UNIT 1

INTRODUCTION TO SOFTWARE ENGINEERING

1.What is Software Engineering?

The term software engineering is the product of two words, software, and engineering. The software is a collection of integrated programs.

Software subsists of carefully-organized instructions and code written by developers on any of various particular computer languages.

Computer programs and related documentation such as requirements, design models and user manuals.

Engineering is the application of scientific and practical knowledge to invent, design, build, maintain, and improve frameworks, processes, etc.

Software Engineering is an engineering branch related to the evolution of software product using well-defined scientific principles, techniques, and procedures. The result of software engineering is an effective and reliable software product.

2.THE EVOLVING ROLE OF SOFTWARE

Today, software takes on a dual role. It is a product and, at the same time, the vehicle for delivering a product. As a product, it delivers the computing potential embodied by computer hardware or, more broadly, a network of computers that are accessible by local hardware. Whether it resides within a cellular phone or operates inside a mainframe computer, software is an information transformer—producing, managing, acquiring, modifying, displaying, or transmitting information that can be as simple as a single bit or as complex as a multimedia presentation.

As the vehicle used to deliver the product, software acts as the basis for the control of the computer (operating systems), the communication of information (networks), and the creation and control of other programs (software tools and environments). Software delivers the most important product of our time—information.

Software transforms personal data (e.g., an individual's financial transactions) so that the data can be more useful in a local context; it manages business information to enhance competitiveness; it provides a gateway to worldwide information networks (e.g., Internet) and provides the means for acquiring information in all of its forms.

The role of computer software has undergone significant change over a time span of little more than 50 years. Dramatic improvements in hardware performance, profound changes in computing architectures, vast increases in memory and storage capacity, and a wide variety of exotic input and output options have all precipitated more sophisticated and complex computer-based systems. The lone programmer of an earlier era has been replaced by a team of software specialists, each focusing on one part of the technology required to deliver a complex application.

3.Changing Nature of Software:

The nature of software has changed a lot over the years.

1S.ystem software: Infrastructure software come under this category like compilers, operating systems, editors, drivers, etc. Basically system software is a collection of programs to provide service to other programs.

2.Real time software: These software are used to monitor, control and analyze real world events as they occur. An example may be software required for weather forecasting. Such software will gather and process the status of temperature, humidity and other environmental parameters to forcast the weather.

3.Embedded software: This type of software is placed in "Read-Only- Memory (ROM)"of the product and control the various functions of the product. The product could be an aircraft, automobile, security system, signalling system, control unit of power plants, etc. he embedded software handles hardware components and is also termed as intelligent software .

4.Business software : This is the largest application area. The software designed to process business applications is called business software. Business software could be payroll, file monitoring system, employee management, account management. It may also be a data warehousing tool which helps us to take decisions based on available data. Management information system, enterprise resource planning (ERP) and such other software are popular examples of business software.

5.Personal computer software: The software used in personal computers are covered in this category. Examples are word processors, computer graphics, multimedia and animating tools, database management, computer games etc. This is a very upcoming area and many big organisations are concentrating their effort here due to large customer base.

6.Artificial intelligence software: Artificial Intelligence software makes use of non numerical algorithms to solve complex problems that are not amenable to computation or straight forward analysis. Examples are expert systems, artificial neural network,signal processing software etc.

7.Web based software: The software related to web applications come under this category. Examples are CGI, HTML, Java, Perl, DHTML etc.

4.LEGACY SOFTWARE

Legacy software are older programs that are developed decades ago. The quality of legacy software is poor because it has inextensible design, convoluted code, poor and nonexistent documentation, test cases and results that are not achieved.

As time passes legacy systems evolve due to following reasons:

The software must be adapted to meet the needs of new computing environment or technology. The software must be enhanced to implement new business requirements.

The software must be extended to make it interoperable with more modern systems or database The software must be rearchitected to make it viable within a network environment.

5. SOFTWARE MYTHS

Myths are widely held but false beliefs and views which propagate misinformation and confusion. Three types of myth are associated with software:

- Management myth
- Customer myth
- Practitioner's myth

MANAGEMENT MYTHS

•Myth(1)-The available standards and procedures for software are enough.

•Myth(2)-Each organization feel that they have state-of-art software development tools since theyhave latest computer.

•Myth(3)-Adding more programmers when the work is behind schedule can catch up.

•Myth(4)-Outsourcing the software project to third party, we can relax and let that party build it.

CUSTOMER MYTHS

•Myth(1)- General statement of objective is enough to begin writing programs, the details can befilled in later.

•Myth(2)-Software is easy to change because software is flexible

PRACTITIONER'S MYTH

•Myth(1)-Once the program is written, the job has been done.

•Myth(2)-Until the program is running, there is no way of assessing the quality.

Myth(3)-The only deliverable work product is the working program

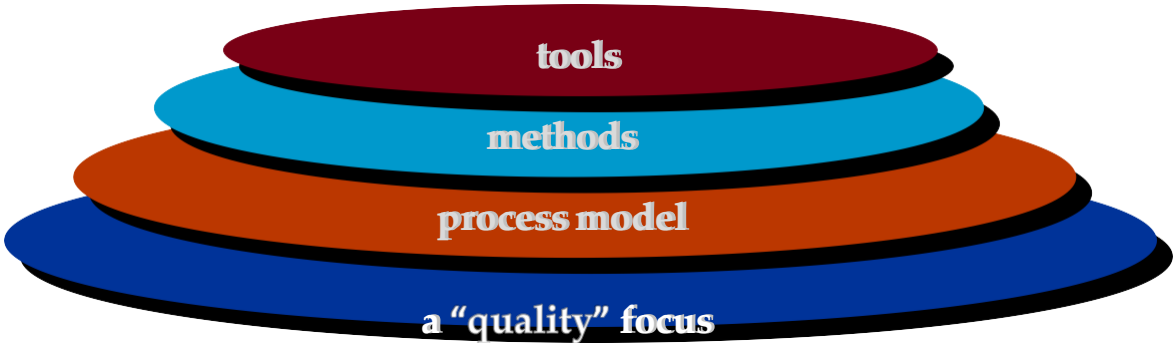
•Myth(4)-Software Engineering creates voluminous and unnecessary documentation and invariably slows down software development.

6. A Generic view of process

Process: A set of activities, methods, practices, and transformations that people use to develop and maintain software and the associated products (e.g., project plans, design documents, code, test cases, and user manuals)

Software Engineering - A Layered Technology

Software engineering encompasses a process, the management of activities, technical methods, and use of tools to develop software products

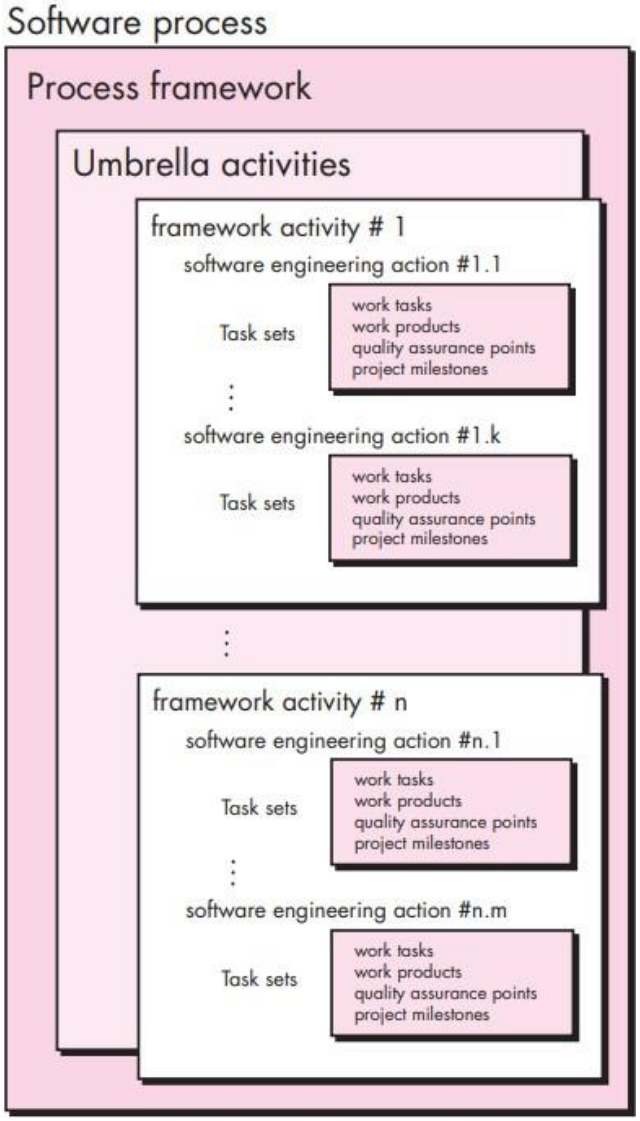


- The foundation for software engineering is the *process* layer. It is the glue that holds the technology layers together and enables rational and timely development of computer software.
- Process defines a framework that must be established for effective delivery of software engineering technology.
- The software process forms the basis for management control of software projects and establishes the context in which technical methods are applied, work products (models, documents, data, reports, etc.) are produced, milestones are established, quality is ensured, and change is properly managed.
- Software engineering *methods* provide the technical —how to's‖ for building software. Methods encompass a broad array of tasks that include communication, req. analysis, design, coding, testing and support.
- Software engineering *tools* provide automated or semi-automated support for the process and the methods.

- When tools are integrated so that info. Created by one tool can be used by another, a system for the support of software development called computer-aided software engineering is established.

7. Software Process Frame work:

FIGURE 2.1
A software process framework



A process was defined as a collection of work activities, actions, and tasks that are performed when some work product is to be created. Each framework activity is populated by a set of software engineering actions. Each software engineering action is defined by a task set that identifies the work tasks that are to be completed, the work products that will be produced, the quality assurance points that will be required, and the milestones that will be used to indicate progress.

A generic process framework for software engineering defines five framework activities—communication, planning, modeling, construction, and deployment.

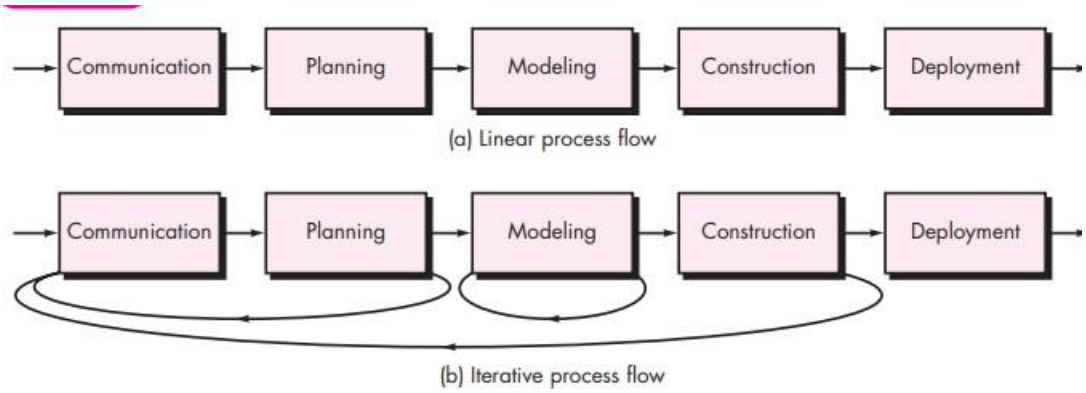
The set of umbrella activities—project tracking and control, risk management, quality assurance, configuration management, technical reviews, and others—are applied throughout the process.

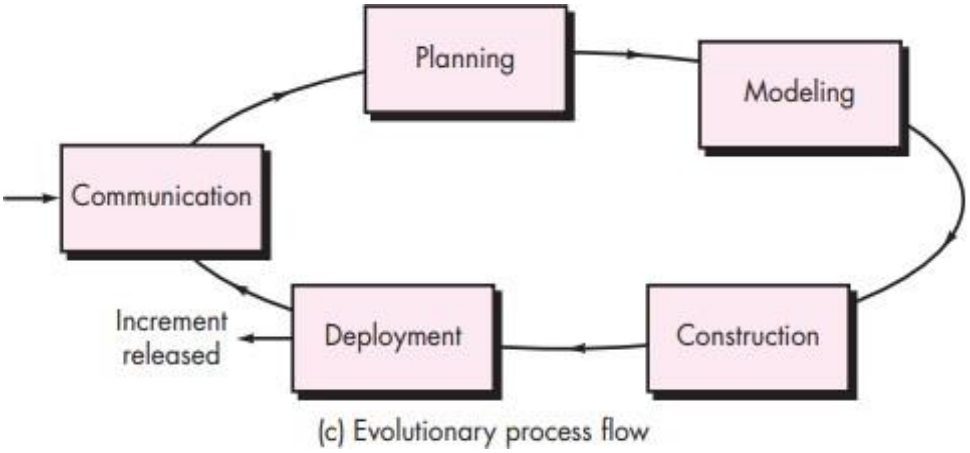
A process flow—describes how the framework activities and the actions and tasks that occur within each framework activity are organized with respect to sequence and time.

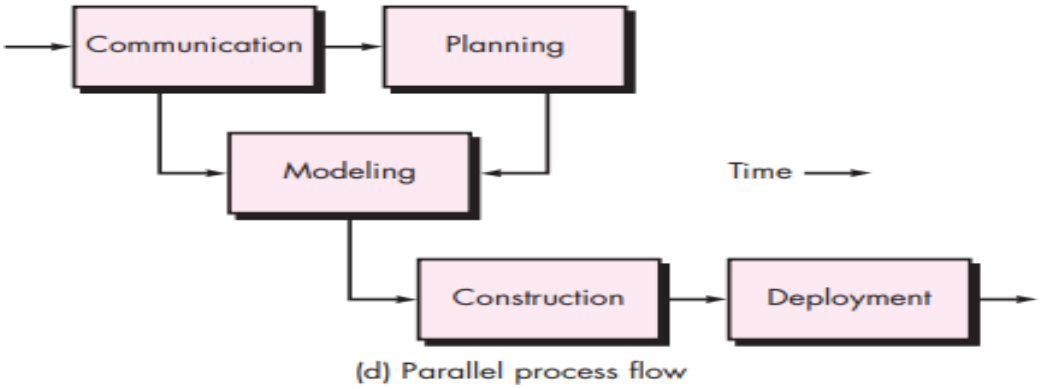
A linear process flow executes each of the five framework activities in sequence, beginning with communication and culminating with deployment. An iterative process flow repeats one or more of the activities before proceeding to the next. An evolutionary process flow executes the activities in a "circular" manner. Each circuit through the five activities leads to a more complete version of the software. A parallel process flow executes one or more activities in parallel with other activities.

1. **Defining a framework activity:**

   To define a framework activity following questions are to be answered - What actions are appropriate for a framework activity, given the nature of the problem to be solved, the characteristics of the people doing the work, and the stakeholders who are sponsoring the project?







(d) Parallel process flow

The work task (communication) (the task set) action encompasses are:

1. Make contact with stakeholder
2. Discuss requirements and take notes.
3. Organize notes into a brief written statement of requirements
4. E-mail to stakeholder for review and approval

If the project was considerably more complex with many stakeholders, each with a different set of (sometime conflicting) requirements, the communication activity might have six distinct actions:

1. inception,
2. elicitation,
3. elaboration,
4. negotiation,
5. specification, and
6. validation.

Each of these software engineering actions would have many work tasks and a number of distinct work products.

Process Framework Activities:

The process framework is required for representing common process activities. Five framework activities are described in a process framework for software engineering. Communication, planning, modeling, construction, and deployment are all examples of framework activities. Each engineering action defined by a framework activity comprises a list of needed work outputs, project milestones, and software quality assurance (SQA) points.

- **Communication:** By communication, customer requirement gathering is done. Comm unication with consumers and stakeholders to determine the system's objectives and the software's requirements.
- **Planning:** Establish engineering work plan, describes technical risk, lists resources requirements, work produced and defines work schedule.
- **Modeling:** Architectural models and design to better understand the problem and for work towards the best solution. The software model is prepared by:

1. Analysis of requirements
2. Design

- Construction: Creating code, testing the system, fixing bugs, and confirming that all criteria are met. The software design is mapped into a code by:
  1. Code generation
  2. Testing
- **Deployment:** In this activity, a complete or non-complete product or software is represented to the customers to evaluate and give feedback. On the basis of their feedback, we modify the product for the supply of better products.

2. Identifying a Task Set:

Each software engineering action can be represented by a number of different task sets—each a collection of software engineering work tasks, related work products, quality assurance points, and project milestones. We should choose a task set that best accommodates the needs of the project and the characteristics of your team. This implies that a software engineering action can be adapted to the specific needs of the software project and the characteristics of the project team.

For example: For a small, relatively simple project, the task set for requirements gathering might look like this:

1. Make a list of stakeholders for the project.
2. Invite all stakeholders to an informal meeting.
3. Ask each stakeholder to make a list of features and functions required.
4. Discuss requirements and build a final list.
5. Prioritize requirements.
6. Note areas of uncertainty.

For a larger, more complex software project, a different task set would be required. It might encompass the following work tasks:

1. Make a list of stakeholders for the project.
2. Interview each stakeholder separately to determine overall wants and needs.
3. Build a preliminary list of functions and features based on stakeholder input.
4. Schedule a series of facilitated application specification meetings.
5. Conduct meetings. 6. Produce informal user scenarios as part of each meeting.
7. Refine user scenarios based on stakeholder feedback.
8. Build a revised list of stakeholder requirements.
9. Use quality function deployment techniques to prioritize requirements.
10. Package requirements so that they can be delivered incrementally.
11. Note constraints and restrictions that will be placed on the system.
12. Discuss methods for validating the system

Both of these task sets achieve "requirements gathering," but they are quite different in their depth and formality. The software team chooses the task set that will allow it to achieve the goal of each action and still maintain quality and agility.

3. **Process Patterns:**

   A process pattern1 describes a process-related problem that is encountered during software engineering work, identifies the environment in which the problem has been encountered, and suggests one or more proven solutions to the problem. Stated in more general terms, a process pattern provides with a template —a consistent method for describing problem solutions within the context of the software process. By combining patterns, a software team can solve problems and construct a process that best meets the needs of a project.

   Patterns can be defined at any level of abstraction.2 In some cases, a pattern might be used to describe a problem (and solution) associated with a complete process model. In other situations, patterns can be used to describe a problem (and solution) associated with a framework activity (e.g., planning) or an action within a framework activity (e.g., project estimating).

Umbrella activities:

Umbrella Activities are that take place during a software development process for improved project management and tracking.

1. **Software project tracking and control:** This is an activity in which the team can assess progress and take corrective action to maintain the schedule. Take action to keep the project on time by comparing the project's progress against the plan.
2. **Risk management:** The risks that may affect project outcomes or quality can be analyzed. Analyze potential risks that may have an impact on the software product's quality and outcome.
3. **Software quality assurance:** These are activities required to maintain software quality. Perform actions to ensure the product's quality.
4. **Formal technical reviews:** It is required to assess engineering work products to uncover and remove errors before they propagate to the next activity. At each level of the process, errors are evaluated and fixed.
5. **Software configuration management:** Managing of configuration process when any change in the software occurs.
6. **Work product preparation and production:** The activities to create models, documents, logs, forms, and lists are carried out.
7. **Reusability management:** It defines criteria for work product reuse. Reusable work items should be backed up, and reusable software components should be achieved.
8. **Measurement:** In this activity, the process can be defined and collected. Also, project and product measures are used to assist the software team in delivering the required software.

8.SOFTWARE PROJECT MANAGEMENT LIFE CYCLE ACTIVITES
SDLC LIFE CYCLE: DEATILED IN RUNNING NOTES

9.SOFTWARE DEVELOPMENT LIFE CYCLE

What is SDLC?
SDLC is a process followed for a software project, within a software organization. It consists of a detailed plan describing how to develop, maintain, replace and alter or enhance specific software. The life cycle defines a methodology for improving the quality of software and the overall development process.

The following figure is a graphical representation of the various stages of a typical SDLC.

Stages of SDLC

A typical Software Development Life Cycle consists of the following stages −

Stage 1: Planning and Requirement Analysis
Requirement analysis is the most important and fundamental stage in SDLC. It is performed by the senior members of the team with inputs from the customer, the sales department, market surveys and domain experts in the industry. This information is then used to plan the basic project approach and to conduct product feasibility study in the economical, operational and technical areas.

Planning for the quality assurance requirements and identification of the risks associated with the project is also done in the planning stage. The outcome of the technical feasibility study is to define the various technical approaches that can be followed to implement the project successfully with minimum risks.

Stage 2: Defining Requirements
Once the requirement analysis is done the next step is to clearly define and document the product requirements and get them approved from the customer or the market analysts. This is done through an SRS (Software Requirement Specification) document which consists of all the product requirements to be designed and developed during the project life cycle.

Stage 3: Designing the Product Architecture
SRS is the reference for product architects to come out with the best architecture for the product to be developed. Based on the requirements specified in SRS, usually more than one design approach for the product architecture is proposed and documented in a DDS - Design Document Specification.

This DDS is reviewed by all the important stakeholders and based on various parameters as risk assessment, product robustness, design modularity, budget and time constraints, the best design approach is selected for the product.

A design approach clearly defines all the architectural modules of the product along with its communication and data flow representation with the external and third party modules (if any). The internal design of all the modules of the proposed architecture should be clearly defined with the minutest of the details in DDS.

Stage 4: Building or Developing the Product
In this stage of SDLC the actual development starts and the product is built. The programming code is generated as per DDS during this stage. If the design is performed in a detailed and organized manner, code generation can be accomplished without much hassle.

Developers must follow the coding guidelines defined by their organization and programming tools like compilers, interpreters, debuggers, etc. are used to generate the code. Different high level programming languages such as C, C++, Pascal, Java and PHP are used for coding. The programming language is chosen with respect to the type of software being developed.

Stage 5: Testing the Product
This stage is usually a subset of all the stages as in the modern SDLC models, the testing activities are mostly involved in all the stages of SDLC. However, this stage refers to the testing only stage of the product where product defects are reported, tracked, fixed and retested, until the product reaches the quality standards defined in the SRS.

Stage 6: Deployment in the Market and Maintenance
Once the product is tested and ready to be deployed it is released formally in the appropriate market. Sometimes product deployment happens in stages as per the business strategy of that organization. The product may first be released in a limited segment and tested in the real business environment (UAT- User acceptance testing).

Then based on the feedback, the product may be released as it is or with suggested enhancements in the targeting market segment. After the product is released in the market, its maintenance is done for the existing customer base.

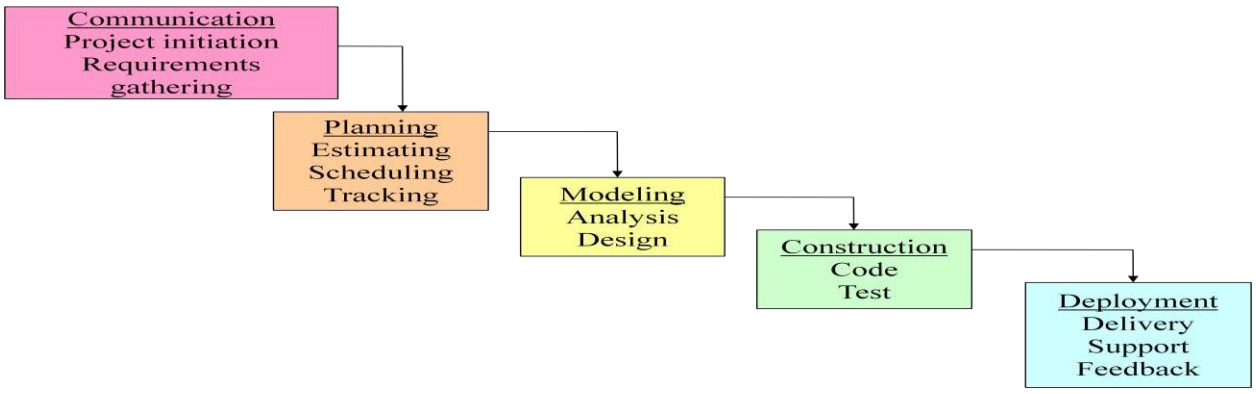
10. Process Models

A software life cycle model also called process model is a descriptive and diagrammatic representation of the software life cycle. A life cycle model represents all the activities required to make a software product transit through its life cycle phases. It also captures the order in which these activities are to be undertaken. In other words, a life cycle model maps the different activities performed on a software product from its inception to retirement.

Different life cycle models may map the basic development activities to phases in different ways. Thus, no matter which life cycle model is followed, the basic activities are included in all life cycle models though the activities may be carried out in different orders in different life cycle models. During any life cycle phase, more than one activity may also be carried out.

1. Classical Waterfall Model
2. Iterative Waterfall Model
3. Prototyping Model
4. Incremental Model
5. RAD Model
6. Spiral Model

1. Classical Waterfall Model



- Oldest software lifecycle model and best understood by upper management
- Used when requirements are well understood and risk is low
- Work flow is in a linear (i.e., sequential) fashion
- Used often with well-defined adaptations or enhancements to current software
- Begins with customer specification of Requirements and progresses through planning, modeling, construction and deployment.

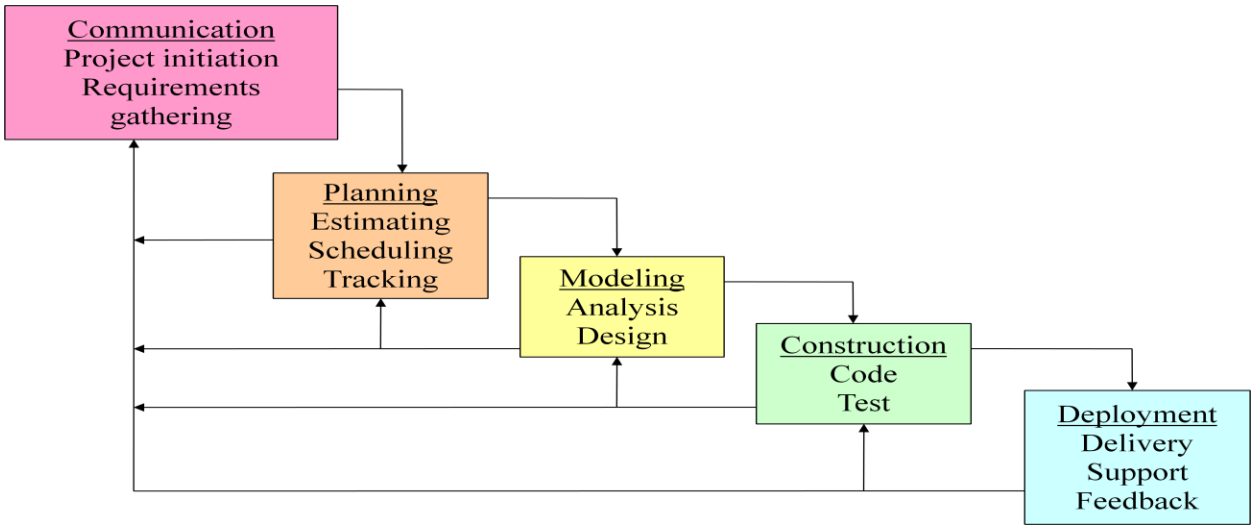
Advantages:

- It is very simple
- It divides the large task of building a software system into a seri es of clearly divided phases.
- Each phase is well documented

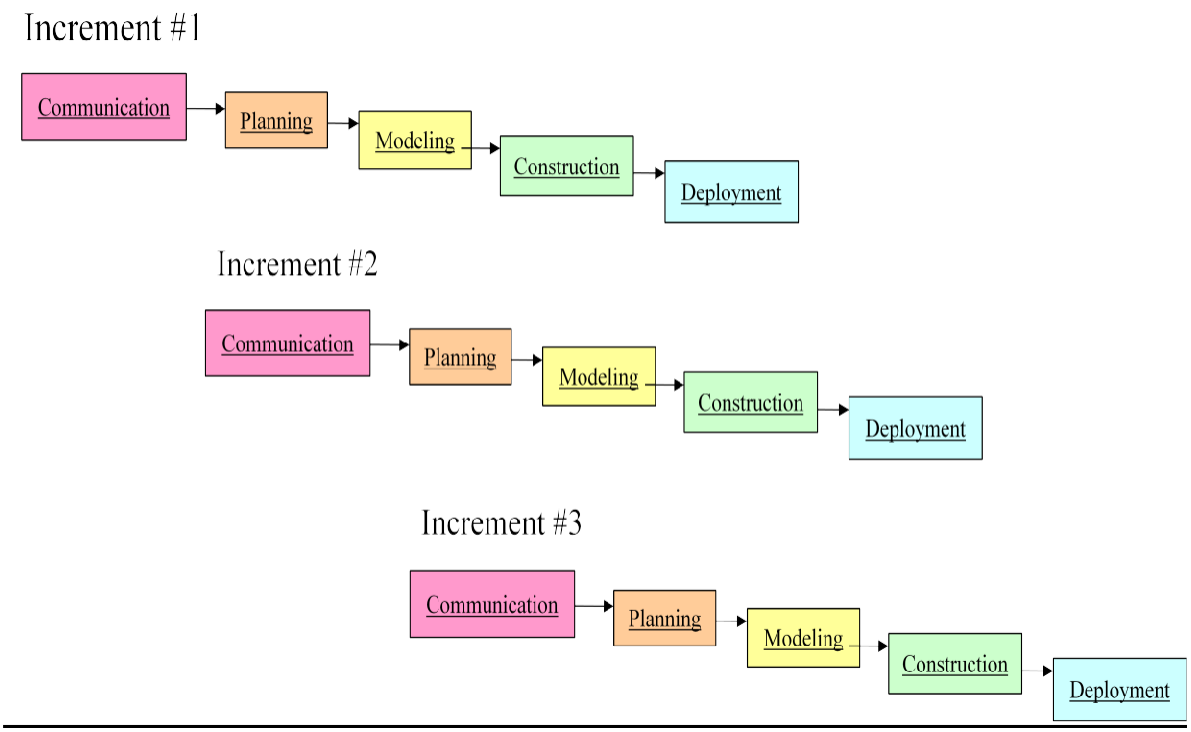
Problems

- Doesn't support iteration, so changes can cause confusion
- Difficult for customers to state all requirements explicitly and up front
- Requires customer patience because a working version of the program doesn't occur until the final phase
- Problems can be somewhat alleviated in the model through the addition of feedback loops

2. **Iterative Waterfall Model**



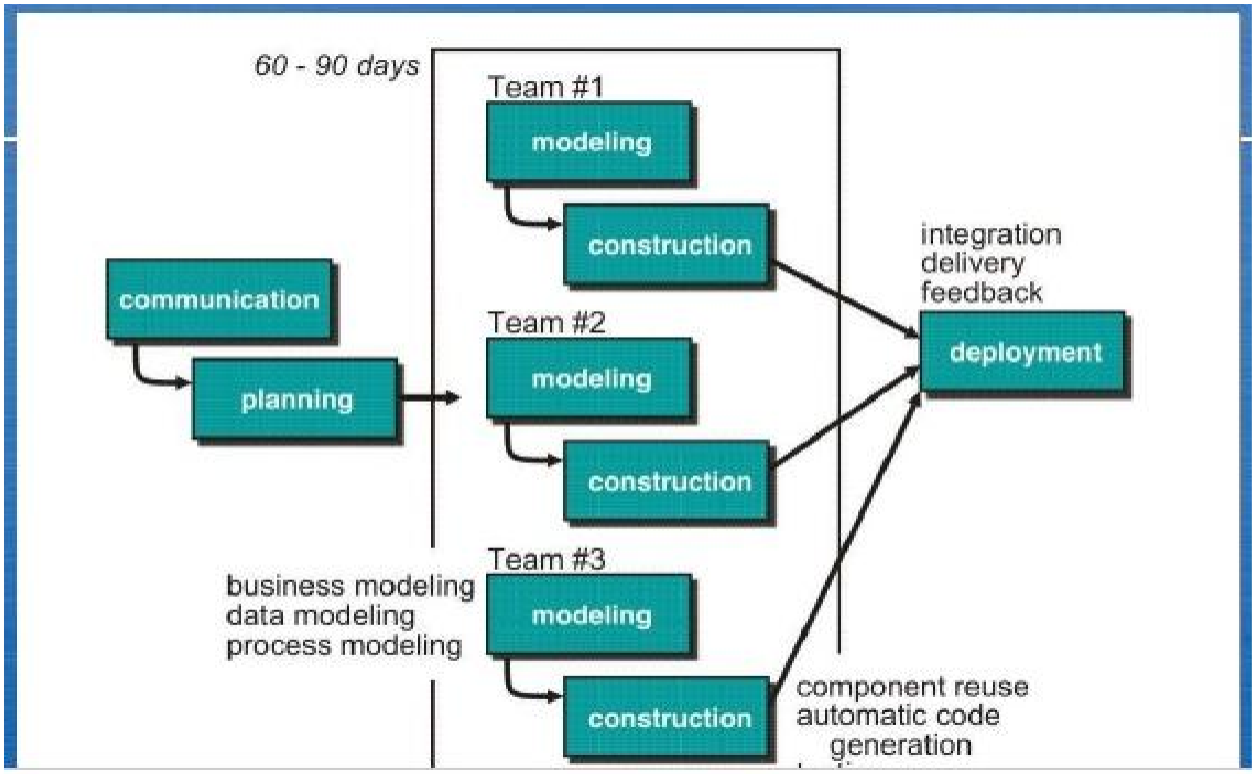
3.INCREMENTAL PROCESS MODELS



- In this life cycle model, the software is first broken down into several modules which can be incrementally constructed and delivered.
- Used when requirements are well understood
- Multiple independent deliveries are identified
- Work flow is in a linear (i.e., sequential) fashion within an increment and is staggered between increments
- Iterative in nature; focuses on an operational product with each increment
- The development team first develops the core modules of the system.
- This initial product skeleton is refined into increasing levels of capability adding new functionalities in successive versions.
- Each evolutionary version may be developed using an iterative waterfall model of development.
- Provides a needed set of functionalities sooner while delivering optional components later
- Useful also when staffing is too short for a full-scale development

4.Rapid Application Model (RAD)

- RAD is a **high speed** adaptation of linear sequential model. It is characterized by a very short development life cycle, in which the objective is to accelerate the development.
- The RAD model follows a component based approach.
- In this approach individual components developed by different people are assembled to develop a large software system.



The RAD model consist of the following phases

- Business Modeling:
  In this phase, define the flow of information within the organization, so that it covers all the functions. This helps in clearly understand the nature, type source and process of information.
- Data Modeling:
  In this phase, convert the component of the information flow into a set of data objects. Each object is referred as an *Entity.*
- Process Modeling:
  In this phase, the data objects defined in the previous phase are used to depict the flow of information . In addition adding , deleting, modifying and retrieving the data objects are included in process modeling.
- Application Designing:
  In this phase, the generation of the application and coding take place. Using fourth generation programming languages or 4 GL tools is the preferred choice for the software developers.
- Testing:
  In this phase, test the new program components.

The RAD has following advantages

- Due to emphasis on rapid development , it results in the delivery of fully functional project in short time period.
- It encourages the development of program component reusable.

The RAD has following disadvantages

- It requires dedication and commitment on the part of the developers as well as the client to meet the deadline. If either party is indifferent in needs of other, the project will run into serious problem.
- For large but scalable projects It is not suitable as RAD requires sufficient human resources to create the right number of RAD teams.
- RAD requires developers and customers who are committed to rapid fire activities
- Its application area is restricted to system that are modular and reusable in nature.
- It is not suitable for the applications that have a high degree of technical risk.
- For large but scalable projects, RAD requires sufficient human resources to create the right number of RAD teams.
- RAD requires developers and customers who are committed to rapid fire activities.
- Not all types of applications are appropriate for RAD.
- RAD is not appropriate when technical risks are high.

5.Evolutionary Process Models:

Software evolves over a period of time

- Business and product requirements often change as development proceeds making a straight-line path to an end product unrealistic
- Evolutionary models are iterative and as such are applicable to modern day applications

Types of evolutionary models

– Prototyping

– Spiral model

– Concurrent development model(Given in running notes)

6.Prototype Models:

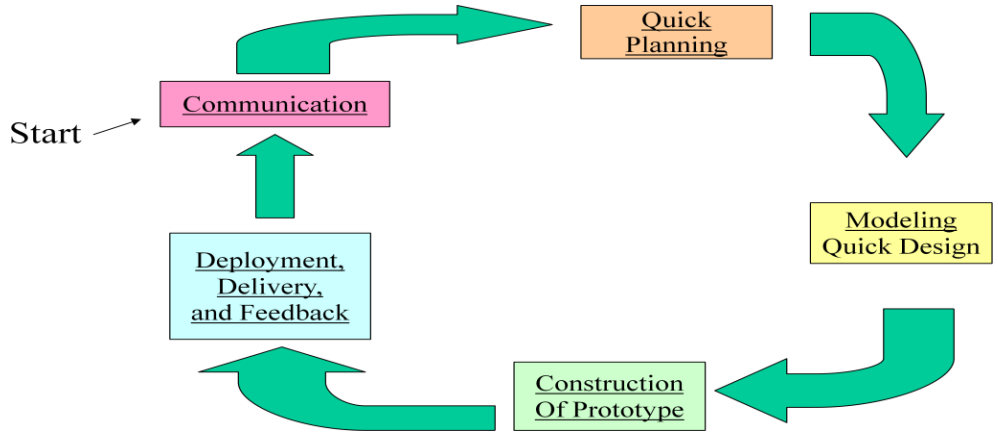
A prototype is a toy implementation of the system. A prototype usually exhibits limited functional capabilities, low reliability, and inefficient performance compared to the actual software. A prototype is usually built using several shortcuts. The shortcuts might involve using inefficient, inaccurate, or dummy functions. The shortcut implementation of a function, for example, may produce the desired results by using a table look-up instead of performing the actual computations.

Need for a prototype in software development

There are several uses of a prototype. An important purpose is to illustrate the input data formats, messages, reports, and the interactive dialogues to the customer. This is a valuable mechanism for gaining better understanding of the customer's needs:

- how the screens might look like
- how the user interface would behave
- how the system would produce outputs

Another reason for developing a prototype is that it is impossible to get the perfect product in the first attempt. Many researchers and engineers advocate that if you want to develop a good product you must plan to throw away the first version. The experience gained in developing the prototype can be used to develop the final product.



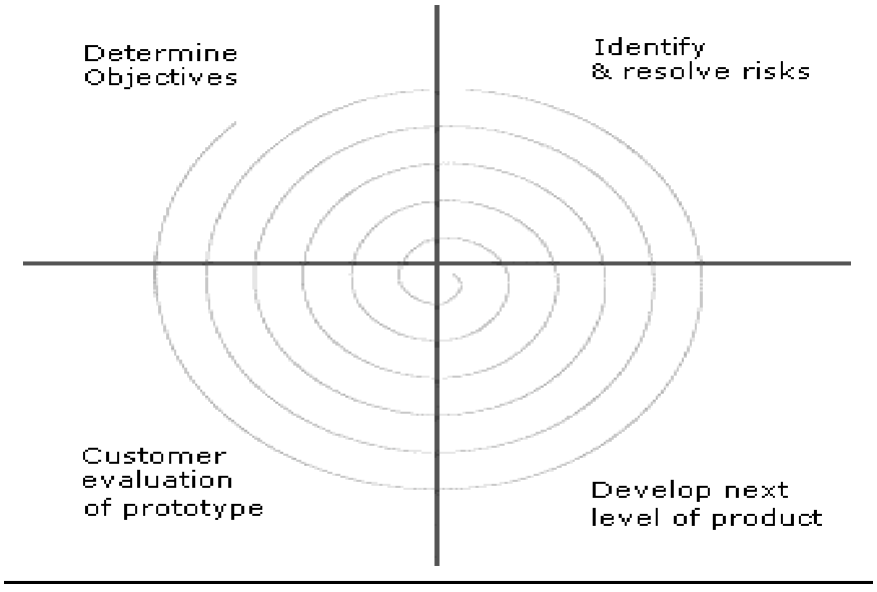
- Follows an evolutionary and iterative approach
- Used when requirements are not well understood
- Serves as a mechanism for identifying software requirements
- Focuses on those aspects of the software that are visible to the customer/user
- In this model, product development starts with an initial requirements gathering phase.
- A quick design is carried out and the prototype is built.
- The developed prototype is submitted to the customer for his evaluation.
- Based on the customer feedback, the requirements are refined and the prototype is suitably modified.
- This cycle of obtaining customer feedback and modifying the prototype continues till the customer approves the prototype.
- The actual system is developed using the iterative waterfall approach. However, in the prototyping model of development, the requirements analysis and specification phase becomes redundant as the working prototype approved by the customer becomes redundant as the working prototype approved by the customer becomes an animated requirements specification.

Disadvantages

- The customer sees a "working version" of the software, wants to stop all development and then buy the prototype after a "few fixes" are made
- Developers often make implementation compromises to get the software running quickly (e.g., language choice, user interface, operating system choice, inefficient algorithms)
- Lesson learned
    - Define the rules up front on the final disposition of the prototype before it is built
    - In most circumstances, plan to discard the prototype and engineer the actual production software with a goal toward quality
    -

7.Spiral Model

- Invented by Dr. Barry Boehm in 1988
- Follows an evolutionary approach
- Used when requirements are not well understood and risks are high
- Inner spirals focus on identifying software requirements and project risks; may also incorporate prototyping
- Outer spirals take on a classical waterfall approach after requirements have been defined, but permit iterative growth of the software
- Operates as a risk-driven model…a go/no-go decision occurs after each complete spiral in order to react to risk determinations
- Requires considerable expertise in risk assessment
- Serves as a realistic model for large-scale software development



First quadrant (Objective Setting)

- During the first quadrant, it is needed to identify the objectives of the phase.
- Examine the risks associated with these objectives.

Second Quadrant (Risk Assessment and Reduction)

- A detailed analysis is carried out for each identified project risk.
- Steps are taken to reduce the risks. For example, if there is a risk that the requirements are inappropriate, a prototype system may be developed.

Third Quadrant (Development and Validation)

- Develop and validate the next level of the product after resolving the identified risks.

Fourth Quadrant (Review and Planning)

- Review the results achieved so far with the customer and plan the next iteration around the spiral.
- Progressively more complete version of the software gets built with each iteration around the spiral.

Spiral Model Advantages

- Focuses attention on reuse options.
- It is a realistic approach to the development of large scale systems and software.
- Focuses attention on early error elimination.
- Puts quality objectives up front.
- Integrates development and maintenance.
- Provides a framework for hardware/software development.

Disadvantages:

- Contractual development often specifies process model
- and deliverables in advance.
- Requires risk assessment expertise.

Circumstances to use spiral model

The spiral model is called a meta model since it encompasses all other life cycle models. Risk handling is inherently built into this model. The spiral model is suitable for development of technically challenging software products that are prone to several kinds of risks. However, this model is much more complex than the other models — this is probably a factor deterring its use in ordinary projects.

Comparison of different life-cycle models

The classical waterfall model can be considered as the basic model and all other life cycle models as embellishments of this model. However, the classical waterfall model can not be used in practical development projects, since this model supports no mechanism to handle the errors committed during any of the phases.

This problem is overcome in the iterative waterfall model. The iterative waterfall model is probably the most widely used software development model evolved so far. This model is simple to understand and use. However, this model is suitable only for well- understood problems; it is not suitable for very large projects and for projects that are subject to many risks.

The prototyping model is suitable for projects for which either the user requirements or the underlying technical aspects are not well understood. This model is especially popular for development of the user-interface part of the projects.

The Incremental approach is suitable for large problems which can be decomposed into a set of modules for incremental development and delivery. This model is also widely used for object-oriented development projects. Of course, this model can only be used if the incremental delivery of the system is acceptable to the customer.

The spiral model is called a meta model since it encompasses all other life cycle models. Risk handling is inherently built into this model. The spiral model is suitable for development of technically challenging software products that are prone to several kinds of risks. However, this model is much more complex than the other models — this is probably a factor deterring its use in ordinary projects.

The different software life cycle models can be compared from the viewpoint of the customer. Initially, customer confidence in the development team is usually high irrespective of the development model followed. During the lengthy development process, customer confidence normally drops off, as no working product is immediately visible.

Specialized Process Models

8. Component-based Development Model

- Consists of the following process steps
  - Available component-based products are researched and evaluated for the application domain in question
  - Component integration issues are considered
  - A software architecture is designed to accommodate the components
  - Components are integrated into the architecture
  - Comprehensive testing is conducted to ensure proper functionality

- Relies on a robust component library
- Capitalizes on software reuse, which leads to documented savings in project cost and time

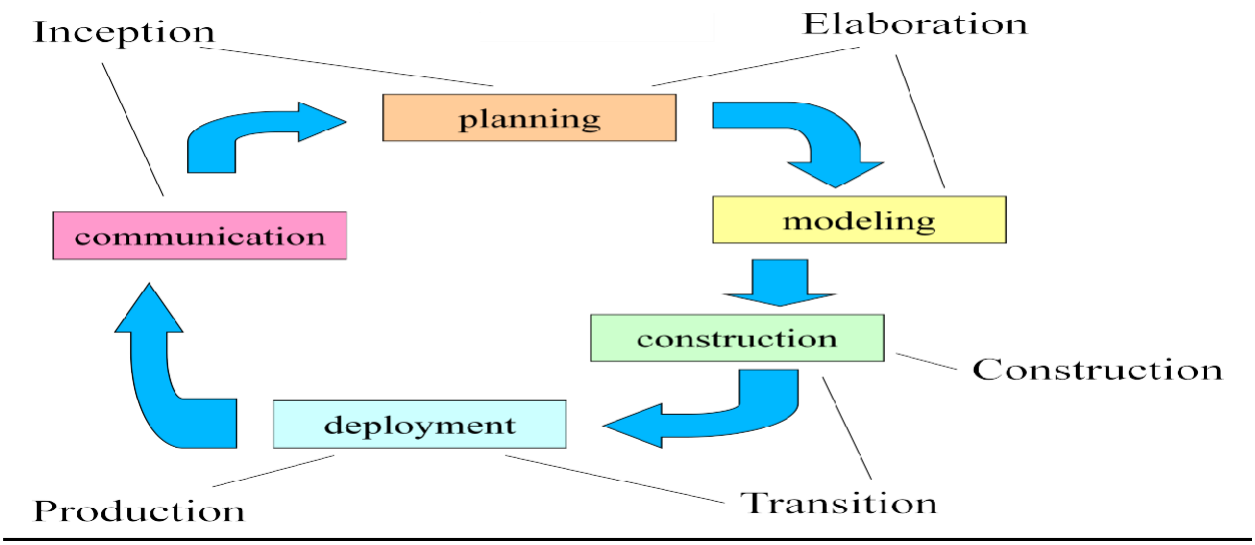
Formal Methods Model

- Encompasses a set of activities that leads to formal mathematical specification of computer software
- Enables a software engineer to specify, develop, and verify a computer-based system by applying a rigorous, mathematical notation
- Ambiguity, incompleteness, and inconsistency can be discovered and corrected more easily through mathematical analysis
- Offers the promise of defect-free software
- Used often when building safety-critical systems

Challenges of Formal Methods

- Development of formal methods is currently quite time-consuming and expensive
- Because few software developers have the necessary background to apply formal methods, extensive training is required
- It is difficult to use the models as a communication mechanism for technically unsophisticated customers

11. The Unified Process



Birthed during the late 1980's and early 1990s when object-oriented languages were gaining wide-spread use
Many object-oriented analysis and design methods were proposed; three top authors were Grady Booch, Ivar Jacobson, and James Rumbaugh

They eventually worked together on a unified method, called the Unified Modelling .
UML is a robust notation for the modelling and development of object-oriented system.

UML became an industry standard in 1997

  - However, UML does not provide the process framework, only the necessary technology for object-oriented development

- Booch, Jacobson, and Rumbaugh later developed the unified process, which is a framework for object-oriented software engineering using UML
  - Draws on the best features and characteristics of conventional software process models
  - Emphasizes the important role of software architecture
  - Consists of a process flow that is iterative and incremental, thereby providing an evolutionary feel
- Consists of five phases: inception, elaboration, construction, transition, and production

A)Inception Phase

- Encompasses both customer communication and planning activities of the generic process
- Business requirements for the software are identified
- A rough architecture for the system is proposed
- A plan is created for an incremental, iterative development
- Fundamental business requirements are described through preliminary use cases
  - A use case describes a sequence of actions that are performed by a user

B)Elaboration Phase

- Encompasses both the planning and modelling activities of the generic process
- Refines and expands the preliminary use cases
- Expands the architectural representation to include five views
  - Use-case model
  - Analysis model
  - Design model
  - Implementation model
  - Deployment model
- Often results in an executable architectural baseline that represents a first cut executable system
- The baseline demonstrates the viability of the architecture but does not provide all features and functions required to use the system

C)Construction Phase

- Encompasses the construction activity of the generic process
- Uses the architectural model from the elaboration phase as input
- Develops or acquires the software components that make each use-case operational

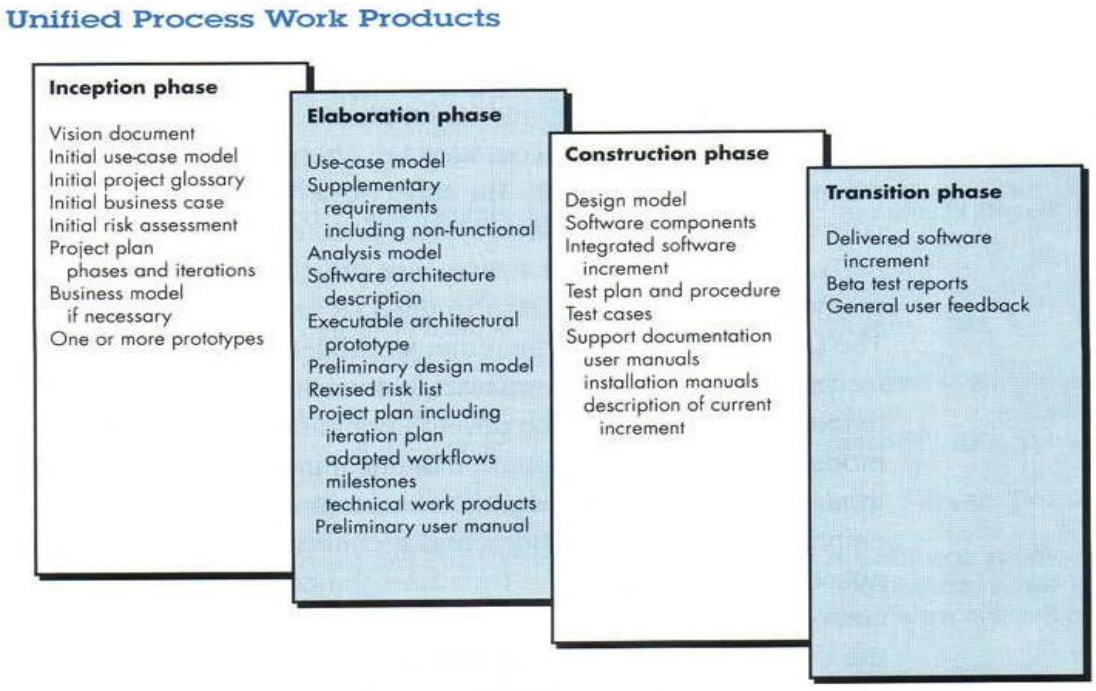
- Analysis and design models from the previous phase are completed to reflect the final version of the increment
- Use cases are used to derive a set of acceptance tests that are executed prior to the next phase

D)Transition Phase

- Encompasses the last part of the construction activity and the first part of the deployment activity of the generic process
- Software is given to end users for beta testing and user feedback reports on defects and necessary changes
- The software teams create necessary support documentation (user manuals, trouble-shooting guides, installation procedures)
- At the conclusion of this phase, the software increment becomes a usable software release

E)Production Phase

- Encompasses the last part of the deployment activity of the generic process
- On-going use of the software is monitored
- Support for the operating environment (infrastructure) is provided
- Defect reports and requests for changes are submitted and evaluated



12.Agility and Agile Process model

The meaning of Agile is swift or versatile. “Agile process model" refers to a software development approach based on iterative development. Agile methods break tasks into smaller iterations, or parts

do not directly involve long term planning. The project scope and requirements are laid down at the beginning of the development process. Plans regarding the number of iterations, the duration and the scope of each iteration are clearly defined in advance

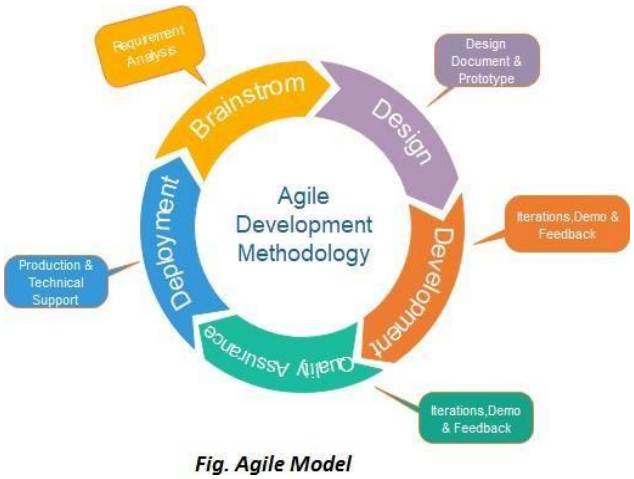


Fig. Agile Model

Phases of Agile model:

1. Requirements gathering
2. Design the requirements
3. Construction/ iteration
4. Testing/ Quality assurance
5. Deployment
6. Feedback

1.Requirements gathering: In this phase, you must define the requirements. You should explain business opportunities and plan the time and effort needed to build the project. Based on this information, you can evaluate technical and economic feasibility.

2.Design the requirements: When you have identified the project, work with stakeholders to define requirements. You can use the user flow diagram or the high-level UML diagram to show the work of new features and show how it will apply to your existing system.

3.Construction/ iteration: When the team defines the requirements, the work begins. Designers and developers start working on their project, which aims to deploy a working product. The product will undergo various stages of improvement, so it includes simple, minimal functionality.

4.Testing: In this phase, the Quality Assurance team examines the product's performance and looks for the bug.

5.Deployment: In this phase, the team issues a product for the user's work environment.

6.Feedback: After releasing the product, the last step is feedback. In this, the team receives feedback about the product and works through the feedback.

Advantages:

1. Frequent Delivery
2. Face-to-Face Communication with clients.
3. Efficient design and fulfils the business requirement.
4. Anytime changes are acceptable.
5. It reduces total development time.

Disadvantages:

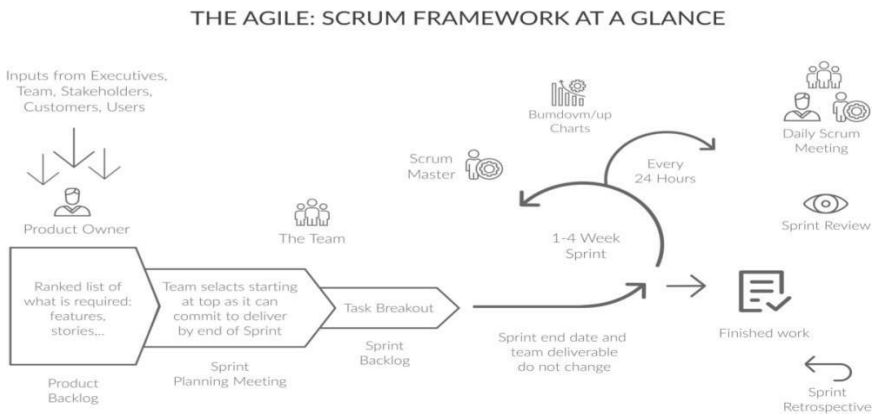
1.Due to the shortage of formal documents, it creates confusion and crucial decisions taken throughout various phases can be misinterpreted at any time by different team members.

2.Due to the lack of proper documentation, once the project completes and the developers allotted to another project, maintenance of the finished project can become a difficulty.

13.Scrum

Scrum is aimed at sustaining strong collaboration between people working on complex products, and details are being changed or added. It is based upon the systematic interactions between the three major roles: Scrum Master, Product Owner, and the Team.

Scrum is one of the most widely adopted Agile frameworks in the software development industry. It provides a structured and comprehensive approach to managing projects, enabling teams to deliver high-quality software efficiently.



Overview of Scrum Framework

The Scrum framework is built on the foundation of Agile principles and is designed to maximize productivity, foster collaboration, and deliver value to customers.

It consists of three essential elements:

1.Scrum Roles

Product Owner: The Product Owner is the voice of the customer and stakeholders. They are responsible for defining and prioritizing the product backlog, ensuring that the development team is working on the most valuable features.

The Product Owner collaborates with stakeholders to gather requirements and provide feedback on delivered increments.

Scrum Master: The Scrum Master acts as a facilitator and servant-leader for the development team. Their primary role is to ensure that the Scrum framework is understood and followed correctly.

They remove any impediments that hinder the team's progress, promote a collaborative team environment, and facilitate the various Scrum ceremonies.

Development Team: The Development Team consists of professionals who do the actual work of delivering a potentially shippable product increment in each sprint. They are self-organizing, cross-functional, and collaborate closely to complete the tasks from the sprint backlog.

2.Scrum Artifacts

Product Backlog: The Product Backlog is a prioritized list of all the work items required to complete the project. These items can include features, enhancements, bug fixes, and technical tasks.

The Product Owner continuously refines and updates the backlog based on feedback and changing requirements.

Sprint Backlog: Before each sprint, the Development Team pulls a set of work items from the Product Backlog

and creates the Sprint Backlog.

The Sprint Backlog contains the tasks the team commits to completing during the sprint. It provides transparency and a clear plan for the upcoming iteration.

Increment: The Increment represents the sum of all completed Product Backlog items at the end of each sprint. It is a potentially shippable piece of software that should be in a usable state and adhere to the team's definition of "done."

3.Scrum Ceremonies

Sprint Planning: At the beginning of each sprint, the Product Owner and Development Team collaborate in the Sprint Planning meeting. They discuss and agree on the sprint goal, select the top items from the Product Backlog, and create the Sprint Backlog with associated tasks.

Daily Standup (Daily Scrum): The Daily Standup is a brief daily meeting where the Development Team synchronizes their work. Each team member shares what they worked on the previous day, what they plan to work on that day, and any impediments they are facing.

Sprint Review: At the end of each sprint, the team holds a Sprint Review meeting to demonstrate the completed Increment to stakeholders. Feedback is gathered, and the Product Backlog is updated based on the stakeholders' input.

Sprint Retrospective: Following the Sprint Review, the team conducts the Sprint Retrospective to reflect on the previous sprint. They identify what went well, what could be improved, and define actionable items to enhance their processes in the upcoming sprints.

Benefits and Advantages of Scrum

Scrum offers a number of benefits that contribute to its popularity and success in Agile software development:

1. Transparency: The use of visible backlogs, frequent progress updates, and regular meetings ensures transparency among team members and stakeholders. This fosters a shared understanding of the project's status.

2. Adaptability: Scrum's iterative nature allows teams to adapt to changing requirements and priorities. This ensures that the delivered product remains aligned with the customer's needs.

3. Continuous Improvement: The Sprint Retrospective encourages continuous improvement by providing a platform for the team to reflect on their practices and identify opportunities for enhancement.

4. Early Value Delivery: The focus on delivering potentially shippable increments at the end of each sprint allows customers to see tangible progress early in the development process.

5. Customer Collaboration: The involvement of the Product Owner and regular Sprint Reviews promote active collaboration with customers, resulting in a product that better meets their expectations.

Scrum Challenges and How to Overcome Them

While Scrum is highly effective, it is not without its challenges. Some common hurdles that teams may encounter include:

1. Overcommitment: Teams might take on too much work in a sprint, leading to incomplete tasks and a compromised Increment. Regularly evaluating capacity and being realistic about commitments can help avoid this pitfall.

2. Lack of Empowerment: If team members are not empowered to make decisions and are overly dependent on the Scrum Master, the efficiency and effectiveness of the team may suffer. Encouraging self-organization and trust within the team can mitigate this challenge.

3. Incomplete Definition of "Done": Ambiguity about what constitutes a "done" user story can lead to misunderstandings and incomplete work. Clearly defining and agreeing upon the team's "definition of done" is crucial for consistent delivery.

4. Product Owner Availability: Insufficient availability of the Product Owner can slow down decision making and result in unclear requirements. Maintaining constant communication and involvement with the team can help alleviate this issue.

Scrum provides a structured and comprehensive approach to Agile Software Development, offering a well-defined set of roles, artifacts, and ceremonies that facilitate collaboration, transparency, and continuous improvement.

By embracing Scrum's core principles and addressing its challenges proactively, development teams can harness the full potential of this framework to deliver successful software projects.

14.Extreme Programming (XP):

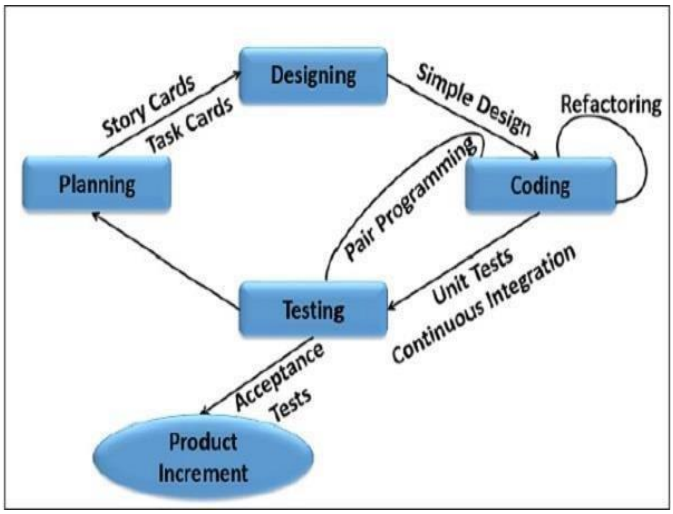
Extreme Programming

XP is a lightweight, efficient, low-risk, flexible, predictable, scientific, and fun way to develop software.

Extreme Programming (XP) was conceived and developed to address the specific needs of software development by small teams in the face of vague and changing requirements.

Extreme Programming is one of the Agile software development methodologies. It provides values and principles to guide the team behavior. The team is expected to self-organize. Extreme Programming provides specific core practices where −

- Each practice is simple and self-complete.
- Combination of practices produces more complex and emergent behaviour.



Extreme Programming (XP) is an Agile software development approach that embraces a set of best practices and values to deliver high-quality software efficiently.

Created by Kent Beck in the late 1990s, XP challenges traditional development practices by promoting a customer-centric and team-oriented philosophy.

In this section, we will explore the key principles and core practices of Extreme Programming and understand its impact on software development teams.

Overview of Extreme Programming

Extreme Programming is based on a set of values that drive the development process. These values include communication, simplicity, feedback, courage, and respect.

XP encourages open and frequent communication between team members and stakeholders. This simplifies processes and solutions and encourages team members to seek and act upon feedback regularly. Team members are also encouraged to have the courage to make necessary changes and respect the expertise and contributions of all team members.

Core Practices of XP

Test-Driven Development (TDD): Test-Driven Development is a fundamental practice in XP where developers write tests before writing the code.

The process involves creating a test that initially fails, then writing the code to pass the test. TDD ensures that the code is thoroughly tested. This makes it easier to identify and fix issues early in the development process.

Pair Programming: In Pair Programming, two developers work collaboratively at the same workstation. One programmer writes the code while the other reviews it in real-time. This promotes continuous feedback, knowledge sharing, and improved code quality.

This practice enhances team communication and leads to the development of more robust solutions.

Continuous Integration: Continuous Integration involves frequently integrating code changes into a shared repository. This ensures that the software is continuously built and tested as new code is added, reducing integration issues and enabling faster feedback on potential defects.

Collective Code Ownership: XP encourages the concept of collective code ownership, where all team members take responsibility for the entire codebase.

This fosters a sense of ownership and accountability within the team, leading to a collaborative and supportive work environment.

On-Site Customer: In XP, having an on-site customer or a dedicated customer representative is vital for effective communication and quick decision-making.

The on-site customer provides real-time feedback and clarifications, ensuring that the team builds the right features and meets customer expectations.

Pros and Cons of Extreme Programming

Pros:

 High-Quality Code: TDD and Pair Programming lead to better-tested and more maintainable code.

 Fast Feedback: Continuous Integration and frequent releases provide rapid feedback on code changes.

 Customer Collaboration: Involving an on-site customer ensures better alignment with customer needs.

 Adaptability: XP's practices allow teams to adapt to changing requirements and priorities effectively.

Cons:

 Learning Curve: Adopting XP may require a cultural shift and training for team members unfamiliar with its practices.

 Resource Intensive: Pair Programming and on-site customer involvement may require additional resources.

 Initial Overhead: Writing tests before code and maintaining continuous integration can add initial overhead.

Extreme Programming (XP) is a development approach grounded in a customer-focused philosophy and driven by a set of core practices.

By emphasizing test-driven development, pair programming, continuous integration, and collective code ownership, XP aims to deliver high-quality software while promoting effective teamwork and continuous improvement.

Like any methodology, XP has its advantages and challenges, but when applied in the right context with committed team members, it can lead to substantial improvements in software development efficiency and customer satisfaction.

15. Project Initiation Management

What is Project Initiation?

Project initiation is like the starting point of a project journey. It's when you decide what you want to do, why you want to do it, and who will be involved. It's all about getting everyone on the same page before you dive into the details of how to make it happen. The initial stage in beginning a new project is called project initiation. Establishing the project's purpose and the commercial value it will provide helps you gain support from important stakeholders

The Significance of the Initiation Phase

The beginning stage is when a project starts and it's the start of managing all the things related to projects.

In this stage, the project's chances of success are checked.

Also, people who have stakes or interests in the venture are found out and the first plans for it start to be made.

The choices you make when starting a project can affect how it goes forward.

They help decide whether the project will be successful or not in the future time.

Benefits of a Well-Executed Initiation Phase

Clear Project Direction: During the beginning stage, it's important to set clear goals for a project. This helps guide the team and keeps them focused in one direction.

Risk Mitigation: By checking things carefully, we can find the possible dangers and problems right at the start of a project.

Resource Optimization: Starting things right means better use of stuff. By knowing the project aims, managers can figure out what talents and tools are needed.

Stakeholder Alignment: The starting stage helps to find and involve top people. This is an important part for all involved in something. Getting everyone on the same page and knowing what is expected during this part of a project helps people support it. This happens if you talk clearly with them about these needs from start to finish in their projects.

Uses of the Initiation Phase

1. Feasibility Analysis

The start part needs a good look at if the project can work. You check stuff like money, computer tech and day-to-day operations Issues that make it happen or not. This study helps to see if the project works and follows what the company is trying to achieve in their big plans.

2. Project Charter Development

From the start phase, a major result is the project charter. This paper tells what the project is about, its goals and who's involved. It also shows the first needs for resources needed to start it up nicely like a guidebook during all the actions of ideal efforts.

3. Goal Definition

Setting up goals for a project is a key job in the starting phase. This means making clear the aim of a project, and setting specific goals that can be measured and accomplished. They should line up with when they need to get done. This will lead on through what you make doing inside it runs its life-cycle work within teams too.

4.Initial Planning

In the beginning, you make an early project plan. Later on, more precise planning is done to go forward with it fully and properly. However, at first, this general idea for a new thing is thought out quietly alone before anyone else does any work or decides whether something can happen easily without problems happening in the different steps each time: what will be accomplished? who'll do the tasks? when they.

16.Project Charter

The Project Charter describes the project vision and objectives (the reasons for doing the work1). It also summarises at a high level the overall project strategy, scope, organisation and implementation. It helps to set the direction for the project and gain buy in from key stakeholders as to how the project will be organised and implemented. It also helps control the scope of the project, by defining exactly what it is that you have to achieve. The Charter is provided to the project manager by the project sponsor2 and helps set the direction for the project

and gain support from key stakeholders as to how the project will be managed. If the Charter is missing or inadequate the project manager should develop or augment the document and have the document agreed and signed by the sponsor or project initiator before starting any other planning process.

The key purpose of a Charter is to define exactly what it is the project has to achieve to be successful

- Identifying the Project Vision: The vision encapsulates the purpose of the project and is business/benefits focused. This is one short, concise paragraph.
- Identify the strategic purpose and alignment4 of the project.
- Identifying the Project Objectives5: Define three to five specific objectives that need to be achieved by the project to fulfil the vision. Each objective should be Specific, Measurable, Achievable, Realistic and Time-bound (SMART) and are output focused6 (major deliverables).
- Defining the overall Project Scope: The scope defines the formal boundaries of the project by describing what will be done by the project and what will not be done by the project. What will be done should be described as specific deliverables.

How this works: Your project is for a new business system;

- Its vision is to improve efficiency.
- o One objective is to provide training. The specific deliverables associated with this objective could include:
  - Develop the courseware
  - Deliver two initial courses
  - Deliver an update of the training materials
  - Run a 'train the trainer' workshop, and
  - Provide on-going support to the business trainer for 6 months.

What's 'not done' by the project is the ongoing training of business staff for the next 3 or 4 years. A high level scope statement would be ‘provide initial training and support for 6 months’ – the detail of the deliverables should be in the scope statement.

The charter should also outline at a high level how the project’s work will be accomplished. This includes:

•Describing the Project Organisation: How the project will be structured including customers, stakeholders, key management roles, responsibilities and reporting lines.

- A customer is a person or entity that is responsible for accepting the deliverables when the project is complete.
- Stakeholders are people or groups who will be impacted by, or can influence the success or failure of the project8.
- Additional roles and responsibilities include the Project Sponsor, Project Board, Project Manager and other key project positions. Each with a summary of their primary

responsibilities.

- Reporting lines: From the roles and responsibilities, the reporting lines between those roles can be defined in a Project Organization Chart.

•Describing the Project Strategy: This describes how the project will be executed and forms the framework for detailed project planning. Some aspects to consider include:

- The methodology to be used PRINCE2 / Agile / Waterfall This sets the overallframework.
- Key Milestones to be achieved by the project. A milestone is typically an important project event, such as the achievement of a key deliverable.
- Key phases and/or activities, and the overall time frames involved in undertaking the project.
- Any environmental implications10 such as opportunities for recycling or waste reduction that should be

considered during the planning and implementation of the project.

- Key external dependencies and their criticality to the project. An external dependency is an activity or event that is likely to impact on the project during its life cycle.
- A summary resource plan (based on the methodology and time frame) including the overall requirements for labour, equipment, materials and financial resources.

Laying out the document:

The overall Charter should be no more than two pages long focused on the clarifying why the project exists, not the detail of how it will be accomplished (this detail will be in the project plan)14. Typically there needs to be six sections:

Section #1: The Executive Summary: This section is written last and should summarise the other sections in 1 to 2 short paragraphs. If the CEO only reads this section he/she will understand at a very high level the definition and purpose of the project, its organization and plan, risks and issues, and any major assumptions or constraints that may have an affect on the project.

Section #2: Project Purpose and Definition: This section defines at a high level what the project involves and why the organization is investing money in accomplishing the work. There should be 4 or 5 short sections:

•Project Vision - What is the project is setting out to accomplish in support of the organisation's strategy?

•Project Objectives - What are the major business objectives that will be met in order to implement the vision?

•Project Scope - What new work, technology, processes, applications, and/or locations are involved or affected by the project?

•Project Deliverables - What are the major elements that will be delivered to meet the scope, objectives, and vision of this project?

•Project phasing, staging or gateway reviews15 - (optional) if there are requirements for reviews or formal authorisations to allow the work to continue, the charter needs to be specific about the extent of the work authorised by this current document, and the processes for extending the authorisation once the defined criteria have been achieved (see also 'change control below).

Section #3: Project Stakeholders16 and Organization: A brief summary of who is involved in this project and what are their roles and responsibilities, including:

Various customer or user groups that will be using the finished result of the project; and ideally a representative for each group that can serve as a spokesperson on behalf of that group.

•Other stakeholders that are interested in the success or failure of the project. Include a brief note as to specifically what interests them about the project.

•The key roles required to undertake the project, including the Sponsor, Project Manager, and key team members. Name the individuals where possible.

Section #4: The Project Plan: Provide a general overview of how the project will be implemented; including the intended method of working (project strategy), key phases or milestones (especially if imposed by the customer) and any other relevant information. Remember this is not the detailed plan developed by the project team, but it is important to describe how and when the project vision and objectives will be achieved. The plan should summarise:

- Project strategy
- The approved budget
- The level of resources required
- The overall timeframe and key milestones
- Phase or stage boundaries (if required)

- The effect of any dependencies and inputs
- Acceptance criteria and processes

Section #5: Risks and Considerations: Identify and include:

Any known Risks (both negative and positive) that could impact the project along with their likelihood of occurring, impact on the project, and what treatments are being considered.

- Any identified opportunities to implement 'green project management' principles to minimise the environmental impact of the work.

Any Issues that are currently open.

•All Assumptions made in the development of the business case and/or the charter

•All identified Constraints and dependencies.

Note: if this section is going to be too long, put the major items in the Project Charter and include an attachment with all of the rest.

Section #6: Change management: The project charter should be a controlled document with any alterations subjected to formal change control processes once the document is approved.

Section #7: References: Key documents used to develop the project charter including the business case, customer contracts, and any other information that should be carried forward into the project planning phase once the Project Charter has been approved.

Key contents checklist:

- Project purpose or justification
- Measurable project objectives and related success criteria
- High-level requirements (including customer needs)
- Assumptions and constraints

•High-level project description and boundaries

•High-level risks

•Summary milestone schedule

•Summary budget

•Stakeholder list

•Project approval requirements (ie, what constitutes project success, who decides the project is successful, and who signs off on the project)

•Change management and document control provisions

•Assigned project manager, responsibility, and authority level, and

Maintaining the Charter:

At the start of each phase of the project, one of the important actions is to review the project charter and ensure the project as it is now defined and understood is still achieving the objectives defined in the charter. If there are discrepancies, the changes need to be processed through the overall change control mechanisms to either update the charter or to re-focus the project on achieving its authorised objectives.

For major projects, there may be valid requirements for a phase-charter defining the elements of the overall project objectives that this particular phase is being initiated to accomplish. Phase charters can be a simplified version of the overall project charter.

17. PROJECT OBJECTIVE

To create objectives that are clearly written and helpful. You can do this by using **the SMART** methodology,

which stands for:

Specific

Measurable

Achievable

Realistic

Time-bound

1. Set your project objectives at the beginning of your project
In order for your objectives to guide the results of the project, you need to set them at the beginning and use them to guide your project. As we mentioned earlier, your project objectives are a key element of your project plan, which you should also create at the beginning of your project.

2. Involve your project team in the goal-setting process
The more buy-in you get, the more successful your project objectives will be. Your stakeholders need to have a clear understanding of the objectives of the project, so they can approach the rest of your project plan and the work that happens during the project most effectively.

3. Create brief, but clear, project objective statements
If this is your first time writing a project objective, you may be tempted to outline every detail—but try to keep your project objective short if you can. Think of it as a statement to guide the results of your project—your project objective statement should be about one to two sentences long. The additional information, like your project budget or stakeholders, will be captured in your project plan.

4. Make sure your objectives are things you can control
This is where the SMART acronym comes in to play to help you create clearly-defined, realistic, and controllable project objectives. There are five elements to this framework:

Specific. Make sure your project objective statement clearly covers the project your team is currently working on. Avoid writing overly broad project objectives that don't directly connect to the result of the project.

Measurable. At the end of your project, you need a way to clearly look back and determine if your project was a success. Make sure your project objectives are clearly measurable things—like percentage change or a specific number of assets.

Achievable. Are your project objectives something you can reasonably hope to achieve within your project? this is connected to your project scope—if your project scope is unrealistic, your project objectives likely will be, too. Without Achievable project goals, your project may suffer from scope creep, delays, or overwork.

Realistic. When you're creating your project objectives, you should have a general sense of your project resources. Make sure your objectives are something you can achieve within the time frame and with the resources you have available for this project.

Time-bound. Your project objectives should take into account how long your project timeline is. Make sure you factor in the time you have available to work on your project.

5. Check in on your project objectives during the project's lifecycle

Employees who understand how their individual work adds value to their organization are 2X as motivated. In order to keep your team aligned and motivated, make sure to check in and update them on your project objectives frequently. In your project status reports, include a section that connects back to your project objectives. Share whether your current project is on track, at risk, or off track. That way, your project team can recalibrate if necessary and move forward in a way that best contributes to your project objectives.

18.Project Scope

1. Definition:

Project scope refers to the boundaries, deliverables, objectives, and constraints of a project. It defines what is included and what is excluded to ensure clarity among stakeholders and prevent scope creep.

2. Importance of Project Scope:

- Ensures clear understanding of project goals.
- Defines deliverables and milestones.
- Helps manage resources, budget, and time.
- Prevents uncontrolled changes (scope creep).

3. Key Components of Project Scope:

- Project Objectives – The goals and intended outcomes.
- Deliverables – The tangible or intangible results expected.
- Stakeholders – People involved in the project.
- Inclusions & Exclusions – Defines what is covered and what is not.
- Constraints – Budget, time, and resource limitations.
- Assumptions – Factors considered to be true for planning.

4. Project Scope in Project Initiation:

- Scope is defined during the project initiation phase.
- Helps create a Project Scope Statement (PSS).
- Used to develop Work Breakdown Structure (WBS).
- Forms the basis for Project Planning and Execution.

Scope Planning – Identifying key elements.
Scope Definition – Creating a detailed scope statement.
Scope Validation – Ensuring stakeholder agreement.
Scope Control – Managing changes and preventing scope creep.