What is Software (CO1)

Software Engineering is

- A program along with proper documentation (requirement analysis, design, coding, testing) and user manuals which includes installation guide and other manuals.
- It is a systematic, disciplined, cost effective techniques for software development
- Engineering approach to develop a software.

What is Software? IEEE Definition

IEEE definition of Software:-

Software is the collection of computer programs procedures, rules and associated documentation and data.

"IEEE : Institute of Electrical and Electronics Engineers"

Importance of Software

- It has become a driving force.
- It is engine that drive business decision making.
- It serve as the basis for modern scientific investigation and engineering problem solving.
- It is embedded in all kind of systems like transportation , medical, telecommunications, military, industrial process, entertainment, office products

Importance of Software

- It affects nearly every aspect of our lives and has become pervasive in our commerce, our culture and our every day activities software impact on our society and culture is significant .
- As software importance grows, the software community continually attempts to develop technologies that will make it easier faster and less expensive to build high quality computer programs.

Introduction to Software Engineering

- IEEE Definition:-

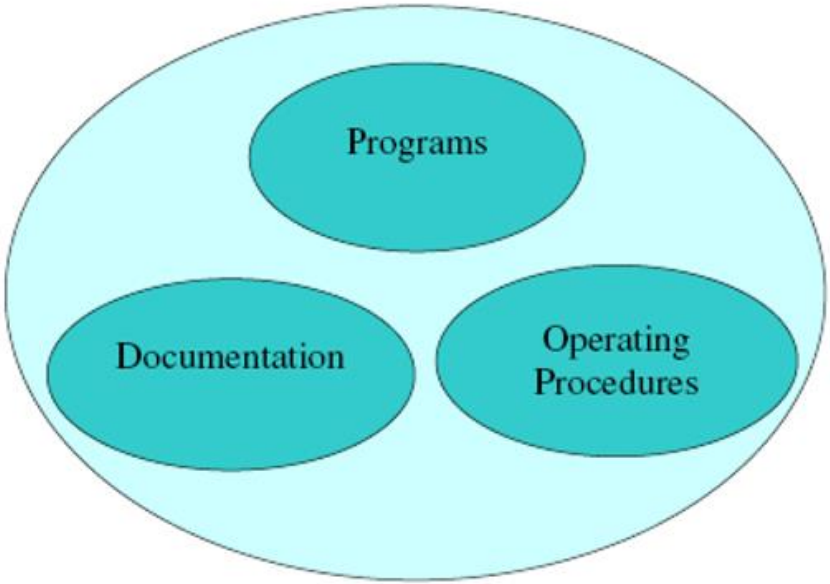
  Software Engineering is the application of systematic, disciplined, quantifiable approach to the development, operation and maintenance of software.

  or

  Software Engineering Is an Engineering discipline which is concerned with all aspects of software production.

Software Components

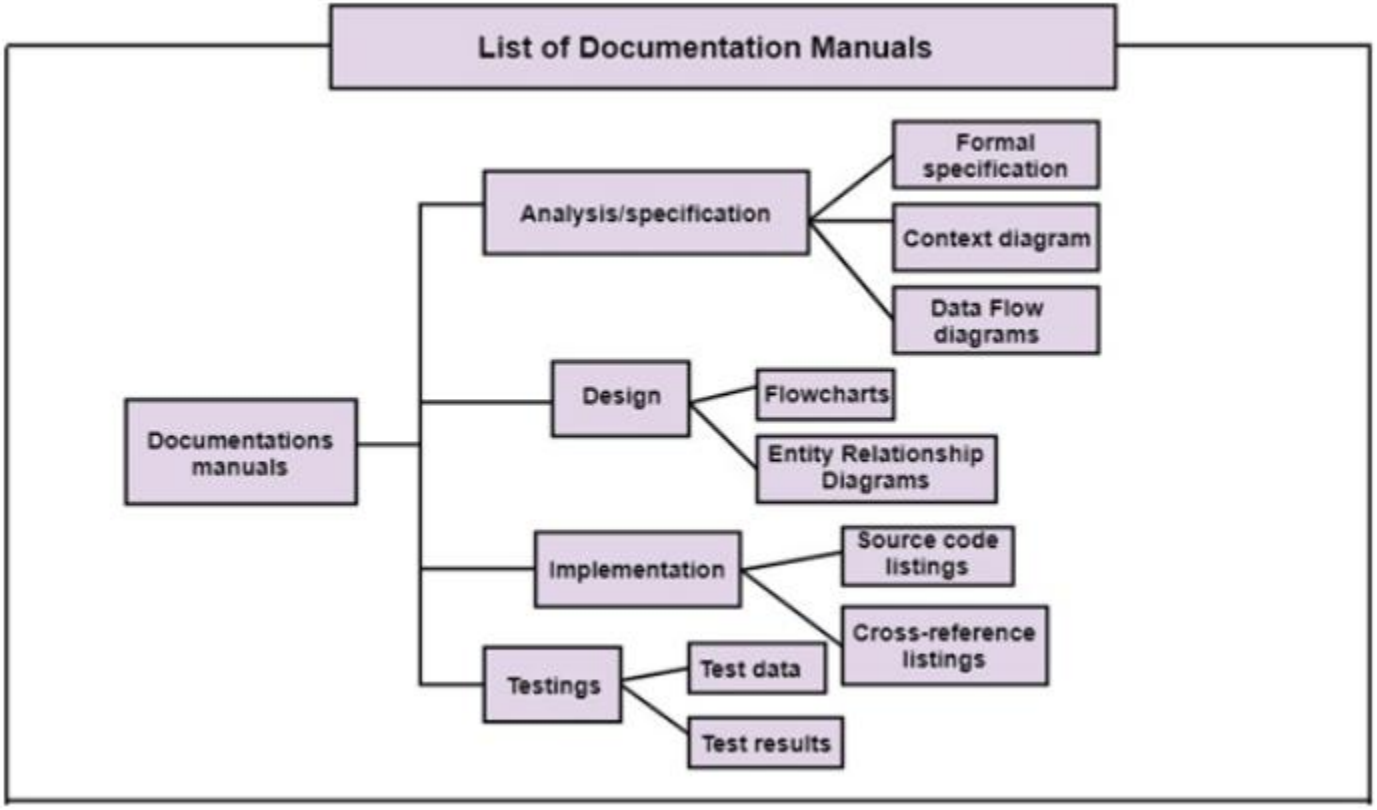
Software components comprises of these 3 components :



Program: Program is a combination of source code & object code.

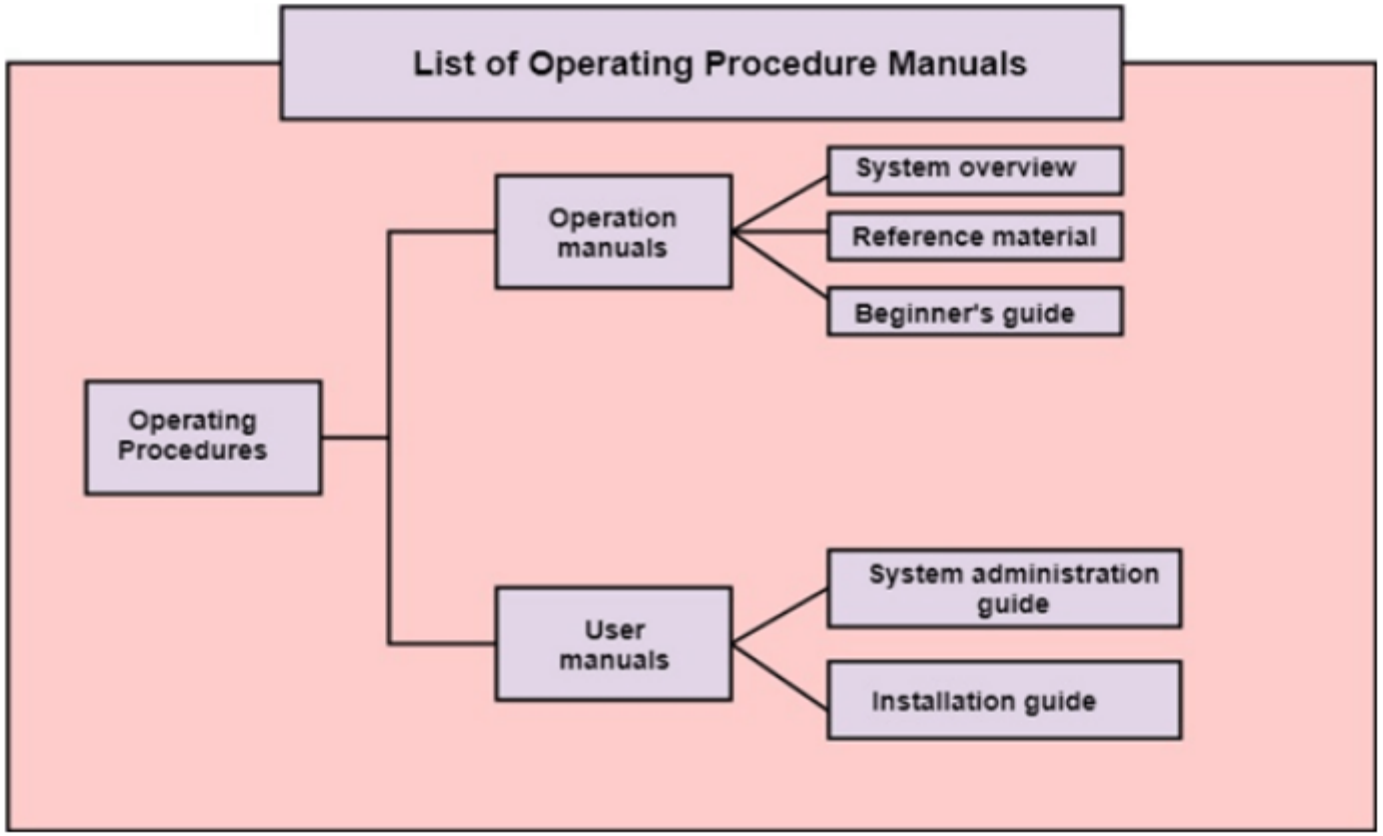
Software Components

Documentation: Documentation consists of different types of manuals. Examples of documentation manuals are: Data Flow Diagram, Flow Charts, ER diagrams, etc.



Software Components

Operating Procedures: Operating Procedures consist of instructions to set up and use the software system and instructions on how react to the system failure



Objective of software Engineering

- The basic objective of software engineering is to develop methods and procedures for software development that can scale up for large systems and that can be used consistently to produce high-quality software at low cost and with a small cycle of time

Software Characteristics (CO1)

The key characteristics of software are -

1. **Correctness:** The software should perform the intended functions accurately and produce the expected outputs.

2. **Reliability:** The software should be dependable, providing consistent and reliable results under different conditions.

3. **Efficiency:**

- **Performance:** The software should execute tasks in a timely manner and with optimal use of system resources.
- **Scalability:** The ability of the software to handle increased loads or demands by scaling its capacity.

4. **Usability:**

- **User Interface (UI):** The software should have a user-friendly interface that is easy to navigate and understand.
- **User Experience (UX):** The overall experience of using the software should be pleasant and efficient.

1. Most software is custom built, rather than being assembled from existing components.

Software Characteristics (CO1)

- Most software is custom built (built or made to a particular customer's order.
-), rather than being assembled from existing components.
- Software is developed or engineered; It is not manufactured in the classical sense.
- Software is flexible and new functionalities can be added on later.
- Software doesn’t wear out.

Software vs Hardware

Software does not have wear out phase

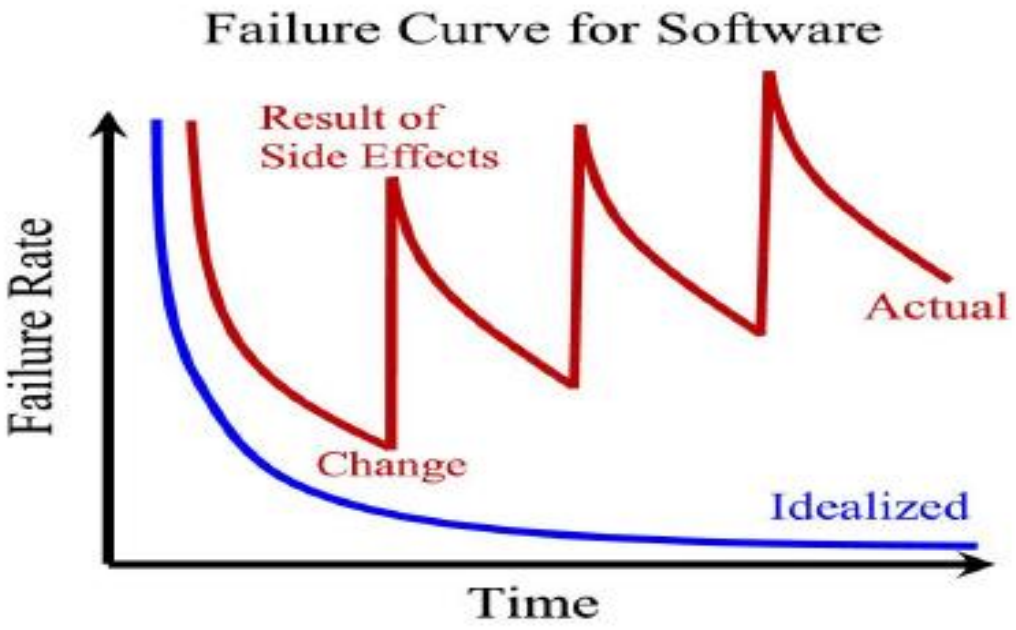


Figure: Software does not have wear out

Software vs Hardware

Failure rate of hardware

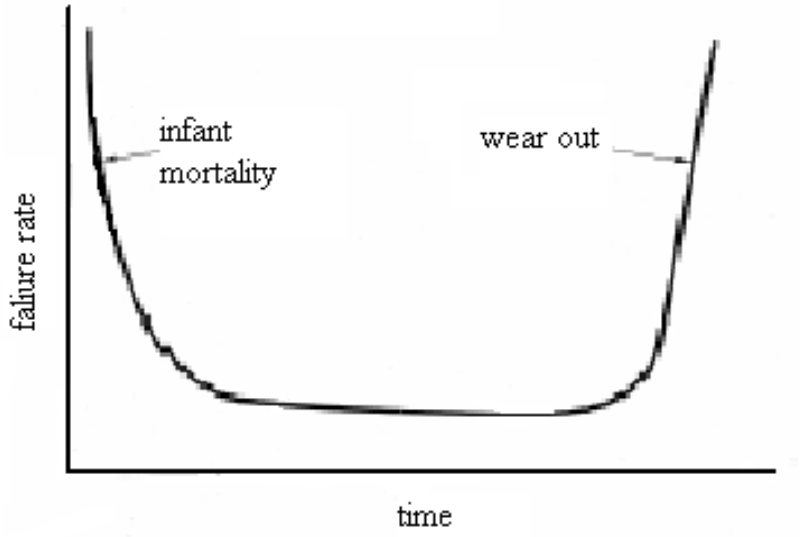


Figure: Failure rate of hardware

Software vs Hardware

- We don’t have this phase for the software as it does not wear out.
- Important point is software becomes reliable overtime instead of wearing out
- It becomes obsolete(no longer in use).
- Software may be retired due to environmental changes , new requirement and new expectation

No Silver Bullet(CO1)

- The familiar software project, at least as seen by the nontechnical manager, has something of this character; it is usually innocent and straightforward, but is capable of becoming a monster of

✓ missed schedules
✓ blown budgets
✓ flawed products.

- So we hear desperate cries for a silver bullet--something to make software costs drop as rapidly as computer hardware costs do.

- We see no silver bullet. There is no single development, in either technology or in management technique, that by itself promises even one order-of-magnitude improvement in

✓ productivity
✓ Reliability
✓ simplicity.

No Silver Bullet (CO1)

The inherent properties of this irreducible essence of modern software systems:

- **Complexity:** Software entities are more complex for their size
- **Conformity:** Much of the complexity that he must master is arbitrary complexity, forced without rhyme or reason by the many human institutions and systems to which his interfaces must conform. These differ from interface to interface, and from time to time, not because of necessity but only because they were designed by different people.
- **Changeability:** The software entity is constantly subject to pressures for change.
- **Invisibility:** Software is invisible and unvisualizable. It has no ready geometric representation

Software Myth (CO1)

What is Software Myth?

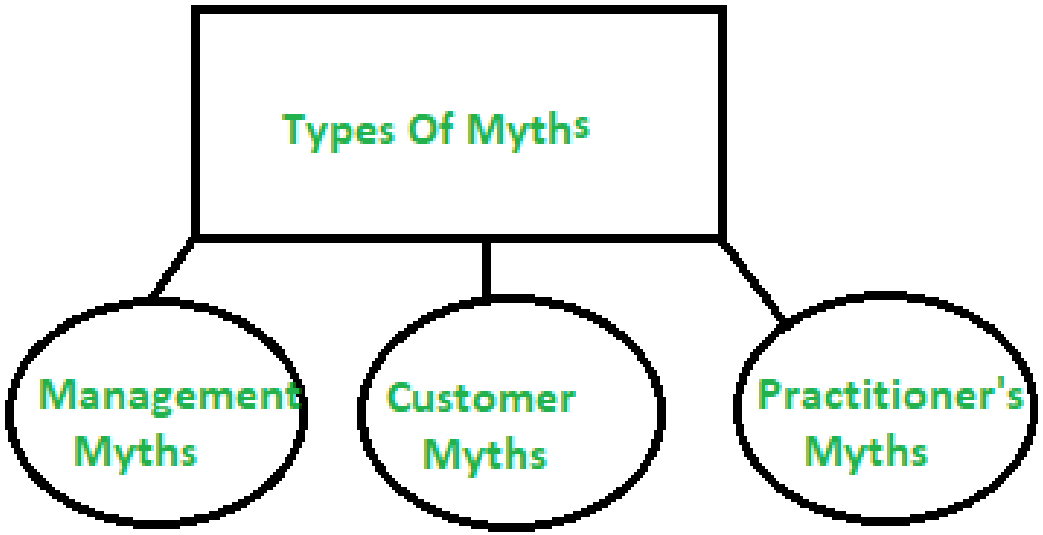
Beliefs about software and the process used to build it. Software myths are misleading attitudes that have caused serious problems for managers and technical people alike. Software myths propagate misinformation and confusion.

Types of Software Myth-

There are 3 types of Software Myth-

1. Management Myth
2. Customer Myth
3. Practitioner's Myth

Software Myth (CO1)



Types of Software Myths

Software Myth (CO1)

1. Management Myths :

Myth 1:

- We have all the standards and procedures available for software development i.e. the software developer has all the reqd.

➢ Fact :

- Software experts do not know that there are all of them levels.
- Such practices may or may not be expired at present / modern software engineering methods.
- And all existing processes are incomplete.

Software Myth (CO1)

➢**Myth 2 :**

There is no need to change approach to s/w development, we can develop same s/w that we developed 10 years ago.

➢Fact:

Software development is a rapidly evolving field, and several factors like technical advances, changing requirements, security etc contribute to the continuous evolution of development approaches, methodologies, and technologies.

➢**Myth 3 :**

Managers think that, with the addition of more people and program planners to Software development can help meet project deadlines (If lagging behind).

➢Fact : Software development is not, the process of doing things like production; here the addition of people in previous stages can reduce the time it will be used for productive development, as the newcomers would take time existing developers of definitions and understanding of the file project.

Software Myth (CO1)

(ii)Customer Myths :

The customer can be the direct users of the software, the technical team, marketing / sales department, or other company. Customer has myths

Leading to false expectations (customer) & that's why you create dissatisfaction with the developer.

- Myth 1 :

A general statement of intent is enough to start writing plans (software development) and details of objectives can be done over time.

- Fact:

Official and detailed description of the database function, ethical performance, communication, structural issues and the verification process are important.

It is happening that the complete communication between the customer and the developer is required.

Software Myth (CO1)

- **Myth 2 :**

Project requirements continue to change, but, change, can be easy accomadate due to the flexible nature of the software.

- Fact :

Changes were made to the final stages of software development but cost to make those changes grow through the latest stages of development. A detailed analysis of user needs should be done to minimize change requirement.

(iii)Practitioner's Myths :

- **Myths 1 :**

They believe that their work has been completed with the writing of the plan and they received it to work.

Software Myth (CO1)

- Fact:

It is true that every 60-80% effort goes into the maintenance phase (as of the latter software release). Efforts are required, where the product is available first delivered to customers.

- **Myths 2**

There is no other way to achieve system quality, behind it done running.

- Fact:

Systematic review of project technology is the quality of effective software verification method. These updates are quality filters and more accessible than test. FTR(Formal Technical review) do all these reviews.

- **Myth 3 :**

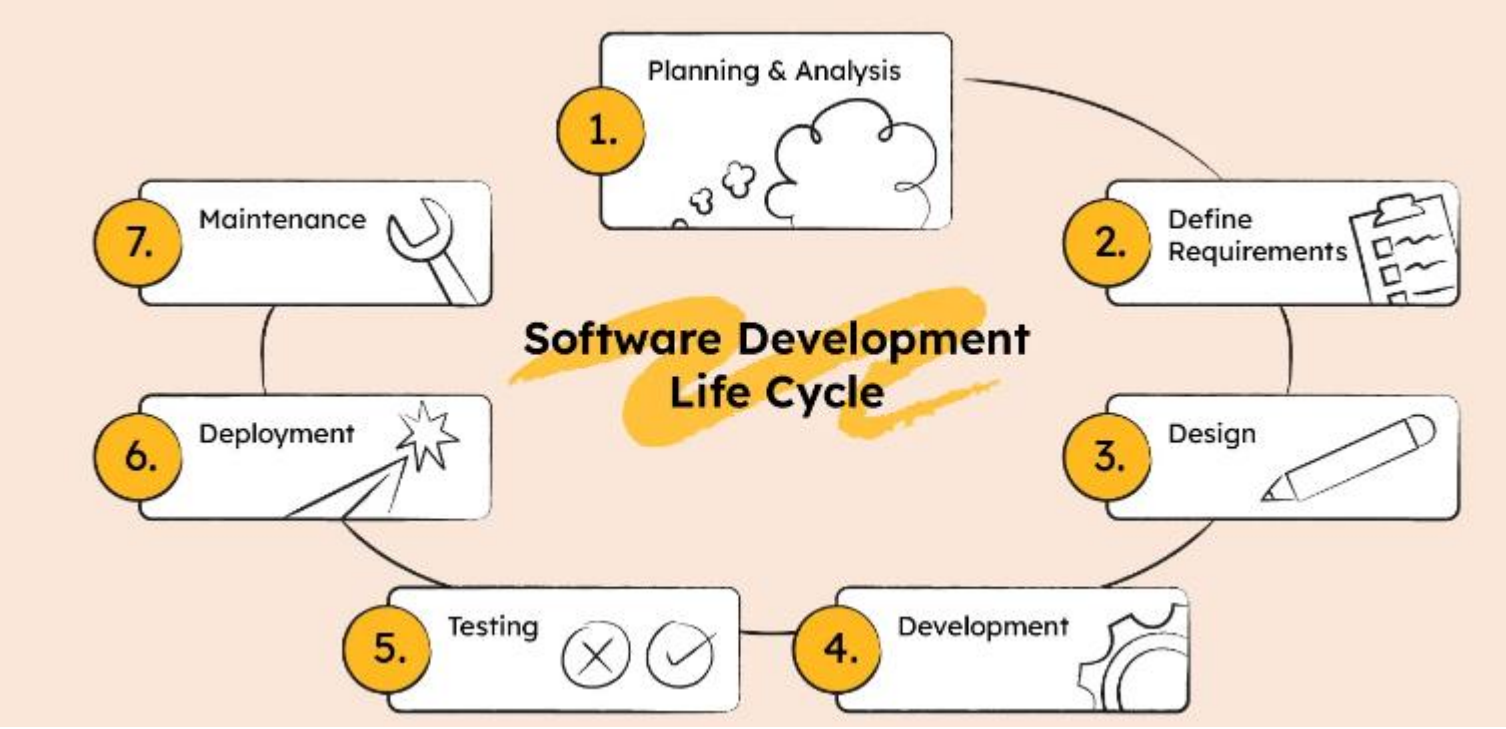
Delivered program is the only working program.

- Fact:

A working system is not enough, it is just the right document brochures and booklets that are required to provide guidance & software support.

Phases of the Software Development Life Cycle

- Software engineering typically involves several phases in the development lifecycle, and these phases may vary slightly depending on the specific development methodology used (e.g., Waterfall, Agile, DevOps). However, a common set of phases includes:



Phases of the Software Development Life Cycle

1. Planning & Analysis

The first phase of the SDLC is the project planning stage where you are gathering business requirements from your client or stakeholders. This phase is when you evaluate the feasibility of creating the product, revenue potential, the cost of production, the needs of the end-users, etc.

2. Define Requirements

This phase is critical for converting the information gathered during the planning and analysis phase into clear requirements for the development team. This process guides the development of several important documents: a software requirement specification (SRS), a Use Case document, and a Requirement Traceability Matrix document.

Phases of the Software Development Life Cycle

3. Design

The design phase is where you put pen to paper—so to speak. The original plan and vision are elaborated into a software design document (SDD) that includes the system design, programming language, templates, platform to use, and application security measures. This is also where you can flowchart how the software responds to user actions.

4. Development

The actual development phase is where the development team members divide the project into software modules and turn the software requirement into code that makes the product.

Phases of the Software Development Life Cycle

- This SDLC phase can take quite a lot of time and specialized development tools. It's important to have a set timeline and milestones so the software developers understand the expectations and you can keep track of the progress in this stage

5. Testing

Before getting the software product out the door to the production environment, it's important to have your quality assurance team perform validation testing to make sure it is functioning properly and does what it's meant to do. The testing process can also help hash out any major user experience issues and security issues.

.

Phases of the Software Development Life Cycle

- The types of testing to do in this phase:
- **Performance testing:** Assesses the software's speed and scalability under different conditions
- **Functional testing:** Verifies that the software meets the requirements
- **Security testing:** Identifies potential vulnerabilities and weaknesses
- **Unit-testing:** Tests individual units or components of the software
- **Usability testing:** Evaluates the software's user interface and overall user experience
- **Acceptance testing:** Also termed end-user testing, beta testing, application testing, or field testing, this is the final testing stage to test if the software product delivers on what it promises

Phases of the Software Development Life Cycle

6. Deployment

During the deployment phase, your final product is delivered to your intended user. You can automate this process and schedule your deployment depending on the type. For example, if you are only deploying a feature update, you can do so with a small number of users (canary release).

7. Maintenance

In the maintenance stage, users may find bugs and errors that were missed in the earlier testing phase. These bugs need to be fixed for better user experience and retention. In some cases, these can lead to going back to the first step of the software development life cycle.

Software Process(CO1)

Software process is the related set of activities and process that are involved in developing and evolving a software system.

or

A set of activities whose goal is the development or evolution of software.

or

A software process is a set of activities and associated results which produce a software product.

or

– A set of interrelated activities, which transform inputs into outputs (*ISO 12207/8402*)

used by an organization or project to plan, manage, execute, monitor, control and improve any software related activity

Software Process (CO1)

- The software industry considers software development as a process. According to Booch and Rambough A process defines *who is doing what, when and how to reach a certain goal ?*

  Software Engineering is a field, which combines process, methods and tools for the development of software.

  The concept of process is the main step in the software engineering approach. When these activities are performed in specific sequence in accordance with ordering constraints, the desired results are produced.

Software Process (CO1)

- A software process is the set of activities and associated outcome that produce a software product. Software engineers mostly carry out these activities. These are four key process activities, which are common to all software processes. These activities are:
- **Software specifications:** The functionality of the software and constraints on its operation must be defined.
- **Software development:** The software to meet the requirement must be produced.
- **Software validation:** The software must be validated to ensure that it does what the customer wants.
- **Software evolution:** The software must evolve to meet changing client needs.

Software Process (CO1)

Software Process Model

- **A workflow model:** This shows the series of activities in the process along with their inputs, outputs and dependencies. The activities in this model perform human actions.
- **A dataflow or activity model:** This represents the process as a set of activities, each of which carries out some data transformations. The activities here may be at a lower level than activities in a workflow model. They may perform transformations carried out by people or by computers.
- **A role/action model:** This means the roles of the people involved in the software process and the activities for which they are responsible.

Team Software Process (CO1)

The Team Software Process (TSP) **guides engineering teams in developing software-intensive products**.

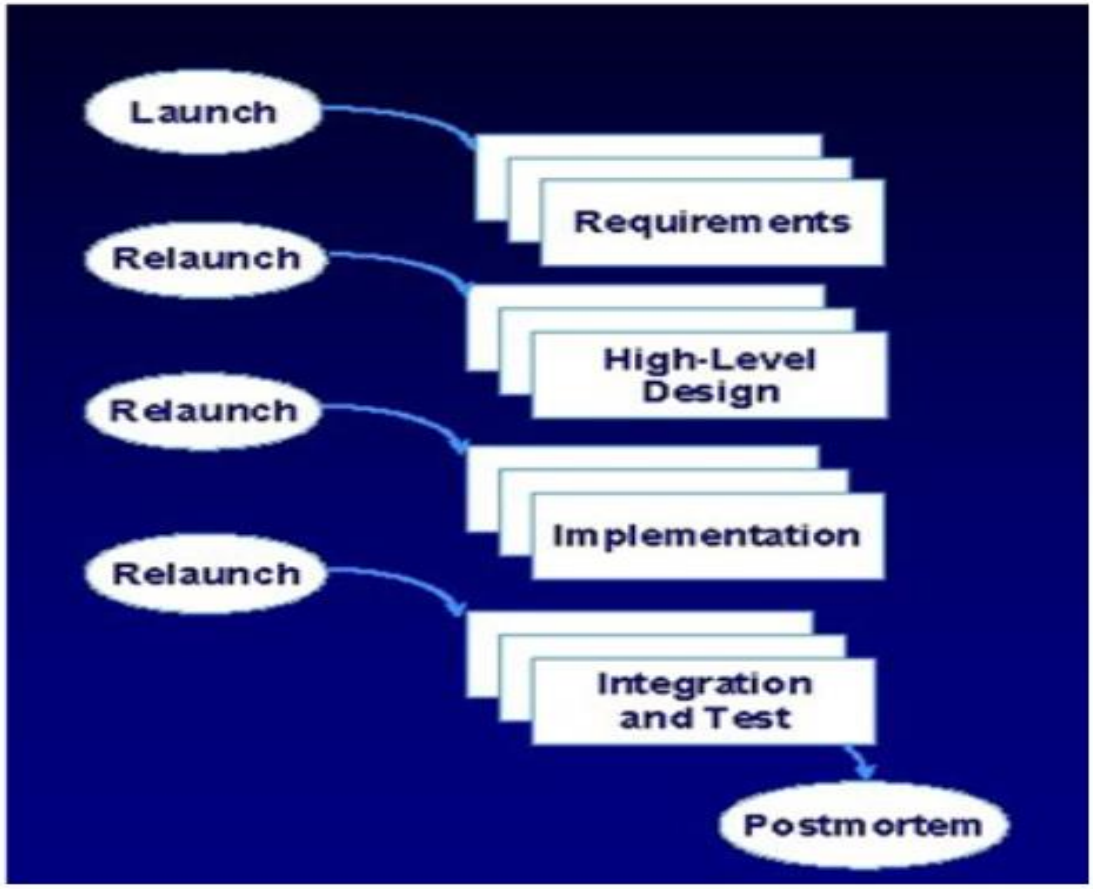
- Goal of TSP is to build "**self-directed and self-motivated**" teams to make a best software.

TSP focuses on helping development teams to improve their quality and productivity to goals of cost and progress.

Objectives of TSP explained by Humphery-

1. Build self-directed teams that plan and track their work, establish goals and own their processes and plans.
2. These can be pure s/w teams or integrated product teams(IPTs) of 3 to about 20 engineers.
3. Show managers how to coach and motivate their team members.
4. Accelerate software process to CMMI(Capability Maturity Model Integration) level 5
5. Provide guidance to high-maturity organizations.
6. It accelerate (increases) the software process development

Team Software Process (CO1)



Team Software Process (CO1)

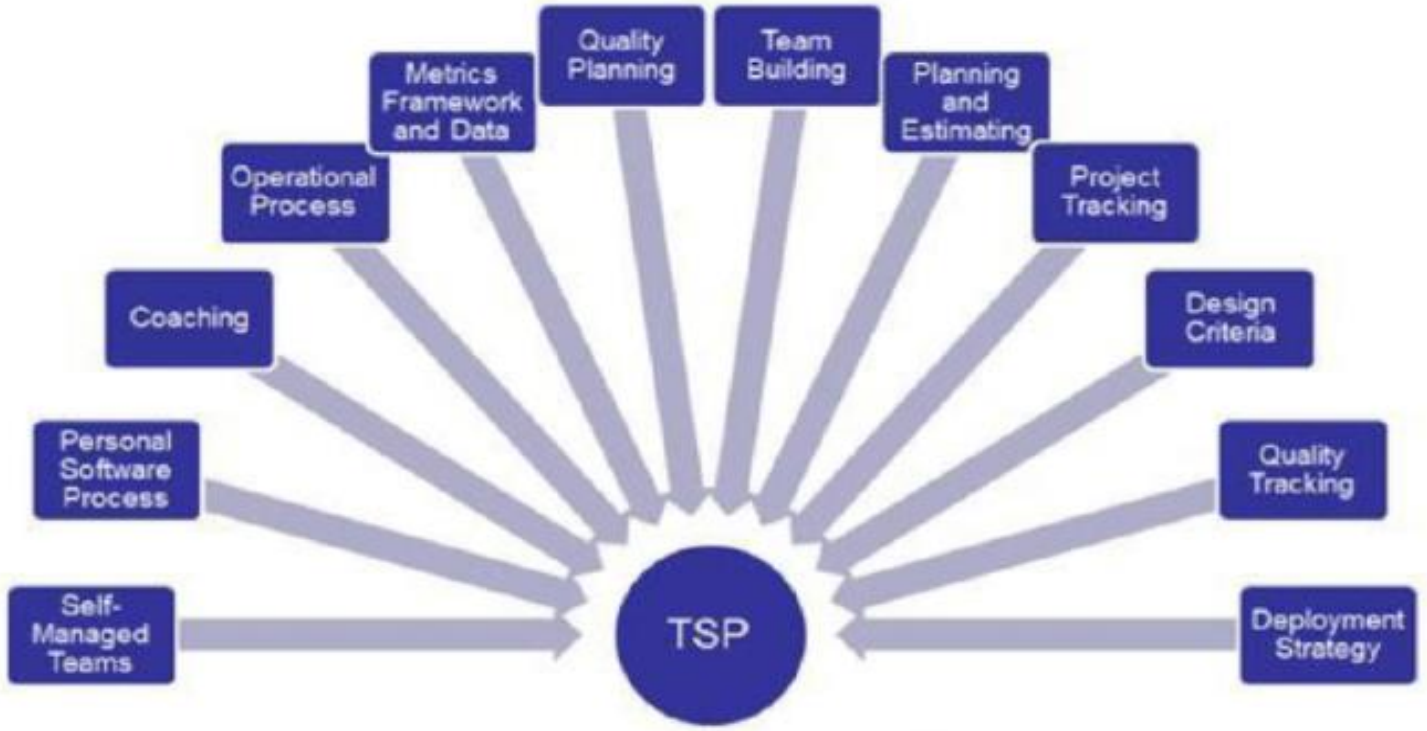
Activities of TSP-

1) **Project Launch**: It reviews project objective and describes the TSP structure and content. It assigns need and roles to the team members and describes the customers need statement.
2) **High Level Design**: it creates the high level design, specify the design, inspect the design and develop the integration plan.
3) **Implementation**: This uses the PSP to implement the modules and the functions.
4) **Integration and Testing**: Testing builds and integrates the system.
5) **Postmortem**: Writes the cycle report and produces peer and team review. Basically it works on maintenance part.

TSP Advantages

- Defines roles and responsibilities for each team member.
- Track quantitative project data.
- Identifies a team process that is appropriate for the project and a strategy for implementing the process.
- Defines local standards that are applicable to the team‘s software engineering work;
- Continually assesses risk and reacts to it;
- Tracks, manages, and reports project status.

Team Software Process (CO1)



Difference between TSP & PSP?

Evolution Of Software (CO1)

- Software Engineering principles have evolved over the past more than fifty years from art to an engineering discipline.
- Development in the field of hardware and software computing make a significant change in the twentieth century.

We can decide software development process in to four eras.

Early Era: During the early years general-purpose hardware became common place. Software, on the other hand, was custom-designed for each application and had a relatively limited distribution. Most software was developed and ultimately used by the same person or organization.

In this era the software are mainly based on (1950-1960)

- Limited Distribution
- Custom Software
- Batch Orientation

Evolution Of Software

Second Era: The second era to computer system evolution introduced new concepts of **Human Computer Interaction (HCI).** Interactive techniques opened a new world of application and new levels of hardware and software specifications. **Real time software** deals with changing environment and one other is **multi-user** in which many user can perform or work on a software at a time.

In this era the software are mainly based on (1960-1972)

- Multi-user
- Data base
- Real time
- Product software
- Multiprogramming

Evolution Of Software

Third Era: In the earlier age the software was custom designed and limited distribution but in this era the software was **consumer designed and the distribution is also not limited**. The cost of the hardware is also very low in this era.

In this era the software are mainly based on(1973-1985)

- Embedded intelligence
- Consumer impact
- Distributed systems
- Low cost hardware

Evolution Of Software

Fourth Era:- The fourth era of computer system evolution moves us away from individual computers and computer programs and towards the collective impact of computers and software. As the fourth era progresses, new technologies have begun to emerge.

In this era the software are mainly based on (1985-)

- Powerful desktop system
- Expert system
- Artificial intelligence
- Network computers
- Parallel computing
- Object oriented technology
  at this time the concept of software making is object oriented technology or network computing etc.

Emergence of Software Engineering (CO1)

Software engineering discipline is the result of advancement in the field of technology. In this section, we will discuss various innovations and technologies that led to the emergence of software engineering discipline.

1. Early Computer Programming

- As we know that in the early 1950s, computers were slow and expensive. Though the programs at that time were very small in size, these computers took considerable time to process them.

- They relied on assembly language which was specific to computer architecture. Thus, developing a program required lot of effort. Every programmer used his own style to develop the programs.

2. High Level Language Programming

- With the introduction of semiconductor technology, the computers became smaller, faster, cheaper, and reliable than their predecessors.
- . One of the major developments includes the progress from assembly language to high-level languages. Early high level programming languages such as COBOL and FORTRAN came into existence. As a result, the programming became easier and thus, increased the productivity of the programmers. However, still the programs were limited in size and the programmers developed programs using their own style and experience.

3. Control Flow Based Design (the sequence in which the program's instructions are executed)

- To help the programmer to design programs having good control flow structure, flowcharting technique was developed.

- In flowcharting technique, the algorithm is represented using flowcharts. A flowchart is a graphical representation that depicts the sequence of operations to be carried out to solve a given problem. But GOTO constructs in the flowchart makes the control flow messy, use of that should be avoided.

- Structured programming became a powerful tool that allowed programmers to write moderately complex programs easily which include loops.
- The decision structures are used for conditional execution of statements (for example, if statement).

- The purpose of structured programming is to make the software code easy to modify when required.
- The purpose of structured programming is to make the software code easy to modify when required. Some languages such as Ada, Pascal, and dBase are designed with features that implement the logical program structure in the software code.

4. Data-Flow Oriented Design

- With the introduction of very Large Scale Integrated circuits (VLSI), the computers became more powerful and faster. As a result, various significant developments like networking and GUIs came into being existence. So, new technique came which is DFD.
- In this technique, the flow of data through business functions or processes is represented using **Data-flow Diagram (DFD).**

- **IEEE** defines a data-flow diagram (also known as **bubble chart** and **work-flow diagram)** as 'a diagram that depicts data sources, data sinks, data storage, and processes performed on data as nodes, and logical flow of data as links between the nodes.'

5. **Object Oriented Design**

- Object-oriented design technique has revolutionized the process of software development. It not only includes the best features of structured programming but also some new and powerful features such as encapsulation, abstraction, inheritance, and polymorphism.
- These new features have tremendously helped in the development of well-designed and high-quality software.
- Object-oriented techniques are widely used these days as they allow reusability of the code. They lead to faster software development and high-quality programs. Moreover, they are easier to adapt and scale, that is, large systems can be created by assembling reusable subsystems.

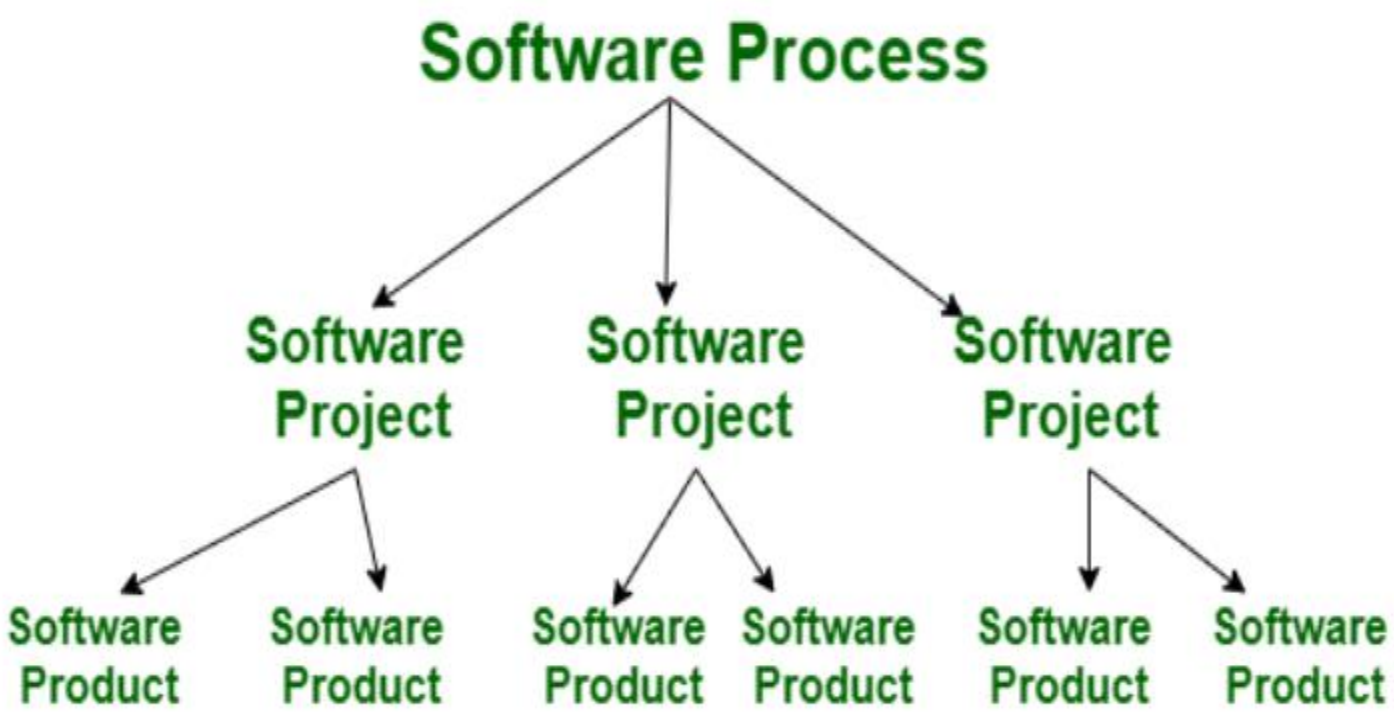
Software Project (CO1)

What is Project?

A project is well-defined task, which is a collection of several operations done in order to achieve a goal (for example, software development and delivery). A Project can be characterized as:

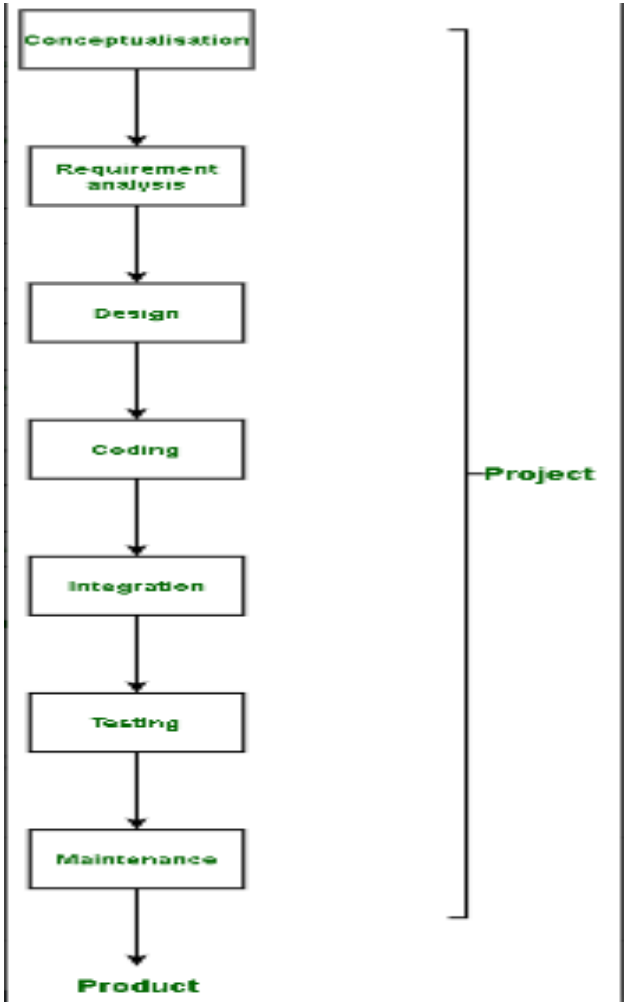
- Every project may has a unique and distinct goal.
- Project is not routine activity or day-to-day operations.
- Project comes with a start time and end time.
- Project ends when its goal is achieved hence it is a temporary phase in the lifetime of an organization.
- Project needs adequate resources in terms of time, manpower, finance, material and knowledge-bank.

Software Project (CO1)



Software Project (CO1)

The product comes into existence after the project is complete.



What is a Software Project?

A Software Project is the complete procedure of software development from requirement gathering to testing and maintenance, carried out according to the execution methodologies, in a specified period of time to achieve intended software product.

Purpose of Software Project

- To meet specific needs of a specific client or organization (known as custom software).
- To meet a perceived need of some set of potential users (known as commercial software).
- For personal use (e.g. a scientist may write software to automate a tedious task).

Software Project (CO1)

- Cost: The project budget, which serves as the financial constraint in a project.
- Scope: The activities necessary to achieve the project's goals.
- Time: The project's schedule based on which the project will be completed.

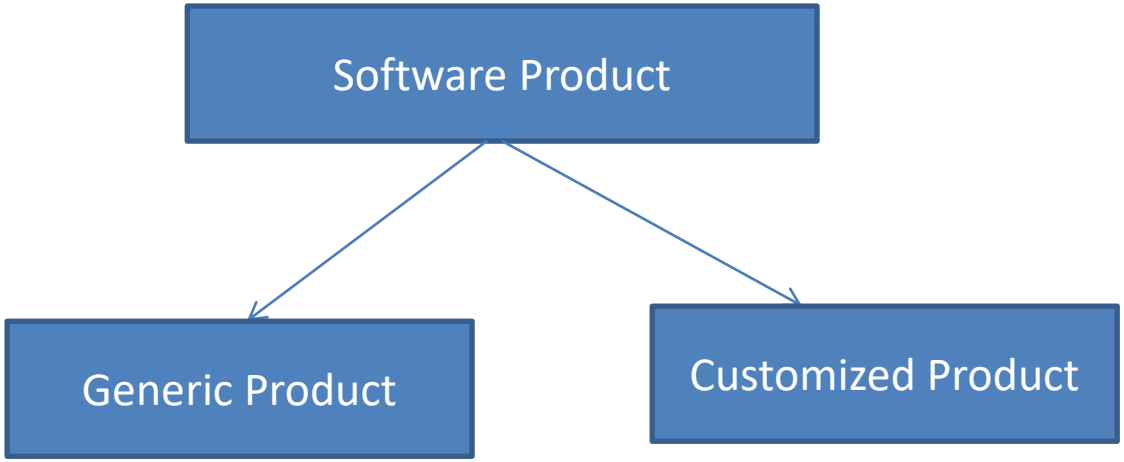
The triple constraint theory says that every project will include three constraints: **budget/cost, time, and scope**. And these constraints are tied to each other. Any change made to one of the triple constraints will have an effect on the other two.

Software Product (CO1)

- **Software Products** are nothing but software systems delivered to the customer with the documentation that describes how to install and use the system.
- In certain cases, software products may be part of system products where hardware, as well as software, is delivered to a customer.
- Software products are produced with the help of the software process. The software process is a way in which we produce software.

Types of Software Product

Software products fall into two broad categories:



Essential characteristics of Well-Engineered Software Product:

A well-engineered software product should possess the following essential characteristics:

- **Efficiency:**

The software should not make wasteful use of system resources such as memory and processor cycles.

- **Maintainability:**

It should be possible to evolve the software to meet the changing requirements of customers.

- **Dependability:**

It is the flexibility of the software that ought to not cause any physical or economic injury within the event of system failure. It includes a range of characteristics such as reliability, security, and safety.

- **In time:**

Software should be developed well in time.

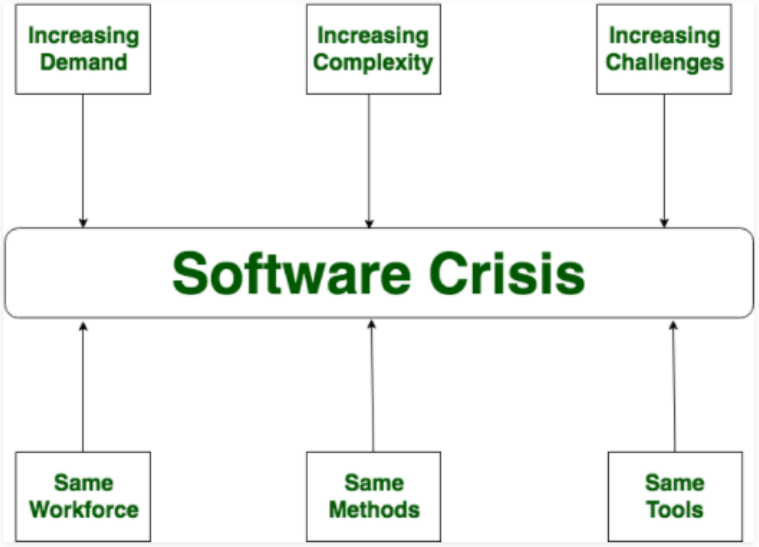
Software Crisis

Software Crisis :

It is a term used in computer science for the difficulty of writing useful and efficient computer programs in the required time.

Software crisis was due to using same workforce, same methods, same tools even though rapidly increasing in software demand, complexity of software and software challenges.

With increase in the complexity of software, many software problems arise because existing methods were insufficient.



Software Crisis

Causes of Software Crisis:

- The cost of owning and maintaining software was as expensive as developing the software
- At that time Projects was running over-time
- At that time Software was very inefficient
- The quality of software was low
- Software often did not meet requirements
- The average software project overshoots its schedule by half, at that time Software was never delivered

Software Crisis

Solution of Software Crisis:

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

- Reduction in software over-budget by proper planning.
- The quality of software must be high
- Less time needed for software project
- Experience working team member on software project
- Software must be delivered

Software Crisis

Size: Software is becoming more expensive and more complex with the growing complexity and expectation out of software. For example, the code in the consumer product is doubling every couple of years.

Quality: Many software products have poor quality, i.e., the software products defects after putting into use due to ineffective testing technique. For example, Software testing typically finds 25 errors per 1000 lines of code.

Cost: Software development is costly i.e. in terms of time taken to develop and the money involved.

Delayed Delivery: Serious schedule overruns are common. Very often the software takes longer than the estimated time to develop, which in turn leads to cost shooting up. For example, one in four large-scale development projects is never completed.

Conventional Engineering Process vs Software Engineering Process

Conventional Engineering Process : It is a engineering process which is highly based on empirical knowledge and is about building cars, machines and hardware.

Software Engineering Process : It is a engineering process which is mainly related to computers and programming and developing different kinds of applications through the use of information technology.

Comparison with conventional Engg. Process

Conventional Engg. Process	Software Engg. Process
• Based on science, mathematics and empirical knowledge	• Based on computer science, information science and discrete mathematics.
• Construct real artifact.	• Construct non real(abstract) artifact.
• Main concern is cost of production and reliability measure by time of failure.	• Main concern is cost of development and reliability is measured by no. of error per thousands lines of source code.
• 1000 years old.	• 50 years old.
• Conventional engineering are able to precisely describe and measure material they used for their work.	• Software engineering are not get able to precisely describe and measure material they used and result of their work.
• Use meter, volt and other units of measurement.	• LOC, function point or complexity measure is used to capture the amount of software created.
• Maintenance try to apply known and tested principle.	• Maintenance apply new and untested elements in software project.

Comparison with conventional Engg. Process

Comparison of constructing a bridge vis-à-vis writing a program.

Sr. No	Constructing a bridge	Writing a program
1.	The problem is well understood	Only some parts of the problem are understood, others are not
2.	There are many existing bridges	Every program is different and designed for special applications.
3.	The requirement for a bridge typically do not change much during construction	Requirements typically change during all phases of development.
4.	The strength and stability of a bridge can be calculated with reasonable precision	Not possible to calculate correctness of a program with existing methods.
5.	When a bridge collapses, there is a detailed investigation and report	When a program fails, the reasons are often unavailable or even deliberately concealed.
6.	Engineers have been constructing bridges for thousands of years	Developers have been writing programs for 50 years or so.
7.	Materials (wood, stone,iron, steel) and techniques (making joints in wood, carving stone, casting iron) change slowly.	Hardware and software changes rapidly.

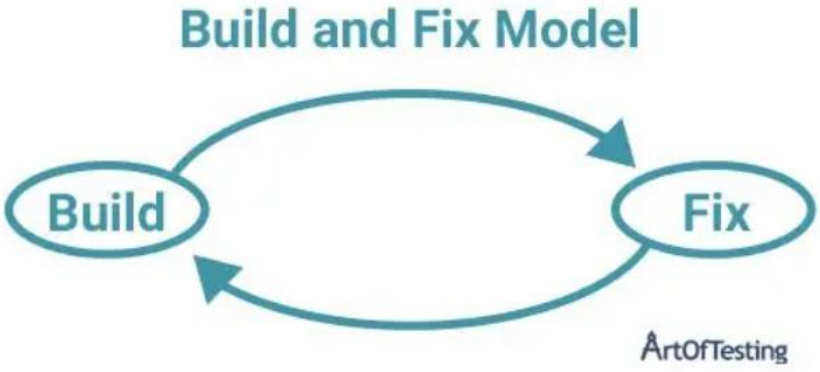
Build and Fix Model (CO1)

In the build and fix model (also referred to as an ad hoc model), the software is developed without any specification or design.

An initial product is built, which is then repeatedly modified until it (software) satisfies the user.

That is, the software is developed and delivered to the user.

Although this approach is good for smaller applications, there are several issues in this approach like – no defined processes, no documentation, maintenance is very difficult, higher cost and low-quality deliverable in case of larger projects, etc.



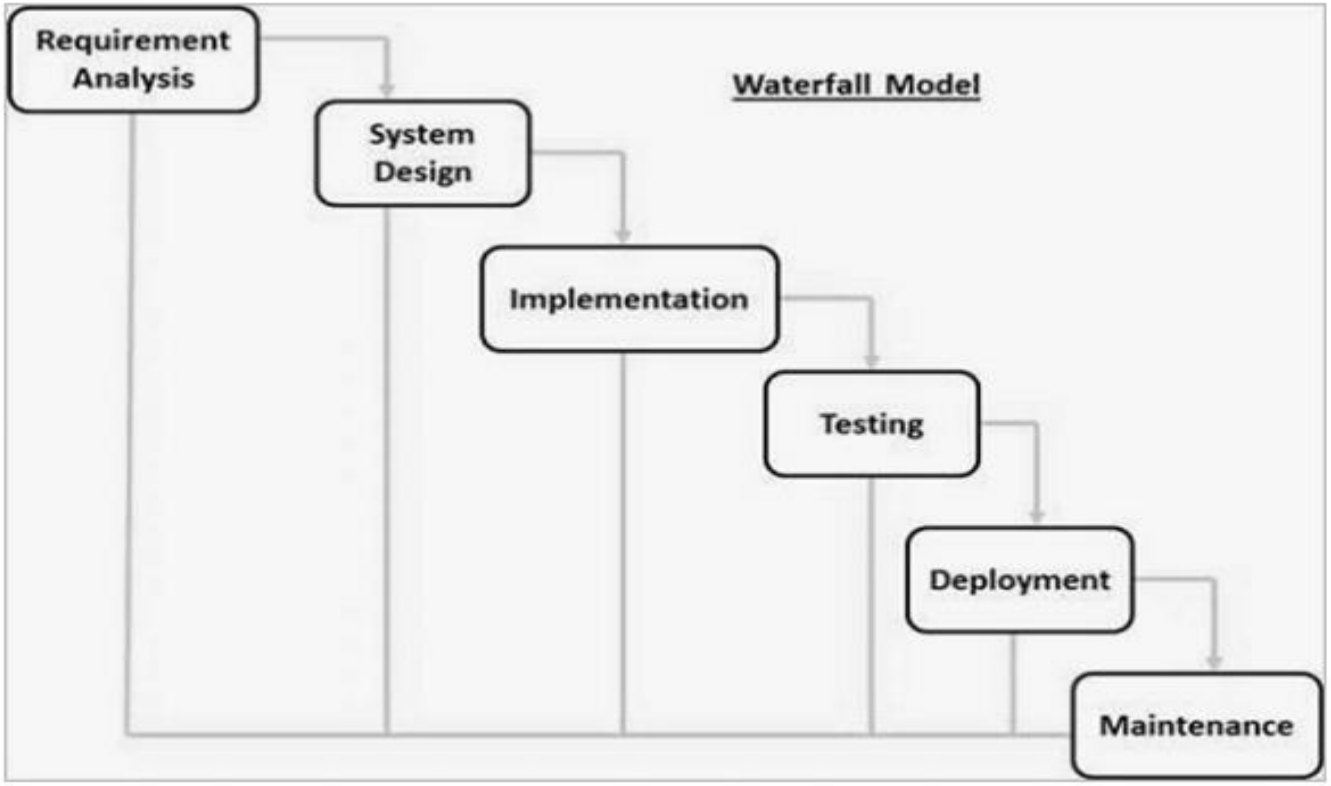
Build and fix Model

Waterfall Model (CO1)

- The waterfall model is one of the earliest models of the Software Development Life Cycle.
- The different phases in the waterfall model progress sequentially downwards, resembling a waterfall, hence the name – “Waterfall Model”.
- Once a phase of the development cycle gets completed, there is no way to go back to that phase again in order to correct it or make any desired change.
- 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.

Waterfall Model (CO1)

- **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.



Waterfall Model (CO1)

- **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.

Waterfall Model (CO1)

- **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.

Waterfall Model (CO1)

- Some situations where the use of Waterfall model is most appropriate are −
- Requirements are very well documented, clear and fixed.
- Product definition is stable.
- Technology is understood and is not dynamic.
- There are no ambiguous requirements.
- Ample resources with required expertise are available to support the product.
- The project is short.

Waterfall Model (CO1)

Waterfall Model Advantages/Pros

- 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.

Waterfall Model (CO1)

Waterfall Model Disadvantages/Cons

- As this model requires freezing of requirements, hence, it not suitable for projects in which changes in requirements are possible or inevitable.
- The working model is only visible in the later phases of the life cycle i.e. after the implementation phase.
- Any correction or update required in the previous phase is not possible.
- 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.

Prototype Model (CO1)

In the case of the prototype model, a working model of the application is created with limited functionality.

The prototype is then shown to the customer to get the feedback.

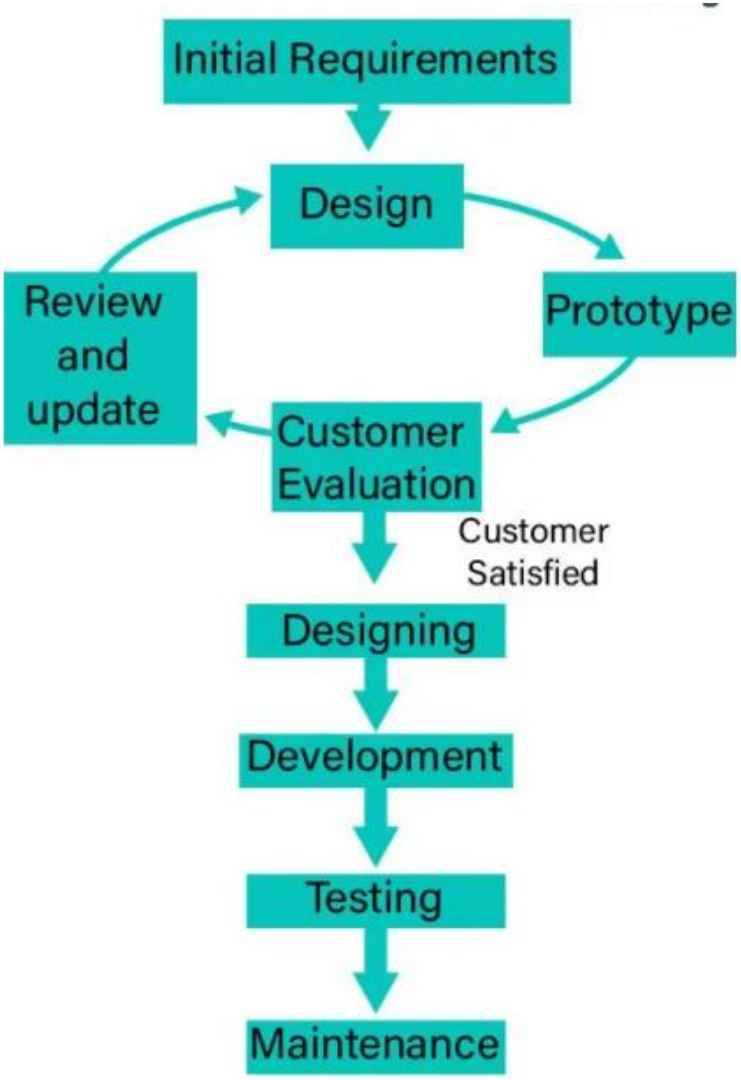
This helps in better requirement understanding.

It have 2 types in general :

1) Throwaway prototype

2) Evolutionary prototype

Prototype Model (CO1)



Prototype Model (CO1)

Prototype Model Advantages/Pros

- Customer satisfaction gets improved as they can see the working software early in the initial phases.
- It is easier to figure out any missing requirements and incorporate them.
- The developed prototype can also be reused by developers in the project.

Prototype Model (CO1)

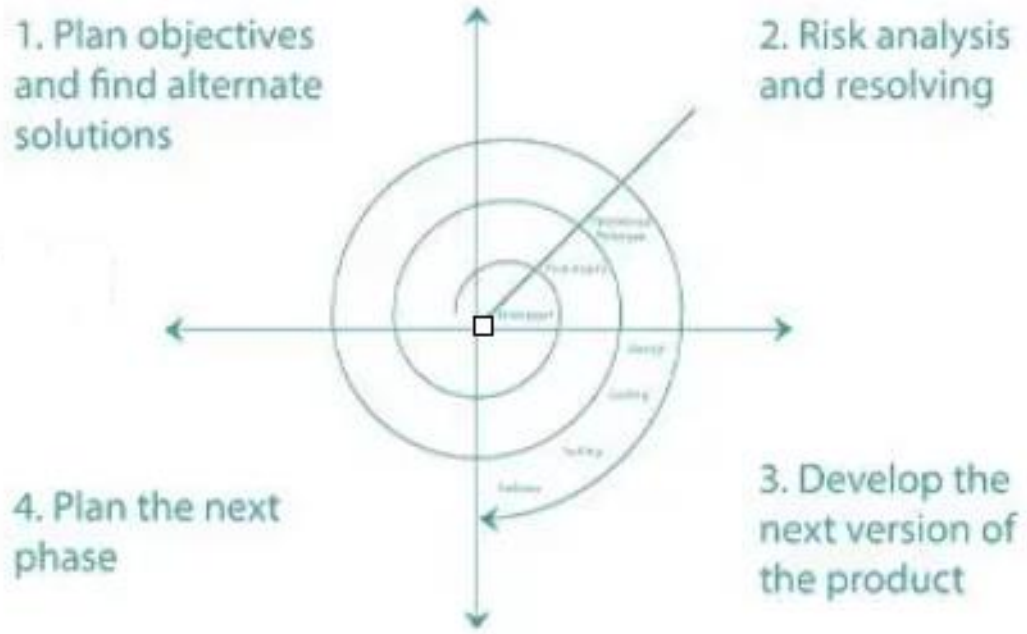
Prototype Model Disadvantages/Cons

- There can be a lack of documentation, since, both the development team as well as the customer rely on the evaluation and feedback of the prototype.
- Overall cost and time can increase due to prototype creation and evaluation.
- There can be uncertainty in the number of iterations that may be required before the prototype gets finally accepted by the customer.

Spiral Model

- The Spiral Model is the combination of the iterative software development model and the waterfall model.
- The most important feature of the model is that even after the project starts, it has the ability to manage and mitigate unknown risks. Hence, it is advisable to use this model for large and complex projects.
- This model has four stages – planning, risk analysis, engineering, and the review phase. A project passes through all these stages repeatedly.

Spiral Model



Spiral Model

Spiral Model Advantages

- The spiral model is perfect for projects that are large and complex in nature as continuous prototyping and evaluation help in mitigating any risk.
- Because of its risk handling ability, the model is best suited for projects which are very critical like software projects related to health domain, space exploration, etc.
- Since customer gets to see a prototype in each phase, so there are higher chances of customer satisfaction.

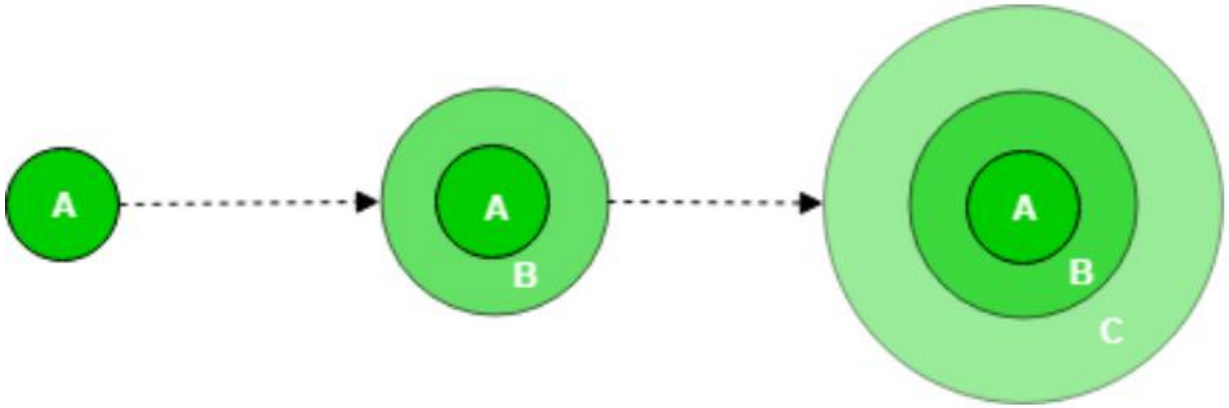
Spiral Model

Spiral Model Disadvantages

- Because of the prototype development and risk analysis in each phase, it is very expensive and time taking.
- It is not suitable for a simpler and smaller project because of the multiple phases.
- Project deadlines can be missed since the number of phases is unknown in the beginning and frequent prototyping and risk analysis can make things worse.

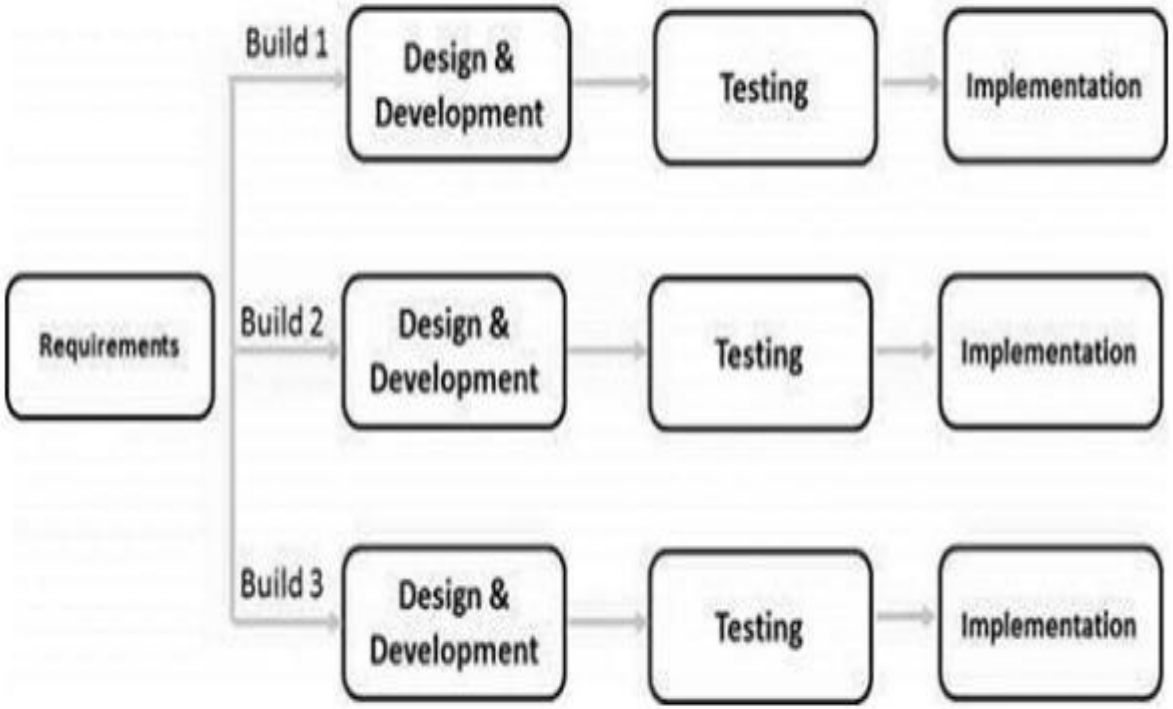
Incremental Model

- The incremental process model is also known as the Successive version model.
- First, a simple working system implementing only a few basic features is built and then that is delivered to the customer. Then thereafter many successive iterations/ versions are implemented and delivered to the customer until the desired system is released.
- A, B, and C are modules of Software Products that are incrementally developed and delivered.



NIET Greater Noida
GET FUTURE READY
Estd 2001
AN AUTONOMOUS INSTITUTE

Incremental Model



Incremental Model

- In this incremental model, the whole requirement is divided into various builds. During each iteration, the development module goes through the requirements, design, implementation and testing phases. Each subsequent release of the module adds function to the previous release. The process continues till the complete system is ready as per the requirement.
- This model is most often used in the following scenarios −

➢ Requirements of the complete system are clearly defined and understood.

➢ Major requirements must be defined; however, some functionalities or requested enhancements may evolve with time.

➢ There is a time to the market constraint.

➢ A new technology is being used and is being learnt by the development team while working on the project.

➢ Resources with needed skill sets are not available and are planned to be used on contract basis for specific iterations.

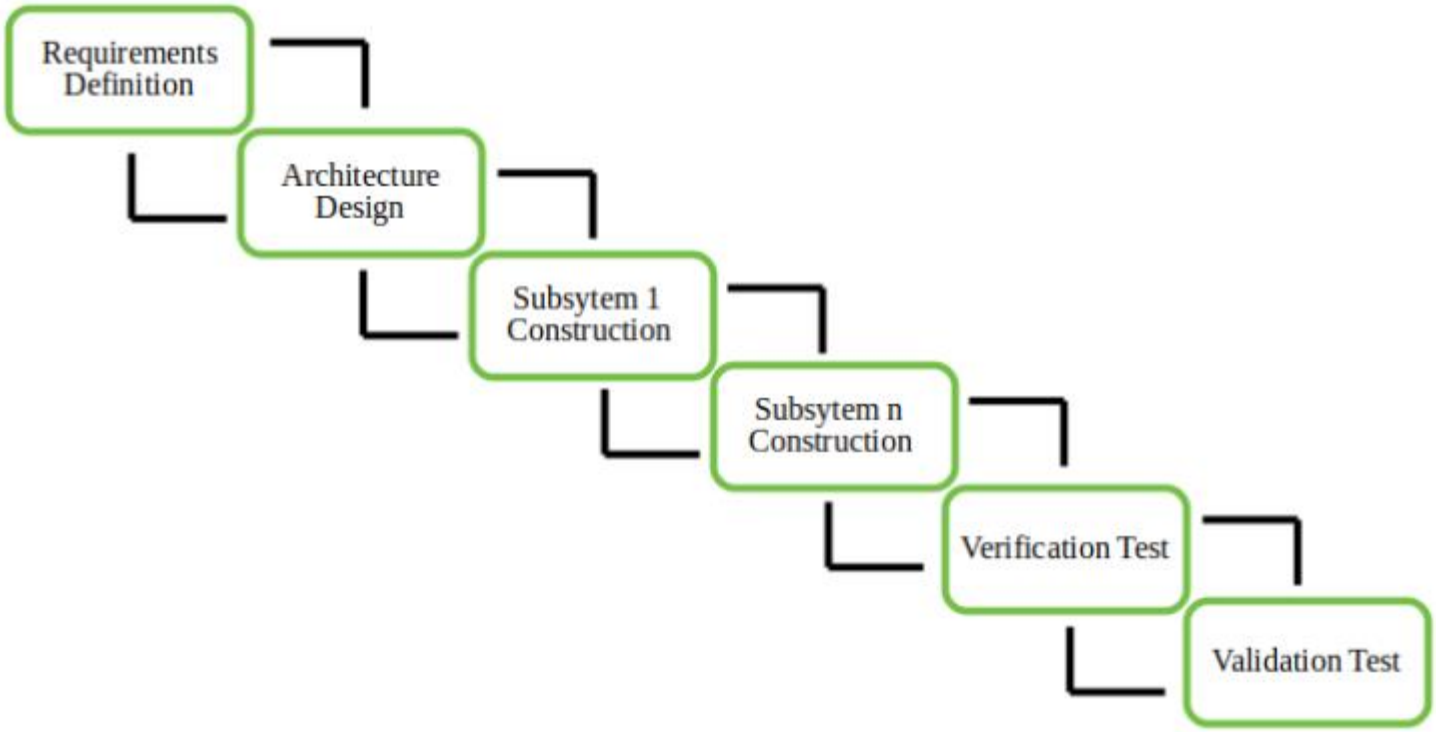
➢ There are some high-risk features and goals which may change in the future.

Incremental Model

Types of Incremental Model

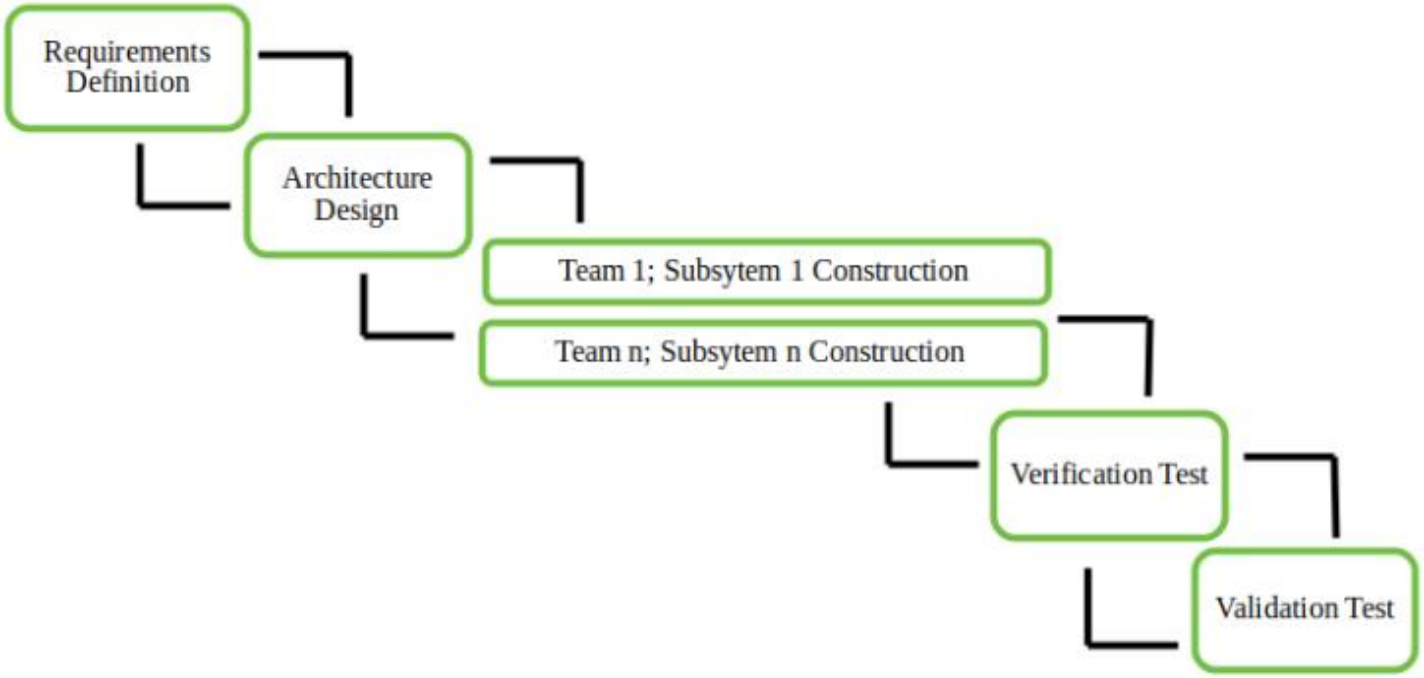
1. Staged Delivery Model

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



Incremental Model

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



Incremental Model

- **Advantages of Incremental Process Model**
- Prepares the software fast.
- Clients have a clear idea of the project.
- Changes are easy to implement.
- Provides risk handling support, because of its iterations.
- Adjusting the criteria and scope is flexible and less costly.
- Comparing this model to others, it is less expensive.
- The identification of errors is simple.

Incremental Model

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

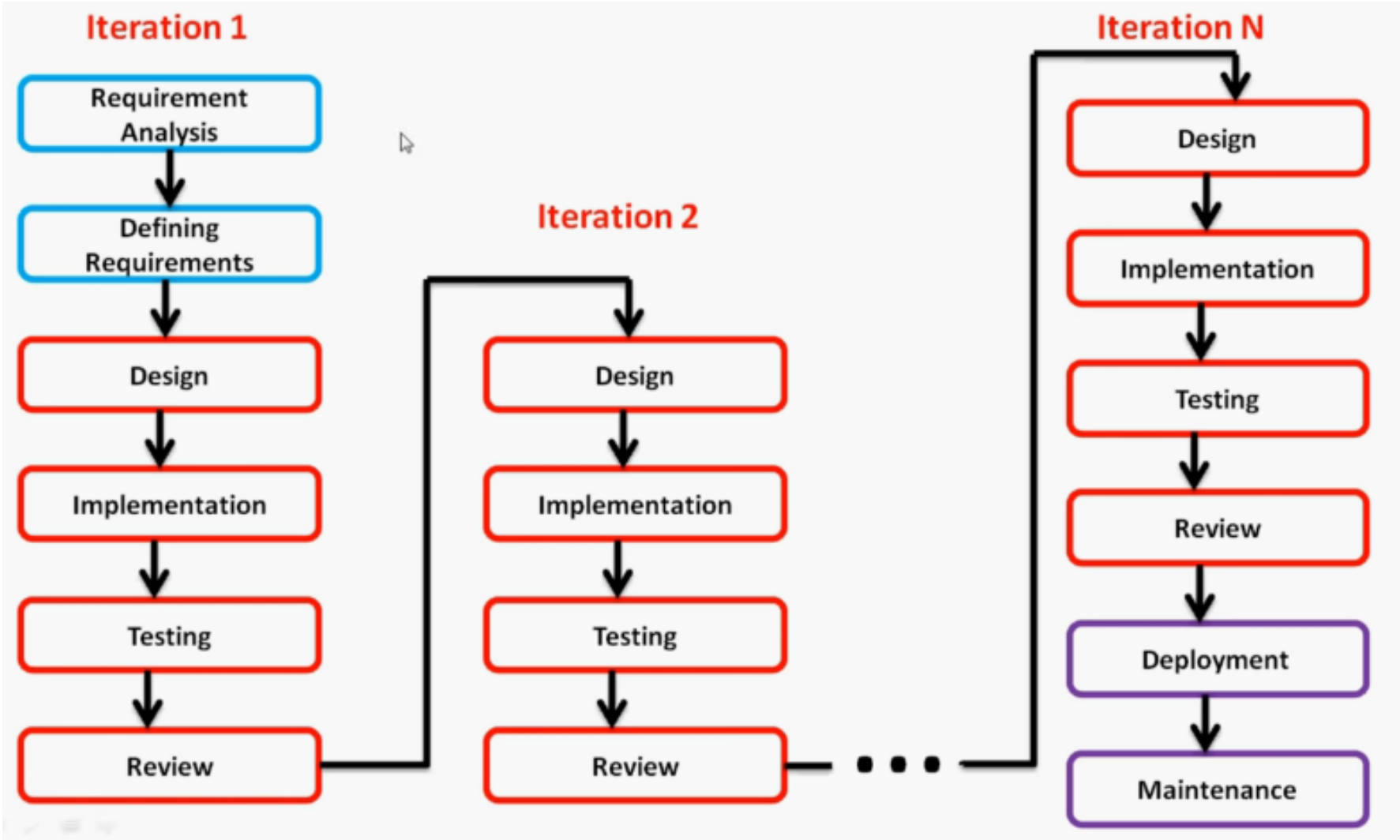
Iterative Enhancement Model (CO1)

- In the case of the Iterative model, we start with the implementation of a small set of software requirements to develop the first version of the software.

- After that, we iteratively implement and incorporate new changes resulting in the creation of new versions of the software until the complete system gets created and deployed.
- At each iteration, design modifications are made and new functional capabilities are added. The basic idea behind this method is to develop a system through repeated cycles (iterative) and in smaller portions at a time (incremental).

Iterative Enhancement Model

Iterative Enhancement Model



Iterative Enhancement Model (CO1)

- **When to use the Iterative Model?**
- When requirements are defined clearly and easy to understand.
- When the software application is large.
- When there is a requirement of changes in future.
- **Advantage(Pros) of Iterative Model:**
- Testing and debugging during smaller iteration is easy.
- A Parallel development can plan.
- It is easily acceptable to ever-changing needs of the project.
- Risks are identified and resolved during iteration.
- Limited time spent on documentation and extra time on designing.

Iterative Enhancement Model(CO1)

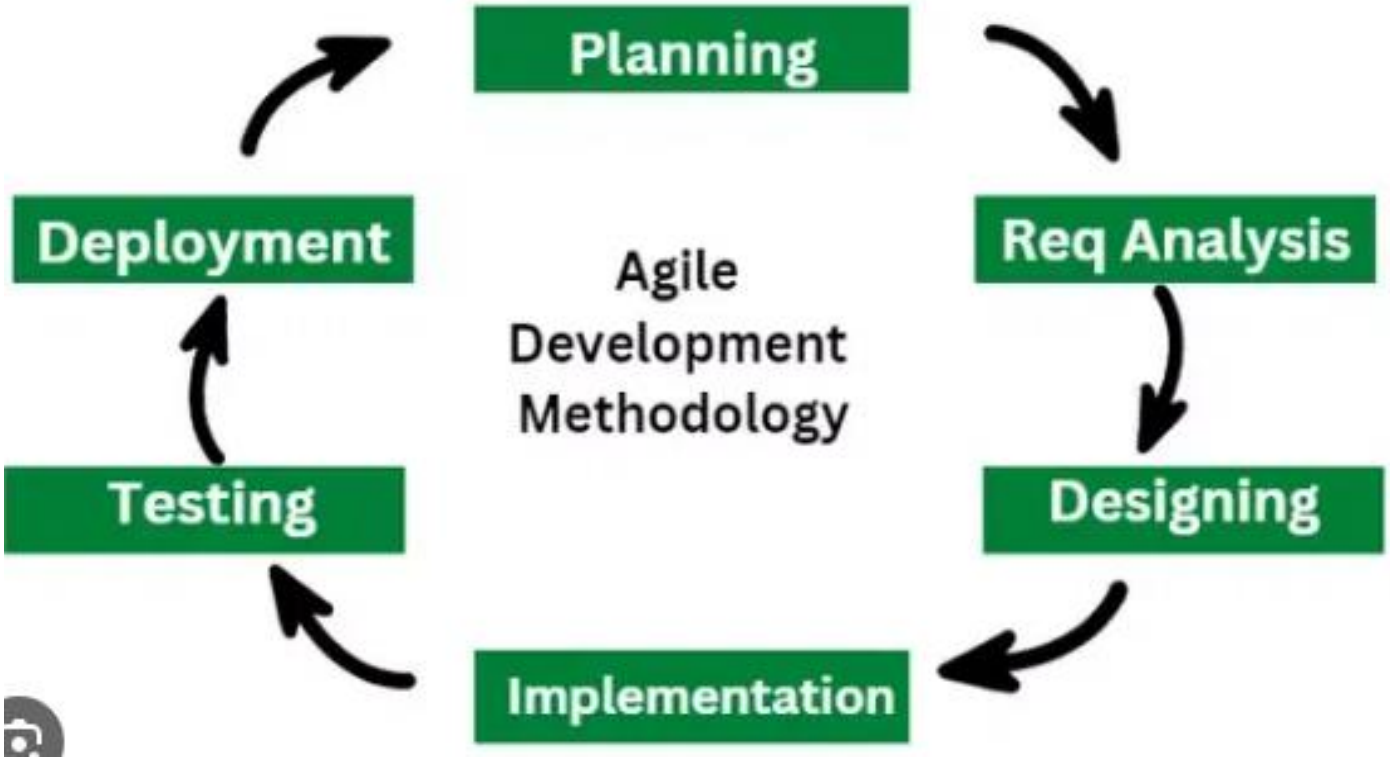
➢ **Disadvantage(Cons) of Iterative Model:**

- It is not suitable for smaller projects.
- More Resources may be required.
- Design can be changed again and again because of imperfect requirements.
- Requirement changes can cause over budget.
- Project completion date not confirmed because of changing requirements.

Agile Methodology

- Agile Software Development is a software development methodology that values flexibility, collaboration, and customer satisfaction.
- It is based on the Agile Manifesto, a set of principles for software development that prioritize individuals and interactions, working software, customer collaboration, and responding to change.
- Agile Software Development is an iterative and incremental approach to software development that emphasizes the importance of delivering a working product quickly and frequently.
- It involves close collaboration between the development team and the customer to ensure that the product meets their needs and expectations.

Agile Methodology



Agile Methodology

Agile Software Development

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

Agile Methodology

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

Agile Methodology

Advantages of Agile Methodology

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.
- Customer satisfaction
- Early and continuous delivery
- Embrace change
- Frequent delivery
- Collaboration of businesses and developers

Agile Methodology

Advantages of Agile Methodology

- Motivated individuals
- Face-to-face conversation
- Functional products
- Technical excellence
- Simplicity
- Self-organized teams
- Regulation, reflection and adjustment

Disadvantages of Agile Methodology

- More time and commitment.
- Greater demands on developers and clients.
- Lack of necessary documentation.
- Maintenance problem.

Agile Methodology: Scrum Model

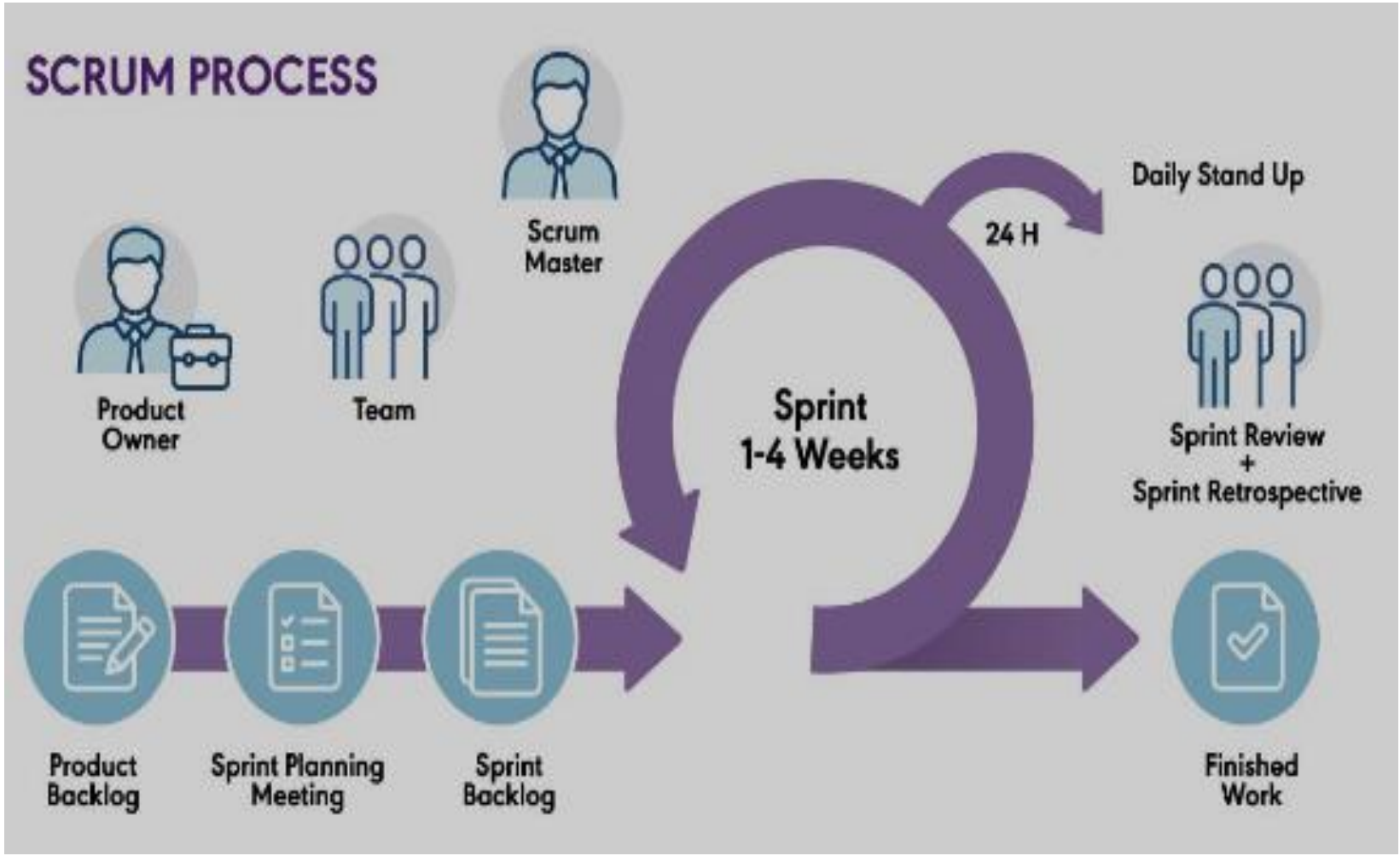
Scrum

- Scrum is a type of agile technology that consists of meetings, roles, and tools to help teams working on complex projects collaborate and better structure and manage their workload.
- It is most often used by software development teams, scrum can be beneficial to any team working toward a common goal.

Who can benefit from scrum?

- Complicated projects
- Companies that value results
- Companies that develop products in accordance with customer preferences and specifications

Agile Methodology: Scrum Model



Agile Methodology: Scrum Model

- **Backlog** – a prioritized (ordered) list of everything that might be developed in a product or service and the single source of requirements for any changes to be made in future releases. The list is typically made up of user stories , describing who wants the feature and why.
- **Sprint** – a fixed timeframe (usually two weeks) that is repeatedly used to deliver selected features from the backlog. The objective is to produce valuable product increments from each sprint.
- **Sprint planning** – a collaborative event to kick off the sprint. The planning session defines the overall goal of the sprint i.e. what backlog items the team will realistically complete, and how that work will be done.
- **Daily standup** – a short team meeting to share progress and intentions, highlight obstacles, and identify opportunities for team members to help each other get the work done. Also known as a huddle or daily scrum, standups support the concept of a self-organised team.

Agile Methodology: Scrum Model

- **Burn chart** – a technique for showing progress for a sprint, where work that is completed and work still to be done are shown with one or more lines. This is updated regularly/daily.
- **Visual management board** – a visual tool (physical or virtual) to display the status of each work item or user story in the sprint. It typically features a series of columns representing different work states and post-its representing the work, which move through the columns as work progresses.
- **User stories** – user stories are well-expressed requirements written from the perspective of an end-user outcome. They are usually short and written in the form of who, what, and why.
- **Epic** – an epic is a large user story, typically too big to be completed in a single sprint, so it needs to be broken down into smaller user stories at some point before development.
- **Retrospective** – a regular, continuous improvement activity where the team looks at how they can work together more effectively – 'what went well', 'what didn't', 'things to try' etc.

Agile Methodology: Scrum Model

Advantages

These are some of the collective benefits of agile scrum methodology:

- Flexibility and adaptability
- Creativity and innovation
- Lower costs
- Quality improvement
- Organizational synergy
- Employee satisfaction
- Customer satisfaction

Agile Methodology: Scrum Model

What are the different roles in agile scrum methodology?

1. Scrum master:

- The scrum master is the facilitator of the scrum development process.
- In addition to holding daily meetings with the scrum team, the scrum master makes certain that scrum rules are being enforced and applied as planned.
- The scrum master's responsibilities also include coaching and motivating the team, removing any hindrance to sprints, and ensuring that the team has the best possible conditions to meet its goals and produce deliverable products.

Agile Methodology: Scrum Model

What are the different roles in agile scrum methodology?

2. Product owner:

- The product owner represents stakeholders, who are typically customers.
- To ensure the scrum team is always delivering value to stakeholders and the business, the product owner determines product expectations, records changes to the product and administers a scrum backlog, a detailed and constantly updated to-do list for the scrum project.
- The product owner is also responsible for prioritizing goals for each sprint, based on their value to stakeholders, such that the most important and deliverable features are built in each iteration.

Agile Methodology: Scrum Model

What are the different roles in agile scrum methodology?

3. Scrum team:

- The scrum team is a self-organized group of three to nine individuals who have the business, design, analytical and development skills to carry out the actual work
- Solve problems and produce deliverable products.
- Members of the scrum team self-administer tasks and are jointly responsible for meeting each sprint's goals.