**Lecture-1: Introduction**

**1) What is software?**

**Ans**: Computer programs and associated documentation. Software products may be developed for a particular customer or may be developed for a general market. Good software should deliver the required functionality and performance to the user, should be maintainable, dependable and usable and ease of use should be high.

**2) What is software engineering?**

Ans: Software engineering is an engineering discipline that is concerned with all aspects of software production.

**3) Two kinds of SW products (SW which can be sold to a customer)**

**Ans: Generic products:** Stand-alone systems that are marketed and sold to any customer who wishes

to buy them. The specification of what the software should do is owned by the software developer and decisions on software change are made by the developer.

**Examples** – PC software such as graphics programs, project management tools; CAD software; software for specific markets such as appointments systems for dentists.

**Customized products:** Software that is commissioned by a specific customer to meet their own needs. The specification of what the software should do is owned by the customer for the software and they make decisions on software changes thatare required.

**Examples** – embedded control systems, air traffic control software, traffic monitoring systems.

Many systems are now being built with a generic product as a base, which is then adapted to suit the requirements of a customer. E.g., Enterprise Resource Planning (ERP), Systems Applications Products.

**4) Importance of software engineering**

**Ans:** More and more, individuals and society rely on advanced software systems. We need to be able to produce reliable and trustworthy systems economically and quickly. It is usually cheaper, in the long run, to use software engineering methods and techniques for software systems rather than just write the programs as if it was a personal programming project. For most types of system, the majority of costs are the costs of changing the software after it has gone into use.

**5) General issues that affect software**

**Ans:** **Heterogeneity:** Increasingly, systems are required to operate as distributed systems across networks that include different types of computer and mobile devices.

**Business and social change:** Business and society are changing incredibly quickly as emerging economies develop and new technologies become available. They need to be able to change their existing software and to rapidly develop new software.

**6) Software Application Domains**

**Ans: a)System software** —A collection of programs written to service other programs. Some system software (e.g., compilers, editors, and file management utilities) processes complex, but determinate,for information structures. Other systems applications (e.g., operating system components, drivers, networking software, telecommunications processors) process largely indeterminate data. In either case, the systems software area is characterized by heavy interaction with computer hardware; heavy usage by multiple users; concurrent operation that requires scheduling, resource sharing, and sophisticated process management; complex data structures; and multiple external interfaces.

**b)** **Application software**—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. In addition to conventional data processing applications, application software is used to control business functions in real time (e.g., point-of-sale transaction processing, real-time manufacturing process control).

**c) 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. However, modern applications within the engineering/scientific area are moving away from conventional numerical algorithms. Computer-aided design, system simulation, and other interactive applications have begun to take on real-time and even system software characteristics.

**d) Embedded software—**Resides within a product or system and is used to implement and control features and functions for the end user andfor the system itself. These are software control systems that control

and manage hardware devices. Numerically, there are probably moreembedded systems than any other type of system. E.g., SW in a microwave oven to control the cooking process.

**e) 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).

**f) 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. However, as Web 2.0 emerges, Web Apps are evolving into sophisticated computing environments that not only provide stand-alone features, computing functions, and content to the end user, but also are integrated with corporate databases and business applications.

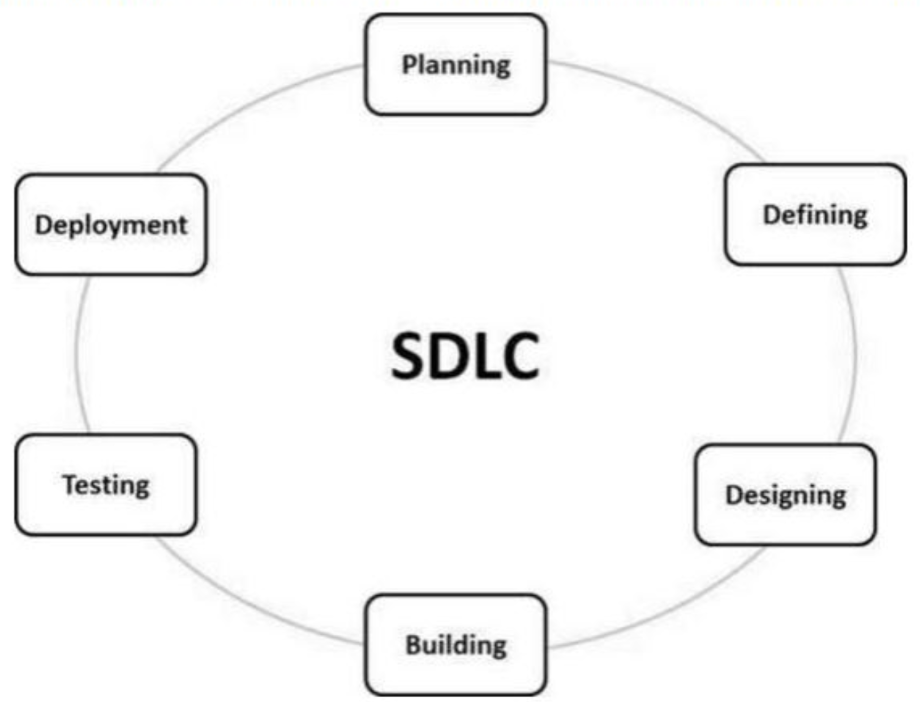
**g) Artificial intelligence software**—makes use of non-numerical algorithms to solve complex problems that are not amenable to computation or straightforward analysis. Applications within this area include robotics,

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

**7) What is SDLC?**

Ans: Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop and test high quality softwares. It consists of a detailed plan describing how to develop, maintain,

replace and alter or enhance specific software. It is also called as Software Development Process. SDLC is a framework defining tasks performed at each step in the software development process.

****

**Stage 1: Planning and Requirement Analysis:** Requirement analysis is the most important and fundamental stage in SDLC. It is performed by the senior members of the team with inputs from the customer, the sales department, market surveys and domain experts in the industry. This information is then used to plan the basic project approach and to conduct product feasibility study in the economical, operational and technical areas. Planning for the quality assurance requirements and identification of the risks associated with the project is also done in the planning stage. The outcome of the technical feasibility study is to define the various technical approaches that can be followed to implement the project successfully with minimum risks.

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

**Stage 3: Designing the Product Architecture:** SRS is the reference for product architects to come out with the best architecture for the product to be developed. Based on the requirements specified in SRS, usually more than one design approach for the product architecture is proposed and documented in a DDS - Design Document Specification. This DDS is reviewed by all the important stakeholders and based on various parameters as risk assessment, product robustness, design modularity, budget and time constraints, the best design approach is selected for the product. A design approach clearly defines all the architectural modules of the product along with its communication and data flow representation with the external and third party

modules (if any). The internal design of all the modules of the proposed architecture should be clearly defined with the minutest of the details in DDS.

**Stage 4: Building or Developing the Product:** In this stage of SDLC the actual development starts and the product is built. The programming code is generated as per DDS during this stage. If the design is performed in a detailed and organized manner, code generation can be accomplished without much hassle. Developers must follow the coding guidelines defined by their organization and programming tools like compilers, interpreters, debuggers, etc. are used to generate the code. Different high level programming languages such as C, C++, Pascal, Java and PHP are used for coding. The programming language is chosen with respect to the type of software being developed.

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

**Stage 6: Deployment in the Market and Maintenance:** Once the product is tested and ready to be deployed it is released formally in the appropriate market. Sometimes product deployment happens in stages as per the business strategy of that organization. The product may first be released in a limited segment and tested in the real business environment (UAT- User acceptance testing). Then based on the feedback, the product may be released as it is or with suggested enhancements in the targeting market segment. After the product is released in the market, its maintenance is done for the existing customer base.

**Lecture 1.5 : Use Case diagram**

**1) What is a use case diagram?**

Ans: In the Unified Modeling Language (UML), a use case diagram can summarize the details of your system's users (also known as actors) and their interactions with the system. To build one, you'll use a set of specialized symbols and connectors.

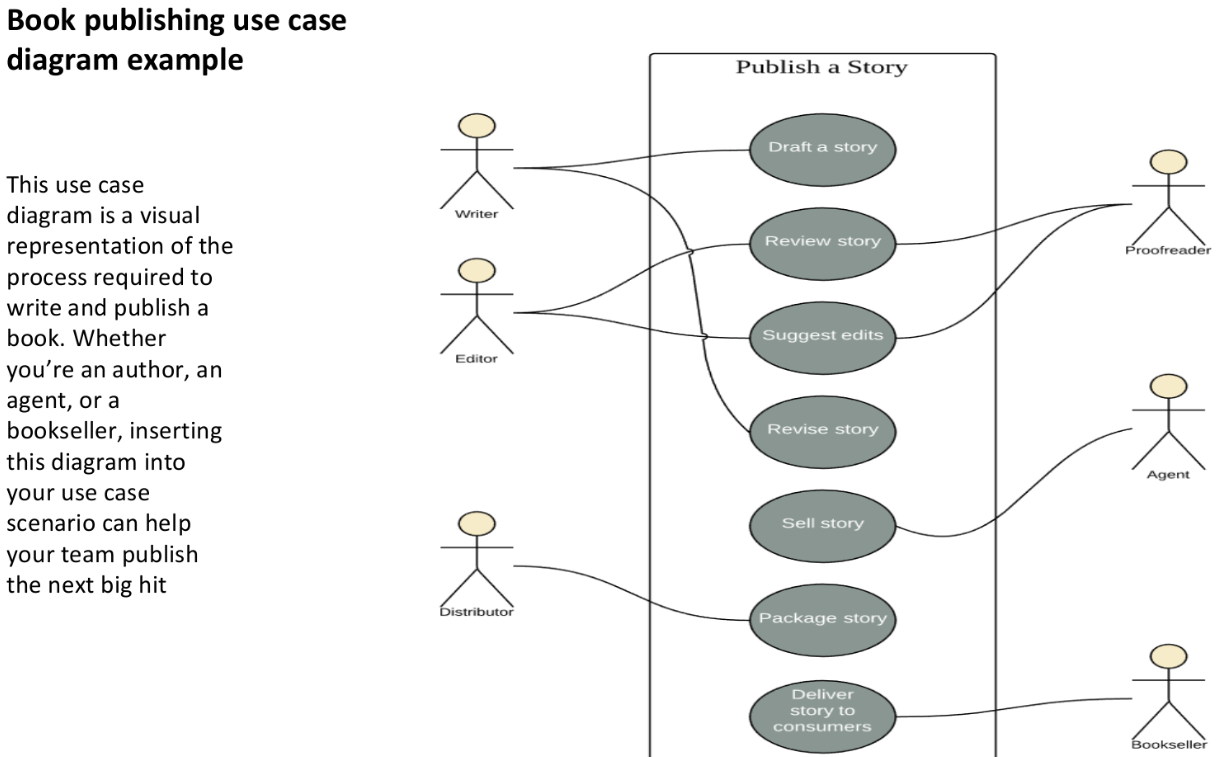
**2) Use case diagram components?**

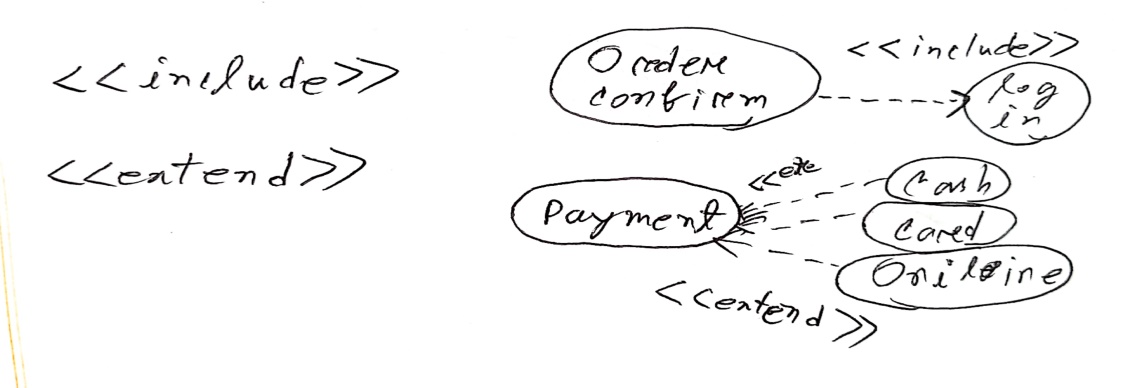
Ans: **Actors:** The users that interact with a system. An actor can be a person, an organization, or an outside system that interacts with your application or system. They must be external objects that produce or consume data.

**System:** A specific sequence of actions and interactions between actors and the system. A system may also be referred to as a scenario.

**Goals:** The end result of most use cases. A successful diagram should describe the activities and variants

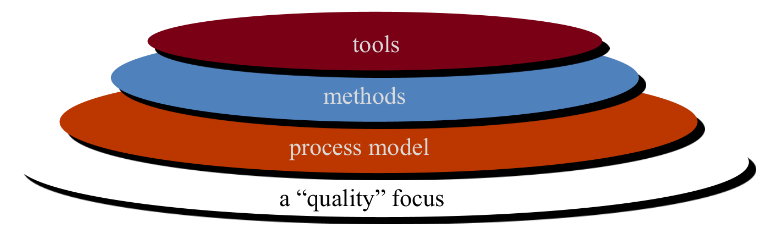
used to reach the goal.





**Lecture-2: Process: A Generic View**

1**) A Layered Technology**



**a) Quality Focus**: Software engineering is a layered technology. Any engineering approach (including software engineering) must rest on an organizational commitment to quality. The bedrock that supports software engineering is a quality focus.

**b) Process layer**: The foundation for software engineering is the process layer. The software engineering process is the glue that holds the technology layers together and enables rational and timely development of computer software Process defines a framework that must be established for effective delivery of software engineering technology.

**c) Method:** Software engineering methods provide the technical how-to’s for building software. Methods encompass a broad array of tasks that include communication, requirements analysis, design modeling, program construction, testing, and support.

**d) 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) Some important terms:**

**a) Task:** A task focuses on a small, but well-defined objective (e.g., conducting a unit test) that produces a tangible outcome.

**b) Action**: An action encompasses a set of tasks that produce a major work product (e.g., an architectural design model).

**c) Activity**: An activity strives to achieve a broad objective(e.g., communication with stakeholders) and is applied regardless of the application domain, size of the project, complexity of the effort, or degree of rigor with which software engineering is to be applied.

**d) Process**: A process is a collection of activities, actions, and tasks that are performed when some work product is to be created.

**3) Process framework**: A process framework establishes the foundation for a complete software engineering process by identifying a small number of framework activities that are applicable to all software projects, regardless of their size or complexity. In addition, the process framework encompasses a set of

umbrella activities that are applicable across the entire software process.

**Process frameworks**: Framework activities, work tasks, work products, milestones & deliverables, QA checkpoints.

**4) Framework Activities**: a) Communication

b) Planning

c) Modeling

- Analysis of requirements

- Design

d) Construction

- Code generation

- Testing

e) Deployment

**a) Communication:**  Before any technical work can commence, it is critically important to communicate and collaborate with the customer (and other Stakeholders). The intent is to understand stakeholders’ objectives for the project and to gather requirements that help define software features and functions.

**b) Planning:** Any complicated journey can be simplified if a map exists. A software project is a complicated journey, and the planning activity creates a “map” that helps guide the team as it makes the journey. The map—called a software project plan—defines the software engineering work by describing the technical tasks to be conducted, the risks that are likely, the resources that will be required, the work products to be produced, and a work schedule.

**c) Modeling**: Modling refine the sketch into greater and greater detail in an effort to better understand the problem and how we are going to solve it. A software engineer does the same thing by creating models to better understand software requirements and the design that will achieve those requirements.

**d) Construction:** This activity combines code generation (either manual or automated) and the testing that is required to uncover errors in the code.

**e) Deployment**: The software (as a complete entity or as a partially completed increment) is delivered to the customer who evaluates the delivered product and provides feedback based on the evaluation.

**5) Umbrella activities:**

**a) Software project tracking and control**—allows the software team to assess progress against the project plan and take any necessary action to maintain the schedule.

**b) Risk management**—assesses risks that may affect the outcome of the project or the quality of the product.

**c) Software quality assurance**—defines and conducts the activities required to ensure software quality.

**d) Technical reviews**—assesses software engineering work products in an effort to uncover and remove errors before they are propagated to the next activity.

**e) Measurement**—defines and collects process, project, and product measures that assist the team in delivering software that meets stakeholders’ needs; can be used in conjunction with all other framework and umbrella activities.

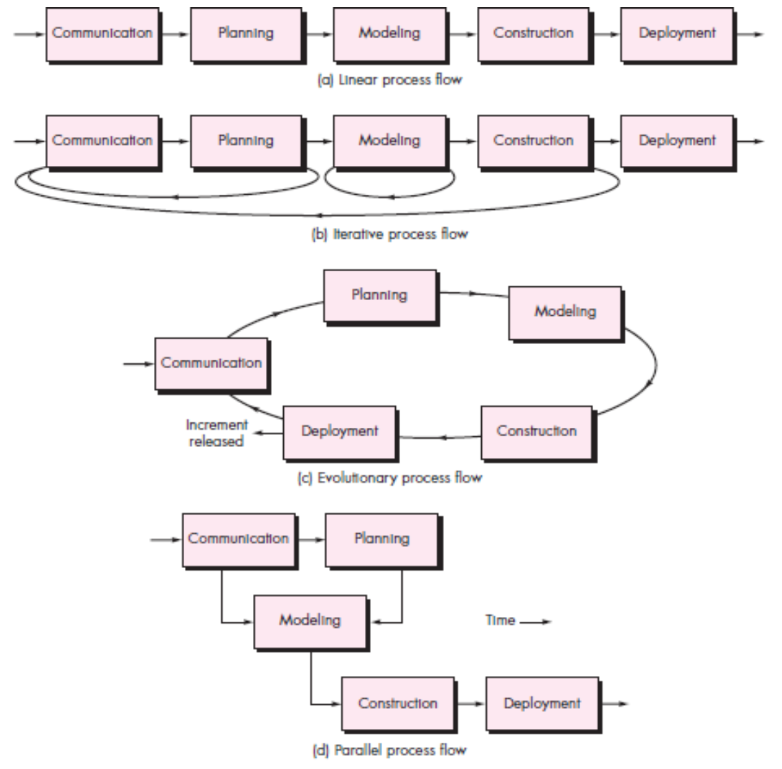
**f) Software configuration management**—manages the effects of change throughout the software process.

**g) Reusability management**—defines criteria for work product reuse (including software components) and establishes mechanisms to achieve reusable components.

**h) Work product preparation and production**—encompasses the activities required to create work products such as models, documents, logs, forms, and lists.

**6) A generic process model:** A process was defined as a collection of work activities, actions, and tasks that are performed when some work product is to be created. Each of these activities, actions, and tasks reside within a framework or model that defines their relationship with the process and with one another.

**7) Process Flow: D**escribes how the framework activities and the actions and tasks that occur with in each framework activity are organized with respect to sequence and time.



**A linear process flow** executes each of the five framework activities in sequence, beginning with communication and culminating with deployment.

**An iterative process flow** repeats one or more of the activities before proceeding to the next.

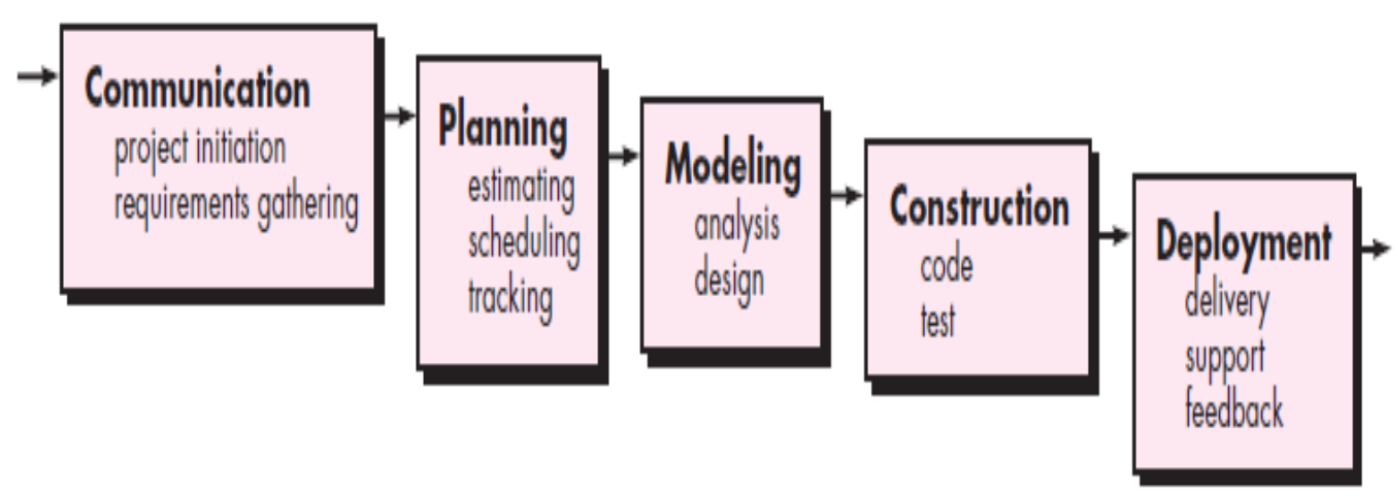
**An evolutionary process flow** executes the activities in a “circular” manner. Each circuit through the five activities leads to a more complete version of the software.

**A parallel process flow** executes one or more activities in parallel with other activities (e.g., modeling for one aspect of the software might be executed in parallel with construction of another aspect of the software).

**Lecture - 3**

**1. Waterfall model** The waterfall model, sometimes called the classic life cycle, suggests a systematic

sequential approach to software development that begins with customer specification of requirements and progresses through planning, modeling, construction, and deployment, culminating in ongoing support of the completed software.



**2. Waterfall Model Advantages: a)** Simple and easy to understand and use

**b)**Easy to manage due tothe rigidity of the model. Each phase has specific deliverables and a review process.

**c)** Phases are processed and completed one at a time.

**d)** Works well for smaller projects where requirements are very well understood.

**e)** Clearly defined stages. **f)** Well understood milestones. **g)** Easy to arrange tasks.

**h)** Process and results are well documented.

**3. Disadvantages of Waterfall Model :**

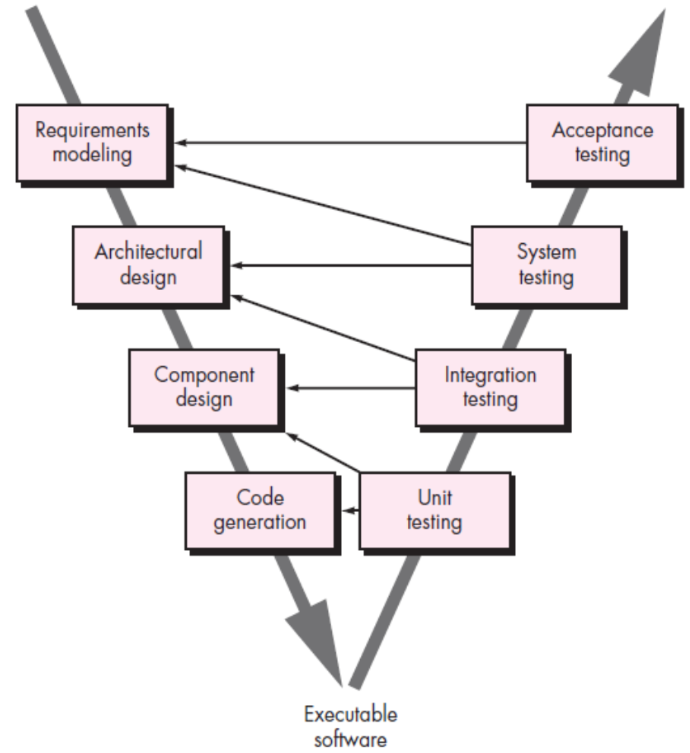
**a)** Real projects rarely follow the sequential flow that the model proposes. Although the linear model can accommodate iteration, it does so indirectly. As a result, changes can cause confusion as the project team proceeds.

**b)** It is often difficult for the customer to state all requirements explicitly. The waterfall model requires this and has difficulty accommodating the natural uncertainty that exists at the beginning of many projects.

**c)** The customer must have patience. A working version of the program(s) will not be available until late in the project time span. A major blunder, if undetected until the working program is reviewed, can be disastrous. **d)** Blocking Stages.

**4) V model:** A variation in the representation of the waterfall model is called the V-model. the V-model depicts the relationship of quality. assurance actions to the actions associated with communication, modeling, andearly construction activities. As a software team moves down the left side of the V,basic problem requirements are refined into progressively more detailed and technical representations of the problem and its solution. Once code has been generated,the team moves up the right side of the V, Essentially performing a series of tests (quality assurance actions) that validate each of the models created as the team moved down the left side.7 In reality, there is no fundamental difference between the classic life cycle and the V-model. The V-model provides a way of visualizing how verification and validation actions are applied to earlier engineering work.

**4)**



**5) The advantages of the V-Model:**

**a)** This is a highly-disciplined model and Phasesare completed one at a time.

**b)** Works well for smaller projects where requirements are very well understood.

**c)** Simple and easy to understand and use.

**d)** Easy to manage due tothe rigidity ofthe model. Each phase has specific deliverables and a review process.

**6) Disadvantages of the V-Model : a)** High risk and uncertainty.

**b)** Not a good model for complex and object-oriented projects.

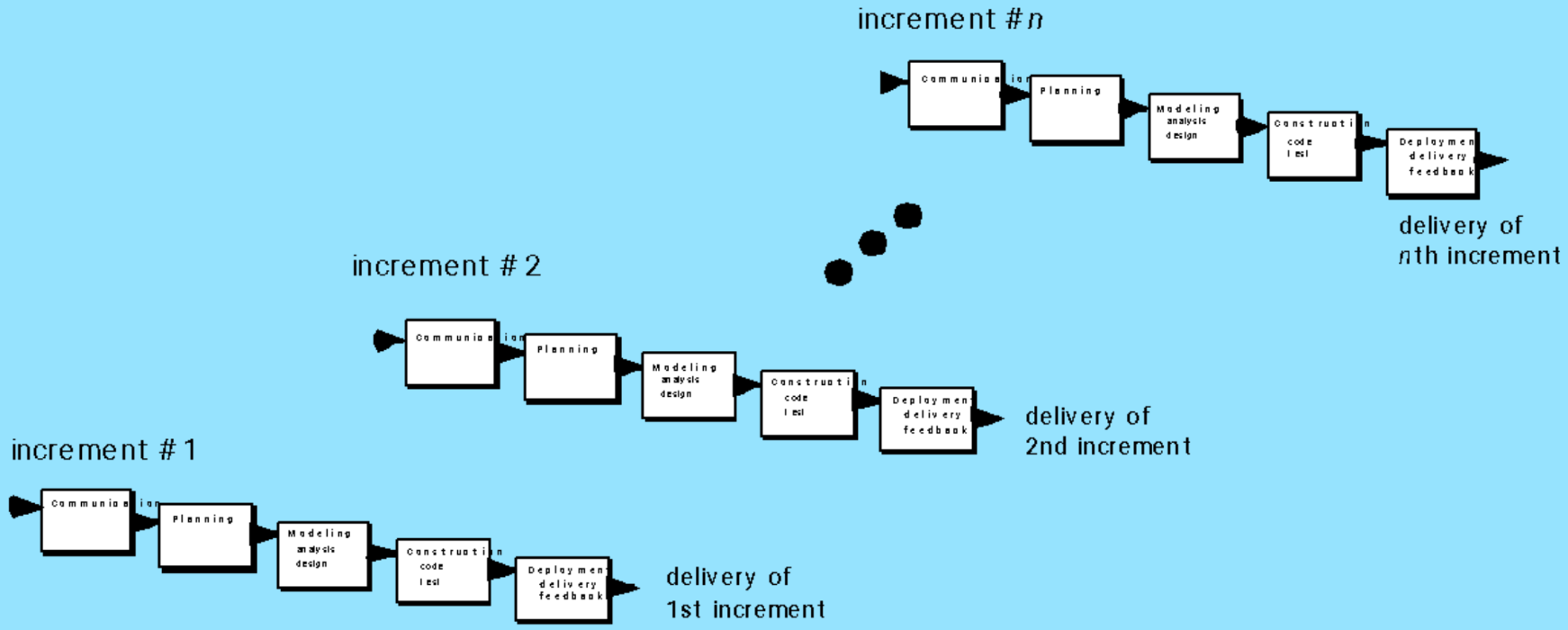
**c)** Poor model for long and ongoing projects.

**d)** Not suitable for the projects where requirements are at a moderate to high risk of changing.

**e)** Once an application is in the testing stage, it is difficult to go back and change a functionality.

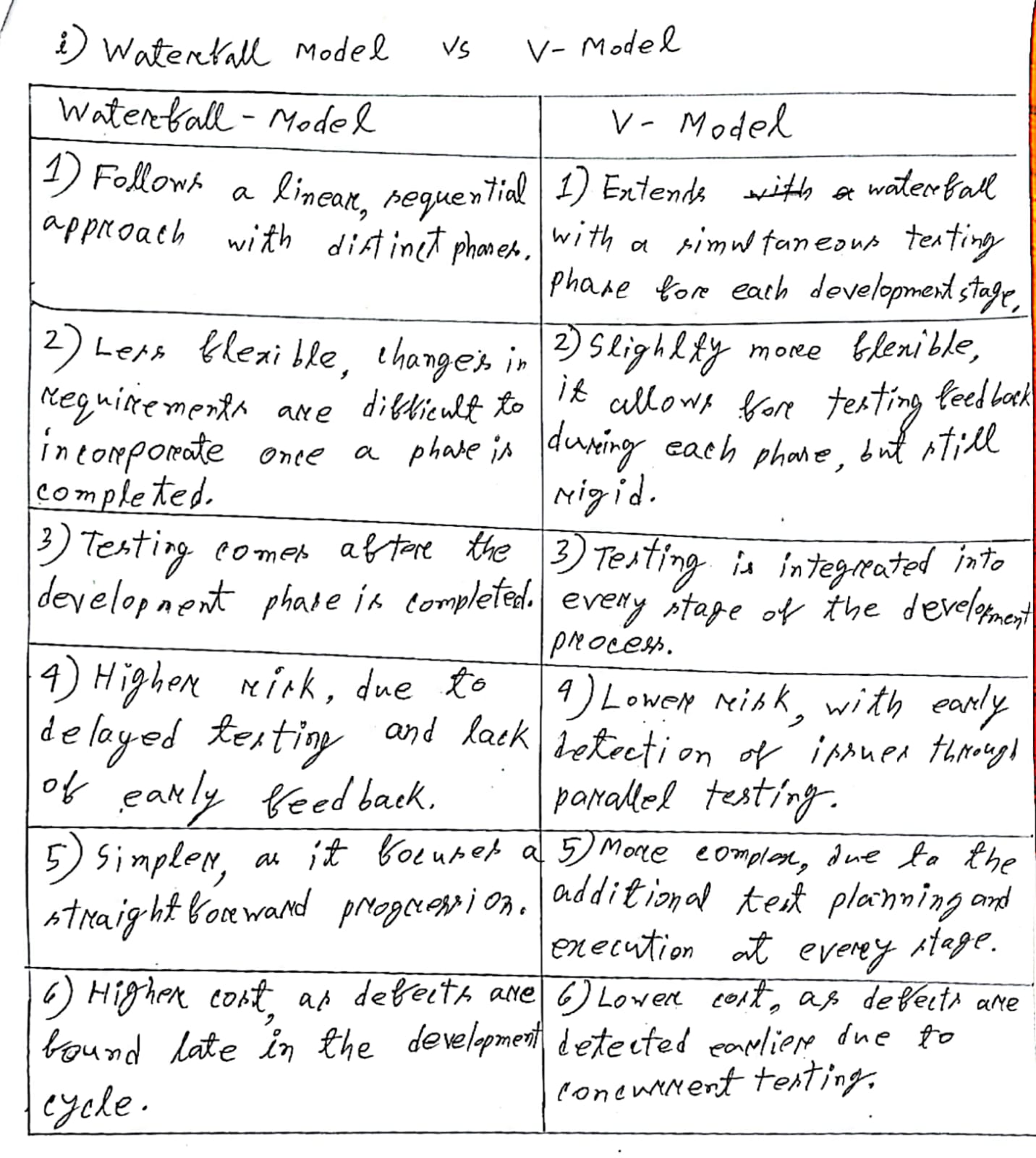
**f)** No working software is produced until late during the life cycle.

**7) Incremental Model:** The incremental model combines elements of linear and parallel process flows in the incremental model applies linear sequences in a staggered fashion as calendar time progresses. Each linear sequence produces deliverable “increments” of the software in a manner that is similar to the increments produced by an evolutionary process flow.

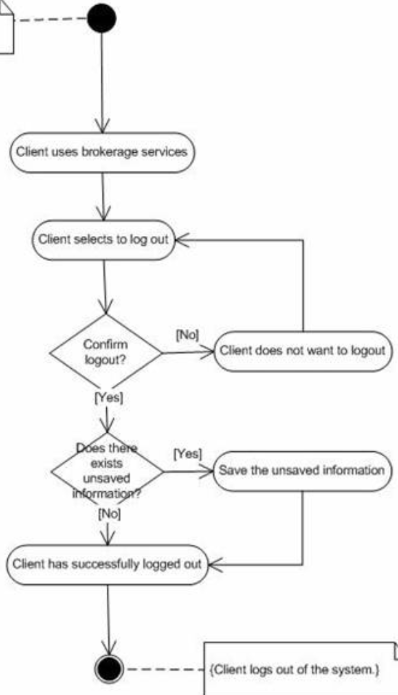


**8) Advantages of incremental model:** Incremental development is particularly useful when staffing is

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 (if required) can be added to implement the next increment. In addition, increments can be planned to manage technical risks. For example,a major system might require the availability of new hardware that is under development and whose delivery date is uncertain. It might be possible to plan early increments in a way that avoids the use of this hardware, thereby enabling partial functionality to be delivered to end users without inordinate delay.

****

**Lecture - 3.5 (Activity diagram)**



**Activity diagram** is another important diagram in UML to describe the dynamic aspects of the system. It is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system. The control flow is drawn from one operation to another.

**Swim Lane Diagram:**

