Unit-1

Chapter-1:

1) Explain about software engineering role & characteristics of a software.

Ans: Software is defined as a collection of programs, documentation and operating procedures. The **Institute of Electrical and Electronic Engineers (IEEE)** defines software as a 'collection of computer programs, procedures, rules and associated documentation and data.'

Software controls, integrates, and manages the hardware components of a computer system. It also instructs the computer what needs to be done to perform a specific task and how it is to be done. For example, software instructs the hardware how to print a document, take input from the user, and display the output.

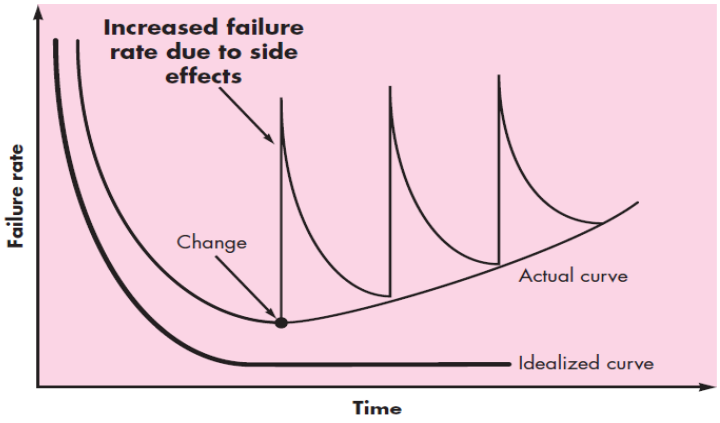
Typical formal **definitions** of **software engineering** are: "the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of **software**". "an **engineering** discipline that is concerned with all aspects of **software** production".

Software engineering role:

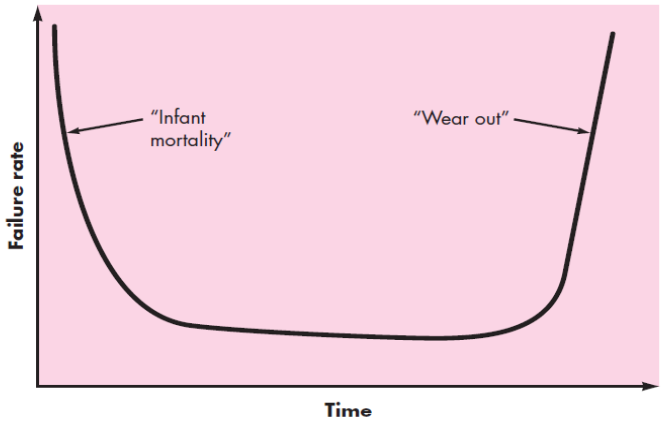
- Today, software takes on a dual role. It is a product, and at the same time, the vehicle for delivering a product. As a product, it delivers the computing potential embodied by computer hardware . It resides within a mobile phone or operates inside a mainframe computer, software is an information transformer—producing, managing, acquiring, modifying, displaying, or transmitting information.
  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.
- It 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., the Internet), and provides the means for acquiring information in all of its forms.

characteristics of a software:

- Software is developed or engineered; it is not manufactured in the classical sense.



- Software doesn't "wear out (exhaust)."



- Although the industry is moving toward component-based construction, most software continues to be custom built (made to a particular customer's order).

• **Functionality:** Refers to the degree of performance of the software against its intended purpose.

• **Reliability:** Refers to the ability of the software to provide desired functionality under the given conditions.

• **Usability:** Refers to the extent to which the software can be used with ease.

• **Efficiency:** Refers to the ability of the software to use system resources in the most effective and efficient manner.

• **Maintainability:** Refers to the ease with which the modifications can be made in a software system to extend its functionality, improve its performance, or correct errors.

• **Portability:** Refers to the ease with which software developers can transfer software from one platform to another, without (or with minimum) changes. In simple terms, it refers to the ability of

software to function properly on different hardware and software platforms without making any changes in it.

2) Explain in detail about changing nature of software.

Ans: Today, seven broad categories of computer software present continuing challenges for software engineers:

- System software
- Application software
- Engineering and scientific software
- Embedded software
- Product-line software
- Web-applications
- Artificial intelligence software

1) System software is a type of computer **program** that is designed to run a computer's hardware and application **programs.** (e.g., compilers, editors, and file management utilities).

2) Application software is a program or group of programs designed for end users. Stand-alone programs that solve a specific business need. Applications in this area process business or technical data in a way that facilitates business operations or management/technical decision making. (e.g.. Microsoft Office, Excel, Google Chrome, Mozilla Firefox and Skype**)**

3) Engineering/scientific software—has been characterized by "number crunching" algorithms. Applications range from astronomy to volcanology, from automotive stress analysis to space shuttle orbital dynamics, and from molecular biology to automated manufacturing.

4) Embedded software—resides within a product or system and is used to implement and control features and functions for the end user and for the system itself. Embedded software can perform limited and esoteric functions (e.g., key pad control for a microwave oven) or provide significant function and control capability (e.g., digital functions in an automobile such as fuel control, dashboard displays, and braking systems).

5) Product-line software—designed to provide a specific capability for use by many different customers. Product-line software can focus on a limited and esoteric marketplace (e.g., inventory control products) or address mass consumer markets (e.g., word processing, spreadsheets, computer graphics, multimedia, entertainment, database management, and personal and business financial applications).

6) Web applications—called "Web-Apps" this network-centric software category spans a wide array of applications. 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.

7) Artificial intelligence software—makes use of non-numerical algorithms to solve complex problems that are not amenable to computation or straightforward analysis. Applications within

this area include robotics, expert systems, pattern recognition (image and voice), artificial neural networks, theorem proving, and game playing.

3) Define Legacy software and explain in detail about quality of legacy software evaluation.

Ans: Legacy software systems were developed decades ago and have been continually modified to meet changes in business requirements and computing platforms. The proliferation of such systems is causing headaches for large organizations who find them costly to maintain and risky to evolve.

- Many legacy systems remain supportive to core business functions and are absolutely to business.
- Unfortunately, there is sometimes one additional characteristic that is present in legacy software—poor quality.
- Legacy systems sometimes have inextensible designs, convoluted code, poor or nonexistent documentation, test cases and results that were never archived, a poorly managed change history—the list can be quite long.
- If the legacy software meets the needs of its users and runs reliably, it isn't broken and does not need to be fixed. However, as time passes, legacy systems often evolve for one or more of the following reasons:
  - 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.

  When these modes of evolution occur, a legacy system must be reengineered so that it remains viable into the future. The goal of modern software engineering is to "devise methodologies that are founded on the notion of evolution"; that is, the notion that software systems continually change, new software systems are built from the old ones, and . . . all must interoperate and cooperate with each other".
  Regardless of its application domain, size, or complexity, computer software will evolve Over time.

Change (software maintenance) drives this process and occurs when error are corrected, when the software is adapted to a new environment, when the customer requests new features or functions, and when the application is reengineered to provide benefit in a modern context.

A Unified Theory for SW Evolution [Lehman 1997]

1. **Continuing change –** software must be continually adapted or it will become less and less satisfactory
2. **Increasing complexity –** as software is changed it becomes increasingly complex unless work is done to mitigate the complexity
3. **Relationship to organization –** the software exists within a framework of people, management, rules and goals which create a system of checks and balances which shape software evolution
4. **Invariant work rate –** over the lifetime of a system the amount of work performed on it is essentially the same as external factors beyond anyone's control drive the evolution
5. **Conservation of familiarity –** developers and users of the software must maintain mastery of its content in order to use and evolve it; excessive growth reduces mastery and acts as a brake
6. **Continuing growth –** seemingly similar to the first law, this observation states that additional growth is also driven by the resource constraints that restricted the original scope of the system
7. **Declining quality –** the quality of the software will decline unless steps are taken to keep it in accord with operational changes
8. **Feedback system –** the evolution in functionality and complexity of software is governed by a multi-loop, multi-level, multi-party feedback system

4) Explain in detail about software myths.

Ans: Software myths—erroneous beliefs about software and the process that is used to build it—can be traced to the earliest days of computing. Myths have a number of attributes that make them insidious (proceeding in a gradual, subtle way, but with very harmful effects).

Today, most knowledgeable software engineering professionals recognize myths for what they are—misleading attitudes that have caused serious problems for managers and practitioners alike. However, old attitudes and habits are difficult to modify, and remnants of software myths remain.

Managers, who own software development responsibility, are often under strain and pressure to maintain a software budget, time constraints, improved quality, and many other considerations.

Table Management Myths

The members of an organization can acquire all-the information, they require from a manual, which contains standards, procedures, and principles;	Standards are often incomplete, inadaptable, and outdated. Developers are often unaware of all the established standards. Developers rarely follow all the known standards because not all the standards tend to decrease the delivery time of software while maintaining its quality.
If the project is behind schedule, increasing the number of programmers can reduce the time gap.	Adding more manpower to the project, which is already behind schedule, further delays the project. New workers take longer to learn about the project as compared to those already working on the project.
If the project is outsourced to a third party, the management can relax and let the other firm develop software for them.	Outsourcing software to a third party does not help the organization, which is incompetent in managing and controlling the software project internally. The organization invariably suffers when it out sources the software project.

Customer myths: A customer who requests computer software may be a person at the next desk, a technical group down the hall, the marketing/sales department or an outside company that has requested software under contract. In many cases, the customer believes myths about software because software managers and practitioners do little to correct misinformation. Myths lead to false expectations (by the customer) and, ultimately, dissatisfaction with the developer.

Table User Myths

Brief requirement stated in the initial process is enough to start development; detailed requirements can be added at the later stages.	Starting development with incomplete and ambiguous requirements often lead to software failure. Instead, a complete and formal description of requirements is essential before starting development. Adding requirements at a later stage often requires repeating the entire development process.

Software is flexible; hence software requirement changes can be added during any phase of the development process.	Incorporating change requests earlier in the development process costs lesser than those that occurs at later stages. This is because incorporating changes later may require redesigning and extra resources.

In the early days of software development, programming was viewed as an art, but now software development has gradually become an engineering discipline. However, developers still believe in some myths-. Some of the common developer myths are listed in Table.

Table Developer Myths

Software development is considered complete when the code is delivered.	50% to 70% of all the efforts are expended after the software is delivered to the user.
The success of a software project depends on the quality of the product produced.	The quality of programs is not the only factor that makes the project successful instead the documentation and software configuration also play a crucial role.
Software engineering requires unnecessary documentation, which slows down the project.	Software engineering is about creating quality at every level of the software project. Proper documentation enhances quality which results in reducing the amount of rework.
The only product that is delivered after the completion of a project is the working program(s).	The deliverables of a successful project includes not only the working program but also the documentation to guide the users for using the software.

2nd chapter

1) Explain in detail about Software Engineering layered technology

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

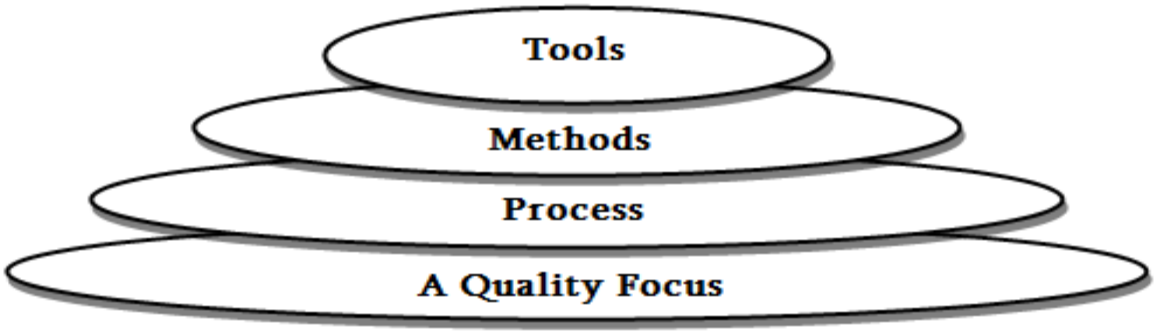


Figure: Flowchart of the Layers of Software Development

01. A Quality Focus: Software engineering must rest on an organizational commitment to quality. Total quality management and similar philosophies foster a continuous process improvement culture, and this culture ultimately leads to the development of increasingly more mature approaches to software engineering. The bedrock that supports software engineering is a quality focus.

02. Process: The foundation for software engineering is the process layer. Process defines a framework for a set of Key Process Areas (KPAs) that must be established for effective delivery of software engineering technology. This establishes the context in which technical methods are applied, work products such as models, documents, data, reports, forms, etc. are produced, milestones are established, quality is ensured, and change is properly managed.

03. Methods: Software engineering *methods* provide the technical how-to's for building software. Methods will include requirements analysis, design, program construction, testing, and support. This relies on a set of basic principles that govern each area of the technology and include modeling activities and other descriptive techniques.

04. Tools: Software engineering *tools* provide automated or semi-automated support for the process and the methods. When tools are integrated so that information created by one tool can be used by another, a system for the support of software development, called computer-aided software engineering, is established.

2) Define Process Framework. Discuss in detail about 5 Generic process framework activities.

Ans:

Software Process

Process defines a framework for a set of Key Process Areas (KPAs) that must be established for effective delivery of Software engineering technology. This establishes the context in which technical methods are applied, work products such as models, documents, data, reports, forms, etc. are produced, milestones are established, quality is ensured, and change is properly managed.

Software Process Framework:

A process framework establishes the foundation for a complete **software process** by identifying a small **number of framework activities** that are applicable to all software projects, regardless of size or complexity. It also includes a set of **Umbrella activities** that are applicable across the entire software process. Some most applicable framework activities are described below.

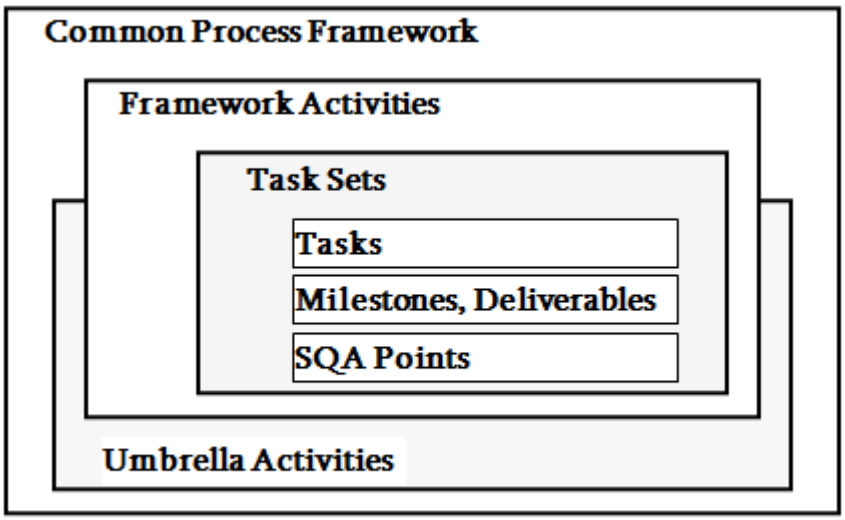


Figure: Chart of Process Framework

Communication:

This activity involves heavy communication with customers and other stakeholders in order to gather requirements and other related activities

Planning:

Here a plan to be followed will be created which will describe the technical tasks to be conducted, risks, required resources, work schedule etc.

Modeling:

A model will be created to better understand the requirements and design to achieve these requirements.

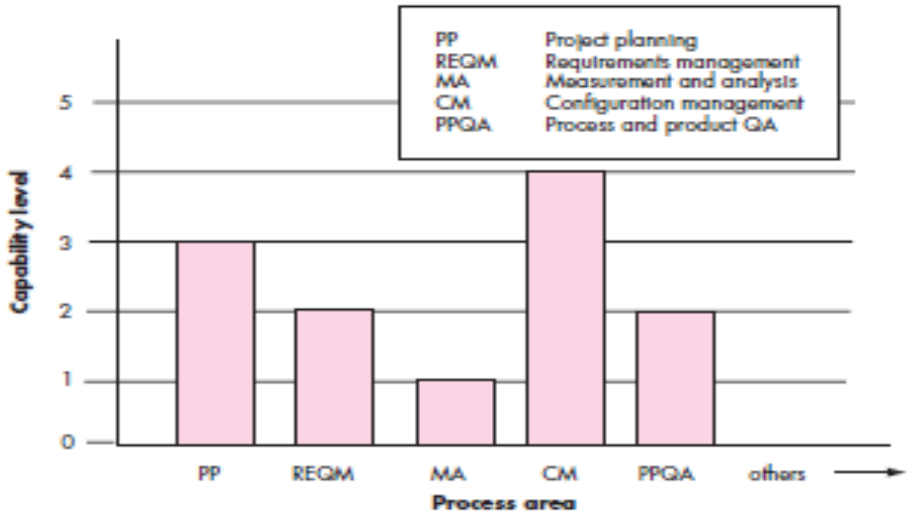
Construction:
Here the code will be generated and tested.

Deployment:
Here, a complete or partially complete version of the software is represented to the customers to evaluate and they give feedbacks based on the evaluation.

These above described five activities can be used in any kind of software development. The details of the software development process may become a little different, but the framework activities remain the same.

3) Explain about CMMI model.

Ans: **CMMI (The Capability Maturity Model Integration):**
The Software Engineering Institute (SEI) has developed a comprehensive model predicated on a set of software engineering capabilities that should be present as organizations reach different levels of process maturity. To determine an organization's current state of process maturity the CMMI uses two ways, (i) as a continuous model and (ii) as a staged model. In the case of a continuous model each process area is rated according to the following capability levels.



Level 0 : Incomplete:
The process are is either not performed or does not achieve all goals and objectives defined by the CMMI for level 1 capability.

Level 1: Performed:
All the specific goals has been achieved and work tasks required to produce defined products has been conducted.

Level 2: Managed:
All level 1 criteria have been achieved. All work associated with the process area conforms to an organizationally defined policy, all people are doing their jobs, stakeholders are actively involved in the process area, all work tasks and work products are monitored, controlled and reviewed.

Level 3: Defined:
All level 2 criteria have been achieved. In addition, the process is "tailored from the organization's set of standard processes according to the organization's tailoring guidelines and contributes work products, measures and other process improvement information to the organizational process assets."

Level 4: Quantitatively Managed:
All level 3 criteria have been achieved. In addition, the process area is controlled and improved using measurement and quantitative assessment. "Quantitative objectives for quality and process performance are established and used as a criteria in managing the process."

Level 5: Optimized:
All capability level 4 criteria have been achieved. In addition, the process area is adapted and optimized using quantitative means to meet changing customer needs and to continually improve the efficacy of the process area under consideration.

The staged CMMI model defines the same process areas, goals and practices as the continuous model. The primary difference is that the staged model defines five maturity levels rather than five capability levels.

Process areas require to achieve maturity levels has been showed in the table below :

Level	Focus	Process Areas
Optimizing	*Continuous process improvement*	Organizational innovation and deployment Causal analysis and resolution
Quantitatively managed	*Quantitative management*	Organizational process performance Quantitative project management
Defined	*Process standardization*	Requirements development Technical solution Product integration Verification Validation Organizational process focus Organizational process definition Organizational training Integrated project management Integrated supplier management Risk management Decision analysis and resolution Organizational environment for integration Integrated teaming
Managed	*Basic project management*	Requirements management Project planning Project monitoring and control Supplier agreement management Measurement and analysis Process and product quality assurance Configuration management
Performed		

4) Define Process Pattern & Explain about various forms of Process patterns.

Ans: The software process can be defined as a collection of patterns that define a set of activities, actions, work tasks, work products and/or related behaviors required to develop computer software.

Patterns can be defined at any level of abstraction. In some cases, a pattern might be used to describe a complete process (e.g., prototyping). In other situations, patterns can be used to describe an important framework activity (e.g., planning) or a task within framework activity (e.g., project-estimating). Ambler has proposed the following template for describing a process pattern. **Pattern Name.** The pattern is given a meaningful name that describes its functions within the software-process-(e.g.,customer-communication).

Intent. The objective of the pattern is described briefly. For example, the intent of customer-communication is "to establish a collaborative relationship with the customer in an effort to define project-scope,business-requirements,and-other_projectsconstraints".

Type.: The_pattern-type-is-specified.Ambler_suggests-three-types:

• Task pattern define a software engineering action or work task that is part of the process and relevant to successful software engineering practice (e.g., requirements gathering is a task pattern).

• Stage patterns represent a framework activity for the process. Since a framework activity encompasses multiple work tasks, a stage pattern incorporates multiple task patterns that are relevant to the stage. An example of a stage pattern might be communication. This pattern would incorporate-the-task_pattern_requirements_gathering_and_others.

• Phase patterns define the sequence of framework activities is iterative in nature. An example of aphase-pattern-might_be_a_spiral_model(or)prototyping.

Initial context. The conditions under which the pattern applies are described. Prior to the initiation of the pattern, we ask (1) what organizational team created activities have already occurred (2) what is the entry state for the process? And (3) what software engineering information or project information_already_exists?

For example, the planning pattern (a stage pattern) requires that (1) customers and software engineers have established a collaborative communication: (2) successful completion of a number of task patterns (specified) for the customer-communication pattern has occurred: and (3) project scope, basic business requirements, and project constraints are known.

An_Example-Process_Pattern:

The following abbreviated process pattern describes an approach that may be applicable when stakeholders have a general idea of what must be done, but are unsure of specific software requirements.

Pattern-name.Prototyping.

Intent: The objective of the pattern is to build a model (a prototype) that can be assessed iteratively by stakeholders in an effort to identify or solidity software requirements.

Type: Phase-pattern.

Initial context: The following conditions must be met prior to the initiation of this pattern:(1) stakeholders have been identified; (2) a mode of communication between stakeholders and the software team has been established; (3) the overriding problem to be solved has been identified by stakeholders: (4) an initial understanding of project scope, basic business requirements, and project constraints-has-been_developed.

Problem: Requirements are hazy or nonexistent, yet there is clear recognition that there is a problem, and the problem must be addressed with a software solution. Stakeholders are unsure of what they want; that is, they cannot describe software requirements in any detail.

Solutions: A description of the prototype that identifies basic requirements (e.g., modes of interaction, computational features, processing functions) is approved by stakeholders. Following this, (1) the prototype may evolve through a series of increments to become the production software or (2) the prototype may be discarded and the production software built using some other process-pattern.

Related Patterns: The following patterns are related to this pattern: **customer-communication; iterative design; iterative development, customer assessment; requirement extraction**.

Known uses/examples: Prototyping is recommended when requirements are uncertain.

5) Explain in detail about personal & Team process models and describe Process assessment.

Ans: The PSP and TSP both use proven techniques to improve individual performance and team in software development. Used together they help reduce costs, produce defect-free and inside deadline by empowering development teams. These technologies are based on the premise that a defined and structured process can improve individual work quality and efficiency. PSP focus is on individuals and TSP focus is on teams.

- The **Personal Software Process** (PSP) provides engineers with a disciplined personal framework for doing software work. The PSP process consists of a set of methods, forms, and scripts that show software engineers how to plan, measure, and manage their work.

  The PSP model defines five framework activities:

  Planning. This activity isolates requirements and develops both size and resource estimates. In addition, a defect 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 (this is a substantial amount of data that should be analyzed statistically), the effectiveness of the process is determined. Measures and metrics should provide guidance for modifying the process to improve its effectiveness.

- The **Team Software Process** (TSP) guides engineering teams that are developing software-intensive products. Using TSP helps organizations establish a mature and disciplined engineering practice that produces secure, reliable software in less time and at lower costs.

  The TSP was introduced in 1998, and builds upon the foundation of PSP to enable engineering teams to build software-intensive products more predictably and effectively. The goal of TSP is to build a "self-directed" project team that organizes itself to produce high-quality software. Humphrey [Hum98] 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 5behavior normal and expected.

  • Provide improvement guidance to high-maturity organizations.

  • Facilitate university teaching of industrial-grade team skills.

  TSP defines the following framework activities: project launch, high-level design, implementation, integration and test, and postmortem.

- The **process assessment** is to ensure that it meets a set of basic process criteria that have been shown to be essential for a successful software engineering. A number of different approaches to software process assessment have been proposed over the past few decades:

  1. Standard CMMI Assessment Method for Process Improvement (SCAMPI) provides a five—step process assessment model that incorporates initiating, diagnosing, establishing, acting, and learning. The SCAMPI methods use the SEI CMMI as-the-basis-for-assessment.

  i. **SPICE (ISO/IEC 15504)** Standard defines a set of requirements for software process assessment. The intent of the standard is to assist organization in developing an objective evaluation of the efficacy of any defined software process.

  ii. **ISO9001: 2000 for Software** is a generic standard that applies to any organization that wants to improve overall quality of the products, systems, or services that it provides. Therefore, the standard is directly applicable to software organizations and companies.

Chapter-3:

1) Discuss in detail about prescriptive process models.

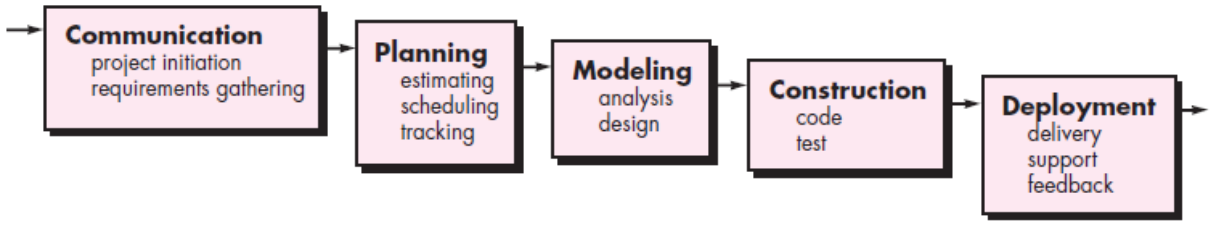
Ans: The following framework activities are carried out irrespective of the process model chosen by-the-organization.
1.Communication
2.Planning
3.Modeling
4.Construction
5.Deployment
The name 'prescriptive' is given because the model prescribes a set of activities, actions, tasks, quality assurance and change the mechanism for every project.

There are three types of prescriptive process models. They are:
1.The-Waterfall-Model
2.Incremental-Process-model
3. RAD model

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.



NOTE: The description of the phases of the waterfall model is same as that of the process model.

An alternative design for 'linear sequential model' is as follows:

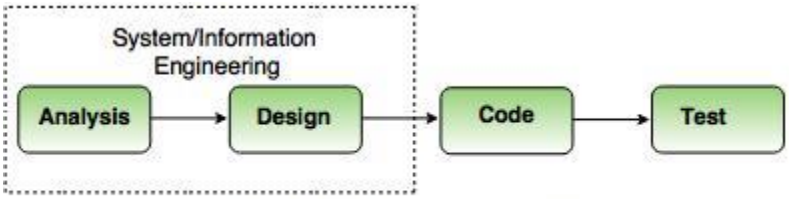


Fig. - The linear sequential model

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. Incremental Process 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.

For example, the word-processing software is developed using the incremental model.

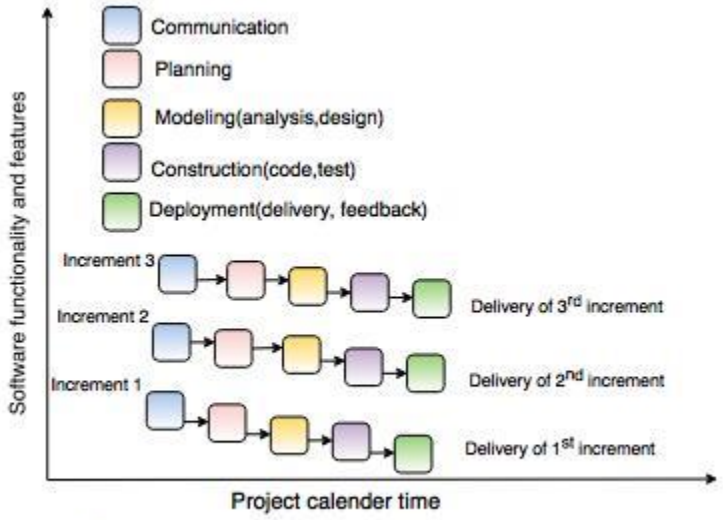


Fig. - Incremental Process Model

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.

3. RAD model

- RAD is a Rapid Application Development model.
- Using the RAD model, software product is developed in a short period of time.
- The initial activity starts with the communication between customer and developer.
- Planning depends upon the initial requirements and then the requirements are divided into groups.
- Planning is more important to work together on different modules.

The-RAD-model-consist-of-following-phases:

1. Business Modeling

- Business modeling consist of the flow of information between various functions in the project.
- For example what type of information is produced by every function and which are the functions to handle that information.
- A complete business analysis should be performed to get the essential business information.

2. Data modeling

- The information in the business modeling phase is refined into the set of objects and it is essential for the business.
- The attributes of each object are identified and define the relationship between objects.

3. Process modeling

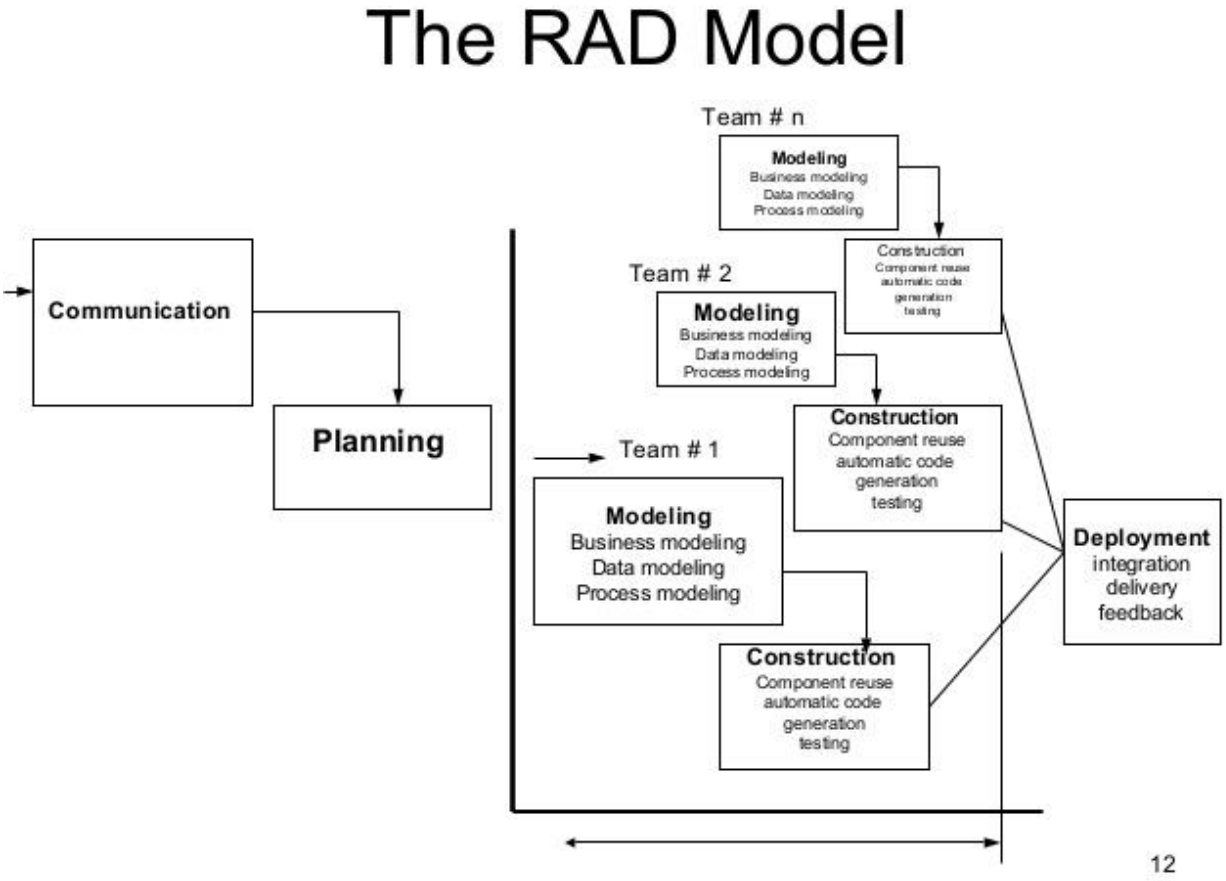
- The data objects defined in the data modeling phase are changed to fulfil the information flow to implement the business model.
- The process description is created for adding, modifying, deleting or retrieving a data object.

4. Application generation

- In the application generation phase, the actual system is built.
- To construct the software the automated tools are used.

5. Testing and turnover

- The prototypes are independently tested after each iteration so that the overall testing time is reduced.
- The data flow and the interfaces between all the components are are fully tested. Hence, most of the programming components are already tested.



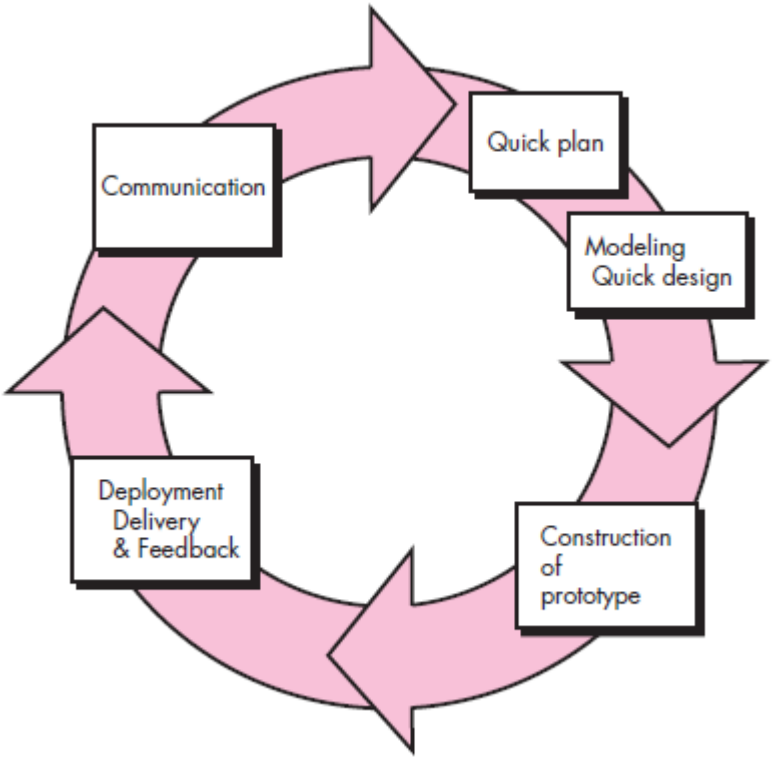
2) Explain about evolutionary process models with a neat diagram.

Ans: Evolutionary software models are iterative. They are characterized in manner that enables the software engineers to develop increasingly more complete version of a software. That is, initially a rapid version of the product is being developed and then the product is developed to more accurate version with the help of the reviewers who review the product after each release and submit improvements. Specification, development and validation are interleaved rather than separate in evolutionary software process model.

These models are applied because as the requirements often change so the end product will be unrealistic, where a complete version is impossible due to tight market deadlines it is better to introduce a limited version to meet the pressure. Thus the software engineers can follow a process

model that has been explicitly designed to accommodate a product that gradually complete over time.

- **Prototyping Model** : Often, a customer defines a set of general objectives for software but does not identify detailed input, processing, or output requirements. In other cases, the developer may be unsure of the efficiency of an algorithm, the adaptability of an operating system, or the form that human/machine interaction should take. In these, and many other situations, a prototyping paradigm may offer the best approach.



(i) **Communication** : Firstly, developer and customer meet and define the overall objectives, requirements, and outline areas where further definition is mandatory.

(ii) **Quick Plan :** Based on the requirements and others of the communication part a quick plane is made to design the software.

(iii) **Modeling Quick Design** : Based on the quick plane, 'A Quick Design' occurs. The quick design focuses on a representation of those aspects of the software that will be visible to the customer/user, such as input approaches and output formats.

(iv) **Construction of Prototype** : The quick design leads to the construction of a prototype.

(v) **Deployment, delivery and feedback** : The prototype is evaluated by the customer/user and used to refine requirements for the software to be developed. All these steps are repeated to tune

the prototype to satisfy user's need. At the same time enable the developer to better understand what needs to be done.

Problems With Prototype Model :

(i) In the rush to get the software working the overall software quality or long-term maintainability will not get consideration. So software, in that way, gets the need of excessive modification to ensure quality.

(ii) The developer may choose inappropriate operating system, algorithms, language in the rush to make the prototype working.

Prototyping is an effective paradigm for software engineering. It necessary to build the prototype to define requirements and then to engineer the software with a eye toward quality.

- **Spiral Model** : The spiral model is an evolutionary software process model that combines the iterative nature of prototyping with the controlled and systematic aspects of the linear sequential model. Using the spiral model, software is developed in a series of incremental releases. During early iterations, the incremental release might be a paper model or prototype. During later iterations, increasingly more complete versions of the engineered system are produced.

A spiral model is divided into a number of framework activities, also called task regions. Typically, there are between three and six task regions. Given figure is of a spiral model that contains five task regions.

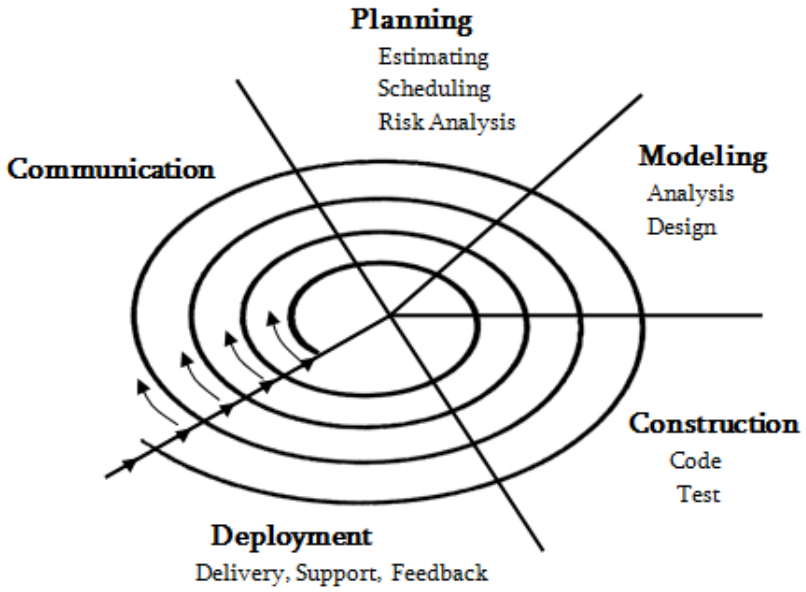


Figure : Spiral Model

model that contains five task regions.

(i) Customer communication — Tasks required to establish effective communication between developer and customer.

(ii) Planning — Tasks required to define resources, timelines, and other project related information.

(iii) Modeling — Tasks required in building one or more representations of the application.

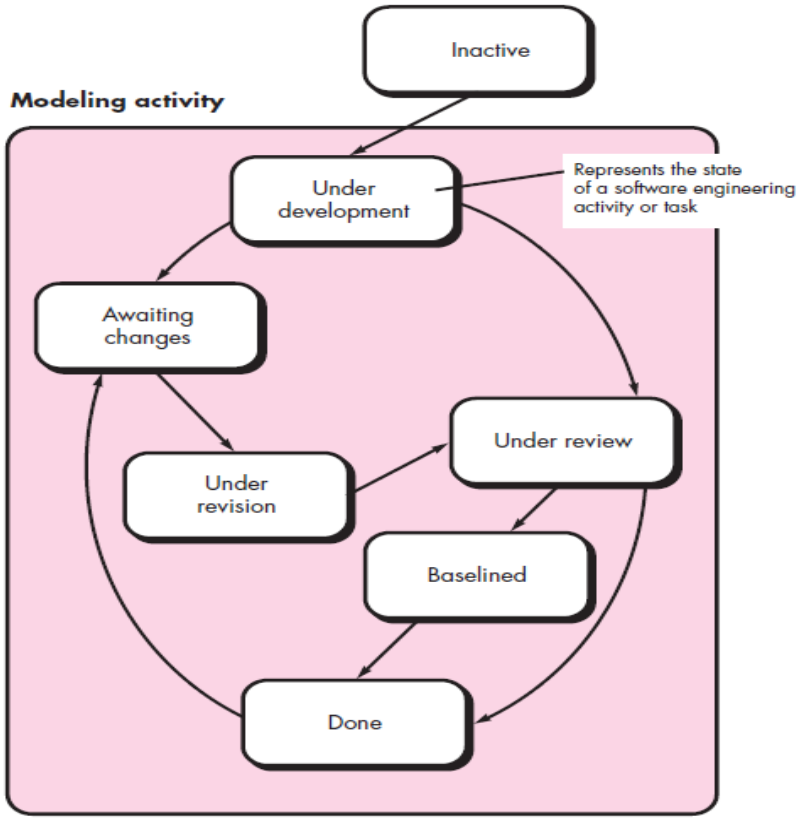
(iv) Construction and release — Tasks required to construct, test, install.

(v) Deployment — Tasks required to deliver the software, getting feedbacks etc.

Software engineering team moves around the spiral in a clockwise direction, beginning at the center. The first circuit around the spiral might result in the development of a product specification; subsequent passes around the spiral might be used to develop a prototype and then progressively more sophisticated versions of the software. Each pass through the planning region results in adjustments to the project plan. Cost and schedule are adjusted based on feedback derived from customer evaluation. In addition, the project manager adjusts the planned number of iterations required to complete the software.

The spiral model is a realistic approach to the development of large-scale systems and software. The spiral model enables the developer to apply the prototyping approach at any stage in the evolution of the product. The spiral model demands a direct consideration of technical risks at all stages of the project and, if properly applied, should reduce risks before they become problematic. It demands considerable risk assessment expertise and relies on this expertise for success.

- **Concurrent process model:**



The concurrent development model, sometimes called concurrent engineering, allows

a software team to represent iterative and concurrent elements of any of the process models.For example, the modeling activity defined for the spiral model is accomplished by invoking one or more of the following software engineering actions: prototyping, analysis, and design

Figure provides a schematic representation of one software engineering activity within the modeling activity using a concurrent modeling approach. The activity—modeling—may be in any one of the states noted at any given time. Similarly, other activities, actions, or tasks (e.g., communication or construction) can be represented in an analogous manner. All software engineering activities exist concurrently but reside in different states.

For example, early in a project the communication activity (not shown in the figure) has completed its first iteration and exists in the awaiting changes state. The modeling activity (which existed in the inactive state while initial communication was completed, now makes a transition into the under development state. If, however, the customer indicates that changes in requirements must be made, the modeling activity moves from the under development state into the awaiting changes state.

Concurrent modeling defines a series of events that will trigger transitions from state to state for each of the software engineering activities, actions, or tasks. For example, during early stages of

design (a major software engineering action that occurs during the modeling activity), an inconsistency in the requirements model is uncovered. This generates the event analysis model correction, which will trigger the requirements analysis action from the done state into the awaiting changes state.

Concurrent modeling is applicable to all types of software development and provides an accurate picture of the current state of a project. Rather than confining software engineering activities, actions, and tasks to a sequence of events, it defines a process network. Each activity, action, or task on the network exists simultaneously with other activities, actions, or tasks. Events generated at one point in the process network trigger transitions among the states.

3) Explain in detail about specialized process models.

Ans: 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.

- **Component based model:**

The component-based development (CBD) model incorporates many of the characteristics of the spiral model. It is evolutionary in nature, demanding an iterative approach to the creation of software. However, the component-based development model composes applications from prepackaged software components, called classes.

Modeling and construction activities begin with the identification of candidate components. Regardless of the technology that has been used to create the components, the model incorporates the following steps

(i) Available component-based products are researched and evaluated for the application domain in question.

(ii) Component integration issues are considered.

(**iii)** A software architecture is designed to accommodate the components.

(iv) Components are integrated into the architecture.

(v) Comprehensive testing is conducted to ensure proper functionality.
The component-based development model leads to software reuse, and re-usability provides software engineers with a number of measurable benefits.

- **Formal model:**

The formal methods model encompasses a set of activities that leads to formal mathematical specification of computer software. Formal methods enable a software engineer to specify, develop, and verify a computer-based system by applying a rigorous, mathematical notation.

When formal methods are used during development, they provide a mechanism for eliminating many of the problems that are difficult to overcome using other software engineering paradigms. Uncertainty, incompleteness, and inconsistency can be discovered and corrected more easily through the application of mathematical analysis. When formal methods are used during design, they serve as a basis for program verification and therefore enable the software engineer to discover and correct errors that might go undetected.

The concerns raised with the formal model are:

• The development of formal models 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.

- **Aspect oriented process model:**

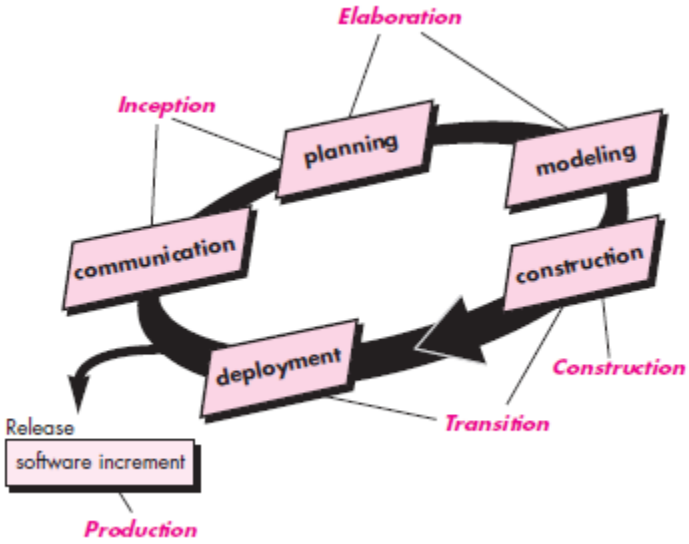
  Regardless of the software process that is chosen, the builders of complex software invariably implement a set of localized features, functions and information content. These localized software characteristics are modeled as components and then constructed within the context of a system architecture. As modern computer-based systems become more sophisticated and complex, certain concerns, customer required properties or areas of technical interest, span the entire architecture. Some concerns are high-level properties of a system; others affect functions or are systemic. When concerns cut across multiple system functions, features, and information they are often referred to as crosscutting concerns. Aspectual requirements define those crosscutting concerns that have impact across the software architecture. Aspects are mechanisms beyond subroutines and inheritance for localizing the expression of a crosscutting concerns. Aspect-oriented software development (AOSD), often referred to as aspect-oriented programming (AOP), is a relatively new software engineering paradigm that provides a process and methodological approach for defining, specifying, designing, and constructing aspects. A distinct aspect-oriented process has not yet matured. However, it is likely that such a process will adopt characteristics of both the spiral and concurrent process models. The evolutionary nature of the spiral is appropriate as aspects are identified and then constructed. The parallel nature of concurrent development is essential because aspects are engineered independently of localized software components and yet, aspects have a direct impact on these components.

4) Explain in detail about unified process model and write all of its 8 phases with a neat diagram.

Ans:

In some ways the unified process (UP) is an attempt to draw on the best features and characteristics of conventional 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.



The above dig represents the unified process model. The life of a software system can be represented as a series of cycles. A cycle ends with the release of a version of the system to customers.

Within the Unified Process, each cycle contains four phases. A phase is simply the span of time between two major milestones, points at which managers make important decisions about whether to proceed with development and, if so, what's required concerning project scope, budget, and schedule.

Inception phase:

The primary goal of the Inception phase is to establish the case for the viability of the proposed system. The tasks that a project team performs during Inception include the following:

- Defining the scope of the system (that is, what's in and what's out)
- Outlining a candidate architecture, which is made up of initial versions of six different models
- Identifying critical risks and determining when and how the project will address them
- Starting to make the business case that the project is worth doing, based on initial estimates of cost, effort, schedule, and product quality

Elaboration phase:

The primary goal of the Elaboration phase is to establish the ability to build the new system given the financial constraints, schedule constraints, and other kinds of constraints that the development project faces.

The tasks that a project team performs during Elaboration include the following:

- Capturing a healthy majority of the remaining functional requirements
- Expanding the candidate architecture into a full architectural baseline, which is an internal release of the system focused on describing the architecture
- Addressing significant risks on an ongoing basis
- Finalizing the business case for the project and preparing a project plan that contains sufficient detail to guide the next phase of the project (Construction)

Construction

The primary goal of the **Construction phase** is to build a system capable of operating successfully in beta customer environments.

During Construction, the project team performs tasks that involve building the system iteratively and incrementally (see "Iterations and Increments" later in this chapter), making sure that the viability of the system is always evident in executable form.

The major milestone associated with the Construction phase is called **Initial Operational Capability**. The project has reached this milestone if a set of beta customers has a more or less fully operational system in their hands.

Transition

The primary goal of the **Transition phase** is to roll out the fully functional system to customers. During Transition, the project team focuses on correcting defects and modifying the system to correct previously unidentified problems. The major milestone associated with the Transition phase is called **Product Release**.

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

Chapter-3

AGILE process models:

AGILE is a methodology that promotes **continuous iteration** of development and testing throughout the software development life cycle of the project. Both development and testing activities are concurrent unlike the Waterfall model

The agile software development emphasizes on four core values.

1. Individual and team interactions over processes and tools
2. Working software over comprehensive documentation
3. Customer collaboration over contract negotiation
4. Responding to change over following a plan

The Agile Alliance defines **12 agility principles** for those who want to achieve agility:

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.

Human Factors

Proponents of agile software development take great pains to emphasize the importance of "people factors." As Cockburn and High-smith [Coc01a] state, "Agile development focuses on the talents and skills of individuals, molding the process to specific people and teams." The key point in this statement is that the process molds to the needs of the people and team, not the other way around.2

If members of the software team are to drive the characteristics of the process that is applied to build software, a number of key traits must exist among the people on an agile team and the team itself:

Competence. In an agile development (as well as software engineering) context, "competence" encompasses innate talent, specific software-related skills, and overall knowledge of the process that the team has chosen to apply. Skill and knowledge of process can and should be taught to all people who serve as agile team members.

Common focus. Although members of the agile team may perform different tasks and bring different skills to the project, all should be focused on one goal—to deliver a working software increment to the customer within the time promised. To achieve this goal, the team will also focus on continual adaptations (small and large) that will make the process fit the needs of the team.

Collaboration. Software engineering (regardless of process) is about assessing, analyzing, and using information that is communicated to the software team; creating information that will help all stakeholders understand the work of the team; and building information (computer software and relevant databases) that provides business value for the customer. To accomplish these tasks, team members must collaborate—with one another and all other stakeholders.

Decision-making ability. Any good software team (including agile teams) must be allowed the freedom to control its own destiny. This implies that the team is given autonomy—decision-making authority for both technical and project issues.

Fuzzy problem-solving ability. Software managers must recognize that the agile team will continually have to deal with ambiguity and will continually be buffeted by change. In some cases, the team must accept the fact that the problem they are solving today may not be the problem that needs to be solved tomorrow. However, lessons learned from any problem-solving activity (including those that solve the wrong problem) may be of benefit to the team later in the project.

Mutual trust and respect. The agile team must become what De-Marco and Lister [DeM98] call a "jelled" team (Chapter 24). A jelled team exhibits the trust and respect that are necessary to make them "so strongly knit that the whole is greater than the sum of the parts." [DeM98]

Self-organization. In the context of agile development, self-organization implies three things: (1) the agile team organizes itself for the work to be done, (2) the team organizes the process to best accommodate its local environment, (3)the team organizes the work schedule to best achieve delivery of the software increment. Self-organization has a number of technical benefits, but more importantly, it serves to improve collaboration and boost team morale. In essence, the team serves as its own management

Agile process models:

-> XP model (Extreme programming model):

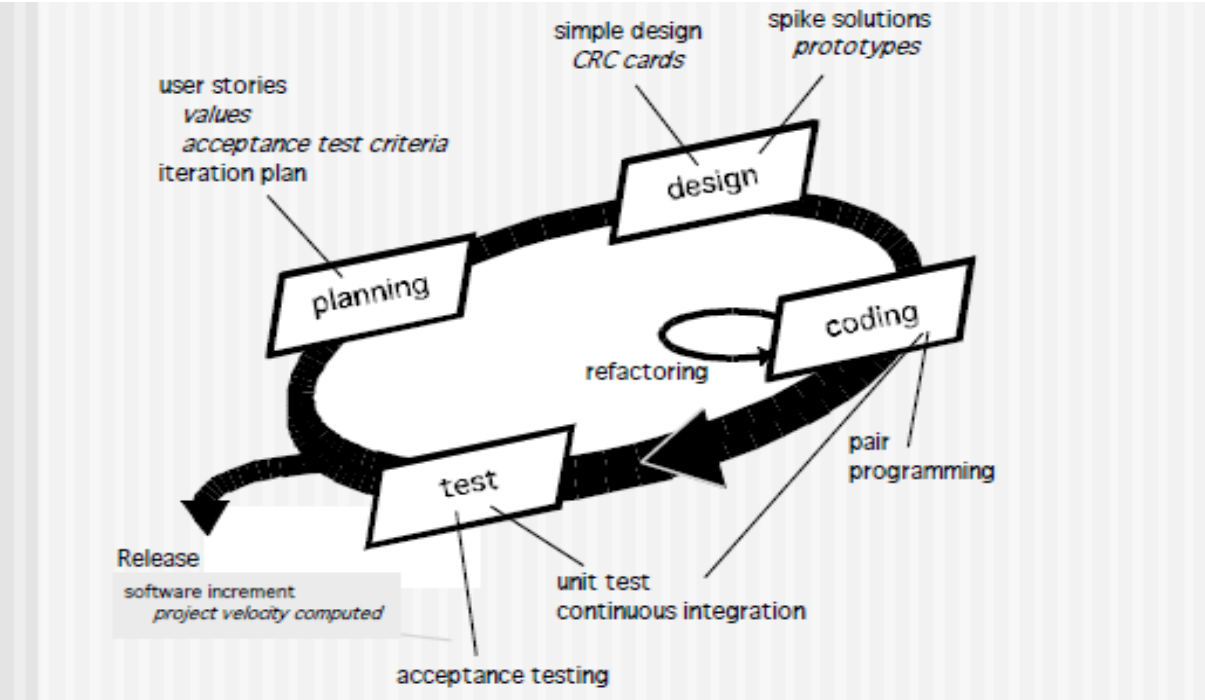
The most widely used agile process, originally proposed by Kent Beck in 2004. It uses an object-oriented approach. !

 XP Planning!

 Begins with the listening, leads to creation of "user stories" that describes required output, features, and functionality. Customer assigns a value (i.e., a priority) to each story. !

 Agile team assesses each story and assigns a cost (development weeks. If more than 3 weeks, customer asked to split into smaller stories)!

Working together, stories are grouped for a deliverable increment next release. !



 A commitment (stories to be included, delivery date and other project matters) is made. Three ways: 1. Either all stories will be implemented in a few weeks, 2. High priority stories first, or 3. the riskiest stories will be implemented first. !

 After the first increment "project velocity", namely number of stories implemented during the first release is used to help define subsequent delivery dates for other increments. Customers can add stories, delete existing stories, change values of an existing story, split stories as development work proceeds.

 XP Design (occurs both before and after coding as refactoring is encouraged)!

 Follows the KIS principle (keep it simple) Nothing more nothing less than the story. !

 Encourage the use of CRC (class-responsibility-collaborator) cards in an object-oriented context. The only design work product of XP. They identify and organize the classes that are relevant to the current software increment.

 For difficult design problems, suggests the creation of "spike solutions"—a design prototype for that portion is implemented and evaluated. !

 Encourages "refactoring"—an iterative refinement of the internal program design. Does not alter the external behavior yet improve the internal structure. Minimize chances of bugs. More efficient, easy to read. !

 XP Coding!

 Recommends the construction of a unit test for a story before coding commences. So implementer can focus on what must be implemented to pass the test. !

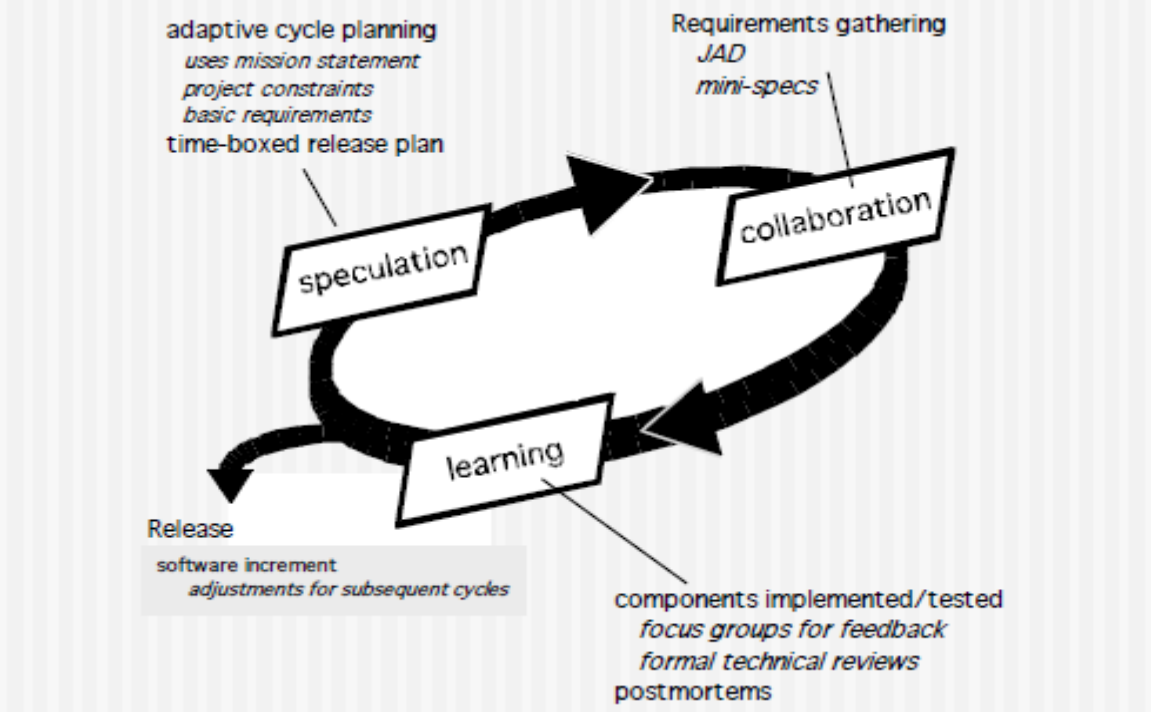
 Encourages "pair programming". Two people work together at one workstation. Real time problem solving, real time review for quality assurance. Take slightly different roles. !

 XP Testing!

 All unit tests are executed daily and ideally should be automated. Regression tests are conducted to test current and previous components. !

 "Acceptance tests" are defined by the customer and executed to assess customer visible functionality!

-> Adaptive software development:



 Originally proposed by Jim High-smith (2000) focusing on human collaboration and team self-organization as a technique to build complex software and system. !

 ASD — distinguishing features!

 Mission-driven planning!

 Component-based focus!

 Explicit consideration of risks!

 Emphasizes collaboration for requirements gathering!

 Emphasizes “learning” throughout the process

Three phase of ASD are:

 1. **Speculation:** project is initiated and adaptive cycle planning is conducted. Adaptive cycle planning uses project initiation information- the customer’s mission statement, project constraints (e.g. delivery date), and basic requirements to define the set of release cycles (increments) that will be required for the project. Based on the information obtained at the completion of the first cycle, the plan is reviewed and adjusted so that planned work better fits the reality.

 2. **Collaborations** are used to multiply their talent and creative output beyond absolute number (1+1>2). It encompasses communication and teamwork, but it also emphasizes individualism, because individual creativity plays an important role in collaborative thinking. !

 It is a matter of trust. 1)criticize without animosity, 2) assist without resentments, 3) work as hard as or harder than they do. 4)have the skill set to contribute to the work at hand, 5) communicate problems or concerns in a way that leads to effective action. !

 3. **Learning:** As members of ASD team begin to develop the components, the emphasis is on “learning”. High-smith argues that software developers often overestimate their own understanding of the technology, the process, and the project and that learning will help them to improve their level of real understanding. Three ways: focus groups, technical reviews and project postmortems.