1) **Explain evolving role of software.**
   - Today, software takes on a dual role. It is a product and, at the same time, the vehicle for delivering a product.
   - As a product, it delivers the computing potential embodied by computer hardware or, more broadly, a network of computers that are accessible by local hardware.
   - Whether it resides within a cellular phone or operates inside a mainframe computer, software is information transformer—producing, managing, acquiring, modifying, displaying, or transmitting information that can be as simple as a single bit or as complex as a multimedia presentation.
   - As the vehicle used to deliver the product, software acts as the basis for the control of the computer (operating Systems), the communication of information (networks), and the creation and control of other programs (software tools and environments).

2) **What is Software? Explain Software characteristics.**
   - Software is (1) instructions (computer programs) that when executed provide desired function and performance, (2) data structures that enable the programs to adequately manipulate information, and (3) documents that describe the operation and use of the programs.

2.1 **Software is developed or engineered; it is not manufactured in the classical sense.**
   - Although some similarities exist between software development and hardware manufacture, the two activities are fundamentally different.
   - In both activities, high quality is achieved through good design, but the manufacturing phase for hardware can introduce quality problems that are nonexistent (or easily corrected) for software.
   - Both activities are dependent on people, but the relationship between people applied and work accomplished is entirely different.
   - Software costs are concentrated in engineering. This means that software projects cannot be managed as if they were manufacturing projects.

2.2 **Software costs are concentrated in engineering. This means that software projects cannot be managed as if they were manufacturing projects.**

![Failure Rate vs. Time Graph](image)

- Figure depicts failure rate as a function of time for hardware. The relationship, often called the "bathtub curve," indicates that hardware exhibits relatively high failure rates early in its life (these failures are often attributable to design or manufacturing defects); defects are corrected and the failure rate drops to a steady-state level (ideally, quite low) for some period of time.
- As time passes, however, the failure rate rises again as hardware components suffer from the

---

Prepared By: Chirag Patel
cumulative affects of dust, vibration, abuse, temperature extremes, and many other environmental maladies.

- Stated simply, the hardware begins to wear out. Software is not susceptible to the environmental maladies that cause hardware to wear out.

![Failure Rate Curve]

- In theory, therefore, the failure rate curve for software should take the form of the “idealized curve” shown in the figure.
- Undiscovered defects will cause high failure rates early in the life of a program. However, these are corrected (ideally, without introducing other errors) and the curve flattens as shown.
- The idealized curve is a gross oversimplification of actual failure models for software. However, the implication is clear—software doesn't wear out. But it does deteriorate! This seeming contradiction can best be explained by considering the “actual curve” shown in Figure.
- During its life, software will undergo change (maintenance). As changes are made, it is likely that some new defects will be introduced, causing the failure rate curve to spike as shown in Figure.
- Before the curve can return to the original steady-state failure rate, another change is requested, causing the curve to spike again. Slowly, the minimum failure rate level begins to rise—the software is deteriorating due to change.

3.3 Although the industry is moving toward component-based assembly, most software continues to be custom built.

- Consider the manner in which the control hardware for a computer-based product is designed and built.
- The reusable components have been created so that the engineer can concentrate on the truly innovative elements of a design, that is, the parts of the design that represent something new.
- In the hardware world, component reuse is a natural part of the engineering process.
- In the software world, it is something that has only begun to be achieved on a broad scale. A software component should be designed and implemented so that it can be reused in many different programs.

3) Explain Software applications.

1. System software:

- System software is a collection of programs written to service other programs. Some system software process complex, but determinate, information structures. Other systems applications (e.g., operating system components, drivers, telecommunications processors) process largely indeterminate data.
2. **Real-time software:**
   - Software that monitors/analyzes/controls real-world events as they occur is called real time.
   - Elements of real-time software include a data gathering component that collects and formats information from an external environment.
   - An analysis component that transforms information as required by the application, a control/output component that responds to the external environment, and a monitoring component that coordinates all other components so that real-time response can be maintained.

3. **Business software:**
   - Business information processing is the largest single software application area. Discrete "systems" have evolved into management information system (MIS) software that accesses one or more large databases containing business information.

4. **Engineering and scientific software:**
   - Engineering and scientific software have 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.

5. **Embedded software:**
   - Intelligent products have become commonplace in nearly every consumer and industrial market. Embedded software resides in read-only memory and is used to control products and systems for the consumer and industrial markets.
   - Embedded software can perform very limited and esoteric functions or provide significant function and control capability. (e.g., digital functions in an automobile such as fuel control, dashboard displays, and braking systems).

6. **Personal computer software:**
   - The personal computer software market has burgeoned over the past two decades.
   - Word processing, spreadsheets, computer graphics, multimedia, entertainment, database management, personal and business financial applications, external network, and database access are only a few of hundreds of applications.

7. **Web-based software:**
   - The Web pages retrieved by a browser are software that incorporates executable instructions (e.g., CGI, HTML, Perl, or Java), and data (e.g., hypertext and a variety of visual and audio formats).
   - In essence, the network becomes a massive computer providing an almost unlimited software resource that can be accessed by anyone with a modem.

8. **Artificial intelligence software:**
   - Artificial intelligence (AI) software makes use of non numerical algorithms to solve complex problems that are not amenable to computation or straightforward analysis.
   - Expert systems, also called knowledge based systems, pattern recognition (image and voice), artificial neural networks, theorem proving, and game playing are representative of applications within this category.
3. **Darwan explain Software Engineering layers OR Layer Technology.**

- Software is (1) instructions (computer programs) that when executed provide desired function and performance, (2) data structures that enable the programs to adequately manipulate information, and (3) documents that describe the operation and use of the programs.

### 3.1 A Quality Focus:

- Software engineering is a layered technology. Any engineering approach (including 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.
- It supports a quality focus. The foundation for software engineering is the process layer. Software engineering process is the glue that holds the technology layers together and enables rational and timely development of computer software.

### 3.2 Process:

- Process defines a framework for a set of key process areas (KPAs) that must be established for effective delivery of software engineering technology.
- The key process areas form the basis for management control of software projects and establish the context in which technical methods are applied, work products are produced, milestones are established, quality is ensured, and change is properly managed.

### 3.3 Methods:

- Software engineering methods provide the technical terms for building software. Methods encompass a broad array of tasks that include requirements analysis, design, program construction, testing, and support.
- Software engineering methods rely on a set of basic principles that govern each area of the technology and include modeling activities and other descriptive techniques.

### 3.4 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.
- CASE combines software, hardware, and a software engineering database (a repository containing important information about analysis, design, program construction, and testing) to create a software engineering environment analogous to CAD/CAE (computer-aided design/engineering) for hardware.

4. **Describe generic view of software Engineering.**

- Engineering is the analysis, design, construction, verification, and management of technical entities. Regardless of the entity to be engineered, the following questions must be asked and answered:
i) What is the problem to be solved?
ii) What characteristics of the entity are used to solve the problem?
iii) How will the entity (and the solution) be realized?
iv) How will the entity be constructed?
v) What approach will be used to uncover errors that were made in the design?
vi) And construction of the entity?
vii) How will the entity be supported over the long term, when corrections, adaptations, and enhancements are requested by users of the entity?

- The work associated with software engineering can be categorized into three generic phases, regardless of application area, project size, or complexity. Each phase addresses one or more of the questions noted previously.

- The definition phase focuses on what. The software engineer attempts to identify what information is to be processed, what function and performance are desired, what system behavior can be expected, what interfaces are to be established, what design constraints exist, and what validation criteria are required to define a successful system.

- The key requirements of the system and the software are identified. Although the methods applied during the definition phase will vary depending on the software engineering that is applied, three major tasks will occur in some form system or information engineering, software project planning, and requirements analysis.

- The development phase focuses on how. That is, during development a software engineer attempts to define
  i) How data are to be structured,
  ii) How function is to be implemented within a software architecture,
  iii) How procedural details are to be implemented,
  iv) How interfaces are to be characterized,
  v) How the design will be translated into a programming language and
  vi) How testing will be performed.

- The methods applied during the development phase will vary, but three specific technical tasks should always occur: software design, code generation and software testing.

- The support phase focuses on change associated with error correction, adaptations required as the software's environment evolves, and changes due to enhancements brought about by changing customer requirements.

- The support phase reapplies the steps of the definition and development phases but does so in the context of existing software. Four types of change are encountered during the support phase:
  1. **Correction.** Even with the best quality assurance activities, it is likely that the customer will uncover defects in the software. Corrective maintenance changes the software to correct defects.
  2. **Adaptation.** Over time, the original environment (e.g., CPU, operating system, business rules, external product characteristics) for which the software was developed is likely to change. Adaptive maintenance results in modification to the software to accommodate changes to its external environment.
  3. **Enhancement.** As software is used, the customer/user will recognize additional functions that will provide benefit. Perfective maintenance extends the software beyond its original functional requirements.
4. **Prevention.** Computer software deteriorates due to change, and because of this, preventive maintenance, often called software reengineering, and must be conducted to enable the software to serve the needs of its end users. In essence, preventive maintenance makes changes to computer programs so that they can be more easily corrected, adapted, and enhanced.

- In addition to these support activities, the users of software require continuing support. In-house technical assistants, telephone-help desks, and application-specific Web sites are often implemented as part of the support phase.
- The phases and related steps described in our generic view of software engineering are complemented by a number of umbrella activities. Typical activities in this category include:
  - Software project tracking and control
  - Formal technical reviews
  - Software quality assurance
  - Software configuration management
  - Document preparation and production
  - Reusability management
  - Measurement
  - Risk management

5. **Explain Process Framework or Draw Common Process Framework.**

- Software Process framework is a set of guidelines, concepts and best practices that describes high level processes in software engineering. It does not talk about how these processes are carried out and in what order.
- Process framework is established by defining a small number of framework activities that are applicable to all software projects, regardless of their size or complexity.
- A number of task sets—each a collection of software engineering work tasks, project milestones, work products, and quality assurance points—enable the framework activities to be adapted to the characteristics of the software project and the requirements of the project team.
- The following generic framework activities are used generally in projects:
  1. **Communication:**
     - It involves heavy communication and collaboration with the customer and it will gather all information from customer. So that he/she can easily serve them.
  2. **Planning**
     - It establishes plan for software engineering work. It describes technical task to be conducted, the required resources, work product to be produced etc.
  3. **Modeling:**
     - It contains phase of analysis and design. It creates model that can be understood by customer and developer easily. So that they can have pre idea about the design of product.
     - Analysis encompasses on a set of tasks that lead to create analysis model. (Negotiation, Requirement Gathering, Validation) While design encompasses work tasks that create a design model. (Architectural design, Interface design etc.)
  4. **Construction:**
     - It combines code generation and testing phase which is required to recover errors in the product.
  5. **Deployment:**
     - Whenever product is ready then it will be deployed to the customer either partially or completely. Then customer provides feedback based on evaluation.
     - These five generic framework activities can be used during the development of small programs, the
creation of large application. The details of software will be very different but the framework activities remain same.

- The framework described in generic view is complemented by a number of umbrella activities. In which:

![Common process framework](image)

1) **Software project tracking and control:**
   It allows the software team to assess progress against the project plan. And take necessary action to maintain a schedule.

2) **Risk Management:**
   It defines risks that may affect the outcome of the project or the quality of the product.

3) **Software quality assurance:**
   It defines and conducts the activities to ensure the software quality.

4) **Formal Technical Reviews:**
   It is necessary to uncover or remove errors before they are propagated to the next action or activity.

5) **Software Configuration Management:**
   It manages the effects of the change throughout the software process.

6) **Re-usability Management:**
   It defines criteria for work product reuse and establishes mechanism to achieve reusable components.

7) **Work product Preparation and Production:**
   It will create work products such as models, documents, logs, forms etc.

8) **Measurement:**
   It defines and collects process, project, and product measures and assists the team to deliver product as per customer’s requirements.

- Thus umbrella activities are applied throughout the software process and it is necessary for success.

6) **Explain Waterfall model or Linear Sequential model or classic life cycle.**
   - The waterfall model is a sequential design process, often used in software development processes, in which progress is seen as flowing steadily downwards (like a waterfall) through the phases of Requirement & Analysis, Design, Code, Testing, Implementation and Maintenance.
   - The waterfall model proceeds from one phase to the next in a sequential manner. For example, one first completes requirements specifications, which after sign-off are considered "set in stone."
   - When requirements are completed, one proceeds to design. The software in question is designed and a blueprint is drawn for implementers (coders) to follow—this design should be a plan for implementing the requirements given.
   - When the design is complete, an implementation of that design is made by coders in appropriate coding
language which is best suited to design.

- Once code is finalized then through testing of code is begin. All testing activities are done in this phase. After testing, the application is being ready to deploy in actual server environment. This phase is implementation phase of actual product.
- Once Application deployed successfully, we can easily modified the application in maintenance phase. The entire customer enhancement, modification has to be done in this phase.

Advantages:
- The Waterfall methodology is the most predictive of development methodologies, stepping through requirements capture, analysis, design, coding, and testing in a strict, pre-planned sequence.
- Provides a disciplined and structured approach which makes it easy to keep projects under control.
- Limits the amount of cross-team interaction needed during development.
- Allows for greater ease in project

Disadvantages:
- While the Waterfall methodology is easy on the manager, it can be grueling for developers. Testing comes at the end of the development phase; subsystem testing can reveal problems with the code that must be rectified quickly. Oversights and flawed design work can seriously affect the budgeted costs and final launch date.
- Debugging can be complicated since developers are often working on other projects at the end of development, and the needed changes can cut into their productivity and work quality.
- Leaves no room for feedback anywhere in the process, except at the end of a phase. There is no room for changes once development has begun. It fails to be open to change due to external factors and requirement modifications.
- When applied to the wrong situations the methodology can give the false expectation of predictability and the reality of cost and time overruns.
- The deliverable (ideas into working software) is a one time.

7) Prototyping Model OR
What is software prototyping? Explain its significance in software engineering with example.
- Prototype is incomplete versions of the software program being developed. Software prototyping is an activity during certain software development. A prototype typically simulates only a few aspects of the features and may be completely different from the eventual implementation.
• The purpose of a prototype is to allow users of the software to evaluate developers' proposals for the design of the product by actually trying them out, rather than having to interpret and evaluate the design based on descriptions.

• 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. In many other situations, a prototyping paradigm may offer the best approach.

The prototyping paradigm begins with requirements gathering. Developer and customer meet and define the overall objectives for the software, identify whatever requirements are known, and outline areas where further definition is mandatory.

• A "quick design" then occurs. The quick design focuses on a representation of those aspects of the software that will be visible to the customer/user. The quick design leads to the construction of a prototype.

• The prototype is evaluated by the customer/user and used to refine requirements for the software to be developed. Iteration occurs as the prototype is tuned to satisfy the needs of the customer, while at the same time enabling the developer to better understand what needs to be done.

• The prototype serves as a mechanism for identifying software requirements. If a working prototype is built, the developer attempts to use existing program fragments or applies tools that enable working programs to be generated quickly.

• The first system built is barely usable. It may be too slow, too big, awkward in use or all three. There is no alternative but to start again, smarting but smarter, and build a redesigned version in which these problems are solved.

• When a new system concept or new technology is used, one has to build a system to throw away, for even the best planning is not so omniscient as to get it right the first time.

• The prototype can serve as "the first system." Users get a feel for the actual system and developers get to build something immediately. Prototyping can also be problematic for the following reasons:

• The customer sees what appears to be a working version of the software. When informed that the product must be rebuilt so that high levels of quality can be maintained, the customer cries foul and demands that "a few fixes" be applied to make the prototype a working product.

• The developer often makes implementation compromises in order to get a prototype working quickly. An inappropriate operating system or programming language may be used simply because it is available and known; an inefficient algorithm may be implemented simply to demonstrate capability.
• After a time, the developer may become familiar with these choices and forget all the reasons why they were inappropriate.
• Although problems can occur, prototyping can be an effective for software engineering. The key is to define the rules of the game at the beginning; that is, the customer and developer must both agree that the prototype is built to serve as a mechanism for defining requirements. It is then discarded and the actual software is engineered with an eye toward quality and maintainability.

**Advantages:**
- Reduced time and costs
- Improved and increased user involvement.

**Disadvantages:**
- Insufficient analysis.
- User confusion of prototype and finished system
- Developer misunderstanding of user objectives
- Expense and time of implementing prototyping

8) **Rapid application development (RAD) Model.**
• Rapid application development (RAD) is an incremental software development process model that emphasizes an extremely short development cycle.
• The RAD model is a “high-speed” adaptation of the waterfall model in which rapid development is achieved by using component-based construction. If requirements are well understood and project scope is constrained, the RAD process enables a development team to create a “fully functional system” within very short time periods (e.g., 60 to 90 days).
• Like other process models, the RAD approach maps into the generic framework activities presented earlier.
• Communication works to understand the business problem and information characteristics that the software must accommodate.
• Planning is essential because multiple software teams work in parallel on different system functions.
• Modeling encompasses three major phases---business modeling, data modeling and process modeling---and establishes design representations that serve as the basis for RAD’s construction activity.
• Construction emphasizes the use of pre-existing software components and the application of automatic code generation.
• Finally, deployment establishes a basis for subsequent iterations, if required.
• Obviously, the time constraints imposed on a RAD project demand “scalable scope”.
• If a business application can be modularized in a way that enables each major function to be completed in less than three months (using the approach described previously), it is a candidate for RAD.
• Each major function can be addressed by a separate RAD team and then integrated to form a whole.
• The RAD process model is illustrated in Figure
Drawbacks:

- For large but scalable projects, RAD requires sufficient human resources to create the right number of RAD teams.
- RAD requires developers and customers who are committed to the rapid-fire activities necessary to get a system complete in a much abbreviated time frame.
- If commitment is lacking from either constituency, RAD projects will fail.
- If a system cannot be properly modularized, building the components necessary for RAD will be problematic.
- If high performance is an issue and performance is to be achieved through tuning the interfaces to system components, the RAD approach may not work.
- RAD is not appropriate when technical risks are high.

9) Spiral Model.

- This model of development combines the features of the prototyping and the waterfall model. The spiral model is intended for large, expensive and complicated projects.
- It provides the potential for rapid development of incremental versions of the software. Using the spiral model, software is developed in a series of incremental releases. In very beginning, this model generates prototype. Then it may create more complete versions of the system. A spiral model is divided into a number of framework activities, also called task regions. spiral model that contains six task regions:
• Customer communication—tasks required to establish effective communication between developer and customer.
• Planning—tasks required to define resources, timelines, and other project-related information.
• Risk analysis—tasks required to assess both technical and management risks.
• Engineering—tasks required to build one or more representations of the application.
• Construction and release—tasks required to construct, test, install, and provide user support (e.g., documentation and training).
• Customer evaluation—tasks required to obtain customer feedback based on evaluation of the software representations created during the engineering stage and implemented during the installation stage.

• Each of the regions is populated by a set of work tasks, called a task set. For small projects, the number of work tasks and their formality is low. For larger, more critical projects, each task region contains more work tasks.
• The software engineering team moves around the spiral in a clockwise direction, beginning at the center. The first circuit around the spiral may result in the development of a product specification. Next passes around the spiral might be used to develop a prototype and then time by time it may generate useful 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. The planned number of iterations required to complete the software.

**Concept Development Project:**
• Start at the core and continues for multiple iterations until it is complete.
• If concept is developed into an actual product, the process proceeds outward on the spiral.

**New Product Development Project:**
• New product will evolve through a number of iterations around the spiral.
• Later, a circuit around spiral might be used to represent a “Product Enhancement Project”

**Product Enhancement Project:**
• There are times when process is dormant or software team not developing new things but change is initiated, process start at appropriate entry point.

**Advantages:**
• High amount of risk analysis.
• Good for large and mission-critical projects.
• Software is produced early in the software life cycle.

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

10) Explain Winwin spiral model
• The objective is to extract project requirements from the customer. In an ideal context, the developer simply asks the customer what is required and the customer provides sufficient detail to proceed. Unfortunately, this rarely happens.
• In reality, the customer and the developer enter into a process of negotiation, where the customer may be asked to balance functionality, performance, and other product or system characteristics against cost and time to market.
• The best negotiations strive for a “win-win” result. That is, the customer wins by getting the system or product that satisfies the majority of the customer’s needs and the developer wins by working to realistic and achievable budgets and deadlines.
• Boehm’s WINWIN spiral model defines a set of negotiation activities at the beginning of each pass around the spiral.
• Rather than a single customer communication activity, the following activities are defined:

  1. Identification of the system stakeholders. That is the people on the organisation that have direct business interest in the product to be built and will be rewarded for a successful outcome or criticised if the effort fails.
  2. Determination of the stakeholder’s “win conditions”
  3. Negotiations of the stakeholder’s win conditions to reconcile them into a set of win-win conditions for all concerned (including the software project team).
  4. Successful completion of these initial steps achieves a win-win result, which becomes the key criterion for proceeding to software and system definition.
  5. In addition to the early emphasis placed on the win-win condition, the model also introduces three
process milestones (anchor points), which help establish the completion of one cycle around the spiral and provide the decision milestones before the software project proceeds. These are,

- **Life Cycle Objectives (LCO)** – Defines a set of objectives for each major software activity (e.g. a set of objectives associated with the definition of top level product requirements)
- **Life Cycle Architecture (LCA)** – Establishes the objectives that must be met as the software architecture is defined.
- **Initial Operational Capability (IOC)** – represents a set of objectives associated with the preparation of the software for installation/distribution, site preparations prior to installations, and assistance required by all parties that will use or support the software.

**Advantages:**
- Faster software production facilitated through collaborative involvement of the relevant stakeholders.
- Cheaper software via rework and maintenance reductions

11) **Concurrent Development Model.**

- The concurrent development model, sometimes called concurrent engineering, can be represented schematically as a series of major technical activities, tasks, and their associated states. For example, the engineering activity defined for the spiral model is accomplished by invoking the following tasks: prototyping and/or analysis modeling, requirements specification, and design.

- Figure below provides a schematic representation of one activity with the concurrent process model.

- The activity—analysis—may be in any one of the states noted at any given time. Similarly, other activities (e.g., design or customer communication) can be represented in an analogous manner

- All activities exist concurrently but reside in different states. For example, early in a project the customer communication activity (not shown in the figure) has completed its first iteration and exists in the awaiting changes state.

- The analysis activity (which existed in the none state while initial customer 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 analysis activity moves from the under development state into the awaiting changes state.

- The concurrent process model defines a series of events that will trigger transitions from state to state for each of the software engineering activities.

- For example, during early stages of design, an inconsistency in the analysis model is uncovered. This
generates the event analysis model correction which will trigger the analysis activity from the done state into the awaiting changes state.

- The concurrent process model 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 to a sequence of events, it defines a network of activities.

- Each activity on the network exists simultaneously with other activities. Events generated within a given activity or at some other place in the activity network trigger transitions among the states of an activity.

12) **Incremental model.**

- Sometimes there may be needed to provide some functionalities at initial level then it may be expanded in later its versions.
- The incremental build model is a method of software development where the model is designed, implemented and tested incrementally until the product is finished. It involves both development and maintenance.
- It has the feature of following 2 models:
  i) Linear Sequential model
  ii) Prototyping model

- The product is decomposed into a number of components, each of which are designed and built separately. Each component is delivered to the client when it is complete. This allows partial utilization of product and avoids a long development time.

- The series of releases is referred to as “increments”, with each increment providing more functionality to the customers. After the first increment, a core product is delivered, which can already be used by the customer.
• Based on customer feedback, a plan is developed for the next increments, and modifications are made accordingly. This process continues with increments being delivered until the complete product is delivered.

• Each linear sequence produces a deliverable “increment” of the software. For example, word-processing software developed using the incremental paradigm might deliver basic file management, editing, and document production functions in the first increment; more sophisticated editing and document production capabilities in the second increment; spelling and grammar checking in the third increment; and advanced page layout capability in the fourth increment.

• It focuses on the delivery of an operational product with each increment. Early increments are stripped down versions of the final product, but they do provide capability that serves the user and also provide a platform for evaluation by the user.

• Incremental development is particularly useful when staffing is unavailable for a complete implementation by the business deadline that has been established for the project. Early increments can be implemented with fewer people. If the core product is well received, then additional staff can be added to implement the next increment. Increments can be planned to manage technical risks.

Advantages
• After each iteration, regression testing should be conducted. During this testing, faulty elements of the software can be quickly identified because few changes are made within any single iteration.
• It is generally easier to test and debug than other methods of software development because relatively smaller changes are made during each iteration.
• Early increments can be developed with few people. It combines iterative nature of prototyping model and linear nature of Linear Sequential Model.
• Number of people required is less
• Easy to add quality
• The system can be designed in such a manner that it can be delivered into pieces.
• Increments are developed one after the other, after feedback has been received from the user.
• Since each increment is simpler than the original system, it is easier to predict resources needed to accomplish the development task within acceptable accuracy bounds.
• Increments can be planned to manage technical risks.

Disadvantages
• Resulting cost may exceed the cost of the organization.
• As additional functionality is added to the product, problems may arise related to system architecture which were not evident in earlier prototypes.

13) Myth in software engineering and its various types.

Software Myths
• Software Myths- beliefs about software and the process used to build it - can be traced to the earliest days of computing. Myths have a number of attributes that have made them insidious. For instance, myths appear to be reasonable statements of fact, they have an intuitive feel, and they are often promulgated by experienced practitioners who "know the score".

Management Myths
• Managers with software responsibility, like managers in most disciplines, are often under pressure to
maintain budgets, keep schedules from slipping, and improve quality. Like a drowning person who grasps at a straw, a software manager often grasps at belief in a software myth, if the belief will lessen the pressure.

- **Myth:** We already have a book that's full of standards and procedures for building software. Won't that provide my people with everything they need to know?
- **Reality:** The book of standards may very well exist, but is it used? - Are software practitioners aware of its existence? - Does it reflect modern software engineering practice? - Is it complete? Is it adaptable? - Is it streamlined to improve time to delivery while still maintaining a focus on Quality? In many cases, the answer to these entire question is no.

- **Myth:** If we get behind schedule, we can add more programmers and catch up (sometimes called the Mongolian horde concept)
- **Reality:** Software development is not a mechanistic process like manufacturing. In the words of Brooks [BRO75]: "Adding people to a late software project makes it later." At first, this statement may seem counterintuitive. However, as new people are added, people who were working must spend time educating the newcomers, thereby reducing the amount of time spent on productive development effort

- **Myth:** If we decide to outsource the software project to a third party, I can just relax and let that firm build it.
- **Reality:** If an organization does not understand how to manage and control software project internally, it will invariably struggle when it out sources 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 led to false expectations and ultimately, dissatisfaction with the developers.

- **Myth:** A general statement of objectives is sufficient to begin writing programs we can fill in details later.

- **Reality:** Although a comprehensive and stable statement of requirements is not always possible, an ambiguous statement of objectives is a recipe for disaster. Unambiguous requirements are developed only through effective and continuous communication between customer and developer.