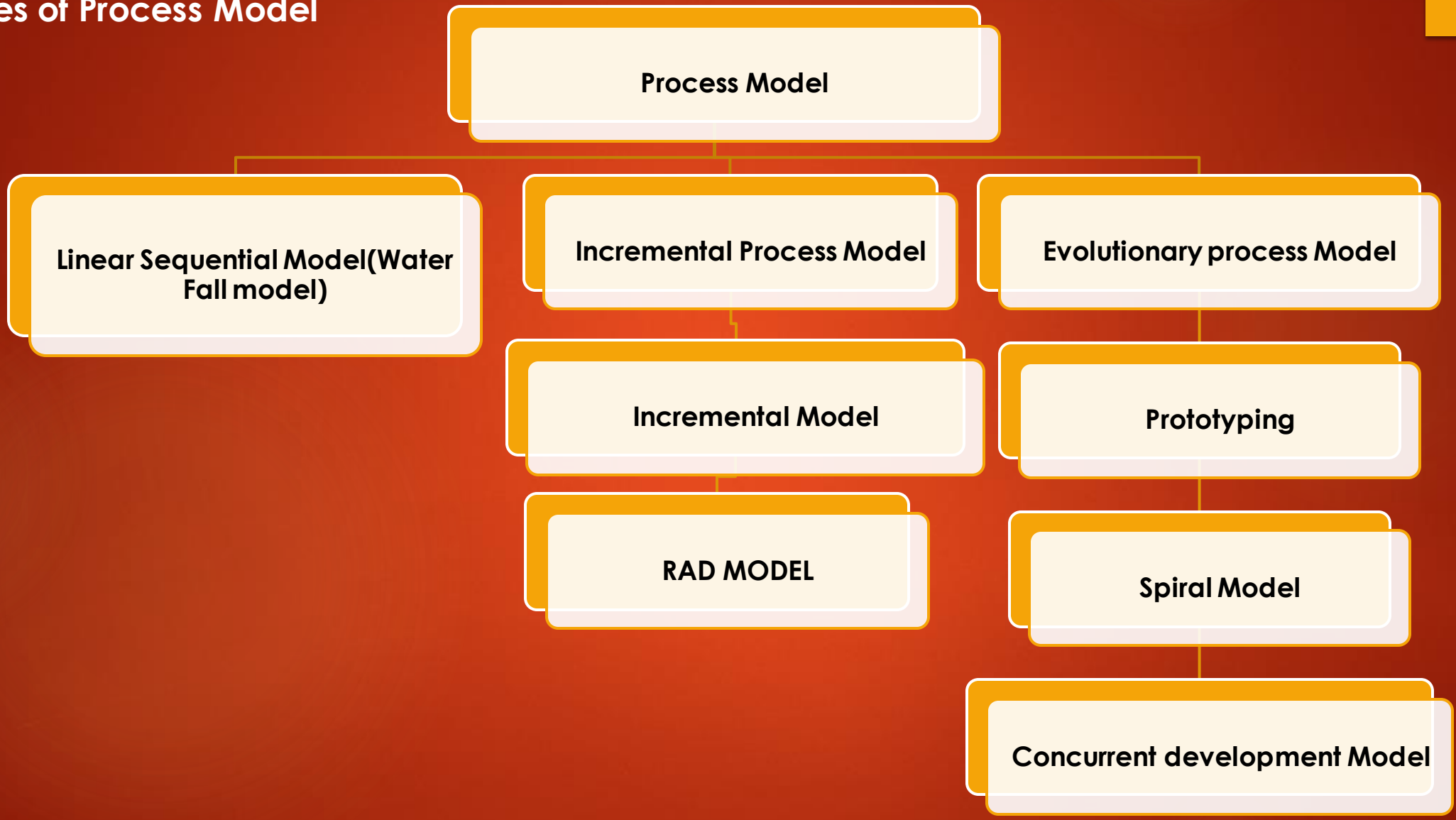
•Besides the phases a Process Model shall also define at least:
The **activities** that have to be carried out in each of the sub-phases, including the sequence in which these activities have to be carried out.

•The **roles** of the executors that have to carry out the activities, including a description of their responsibilities and required skills.
•The **work products** that have to be established or updated in each of the activities. Besides the final product there are usually several other items that have to be generated during the development of a product. These are for example requirements and design document, test specifications and test reports, etc.
•
Therefore, a Process Model provides a fixed framework that guides a project in:
Development of the product
•Planning and organizing the project
•Tracking and running the project

**Why do we need Process Models**

- Develop an informal narrative specification of the system.
- Identify the objects and their attributes.
- Identify the operations on the objects.
- Identify the interfaces between objects, attributes, or operations.
- Implement the operations.

## Types of Process Model

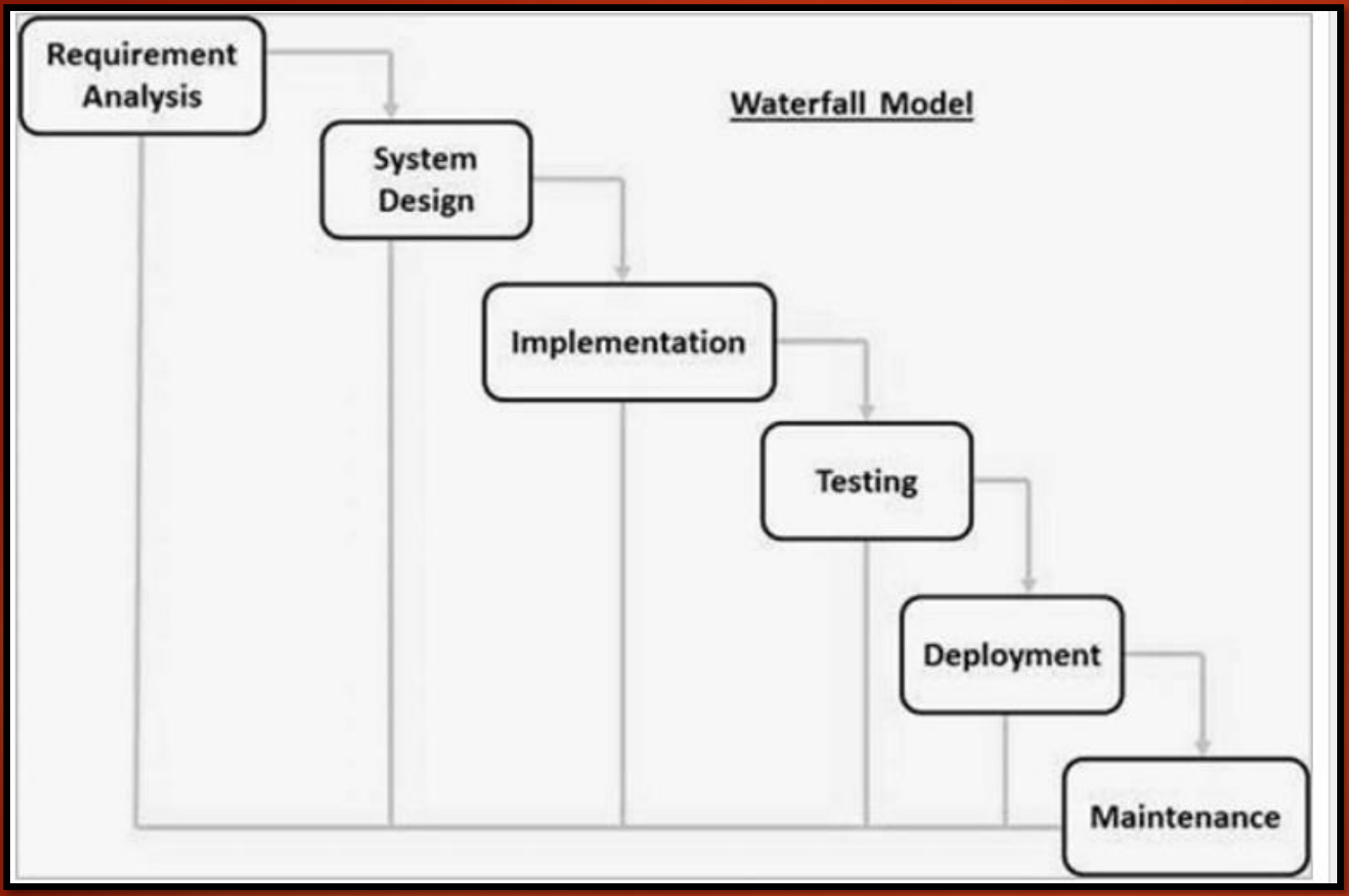


## Linear Sequential Model(Water Fall model)

- The Waterfall Model was the first Process Model to be introduced. It is also referred to as a **linear-sequential life cycle model**. It is very simple to understand and use. In a waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases.
- The waterfall Model illustrates the software development process in a linear sequential flow. This means that any phase in the development process begins only if the previous phase is complete. In this waterfall model, the phases do not overlap.
- "The Waterfall" approach, the whole process of software development is divided into separate phases. In this Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.

## PHASES OF WATER FALL MODEL:

•**Requirement Gathering and analysis** − All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.
•**System Design** − The requirement specifications from first phase are studied in this phase and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture.




•**Implementation** – With inputs from the system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing.

•**Integration and Testing** – All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.

•**Deployment of system** – Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.

•**Maintenance** – There are some issues which come up in the client environment. To fix those issues, patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

- All these phases are cascaded to each other in which progress is seen as flowing steadily downwards (like a waterfall) through the phases. The next phase is started only after the defined set of goals are achieved for previous phase and it is signed off, so the name "Waterfall Model". In this model, phases do not overlap.

•**ADVANTAGES:**

•Simple and easy to understand and use

•Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.

•Phases are processed and completed one at a time.

•Works well for smaller projects where requirements are very well understood.

•Clearly defined stages.

•Well understood milestones.

•Easy to arrange tasks.

•Process and results are well documented.

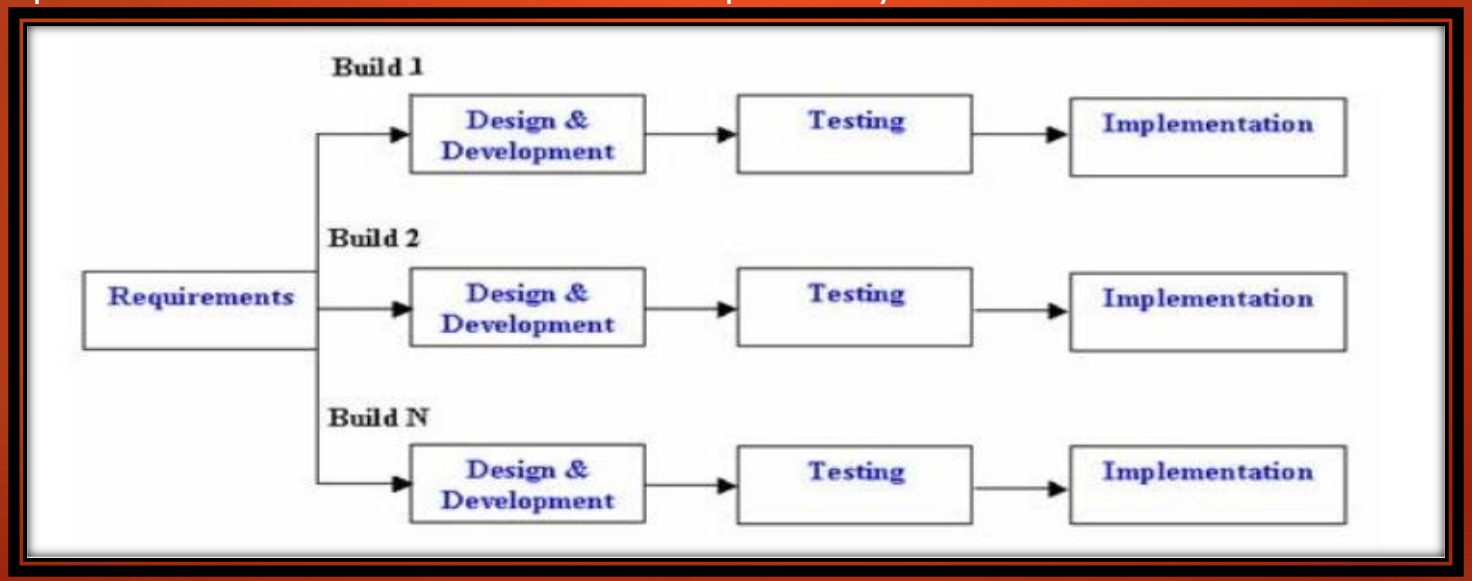
The major **disadvantages** of the Waterfall Model are as follows

- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing. So, risk and uncertainty is high with this process model.
- It is difficult to measure progress within stages.
- Cannot accommodate changing requirements.
- Adjusting scope during the life cycle can end a project.
- Integration is done as a "big-bang. at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early.

## Incremental model

- In incremental model the whole requirement is divided into various builds. Multiple development cycles take place here, making the life cycle a **"multi-waterfall" cycle**. Cycles are divided up into smaller, more easily managed modules.

- In this model, each module passes through the requirements, design, implementation and **testing** phases. A working version of software is produced during the first module, so you have working software early on during the **software life cycle**. Each subsequent release of the module adds function to the previous release. The process continues till the complete system is achieved.



**Advantages of Incremental model:**

- Generates working software quickly and early during the software life cycle.
- This model is more flexible – less costly to change scope and requirements.
- It is easier to test and debug during a smaller iteration.
- In this model customer can respond to each built.
- Lowers initial delivery cost.
- Easier to manage risk because risky pieces are identified and handled during it's iteration.

**Disadvantages of Incremental model:**

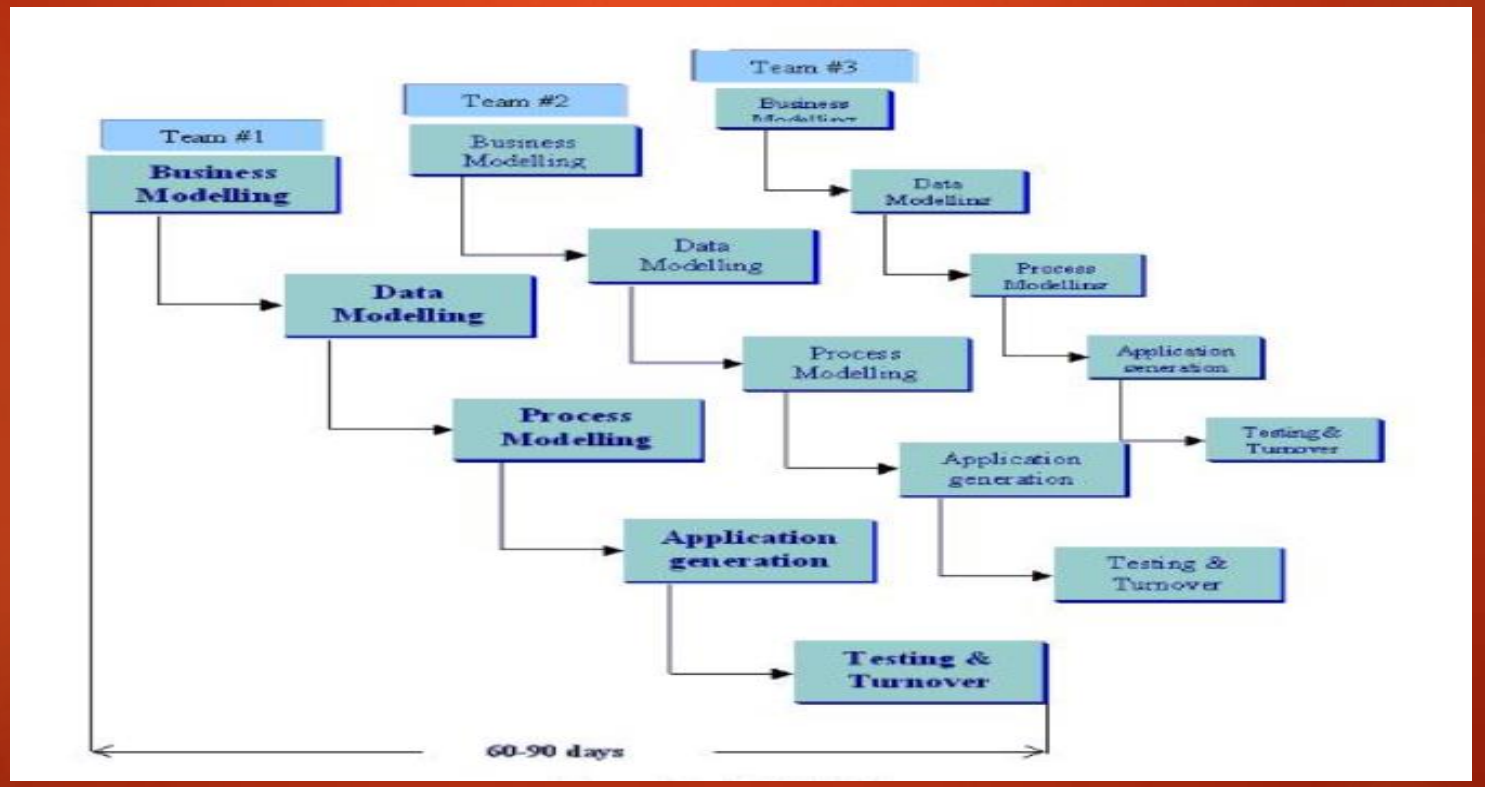
- Needs good planning and design.
- Needs a clear and complete definition of the whole system before it can be broken down and built incrementally.
- Total cost is higher than **waterfall**.

**When to use the Incremental model:**

- This model can be used when the requirements of the complete system are clearly defined and understood.
- Major requirements must be defined; however, some details can evolve with time.
- There is a need to get a product to the market early.
- A new technology is being used
- Resources with needed skill set are not available
- There are some high risk features and goals.

## RAD model

- RAD model is Rapid Application Development model. It is a type of **incremental model**. In RAD model the components or functions are developed in parallel as if they were mini projects. The developments are time boxed, delivered and then assembled into a working prototype. This can quickly give the customer something to see and use and to provide feedback regarding the delivery and their requirements.



The phases in the rapid application development (RAD) model are:

**Business modeling:** The information flow is identified between various business functions.

**Data modeling:** Information gathered from business modeling is used to define data objects that are needed for the business.

**Process modeling:** Data objects defined in data modeling are converted to achieve the business information flow to achieve some specific business objective. Description are identified and created for CRUD of data objects.

**Application generation:** Automated tools are used to convert process models into code and the actual system.

**Testing and turnover:** Test new components and all the interfaces.

**Advantages of the RAD model:**

- Reduced development time.
- Increases reusability of components
- Quick initial reviews occur
- Encourages customer feedback
- Integration from very beginning solves a lot of **integration issues**.

**Disadvantages of RAD model:**

- Depends on strong team and individual performances for identifying business requirements.
- Only system that can be modularized can be built using RAD
- Requires highly skilled developers/designers.
- High dependency on modeling skills
- Inapplicable to cheaper projects as cost of modeling and automated code generation is very high.

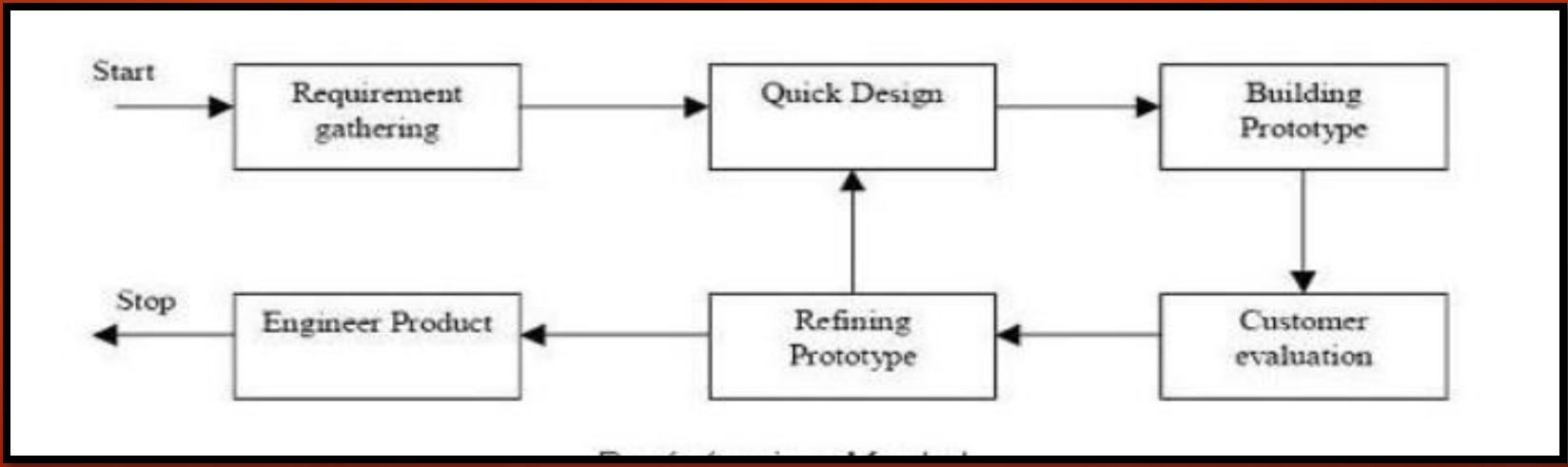
**When to use RAD model:**

- RAD should be used when there is a need to create a system that can be modularized in 2-3 months of time.
- It should be used if there's high availability of designers for modeling and the budget is high enough to afford their cost along with the cost of automated code generating tools.
- RAD **SDLC model** should be chosen only if resources with high business knowledge are available and there is a need to produce the system in a short span of time (2-3 months).

## Prototype model

- The basic idea in Prototype model is that instead of freezing the requirements before a design or coding can proceed, a throw way prototype is built to understand the requirements. This prototype is developed based on the currently known requirements. Prototype model is a software development model. By using this prototype, the client can get an "actual feel" of the system, since the interactions with prototype can enable the client to better understand the requirements of the desired system. Prototyping is an attractive idea for complicated and large systems for which there is no manual process or existing system to help determining the requirements.

- The prototype are usually not complete systems and many of the details are not built in the prototype. The goal is to provide a system with overall functionality.



**Advantages of Prototype model:**

- Users are actively involved in the development
- Since in this methodology a working model of the system is provided, the users get a better understanding of the system being developed.
- Errors can be detected much earlier.
- Quicker user feedback is available leading to better solutions.
- Missing functionality can be identified easily
- Confusing or difficult functions can be identified Requirements validation, Quick implementation of, incomplete, but functional, application.

**Disadvantages of Prototype model:**

- Leads to implementing and then repairing way of building systems.
- Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans.
- Incomplete application may cause application not to be used as the full system was designed Incomplete or inadequate problem analysis.

**When to use Prototype model:**

•Prototype model should be used when the desired system needs to have a lot of interaction with the end users.

•Typically, online systems, web interfaces have a very high amount of interaction with end users, are best suited for Prototype model. It might take a while for a system to be built that allows ease of use and needs minimal training for the end user.

•Prototyping ensures that the end users constantly work with the system and provide a feedback which is incorporated in the prototype to result in a useable system. They are excellent for designing good human computer interface systems.

## SPIRAL MODEL

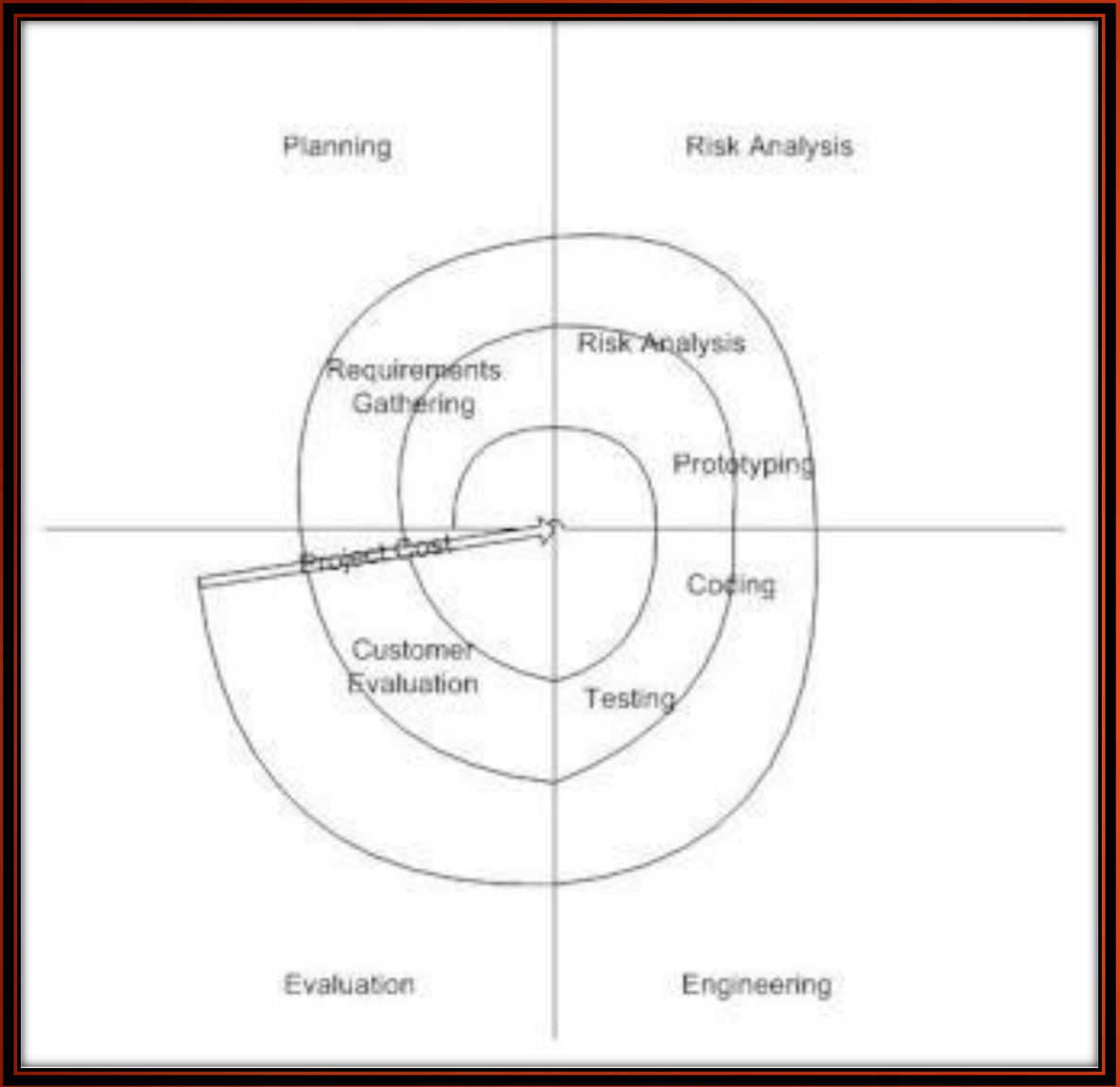
The spiral model is similar to the **incremental model**, with more emphasis placed on risk analysis. The spiral model has four phases: Planning, Risk Analysis, Engineering and Evaluation. A software project repeatedly passes through these phases in iterations (called Spirals in this model). The baseline spiral, starting in the planning phase, requirements are gathered and risk is assessed. Each subsequent spirals builds on the baseline spiral

**Planning Phase:** Requirements are gathered during the planning phase. Requirements like 'BRS' that is 'Business Requirement Specifications' and 'SRS' that is 'System Requirement specifications'

**Risk Analysis:** In the **risk analysis phase**, a process is undertaken to identify risk and alternate solutions. A prototype is produced at the end of the risk analysis phase. If any risk is found during the risk analysis then alternate solutions are suggested and implemented.

**Engineering Phase:** In this phase software is **developed**, along with **testing** at the end of the phase. Hence in this phase the development and testing is done.

E**valuation phase:** This phase allows the customer to evaluate the output of the project to date before the project continues to the next spiral.




**Advantages of Spiral model:**

- High amount of risk analysis hence, avoidance of Risk is enhanced.
- Good for large and mission-critical projects.
- Strong approval and documentation control.
- Additional Functionality can be added at a later date.
- Software is produced early in the **software life cycle**.

**Disadvantages of Spiral model:**

- Can be a costly model to use.
- Risk analysis requires highly specific expertise.
- Project's success is highly dependent on the risk analysis phase.
- Doesn't work well for smaller projects.

- **Component-based architecture**

- Component-based architecture focuses on the decomposition of the design into individual functional or logical components that represent well-defined communication interfaces containing methods, events, and properties. It provides a higher level of abstraction and divides the problem into sub-problems, each associated with component partitions.

- The primary objective of component-based architecture is to ensure **component reusability**. A component encapsulates functionality and behaviors of a software element into a reusable and self-deployable binary unit.

- Component-oriented software design has many advantages over the traditional object-oriented approaches such as −

•Reduced time in market and the development cost by reusing existing components.
•Increased reliability with the reuse of the existing components.

- **What is a Component?**
- A component is a modular, portable, replaceable, and reusable set of well-defined functionality that encapsulates its implementation and exporting it as a higher-level interface.
- A component is a software object, intended to interact with other components, encapsulating certain functionality or a set of functionalities. It has an obviously defined interface and conforms to a recommended behavior common to all components within an architecture.

- **Advantages**

•**Ease of deployment** – As new compatible versions become available, it is easier to replace existing versions with no impact on the other components or the system as a whole.
•**Reduced cost** – The use of third-party components allows you to spread the cost of development and maintenance.
•**Ease of development** – Components implement well-known interfaces to provide defined functionality, allowing development without impacting other parts of the system.
•**Reusable** – The use of reusable components means that they can be used to spread the development and maintenance cost across several applications or systems.
•**Modification of technical complexity** – A component modifies the complexity through the use of a component container and its services.
•**Reliability** – The overall system reliability increases since the reliability of each individual component enhances the reliability of the whole system via reuse.
•**System maintenance and evolution** – Easy to change and update the implementation without affecting the rest of the system.
•**Independent** – Independency and flexible connectivity of components. Independent development of components by different group in parallel. Productivity for the software development and future software development.

- **It has following Silent features –**

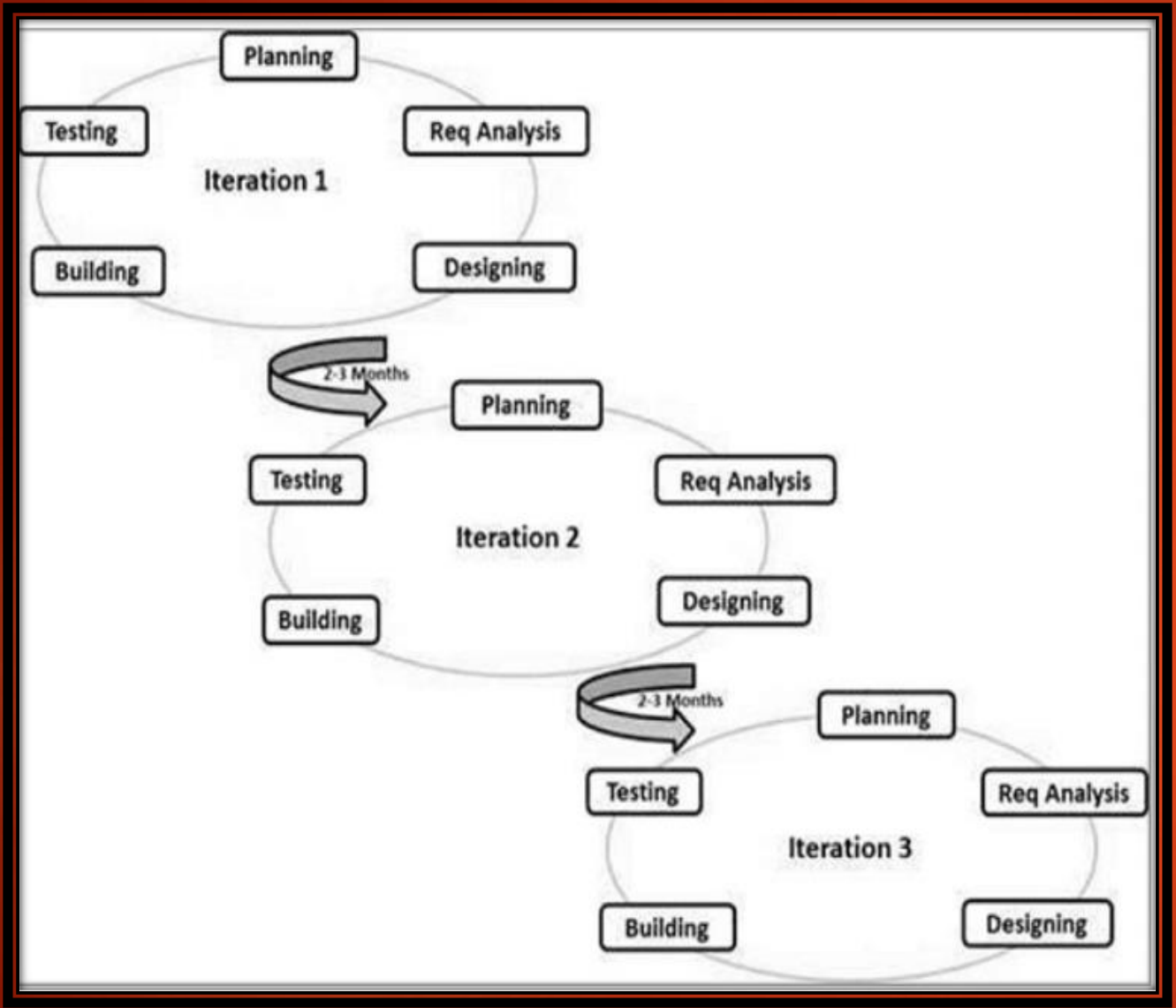
•The software system is decomposed into reusable, cohesive, and encapsulated component units.
•Each component has its own interface that specifies required ports and provided ports; each component hides its detailed implementation.
•A component should be extended without the need to make internal code or design modifications to the existing parts of the component.
•Depend on abstractions component do not depend on other concrete components, which increase difficulty in expendability.
•Connectors connected components, specifying and ruling the interaction among components. The interaction type is specified by the interfaces of the components.
•Components interaction can take the form of method invocations, asynchronous invocations, broadcasting, message driven interactions, data stream communications, and other protocol specific interactions.
•For a server class, specialized interfaces should be created to serve major categories of clients. Only those operations that are relevant to a particular category of clients should be specified in the interface.
•A component can extend to other components and still offer its own extension points. It is the concept of plug-in based architecture. This allows a plugin to offer another plugin API.

## Agile model

- Agile model believes that every project needs to be handled differently and the existing methods need to be tailored to best suit the project requirements. In Agile, the tasks are divided to time boxes (small time frames) to deliver specific features for a release.
- Iterative approach is taken and working software build is delivered after each iteration. Each build is incremental in terms of features; the final build holds all the features required by the customer.

- **Following are the Agile Manifesto principles –**

1. **Individuals and interactions** − In Agile development, self-organization and motivation are important, as are interactions like co-location and pair programming.
2. **Working software** − Demo working software is considered the best means of communication with the customers to understand their requirements, instead of just depending on documentation.
3. **Customer collaboration** − As the requirements cannot be gathered completely in the beginning of the project due to various factors, continuous customer interaction is very important to get proper product requirements.
4. **Responding to change** − Agile Development is focused on quick responses to change and continuous development.




**•The advantages of the Agile Model are as follows –**

•Is a very realistic approach to software development.
•Promotes teamwork and cross training.
•Functionality can be developed rapidly and demonstrated.
•Resource requirements are minimum.
•Suitable for fixed or changing requirements
•Delivers early partial working solutions.
•Good model for environments that change steadily.
•Minimal rules, documentation easily employed.
•Enables concurrent development and delivery within an overall planned context.
•Little or no planning required.
•Easy to manage.
•Gives flexibility to developers.

**The disadvantages of the Agile Model are as follows –**

•Not suitable for handling complex dependencies.

•More risk of sustainability, maintainability and extensibility.

•An overall plan, an agile leader and agile PM practice is a must without which it will not work.

•Strict delivery management dictates the scope, functionality to be delivered, and adjustments to meet the deadlines.

•Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrong direction.

•There is a very high individual dependency, since there is minimum documentation generated.

•Transfer of technology to new team members may be quite challenging due to lack of documentation.

| SPIRAL MODEL | WATERFALL MODEL |
|---|---|
| Spiral model is not suitable for small projects. | Waterfall model is suitable for small projects. |
| Better risk management. | High amount of risk and uncertainty. |
| Process is complex. | Easy to understand. |
| The process may go indefinitely. | Stages are clearly defined. |
| This model is suitable for long and ongoing projects. | This model is not suitable for long and ongoing projects. |
| Iterations are followed | Sequence is followed |
| Flexible with user requirements | Requirements once fixed cannot be modified |
| Refinements are easily possible | Refinements are not so easy |
| Phases are repeated itself | Phases are processed and completed one at a time. |
| | |

**Important:**
**Difference between all process models,**