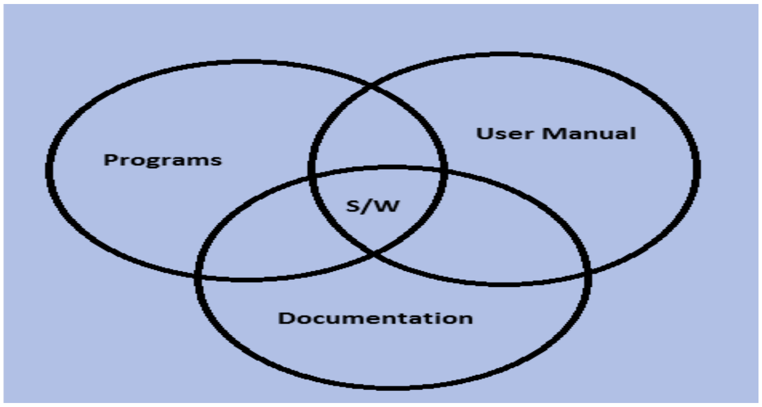
Module-1: The Software Process

Software is a set of instructions, data or programs used to operate computers and execute specific tasks. It is the opposite of hardware, which describes the physical aspects of a computer. Software is a generic term used to refer to applications, scripts and programs that run on a device. It can be thought of as the variable part of a computer, while hardware is the invariable part.

Software:

“Software is a set of inter-related programs used to operate computers and execute specific well defined tasks”

Software = Programs + User Manual + Documentation



The two main categories of software are application software and system software. An application is software that fulfills a specific need or performs tasks. System software is designed to run a computer's hardware and provides a platform for applications to run on top of.

Other types of software include programming software, which provides the programming tools software developers need; middleware, which sits between system software and applications; and driver software, which operates computer devices and peripherals.

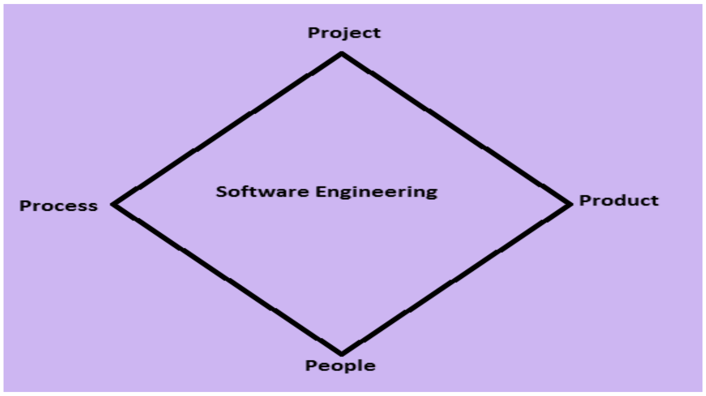
Examples and types of software

Among the various categories of software, the most common types include the following:

- **Application software.** The most common type of software, application software is a computer software package that performs a specific function for a user, or in some cases, for another application. An application can be self-contained, or it can be a group of programs that run the application for the user. Examples of modern applications include office suites, graphics software, databases and database management programs, web browsers, word processors, software development tools, image editors and communication platforms.
- **System software.** These software programs are designed to run a computer's application programs and hardware. System software coordinates the activities and functions of the hardware and software. In addition, it controls the operations of the computer hardware and provides an environment or platform for all the other types of software to work in. The OS is the best example of system software; it manages all the other computer programs. Other examples of system software include the firmware, computer language translators and system utilities.
- **Driver software.** Also known as device drivers, this software is often considered a type of system software. Device drivers control the devices and peripherals connected to a computer, enabling them to perform their specific tasks. Every device that is connected to a computer needs at least one device driver to function. Examples include software that comes with any nonstandard hardware, including special game controllers, as well as the software that enables standard hardware, such as USB storage devices, keyboards, headphones and printers.
- **Middleware.** The term *middleware* describes software that mediates between application and system software or between two different kinds of application software. For example, middleware enables Microsoft Windows to talk to Excel and Word. It is also used to send a remote work request from an application in a computer that has one kind of OS, to an application in a computer with a different OS. It also enables newer applications to work with legacy ones.
- **Programming software.** Computer programmers use programming software to write code. Programming software and programming tools enable developers to develop, write, test and debug other software programs. Examples of programming software include assemblers, compilers, debuggers and interpreters.

Software Engineering:

> *"Software engineering is a computer technology defined as a process of analyzing user requirements and then designing, building, and testing software application which will satisfy those requirements"*



Software Engineering Objectives:

1. To understand the software life cycle models.
2. To understand the software requirements and SRS document.
3. To understand the importance of modeling and modeling languages.
4. To design and develop correct and robust software products.
5. To understand the quality control and how to ensure good quality software.
6. To understand the planning and estimation of software projects.
7. To understand the implementation issues, validation and verification procedures.
8. To understand the maintenance of software

Software Engineering Outcomes:

1. Define and develop a software project from requirement gathering to implementation.
2. Ability to code and test the software
3. Ability to plan, Estimate and Maintain software systems

The Nature of Software:

The software is instruction or computer program that when executed provide desired features, function, and performance. A data structure that enables the program to adequately manipulate information and document that describes the operation and use of the program.
Characteristic of software: There is some characteristic of software which is given below:

1. Functionality
2. Reliability
3. Usability
4. Efficiency

5. Maintainability
6. Portability

Changing Nature of Software:
Nowadays, seven broad categories of computer software present continuing challenges for software engineers .which is given below:

1. **System Software:**

   System software is a collection of programs which are written to service other programs. Some system software processes complex but determinate, information structures. Other system application process largely indeterminate data. Sometimes when, the system software area is characterized by the heavy interaction with computer hardware that requires scheduling, resource sharing, and sophisticated process management.
2. **Application Software:**

   Application software is defined as programs that solve a specific business need. Application in this area process business or technical data in a way that facilitates business operation or management technical decision making. In addition to convention data processing application, application software is used to control business function in real time.
3. **Engineering and Scientific Software:**

   This software is used to facilitate the engineering function and task. however modern application within the engineering and scientific area are moving away from the conventional numerical algorithms. Computer-aided design, system simulation, and other interactive applications have begun to take a real-time and even system software characteristic.
4. **Embedded Software:**

   Embedded software resides within the system or product and is used to implement and control feature and function for the end-user and for the system itself. Embedded software can perform the limited and esoteric function or provided significant function and control capability.
5. **Product-line Software:**

   Designed to provide a specific capability for use by many different customers, product line software can focus on the limited and esoteric marketplace or address the mass consumer market.
6. **Web Application:**

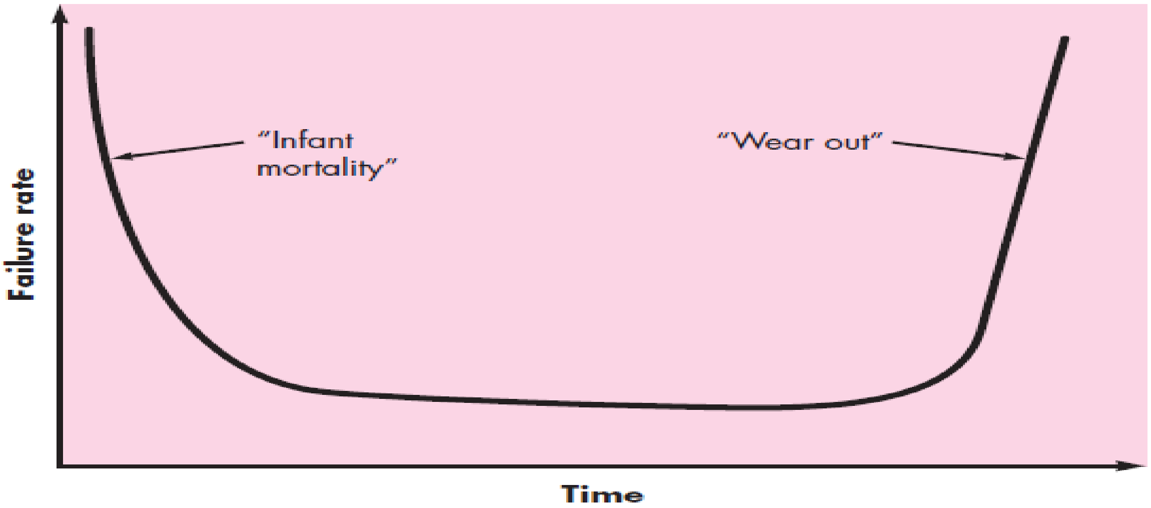
   It is a client-server computer program which the client runs on the web browser. In their simplest form, Web apps can be little more than a set of linked hypertext files that present information using text and limited graphics. However, as e-commerce and B2B application grow in importance. Web apps are evolving into a sophisticate computing environment that not only provides a standalone feature, computing function, and content to the end user.

7. **Artificial Intelligence Software:**

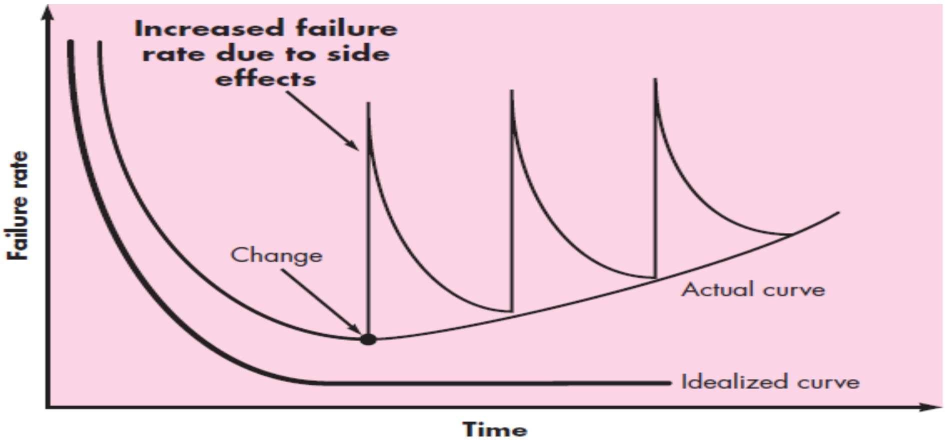
Artificial intelligence software makes use of a nonnumerical algorithm to solve a complex problem that is not amenable to computation or straightforward analysis. Application within this area includes robotics, expert system, pattern recognition, artificial neural network, theorem proving and game playing.

1. Why does it take so long to get software finished?
2. Why are development costs so high?
3. Why can't we find all errors before we give the software to our customers?
4. Why do we spend so much time and effort maintaining existing programs?
5. Why do we continue to have difficulty in measuring progress as software is being developed and maintained?

Failure curve for hardware:



Failure curves for software:



The unique nature of web apps:

Today, WebApps have evolved into sophisticated computing tools that not only provide stand-alone function to the end user, but also have been integrated with corporate databases and business applications due to the development of HTML, JAVA, XML etc.

Attributes of WebApps:

1. Network Intensiveness
2. Concurrency
3. Unpredictable load
4. Performance
5. Availability
6. Data driven
7. Content Sensitive
8. Continuous evolution
9. Immediacy
10. Security
11. Aesthetic

Network intensiveness:
A WebApp resides on a network and must serve the needs of a diverse community of clients.
The network may enable worldwide access and communication (i.e., the Internet) or more limited access and communication
(e.g., a corporate Intranet Network Intensiveness)

Concurrency: [Operation at the same time]
A large number of users may access the WebApp at one time. In many cases, the patterns of usage among end users will vary greatly.

Unpredictable load:
The number of users of the WebApp may vary by orders of magnitude from day to day. One hundred users may show up on Monday; 10,000 may use the system on Thursday.

Performance:
If a WebApp user must wait too long (for access, for server side processing, for client-side formatting and display), he or she may decide to go elsewhere.

Availability:
Although expectation of 100 percent availability is unreasonable, users of popular WebApps often demand access on a 24/7/365 basis.

Data driven:
The primary function of many WebApps is to use hypermedia to present text, graphics, audio, and video content to the end user.

In addition, WebApps are commonly used to access information that exists on databases that are not an integral part of the Web-based environment (e.g., e-commerce or financial applications).

Content sensitive:
The quality and artistic nature of content remains an important Determinant of the quality of a WebApp.

Continuous evolution:
Unlike conventional application software that evolves over a series of planned, chronologically spaced releases, Web applications evolve continuously. It is not unusual for some WebApps (specifically, their content) to be updated on a minute-by-minute schedule or for content to be independently computed for each request.

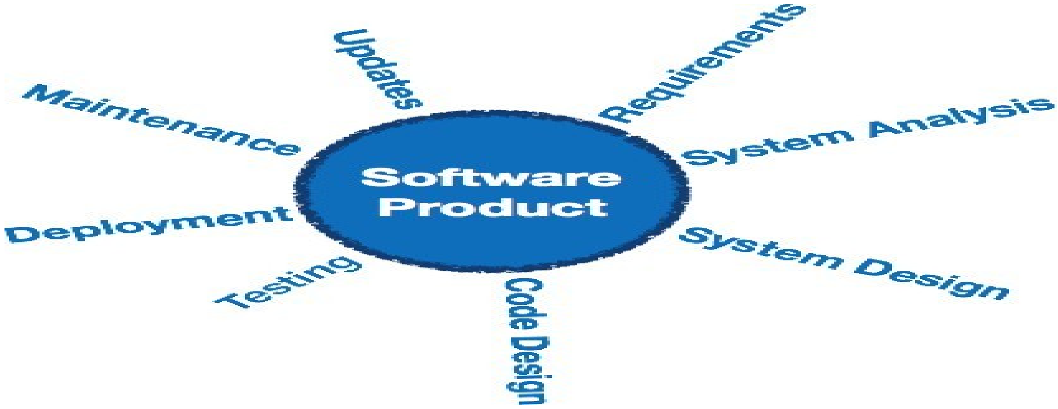
Immediacy:
Although immediacy—the compelling (forceful) need to get software to market quickly—is a characteristic of many application domains, WebApps often exhibit a time-to-market that can be a matter of a few days or weeks.

Security:
Because WebApps are available via network access, it is difficult, if not impossible, to limit the population of end users who may access the application. In order to protect sensitive content and provide secure mode of data transmission, strong security measures must be implemented.

Aesthetics : [Artistic / Visual]
An undeniable part of the appeal of a WebApp is its look and feel. When an application has been designed to market or sell products or ideas, aesthetic may have as much to do with success as technical design.

Software Engineering:

Software engineering is a branch of computer science which includes the development and building of computer systems software and applications software. Computer systems software is composed of programs that include computing utilities and operations systems.



The Software Process:

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

There are some fundamental activities that are common to all software processes:

1. **Software specification**. In this activity the functionality of the software and constraints on its operation must be defined.

2. **Software design and implementation**. The software that meets the specification is produced.

3. **Software validation**. The software must be validated to ensure that it has all the functionalities what the customer needs.

4. **Software evolution**. The software must evolve to meet changing customer needs.

When we talk about a process, we usually talk about the activities in it. However, a process also includes the process description, which includes:

1. **Products**: The outcomes of the an activity. For example, the outcome of architectural design maybe a model for the software architecture.

2. **Roles**: The responsibilities of the people involved in the process. For example, the project manager, programmer, etc.

3. **Pre and post conditions**: The conditions that must be true before and after an activity. For example, the pre condition of the architectural design is the requirements have been approved by the customer, while the post condition is the diagrams describing the architectural have been reviewed.

Software process is complex, it relies on making decisions. There’s no ideal process and most organizations have developed their own software process.

For example, an organization works on critical systems has a very structured process, while with business systems, with rapidly changing requirements, a less formal, flexible process is likely to be more effective.

A process is a collection of activities, actions, and tasks that are performed when some work product is to be created. An activity strives to achieve a broad objective

It Consists of the following Umbrella Activities. They are

1. **Communication.**
2. **Planning.**
3. **Modeling.**
4. **Construction.**
5. **Deployment**

Software Engineering Practice:

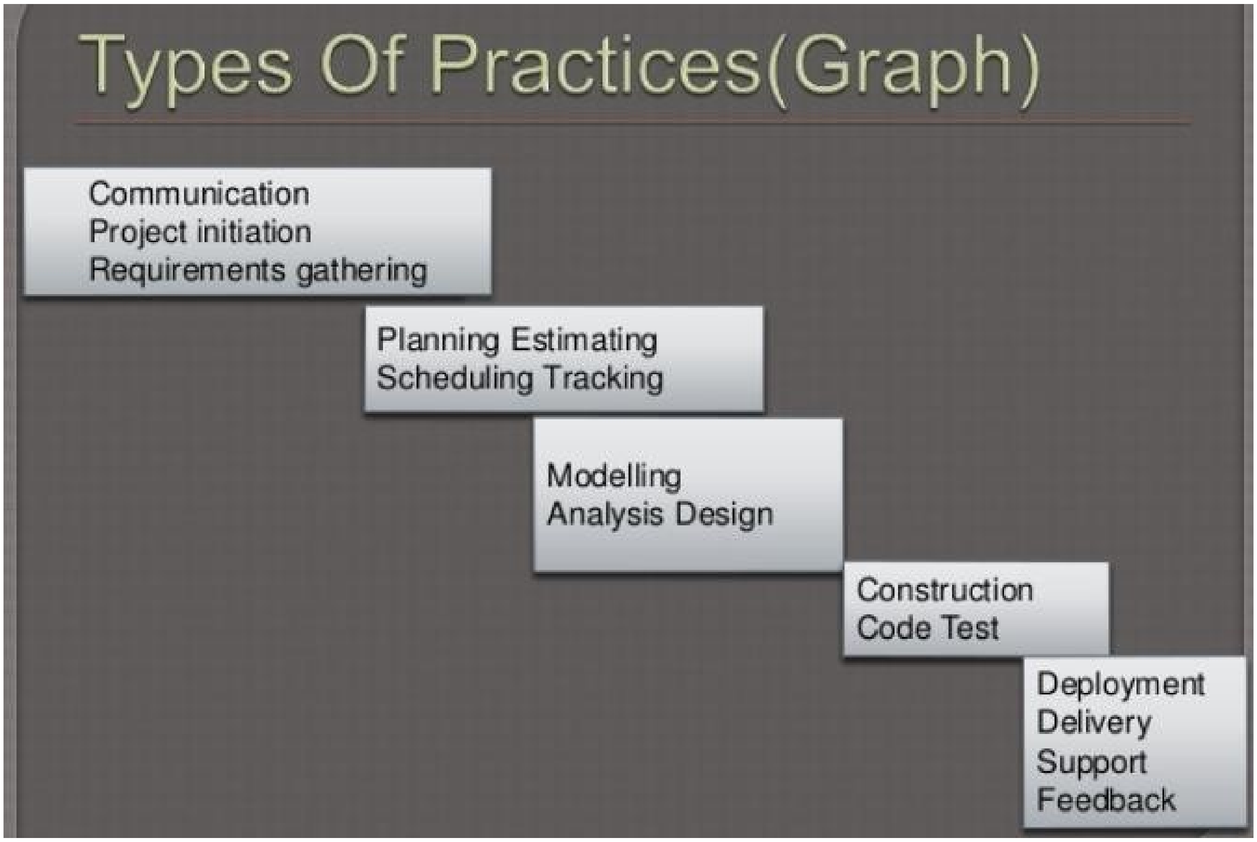
A software engineer applies engineering practices to software development, and typically handles the overall system design of the software application. Once the coding is complete, they test the software and make sure it performs within the engineered requirements.

Principle 1 - Understand the

problem Principle 2 - Plan a solution

Principle 3 - Carry out the plan

Principle 4 - Examine the result for accuracy



Types Of Practices(Graph)

Communication
Project initiation
Requirements gathering

Planning Estimating
Scheduling Tracking

Modelling
Analysis Design

Construction
Code Test

Deployment
Delivery
Support
Feedback

Software Myths:

The development of software requires dedication and understanding on the developers' part. Many software problems arise due to myths that are formed during the initial stages of software development. Unlike ancient folklore that often provides valuable lessons, "software myths propagate false beliefs and confusion in the minds of management, users and developers."

There are 3 types of Myths:

1. Software Customer Myths:
2. Developer Myths:
3. Software Management Myths.

1. Software Customer Myths.
a) A general statement of need is sufficient to start coding
b) Change is easily accommodated, since software is malleable.

2. Software Developer(Practitioner's)Myths.

a) *Once we write the program and get it to work, our job is done.*

b) *Until I get the program "running" I have no way of assessing its quality.*

c) The only deliverable work product for a successful project is the working program.

d) *Software engineering will make us create voluminous and unnecessary documentation and will invariably slow us down.*

3. Software Management Myths.

a) Development problems can be solved by developing and documenting standards.

b) Development problems can be solved by using state-of-the art tools.

c) When schedules slip, just add more people

- The software must be adapted to meet the needs of new computing environments or technology.
- The software must be enhanced to implement new business requirements.
- The software must be extended to make it interoperable with other more modern systems or databases.
- The software must be re-architected to make it viable within a network environment.

General Principles:

The First Principle: *The Reason It All Exists*

The Second Principle: *KISS (Keep It Simple, Stupid!)*

The Third Principle: *Maintain the Vision*

The Fourth Principle: *What You Produce, Others Will Consume*

The Fifth Principle: *Be Open to the Future*

The Sixth Principle: *Plan Ahead for Reuse*

The Seventh principle: *Think!*

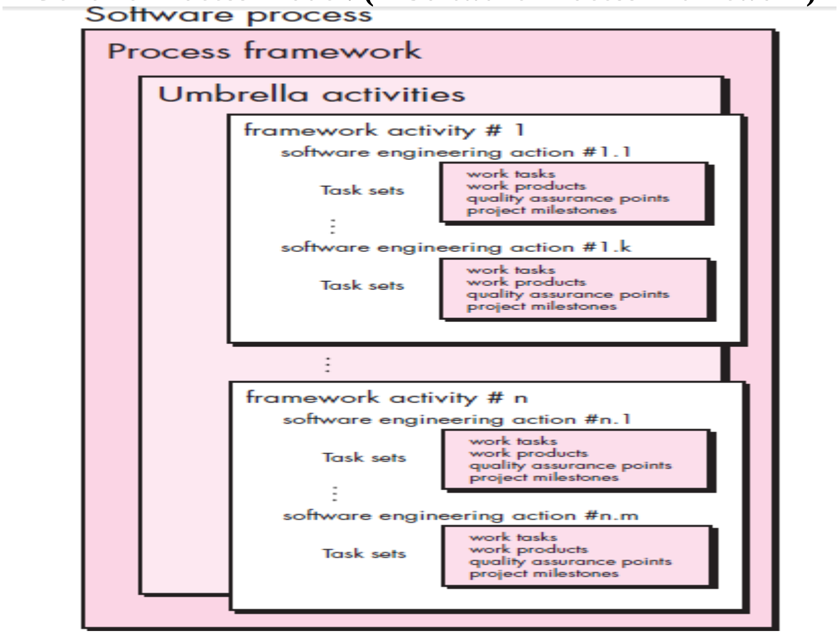
Process Models

A software process model is a simplified representation of a software process. Each model represents a process from a specific perspective. Software Processes is a coherent set of activities for specifying, designing, implementing and testing software systems. A software process model is an abstract representation of a process that presents a description of a process from some particular perspective.

There are many different software processes:
1. Specification – defining what the system should do.
2. Design and implementation – defining the organization of the system and implementing the system.
3. Validation – checking that it does what the customer wants.
4. Evolution – changing the system in response to changing customer needs.

Popular software Process Models:
1. Waterfall model
2. V model
3. Incremental model
4. RAD model
5. Agile model
6. Iterative model
7. Spiral model
8. Prototype model

A Generic Process Model: (A Software Process Framework)



Framework is a Standard way to build and deploy applications. **Software Process Framework** is a foundation of complete software engineering process. Software process framework includes all set of umbrella activities. It also includes number of framework activities that are applicable to all software projects.

There are five generic process framework activities:

1. Communication:
The software development starts with the communication between customer and developer.

2. Planning:
It consists of complete estimation, scheduling for project development and tracking.

3. Modeling:

- Modeling consists of complete requirement analysis and the design of the project like algorithm, flowchart etc.
- The algorithm is the step-by-step solution of the problem and the flow chart shows a complete flow diagram of a program.

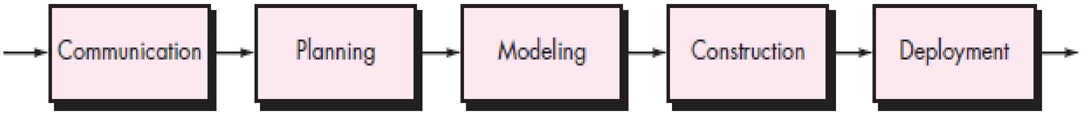
4. Construction:

- Construction consists of code generation and the testing part.
- Coding part implements the design details using an appropriate programming language.
- Testing is to check whether the flow of coding is correct or not.
- Testing also check that the program provides desired output.

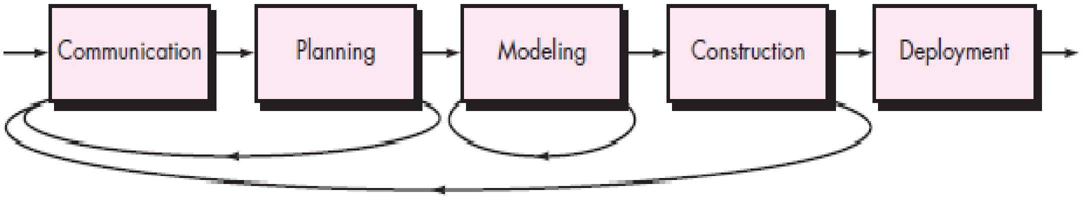
5. Deployment:

- Deployment step consists of delivering the product to the customer and take feedback from them.
- If the customer wants some corrections or demands for the additional capabilities, then the change is required for improvement in the quality of the software.

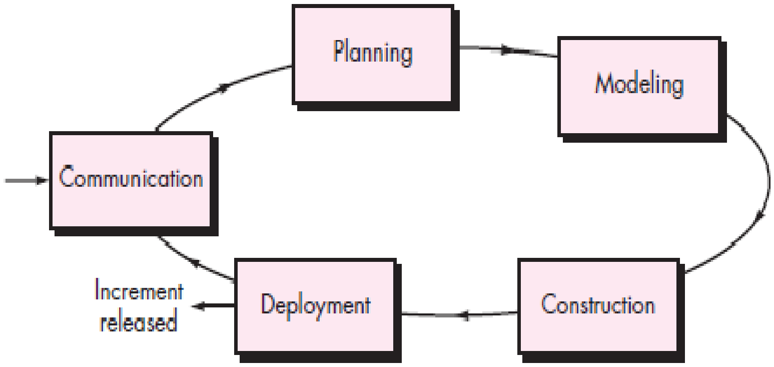
The following activities are taken into consideration:
1. Make contact with stakeholder via telephone.
2. Discuss requirements and take notes.
3. Organize notes into a brief written statement of requirements.
4. E-mail to stakeholder for review and approval.



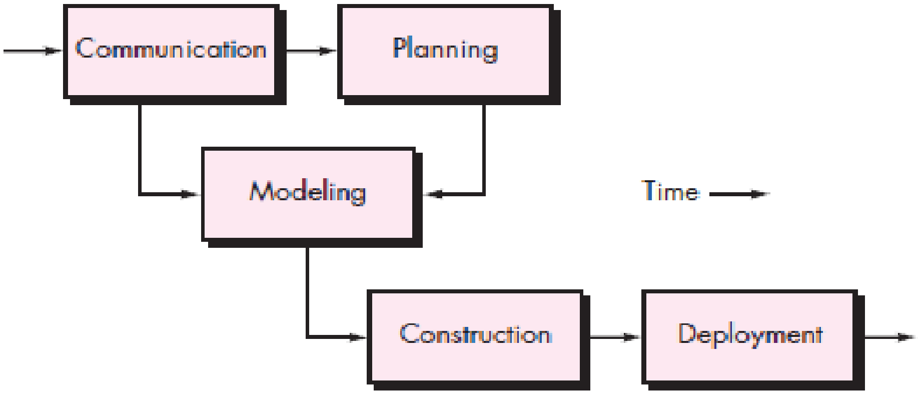
(a) Linear process flow



(b) Iterative process flow



(c) Evolutionary process flow



(d) Parallel process flow

Process Assessment and Improvement:

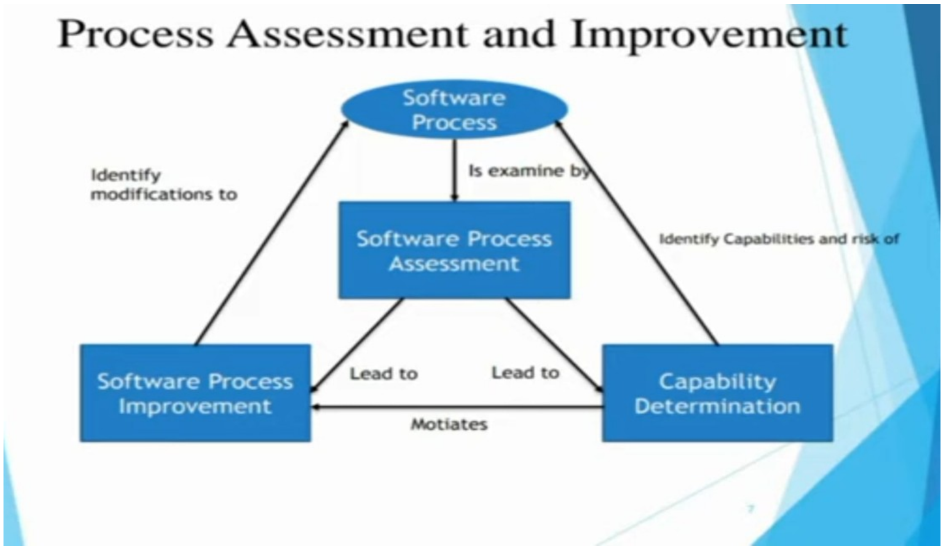
A software process assessment is a disciplined examination of the software processes used by an organization, based on a process model. The assessment includes the identification and characterization of current practices, identifying areas of strengths and weaknesses, and the ability of current practices to control or avoid significant causes of poor (software) quality, cost, and schedule.

A software assessment (or audit) can be of three types.

- A **self-assessment (first-party assessment)** is performed internally by an organization's own personnel.
- A **second-party assessment** is performed by an external assessment team or the organization is assessed by a customer.
- A **third-party assessment** is performed by an external party or (e.g., a supplier being assessed by a third party to verify its ability to enter contracts with a customer).

Software process assessments are performed in an open and collaborative environment. They are for the use of the organization to improve its software processes, and the results are confidential to the organization. The organization being assessed must have members on the assessment team.

1. Standard CMMI Assessment Method for Process Improvement (SCAMPI)
2. CMM-Based Appraisal for Internal Process Improvement (CBA IPI)
3. SPICE (ISO/IEC15504)
4. ISO 9001:2000 for Software.


Process Assessment and Improvement
Software Process
Is examine by
Identify modifications to
Software Process Assessment
Identify Capabilities and risk of
Software Process Improvement
Lead to
Lead to
Capability Determination
Motiates

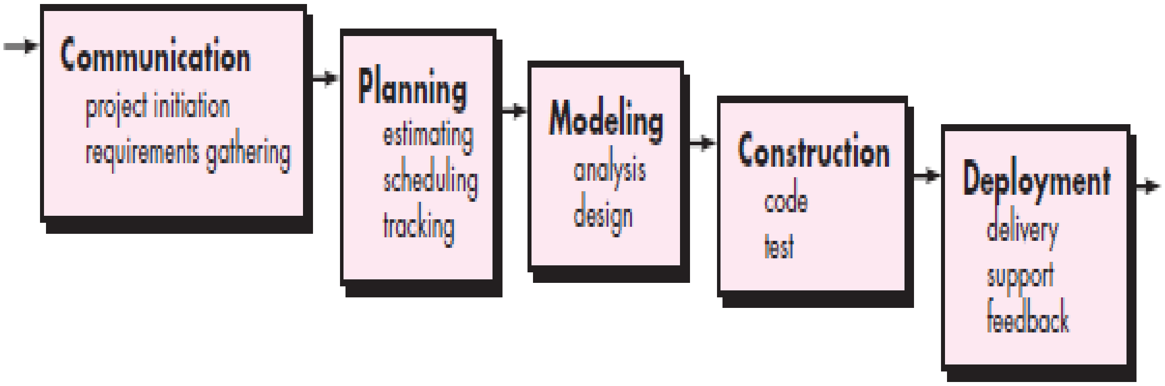
Prescriptive Process Models:

The name 'prescriptive' is given because the model prescribes a set of activities, actions, tasks, quality assurance and change the mechanism for every project.

- Communication – Involves communication among the customer and other stake holders; encompasses requirements gathering
- Planning – Establishes a plan for software engineering work; addresses technical tasks, resources, work products, and work schedule
- Modeling (Analyze, Design) – Encompasses the creation of models to better under the requirements and the design
- Construction (Code, Test) – Combines code generation and testing to uncover errors
- Deployment – Involves delivery of software to the customer for evaluation and feedback

1. The Waterfall Model

- The waterfall model is also called as **'Linear sequential model'** or **'Classic life cycle model'.**
- In this model, each phase is fully completed before the beginning of the next phase.
- This model is used for the small projects.
- In this model, feedback is taken after each phase to ensure that the project is on the right path.
- Testing part starts only after the development is complete.



Advantages of waterfall model

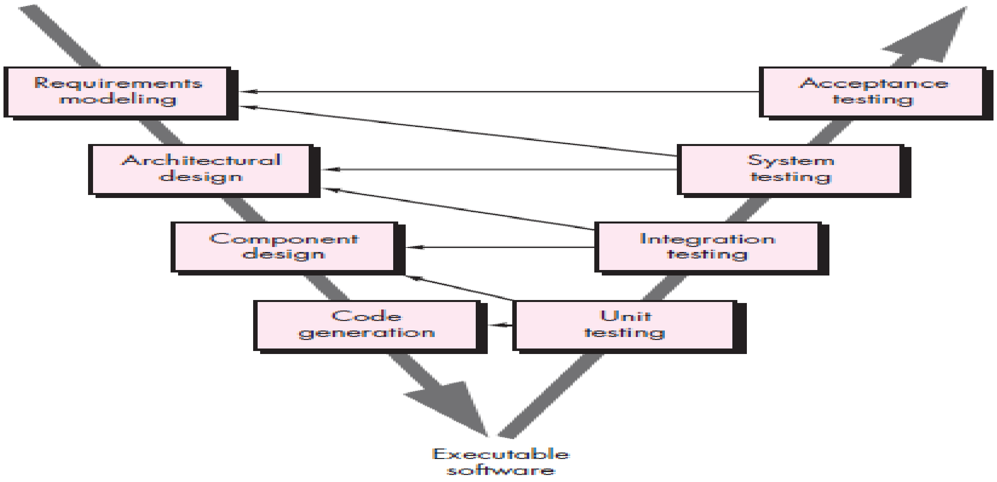
- The waterfall model is simple and easy to understand, implement, and use.
- All the requirements are known at the beginning of the project, hence it is easy to manage.
- It avoids overlapping of phases because each phase is completed at once.
- This model works for small projects because the requirements are understood very well.
- This model is preferred for those projects where the quality is more important as compared to the cost of the project.

Disadvantages of the waterfall model

- This model is not good for complex and object oriented projects.
- It is a poor model for long projects.
- The problems with this model are uncovered, until the software testing.
- The amount of risk is high.

2. V (Validation & Verification) model:

The V-model is a type of SDLC model where process executes in a sequential manner in V-shape. It is also known as Verification and Validation model. It is based on the association of a testing phase for each corresponding development stage. Development of each step directly associated with the testing phase. The next phase starts only after completion of the previous phase i.e. for each development activity, there is a testing activity corresponding to it .



Verification: It involves static analysis technique (review) done without executing code. It is the process of evaluation of the product development phase to find whether specified requirements meet.

Validation: It involves dynamic analysis technique (functional, non-functional), testing done by executing code. Validation is the process to evaluate the software after the completion of the development phase to determine whether software meets the customer expectations and requirements.

So V-Model contains Verification phases on one side of the Validation phases on the other side. Verification and Validation phases are joined by coding phase in V-shape. Thus it is called V-Model.

Advantages:

- This is a highly disciplined model and Phases are completed one at a time.
- V-Model is used for small projects where project requirements are clear.
- Simple and easy to understand and use.
- This model focuses on verification and validation activities early in the life cycle thereby enhancing the probability of building an error-free and good quality product.
- It enables project management to track progress accurately.

Disadvantages:

- High risk and uncertainty.
- It is not a good for complex and object-oriented projects.
- It is not suitable for projects where requirements are not clear and contains high risk of changing.
- This model does not support iteration of phases.
- It does not easily handle concurrent events.

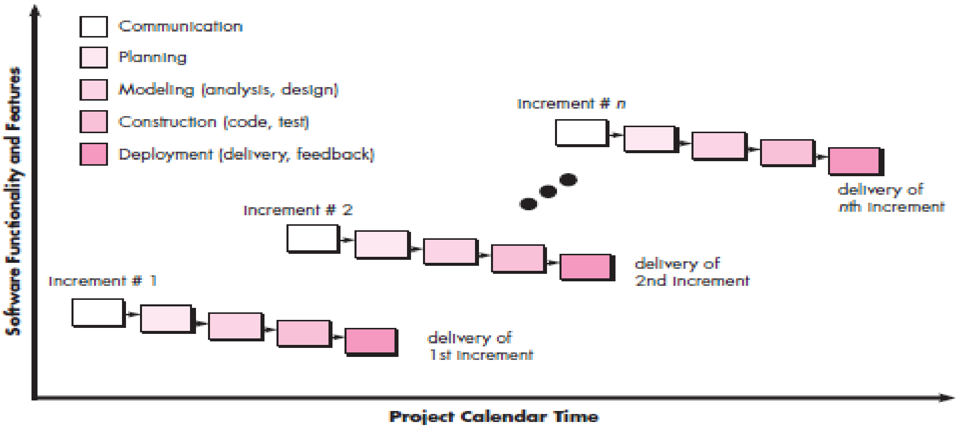
Principles of V-Model:

- **Large to Small:** In V-Model, testing is done in a hierarchical perspective, For example, requirements identified by the project team, create High-Level Design, and Detailed Design phases of the project. As each of these phases is completed the requirements, they are defining become more and more refined and detailed.
- **Data/Process Integrity:** This principle states that the successful design of any project requires the incorporation and cohesion of both data and processes. Process elements must be identified at each and every requirements.
- **Scalability:** This principle states that the V-Model concept has the flexibility to accommodate any IT project irrespective of its size, complexity or duration.
- **Cross Referencing:** Direct correlation between requirements and corresponding testing activity is known as cross-referencing.
- **Tangible Documentation:** This principle states that every project needs to create a document. This documentation is required and applied by both the project development team and the support team. Documentation is used to maintaining the application once it is available in a production environment.

3. The Incremental Model:

- The incremental model combines the elements of waterfall model and they are applied in an iterative fashion.
- The first increment in this model is generally a core product.
- Each increment builds the product and submits it to the customer for any suggested modifications.

- The next increment implements on the customer's suggestions and add additional requirements in the previous increment.
- This process is repeated until the product is finished.


Communication
Planning
Modeling (analysis, design)
Construction (code, test)
Deployment (delivery, feedback)
Increment # n
delivery of nth increment
Increment # 2
delivery of 2nd increment
Increment # 1
delivery of 1st increment
Software Functionality and Features
Project Calendar Time

Advantages of incremental model

- This model is flexible because the cost of development is low and initial product delivery is faster.
- It is easier to test and debug during the smaller iteration.
- The working software generates quickly and early during the software life cycle.
- The customers can respond to its functionalities after every increment.

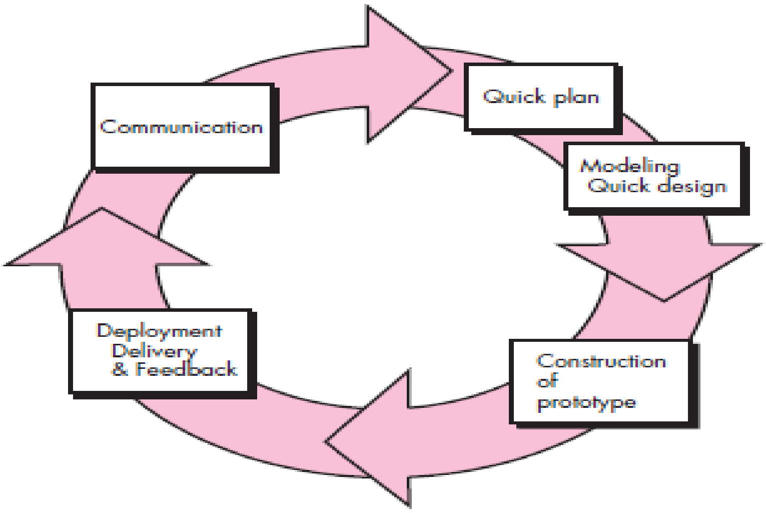
Disadvantages of the incremental model

- The cost of the final product may cross the cost estimated initially.
- This model requires a very clear and complete planning.
- The planning of design is required before the whole system is broken into small increments.
- The demands of customer for the additional functionalities after every increment causes problem during the system architecture.

4. The Prototype Model

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

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



Advantages –

- The customers get to see the partial product early in the life cycle. This ensures a greater level of customer satisfaction and comfort.
- New requirements can be easily accommodated as there is scope for refinement.
- Missing functionalities can be easily figured out.
- Errors can be detected much earlier thereby saving a lot of effort and cost, besides enhancing the quality of the software.
- The developed prototype can be reused by the developer for more complicated projects in the future.

- Flexibility in design.

Disadvantages –

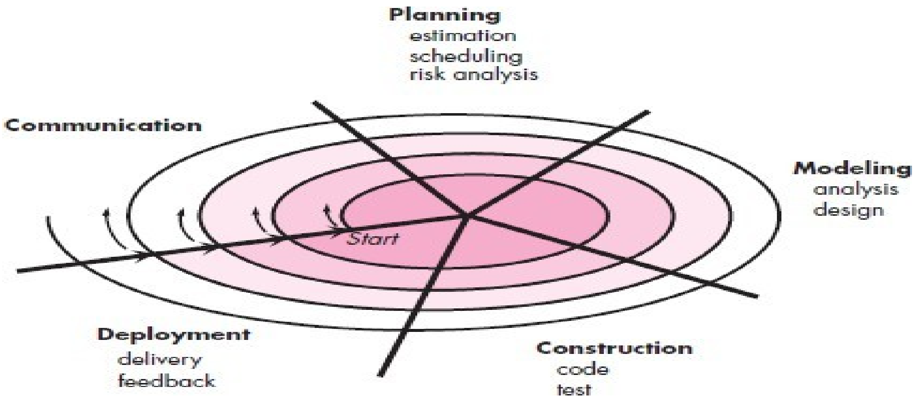
- Costly w.r.t time as well as money.
- There may be too much variation in requirements each time the prototype is evaluated by the customer.
- Poor Documentation due to continuously changing customer requirements.
- It is very difficult for developers to accommodate all the changes demanded by the customer.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

5. The Spiral Model:

The spiral model combines the idea of iterative development with the systematic, controlled aspects of the waterfall model. This Spiral model is a combination of iterative development process model and sequential linear development model i.e. the waterfall model with a very high emphasis on risk analysis. It allows incremental releases of the product or incremental refinement through each iteration around the spiral.

The advantage of spiral lifecycle model is that it allows elements of the product to be added in, when they become available or known. This assures that there is no conflict with previous requirements and design.

This method is consistent with approaches that have multiple software builds and releases which allows making an orderly transition to a maintenance activity. Another positive aspect of this method is that the spiral model forces an early user involvement in the system development effort.



The advantages of the Spiral SDLC Model are as follows −

- Changing requirements can be accommodated.
- Allows extensive use of prototypes.
- Requirements can be captured more accurately.
- Users see the system early.
- Development can be divided into smaller parts and the risky parts can be developed earlier which helps in better risk management.

The disadvantages of the Spiral SDLC Model are as follows −

- Management is more complex.
- End of the project may not be known early.
- Not suitable for small or low risk projects and could be expensive for small projects.

- Process is complex
- Spiral may go on indefinitely.
- Large number of intermediate stages requires excessive documentation.

Specialized Process Models:

Specialized process models use many of the characteristics of one or more of the conventional **models** presented so far, however, they tend to be applied when a narrowly defined **software engineering** approach is chosen. They include Components based development.
Software engineers and their managers adopt a process model to their needs and then follow it. Each prescriptive model can accommodate generic process framework that encompasses ... Types in Specialized process models
Special process models take on many of the characteristics of one or more of the conventional models. However, specialized models tend to be applied when a narrowly defined software engineering approach is chosen.
Commercial off-the-shelf (COST) software components, developed by vendors who offer them as products, can be used when software is to be built. These components provide targeted functionality with well-defined interfaces that enable the component to be integrated into the software.

1. Available component-based products are researched and evaluated for the application domain in question.
2. Component integration issues are considered.
3. A software architecture is designed to accommodate the components.
4. Components are integrated into the architecture.
5. Comprehensive testing is conducted to ensure proper functionality.

The Unified Process:

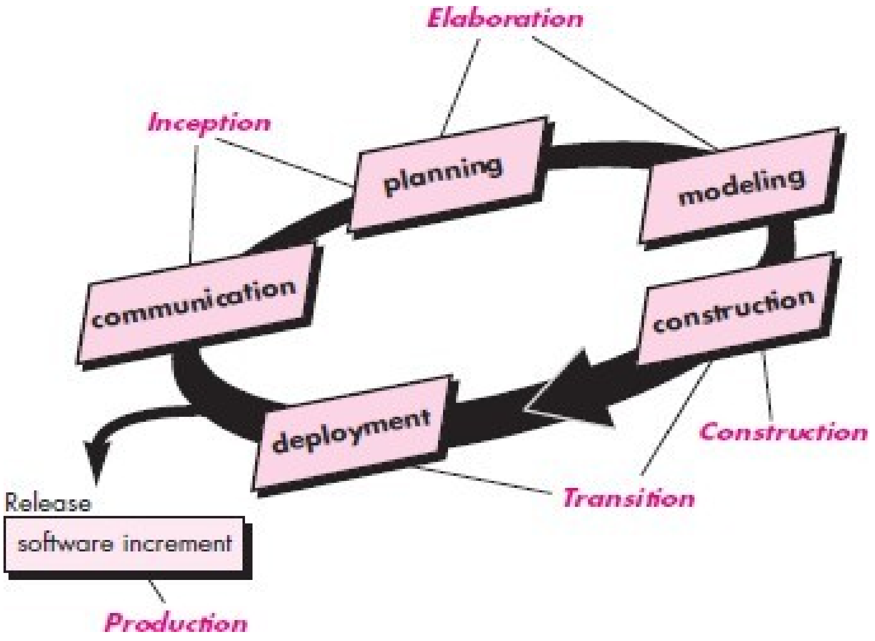
Unified Process is based on the enlargement and refinement of a system through multiple iterations, with cyclic feedback and adaptation. The system is developed incrementally over time, iteration by iteration, and thus this approach is also known as iterative and incremental software development.

The Unified Process is an attempt to draw on the best features and characteristics of traditional software process models, but characterize them in a way that implements many of the best principles of agile software development. The Unified Process recognizes the importance of customer communication and streamlined methods for describing the customer's view of a system. It emphasizes the important role of software architecture and "helps the architect focus on the right goals, such as understandability, reliance to future changes, and reuse" . It suggests a process flow that is iterative and incremental, providing the evolutionary feel that is essential in modern software development.

Phases of the Unified Process

This process divides the development process into five phases:

- Inception
- Elaboration
- Conception
- Transition
- Production



1. The Inception phase of the UP encompasses both customer communication and planning activities. By collaborating with stakeholders, business requirements for the software are identified; a rough architecture for the system is proposed; and a plan for the iterative, incremental nature of the ensuing project is developed.

2. The Elaboration phase encompasses the communication and modeling activities of the generic process model. Elaboration refines and expands the preliminary use cases that were developed as part of the inception phase and expands the architectural representation to include five different views of the software the use case model, the requirements model, the design model, the implementation model, and the deployment model.

3. The Construction phase of the UP is identical to the construction activity defined for the generic software process. Using the architectural model as input, the construction phase develops or acquires the software components that will make each use case operational for end users. To accomplish this, requirements and design models that were started during the elaboration phase are completed to reflect the final version of the software increment. All necessary and required features and functions for the software increment (the release) are then implemented in source code.

4. The Transition phase of the UP encompasses the latter stages of the generic construction activity and the first part of the generic deployment (delivery and feedback) activity. Software is given to end users for beta testing and user feedback reports both defects and necessary changes. At the conclusion of the transition phase, the software increment becomes a usable software release.

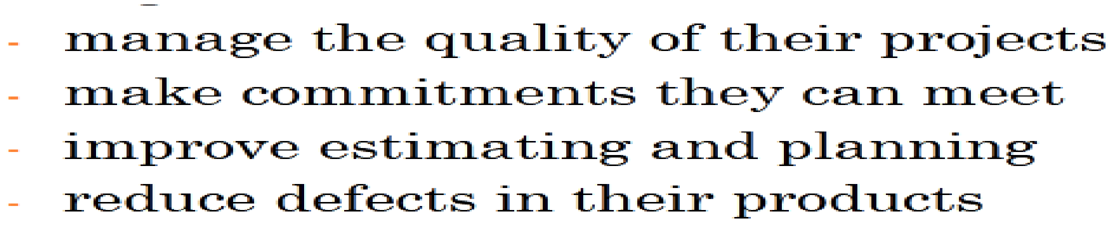
5. The Production phase of the UP coincides with the deployment activity of the generic process. During this phase, the ongoing use of the software is monitored, support for the operating environment (infrastructure) is provided, and defect reports and requests for changes are submitted and evaluated. It is likely that at the same time the construction, transition, and production phases are being conducted, work may have already begun on the next software increment. This means that the five UP phases do not occur in a sequence, but rather with staggered concurrency.

Personal and Team Process Models:

The Personal Software Process (PSP) emphasizes personal measurement of both the work product that is produced and the resultant quality of the work product. In addition PSP makes the practitioner responsible for project planning and empowers the practitioner to control the quality of all software work products that are developed. The PSP model defines five framework activities:

- ***Planning.*** This activity isolates requirements and develops both size and resource estimates. In addition, defects estimate (the number of defects projected for the work) is made. All metrics are recorded on worksheets or templates. Finally, development tasks are identified and a project schedule is created.
- ***High level design.*** External specifications for each component to be constructed are developed and a component design is created. Prototypes are built when uncertainty exists. All issues are recorded and tracked.
- ***High level design review.*** Formal verification methods are applied to uncover errors in the design. Metrics are maintained for all important tasks and work results.
- ***Development.*** The component level design is refined and reviewed. Code is generated, reviewed, compiled, and tested. Metrics are maintained for all important tasks and work results.
- ***Postmortem.*** Using the measures and metrics collected, the effectiveness of the process is determined. Measures and metrics should provide guidance for modifying the process to improve its effectiveness.

The Personal Software Process (PSP) shows engineers how to



- manage the quality of their projects
- make commitments they can meet
- improve estimating and planning
- reduce defects in their products

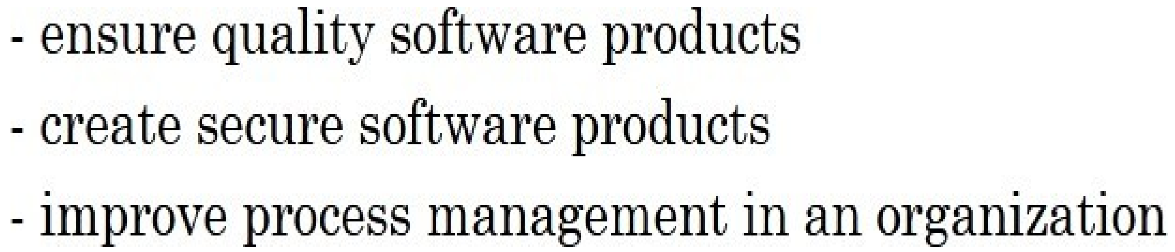
Watts Humphrey extended the lessons learned from the introduction of PSP and proposed a Team Software Process (TSP). The goal of TSP is to build a “self directed” project team that organizes itself to produce high quality software.

Humphrey defines the following objectives for TSP:

- Build self directed teams that plan and track their work, establish goals, and own their processes and plans. These can be pure software teams or integrated product teams (IPTs) of 3 to about 20 engineers.
- Show managers how to coach and motivate their teams and how to help them sustain peak performance

- Accelerate software process improvement by making CMM23 Level 5 behavior normal and expected.
- Provide improvement guidance to high-maturity organizations.
- Facilitate university teaching of industrial-grade team skills.

The Team Software Process (TSP) shows engineers how to



- ensure quality software products
- create secure software products
- improve process management in an organization

Process Technology:

Process technology is the software and tools—including digital platforms and intelligent process automation (IPA)—used to efficiently and intelligently create and deliver products and services, as well as improves speed and agility of processes across the enterprise.

Process technology can be used to automate and simplify workflows and accelerate business processes. It can also improve accuracy, quality, and reliability of those processes, helping companies operate with greater agility so they can be more flexible and adaptable to shifting business and market priorities. Finally, process technology can play a key role in reducing costs across business operations.

Product and Process:

In the context of software engineering, Product includes any software manufactured based on the customer’s request. This can be a problem solving software or computer based system. It can also be said that this is the result of a project.

* The product is the final result of a development cycle.

* Product development focus is on final outcome.

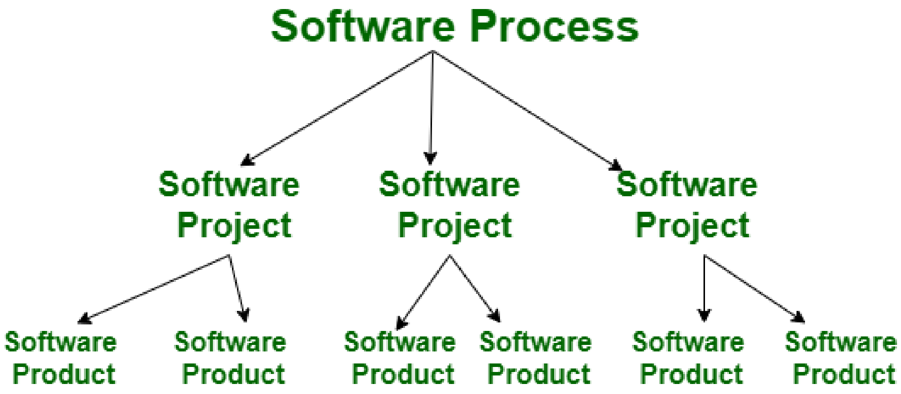
Process:

Process is a set of sequence steps that have to be followed to create a project. The main purpose of a process is to improve the quality of the project. The process serves as a template that can be used through the creation of its examples and is used to direct the project.

* The process focuses on each step to be followed during software product development.

* The process is a sequence or set of steps that should be followed to create a product

The main difference between a process and a product is that the process is a set of steps that guide the project to achieve the convenient product. while on the other hand, the product is the result of a project that is manufactured by a wide variety of people.



Agile Development

Agile software development refers to **a group of software development methodologies based on iterative development**, where requirements and solutions evolve through collaboration between self-organizing cross-functional teams.

Agile model believes that every project needs to be handled differently and the existing methods need to be tailored to best suit the project requirements. In Agile, the tasks are divided to time boxes (small time frames) to deliver specific features for a release.

Iterative approach is taken and working software build is delivered after each iteration. Each build is incremental in terms of features; the final build holds all the features required by the customer.

Agile software engineering combines a philosophy and a set of development guidelines. The philosophy encourages customer satisfaction and early incremental delivery of software.

- Small, highly motivated project teams
- Informal methods; minimal software engineering work products;
- Overall development simplicity.
- The development guidelines stress delivery over analysis
- Design active and continuous communication between developers and customers.

What is Agility?

Agile is the ability to create and respond to change. It is a way of dealing with, and ultimately succeeding in, an uncertain and turbulent environment.

The authors of the Agile Manifesto chose “Agile” as the label for this whole idea because that word represented the addictiveness and response to change which was so important to their approach.

It’s really about thinking through how you can understand what’s going on in the environment that you’re in today, identify what uncertainty you’re facing, and figure out how you can adapt to that as you go along.

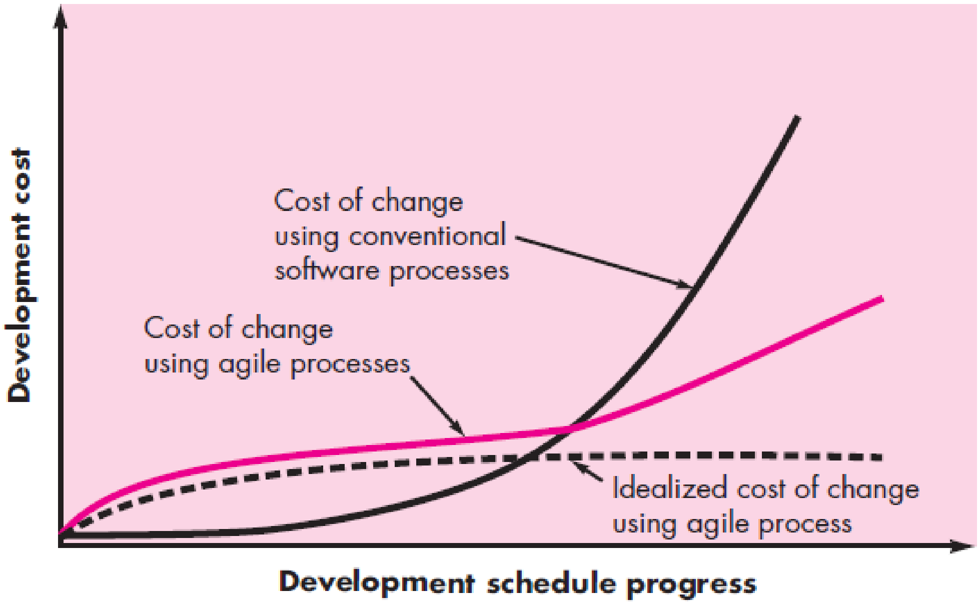
- Agility means effective (rapid and adaptive) response to change, effective communication among all stockholder.
- Drawing the customer onto team and organizing a team so that it is in control of work performed. -The Agile process, light-weight methods are People-based rather than plan-based methods.
- The agile process forces the development team to focus on software itself rather than design and documentation.
- The agile process believes in iterative method.
- The aim of agile process is to deliver the working model of software quickly to the customer

General Principles:

1. Highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. It welcomes changing requirements, even late in development.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shortest timescale.
4. Build projects around motivated individuals. Give them the environment and the support they need, and trust them to get the job done.
5. Working software is the primary measure of progress.
6. Simplicity the art of maximizing the amount of work not done is essential.
7. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Advantages:

- Deployment of software is quicker and thus helps in increasing the trust of the customer.
- Can better adapt to rapidly changing requirements and respond faster.
- Helps in getting immediate feedback which can be used to improve the software in the next increment.
- People – Not Process. People and interactions are given a higher priority rather than process and tools.
- Continuous attention to technical excellence and good design.



Agile Software Process:

The agile model encourages to practice of continuous iteration of development and testings in the software development life cycle SDLC. Here both the development and testing activities are performed together so that better track on defects can be identified. This was one of the major shortcomings of the waterfall model.

Agile Process models are below:

- Scrum
- Extreme Programming
- Adaptive Software Development ASD
- Dynamic System Development Model DSDM
- Crystal
- Feature Driven Development

Agility Principles:

1. Our highest priority is to **satisfy the customer** through early and continuous delivery of valuable software.

2. Welcome **changing requirements**, even late in development. Agile processes harness change for the customer’s competitive advantage.

3. Deliver **working software** frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

4. Business people and developers must **work together** daily throughout the project.

5. Build projects around **motivated individuals**. Give them the environment and support they need, and trust them to get the job done.

6. The most efficient and effective method of conveying information to and within a development team is **face-to-face conversation**.

7. **Working software** is the primary measure of progress.

8. **Agile processes** promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

9. **Continuous attention** to technical excellence and good design enhances agility.

10. **Simplicity**—the art of maximizing the amount of work not done—is essential.

11. The **best architectures, requirements, and designs** emerge from self–organizing teams.

12. At regular intervals, the team reflects on how to become **more effective**, then tunes and adjusts its behavior accordingly.

Extreme Programming (XP):

Extreme programming is a software development methodology that's part of what's collectively known as agile methodologies. XP is built upon values, principles, and practices, and its goal is to allow small to mid-sized teams to produce high-quality software and adapt to evolving and changing requirements.

Extreme Programming uses an object-oriented approach as its preferred development paradigm and encompasses a set of rules and practices that occur within the context of four framework activities: planning, design, coding, and testing.

Extreme programming (XP) is one of the most important software development frameworks of Agile models. It is used to improve software quality and responsive to customer requirements. The extreme programming model recommends taking the best practices that have worked well in the past in program development projects to extreme levels.

Software Engineering involves −

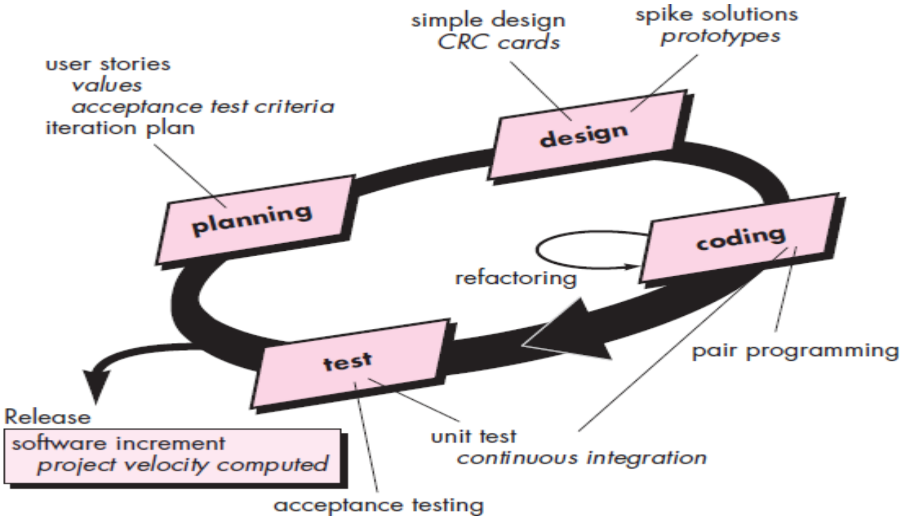
- Creativity
- Learning and improving through trials and errors
- Iterations

Extreme Programming builds on these activities and coding. It is the detailed (not the only) design activity with multiple tight feedback loops through effective implementation, testing and refactoring continuously.

Extreme Programming is based on the following values −

- Communication
- Simplicity
- Feedback
- Courage
- Respect

- **Code Review:** Code review detects and corrects errors efficiently. It suggests pair programming as coding and reviewing of written code carried out by a pair of programmers who switch their works between them every hour.
- **Testing:** Testing code helps to remove errors and improves its reliability. XP suggests test-driven development (TDD) to continually write and execute test cases. In the TDD approach test cases are written even before any code is written.
- **Incremental development:** Incremental development is very good because customer feedback is gained and based on this development team come up with new increments every few days after each iteration.
- **Simplicity:** Simplicity makes it easier to develop good quality code as well as to test and debug it.
- **Design:** Good quality design is important to develop a good quality software. So, everybody should design daily.
- **Integration testing:** It helps to identify bugs at the interfaces of different functionalities. Extreme programming suggests that the developers should achieve continuous integration by building and performing integration testing several times a day.



Applications of Extreme Programming (XP): Some of the projects that are suitable to develop using XP model are given below:

- **Small projects:** XP model is very useful in small projects consisting of small teams as face to face meeting is easier to achieve.
- **Projects involving new technology or Research projects:** This type of projects faces changing of requirements rapidly and technical problems. So XP model is used to complete this type of projects.

Model Question Bank: (Module-1: The Software Process)

S.NO	QUESTION	CO	BL	MARKS
1.	Define Software and explain Software Engineering process in detailed	1	1	12
2.	What is the nature of software? Discuss about the unique nature of WebApps	1	1,2	12
3.	Define Software Process? Explain about Software Engineering Practice?	1	1	12
4.	What is Myth? Discuss about various types of software related Myths	1	2	12
5.	Define Process Model? Explain about various types of process models in detailed	1	1,2	12
6.	Discuss in detail about Process Assessment and Improvement	1	2	12
7	Explain about Prescriptive Process Models and Specialized Process Models	1	2	12
8.	Define The Unified Process, Personal and Team Process Models	1	2	12
9.	Explain Process Technology, differentiate between Product and Process	1	2	12
10.	Discuss about Agile Development in detailed	1	1	12
11.	What is Agility? Explain Agile Process	1	1,2	12
12	What is XP? Discuss about Extreme Programming(XP)	1	1	12

Assignment Questions:

S.NO	QUESTION	CO	BL	MARKS
1.	Define Software and explain Software Engineering process in detailed	1	1	12
2.	What is the nature of software? Discuss about the unique nature of WebApps	1	1,2	12
3.	Define Software Process? Explain about Software Engineering Practice?	1	1	12
4.	What is Myth? Discuss about various types of software related Myths	1	2	12
5.	Define Process Model? Explain about various types of process models in detailed	1	1,2	12
6.	Explain about Prescriptive Process Models and Specialized Process Models	1	2	12
7.	Define The Unified Process, Personal and Team Process Models	1	2	12
8.	What is Agility? Explain Agile Process	1	1,2	12