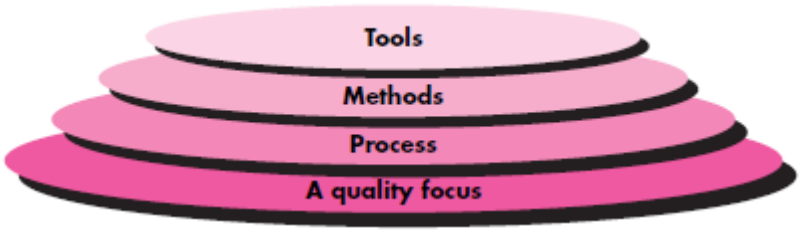
Software Engineering is a layered technology



Layer 1: Quality Focus

- Organizational commitment to quality forms the base to any engineering approach.
- Total quality management, six sigma and such principles need to be applied.
- There are some typical **umbrella activities** that are applied throughout the software project and help a software team manage and control progress, quality, change and risk:

1. **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**.

2. **Risk management**
   - Assesses **risks** that may affect the **outcome** of the project or the **quality** of the product.

3. **Software quality assurance**
   - Defines and conducts the activities required to **ensure software quality**.

4. **Technical reviews**
   - **Assess** software engineering work products in an effort **to uncover and remove errors** before they are propagated to the next activity.

5. **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.

6. **Software configuration management**
   - Manages the effects of **change** throughout the software process.

7. **Reusability management**
   - Defines criteria for work **product reuse** (including **software components**) and establishes mechanisms to achieve reusable components.

8. **Work product preparation and production**
   - Encompasses the activities required to **create work products** such as models, documents, logs, forms, and lists.

Layer 2: Software process framework

- This includes models, documentation reports, forms, etc.
- A process is a collection of activities, actions, and tasks that are performed when some work product is to be created.

  An **activity** strives to achieve a broad objective (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 (inflexibility) with which software engineering is to be applied.

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

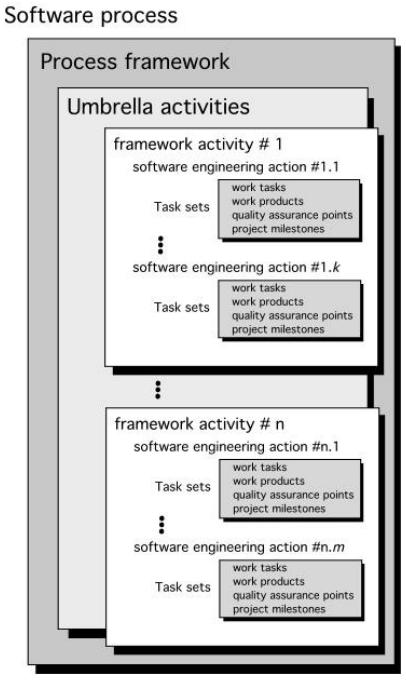
  A **task** focuses on a small, but well-defined objective (e.g., conducting a unit-test) that produces a tangible (concrete) outcome.

- In the context of software engineering, a process is ***not a rigid prescription*** for how to build computer software.
- Rather, it is an ***adaptable approach*** that enables the people doing the work (the software team) to pick and choose the appropriate set of work actions and tasks.

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.

- The **software process** is represented schematically in Figure.
- Each **framework activity** is populated by a set of software engineering **actions**.
- Each software engineering **action** is defined by a ***task set*** that identifies the following:
  - The **work tasks** that are to be completed
  - The **work products** that will be produced
  - The **quality assurance points** that will be required
  - The **milestones** that will be used to indicate progress.



Layer 3: 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.

5 generic process framework activities

These are the basic steps (process) of Software Development Life Cycle (SDLC)

1. **Communication**:
    - Project initiation
    - Requirements gathering
    - SRS: Software Requirement Specification

2. **Planning**:
    - Estimating (size, time, effort)
    - Scheduling
    - Tracking

3. **Modeling**:
    - Analysis
    - Design

4. **Construction**:
    - Code
    - Test

5. **Deployment**:
    - Delivery
    - Support
    - Feedback

Layer 4: Tools

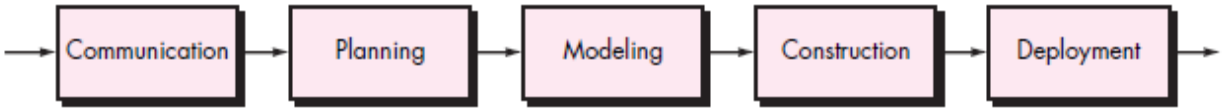
- Software engineering *tools* provide automated or semi-automated support for the process and the methods.
- When tools are integrated so that information created by one tool can be used by another, a system for the support of software development, called ***computer-aided software engineering*** **(CASE)**,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

Process flow

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

a) Linear Process Flow

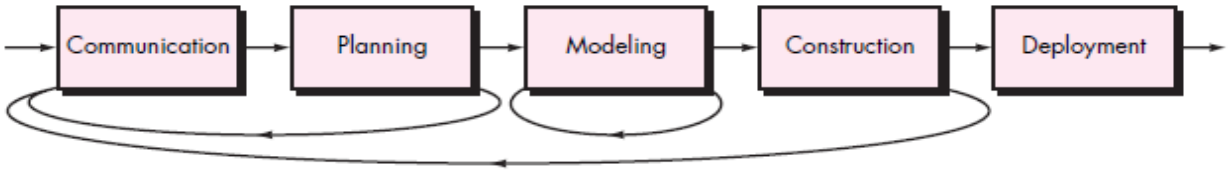
A linear process flow executes each of the five framework activities in sequence, beginning with communication and concluding with deployment.



(a) Linear process flow

b) Iterative Process Flow

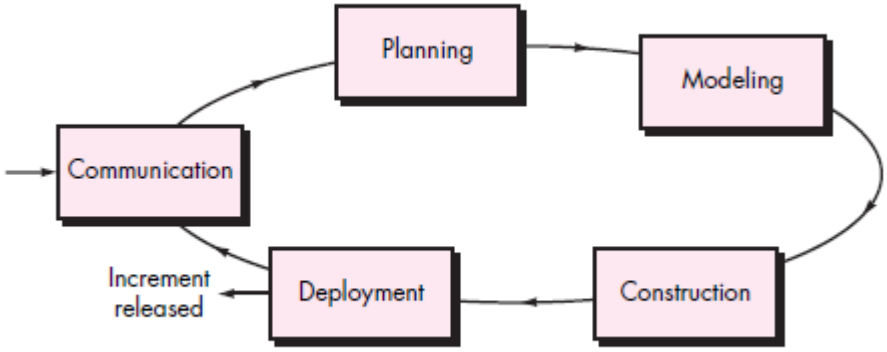
An iterative process flow repeats one or more of the activities before proceeding to the next.



(b) Iterative process flow

c) Evolutionary Process Flow

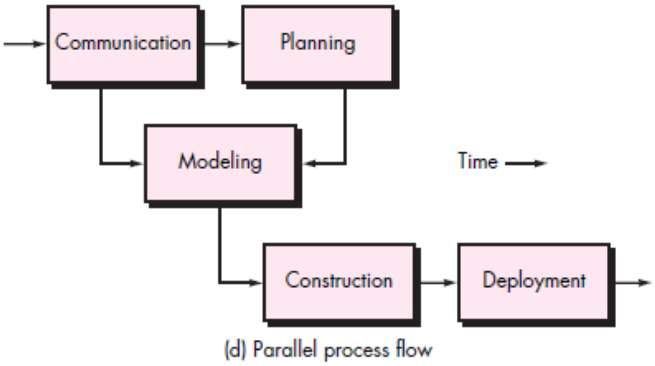
An evolutionary process flow executes the activities in a "circular" manner.
Each circuit through the five activities leads to a more complete version of the software.



(c) Evolutionary process flow

d) Parallel Process Flow

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.



(d) Parallel process flow

Process Models

They define the structure and order in which framework activities should be performed.

1. Prescriptive Process Models
2. Specialized Process Models
3. Unified Process

1. **Prescriptive process models** define a prescribed set of process elements and a predictable process work flow.

   Prescriptive process models are sometimes referred to as "traditional" process models.

   Process elements—framework activities, software engineering actions, tasks, work products, quality assurance, and change control mechanisms for each project.

   Each process model also prescribes **a process flow** (also called a work flow)—that is, the manner in which the process elements are interrelated to one another.

2. **Specialized process models** take on many of the characteristics of one or more of the traditional models.

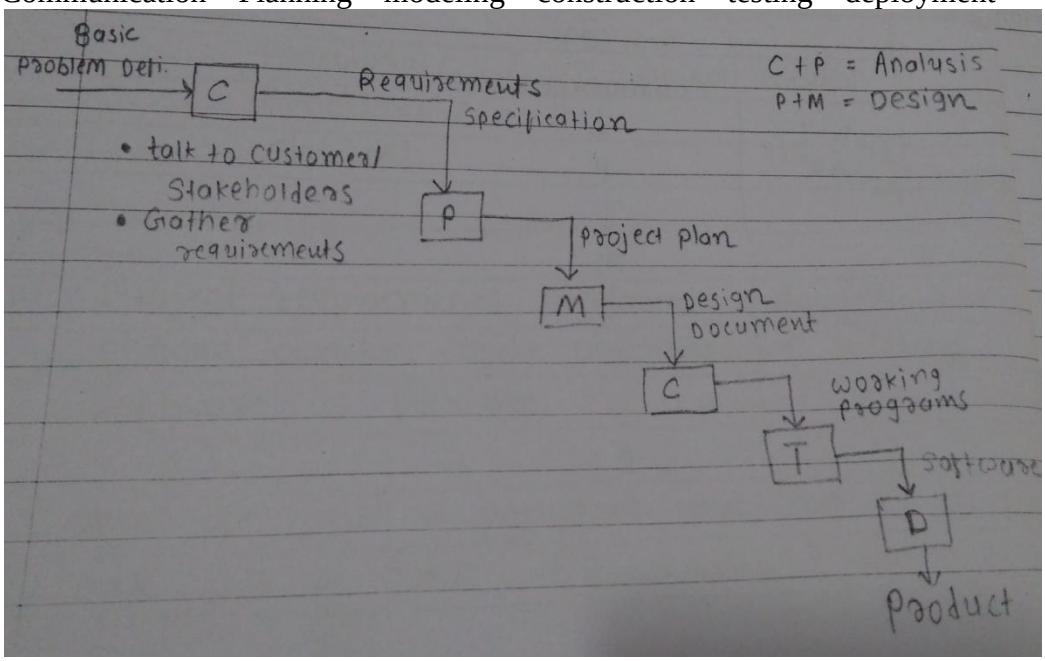
   These models tend to be applied when a specialized or narrowly defined software engineering approach is chosen.

3. The **Unified Process** is an attempt to draw on the best features and characteristics of traditional software process models, but characterize them in a way that implements many of the best principles of agile software development.

   It suggests a process flow that is iterative and incremental, providing the evolutionary feel that is essential in modern software development.

Waterfall model or Linear Sequential Model

This model performs software engineering task in sequential manner. C-P-M-C-T-D Communication – Planning – modeling – construction – testing – deployment



Drawbacks of Waterfall model

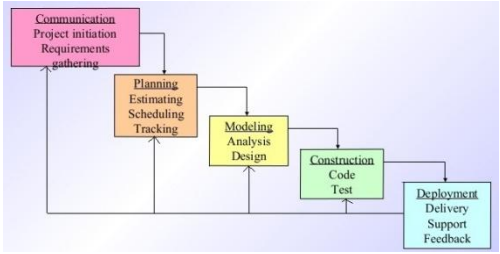
- No Feedback paths:
  - This model does not allow jumping backward.
- No mechanism for error correction :
  - The classical waterfall model is idealistic in the sense that it assumes that no error is ever committed by the developers during any of the life cycle phases.
- Inefficient error corrections and discovering of errors is usually too late.
- Difficult to accommodate change requests:
  - There is much emphasis on creating an unambiguous and complete set of requirements.
  - But, it is hard to achieve this even in ideal project scenarios.
  - The customers' requirements usually keep on changing with time.
- Customer needs to have patience for the project time to complete.
- No overlapping of phases:
  - Large number of team members to idle for extended periods. This is due to "blocking states".
  - Customer is involved in communication phase.
  - In a practical software development scenario, rather than having a precise point in time at which a phase transition occurs, the different phases need to overlap for cost and efficiency reasons.

So why use waterfall model???

- This is a theoretical model.
- This model is useful for basic understanding and adaption of the process.
- When the problem is well understood with fixed requirements, this model can be used.

Iterative Waterfall Model

- The main change brought about by the iterative waterfall model to the classical waterfall model is in the form of **providing feedback paths from every phase to its preceding phases.**
- For example, if during the testing phase a design error is identified, then the feedback path allows the design to be reworked and the changes to b e reflected in the design documents and all other subsequent documents.



- No matter how careful a programmer may be, he might end up committing some mistake or other while carrying out a life cycle activity which results in bugs in the work product.
- It is advantageous to detect these errors in the same phase in which they take place, since early detection of bugs reduces the effort and time required for correcting those.

- The principle of detecting errors as close to their points of commitment as possible is known as **phase containment of errors**.

- The end product of many phases is text or graphical documents, e.g. SRS document, design document, test plan document, etc. A popular technique is to rigorously review the documents produced at the end of a phase.

- **Phase overlap**
  - When some error in some phase is detected, the activities of that phase may have to be reworked.
  - If we consider such rework after a phase is complete, we can say that the activities pertaining to a phase do not end at the completion of the phase, but overlap with other phases.
  - Also wastage of resources, cost escalation and inefficiency can be avoided by phase overlap as once a developer completes his work assignment for a phase, proceeds to start the work for the next phase, without waiting for all his team members to complete their respective work allocations.

Drawbacks of Iterative Waterfall model

- Difficult to accommodate change in requests
- Incremental delivery not supported
- Phase overlap not supported
- Error correction is unduly expensive
- Limited customer interaction
- Heavy weight due to overemphasis on documentations
- No support for risk handling and code reuse.