

# UNIT 1

## Introduction to Software & software engineering

Prepared by :  
Dr.Pooja M. bhatt

# Index

- Why SE?
- SDLC
- Evolving Role of Software
- What is software?
- Program vs. software
- Characteristics of Software
- Hardware vs. Software
- Changing nature of software
- Software myths
- Software Process
- What is software engineering?
- Layered Approach
- Process framework activities
- Software Process models
- Product & Process

# Why SE?



1

How the  
Customer  
Explains  
Requirement



2

How the  
Project  
Leader  
understand it



3

How the  
System  
Analyst  
design it



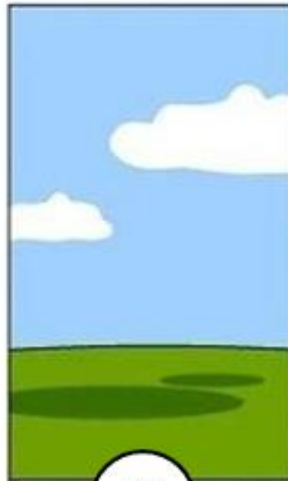
4

How the  
Programmer  
Works  
on it



5

How the  
Business  
Consultant  
describe it



6

How the  
Project  
was  
documented



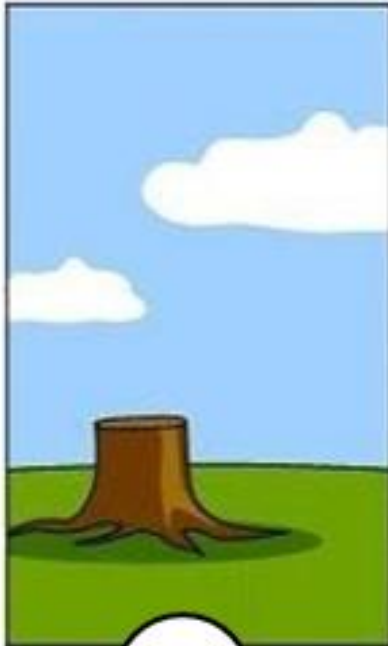
7

What  
Operations  
Installed



8

How the  
Customer  
was  
billed



**9**

**How it  
was  
supported**





**10**

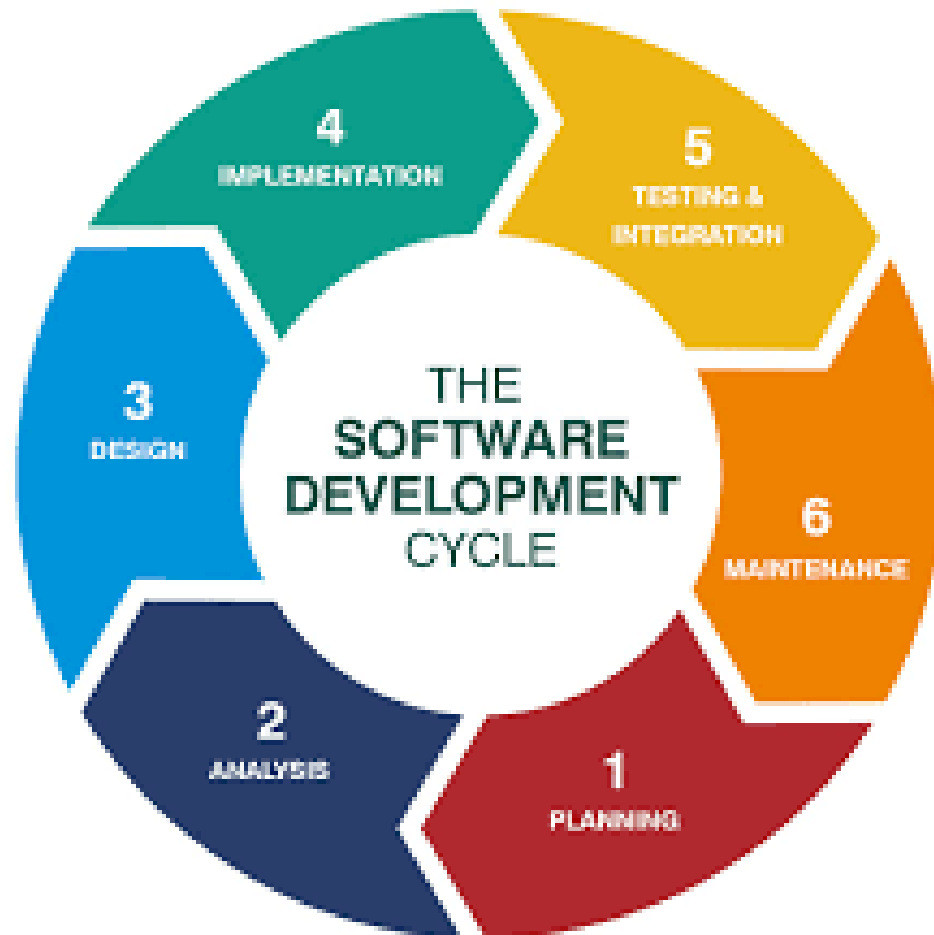
**What the  
customer  
really needed**

# Example

## SDLC **without** Software Engineering

Customer Requirement	Solution
<ul style="list-style-type: none"><li>• Have one trunk</li><li>• Have four legs</li><li>• Should carry load both passenger &amp; cargo</li><li>• Black in color</li><li>• Should be herbivorous</li></ul>	<ul style="list-style-type: none"><li>• Have one trunk</li><li>• Have four legs</li><li>• Should carry load both passenger &amp; cargo</li><li>• Black in color</li><li>• Should be herbivorous</li></ul>
	 <p>Our value added Also gives milk</p>

# SDLC



# Evolving Role of Software

## Software is a product

- Transforms information - produces, manages, acquires, modifies, displays, or transmits information
- Delivers computing potential of hardware and networks

## Software is a vehicle for delivering a product

- Controls other programs (operating system)
- Effects communications (networking software)
- Helps build other software (software tools & environments)



# What is software?

- Software can define as:
  - ❑ Instruction – executed provide desire features, function & performance.
  - ❑ Data structure – to adequately manipulate operation.
  - ❑ Documents – operation and use of the program.

Software products may be developed for a particular customer or may be developed for a general market.

- ❑ Software products may be
  - ❑ **Generic** - developed to be sold to a range of different customers e.g. PC software such as Excel or Word.
  - ❑ **Bespoke (custom)** - developed for a single customer according to their specification.

# Difference between Program and Software

Program	Software
Small in size	Large in size.
Authors himself is user-soul.	Large number.
Single developer	Team developer.
Adopt development.	Systematic development.
Lack of proper interface.	Well define interface.
Lack of proper documentation.	Well documented

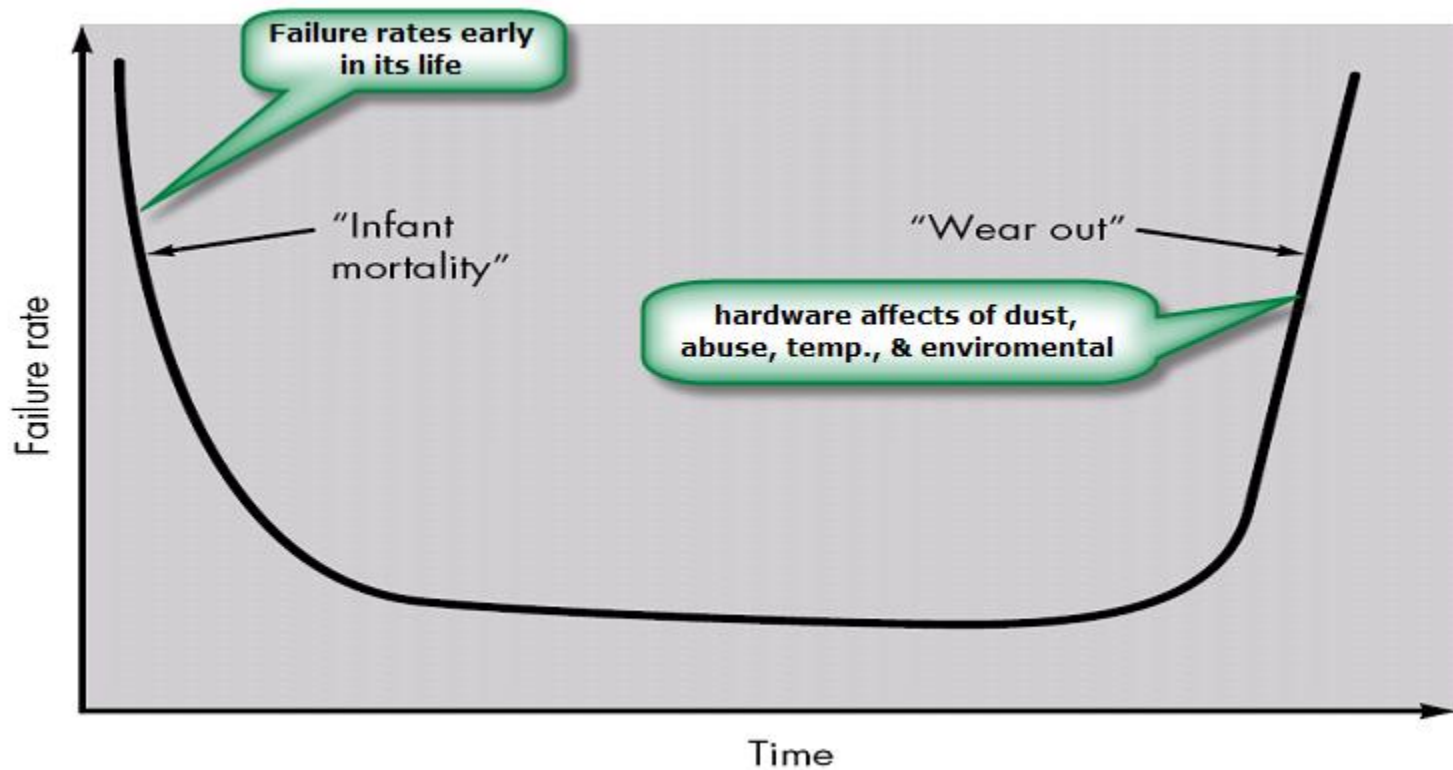
# Characteristics of software

- 1) Software is developed or engineered; it is not manufactured.
- 2) Software does not “wear out” but it does deteriorate.
- 3) Software continues to be custom built, as industry is moving toward component based construction.

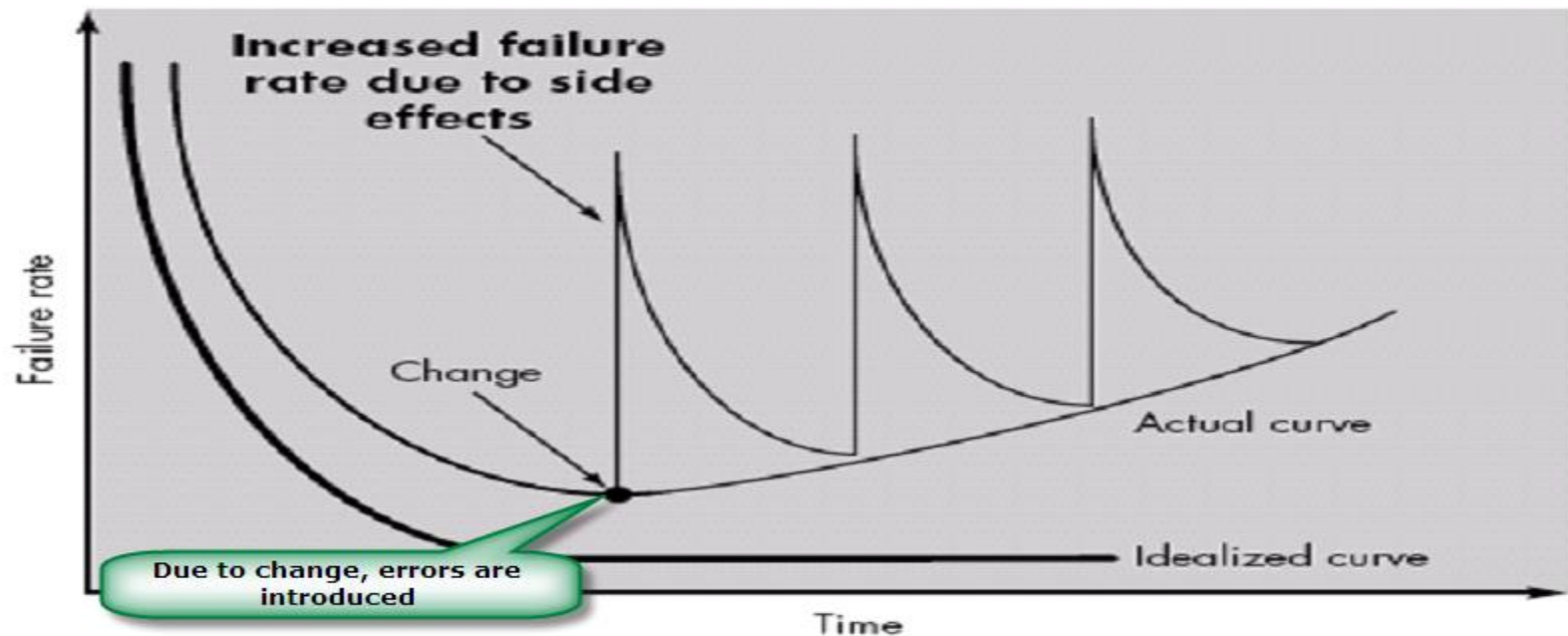
# Manufacturing vs. Development

- Once a hardware product has been manufactured, it is difficult or impossible to modify. In contrast, software products are routinely modified and upgraded.
- In hardware, hiring more people allows you to accomplish more work, but the same does not necessarily hold true in software engineering.
- Unlike hardware, software costs are concentrated in design rather than production.

# Failure curve for Hardware



# Failure curve for Software



When a hardware component wears out, it is replaced by a spare part.

There are no software spare parts. Every software failure indicates an error in design or in the process through which design was translated into machine executable code. Therefore, software maintenance involves considerably more complexity

# Component Based vs. Custom Built

- Hardware products typically employ many standardized design components.
- Most software continues to be custom built.
- The software industry does seem to be moving (slowly) toward component-based construction

# Types of software

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



## System Software:

- System software is a collection of programs written to service other programs.
- It 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.

Ex. Compilers, operating system, drivers etc.

## Application Software :

- Application software consists of standalone programs that solve a specific business need.
- Application software is used to control the business function in real-time.

## Engineering / Scientific software:

- Characterized by "number crunching" algorithms.
- Applications range from astronomy to volcano logy, from automotive stress analysis to space shuttle orbital dynamics, and from molecular biology to automated manufacturing.

Ex. Computer Aided Design (CAD), system stimulation etc.

## **Embedded Software:**

- It resides in read-only memory and is used to control products and systems
- Embedded software can perform limited and esoteric functions.

**Ex.** keypad control for a microwave oven.

## **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.

**Ex.** Word processing, spreadsheet, CG, multimedia, etc.

## **Web Applications:**

- Web apps can be little more than a set of linked hypertext files.
- It evolving into sophisticated computing environments that not only provide standalone features, functions but also integrated with corporate database and business applications.

## **Artificial Intelligence software**

- AI software makes use of non-numerical algorithms to solve complex problems that are not amenable to computation or straightforward analysis

**Ex.** Robotics, expert system, game playing, etc.

# Software Myths

- Software myths are beliefs about software and the process used to build it.

## (1) Management Myths:

- **Myth 1:** 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 all of these questions is no.

# Software Myths

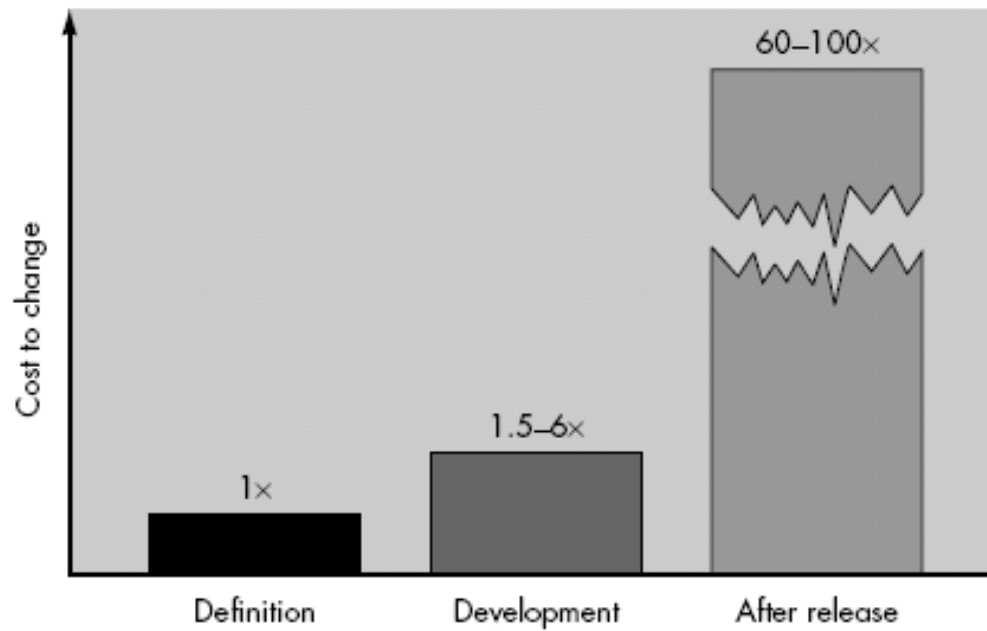
- **Myth 2:** If we get behind schedule, we can add more programmers and catch up.
- **Reality:**
- Software development is not a mechanistic process like manufacturing.
- As new people are added, people who were working must spend time educating the newcomers, thereby reducing the amount of time spent on productive development effort.
- People can be added but only in a planned and well-coordinated manner.
- **Myth 3:**
- If I 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 projects internally, it will invariably struggle when it outsources software projects.

## (2) Customer Myths:

- **Myth 1:**
- *A general statement of objectives is sufficient to begin writing programs—we can fill in the 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.

- **Myth 2:**
- Software requirements continually change, but change can be easily accommodated because software is flexible.
- **Reality:**
- It is true that software requirements change, but the impact of change varies with the time at which it is introduced.
- When requirements changes are requested early the cost impact is relatively small.
- However, as time passes, the cost impact grows rapidly—resources have been committed, a design framework has been established, and change can cause disruption that requires additional resources and major design modification.

# Customer Myths



### (3) Practitioner's Myths:

- **Myth 1:**
- Once we write the program and get it to work, our job is done.
- **Reality:**
- Industry data indicate that between 60 and 80 percent of all effort expended on software will be expended after it is delivered to the customer for the first time.
- **Myth 2:**
- Until I get the program “running” I have no way of assessing its quality.
- **Reality:**
- One of the most effective software quality assurance mechanisms can be applied from the inception of a project—the technical review.
- Software reviews are a “quality filter” that have been found to be more effective than testing for finding certain classes of software defects.



- **Myth 3:**

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

- **Reality:**

- A working program is only one part of a software configuration that includes many elements.
- A variety of work products (e.g., models, documents, plans) provide a foundation for successful engineering and, more important, guidance for software support.

- **Myth 4:**

- Software engineering will make us create voluminous and unnecessary documentation and will invariably slow us down.

- **Reality:**

- Software engineering is not about creating documents.
- It is about creating a quality product.
- Better quality leads to reduced rework.
- Reduced network results in faster delivery times.

# Software process

- **What? A software process** — as a framework for the tasks that are required to build high-quality software.
- **Who?** Managers, software engineers, and customers.
- **Why?** Provides stability and control to an organization otherwise disordered activity.
- **Steps?** A handful of activities are common to all software processes, details.
- **Work product?** Programs, documents, and data.

# What is software engineering?

**“The application of systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software”**

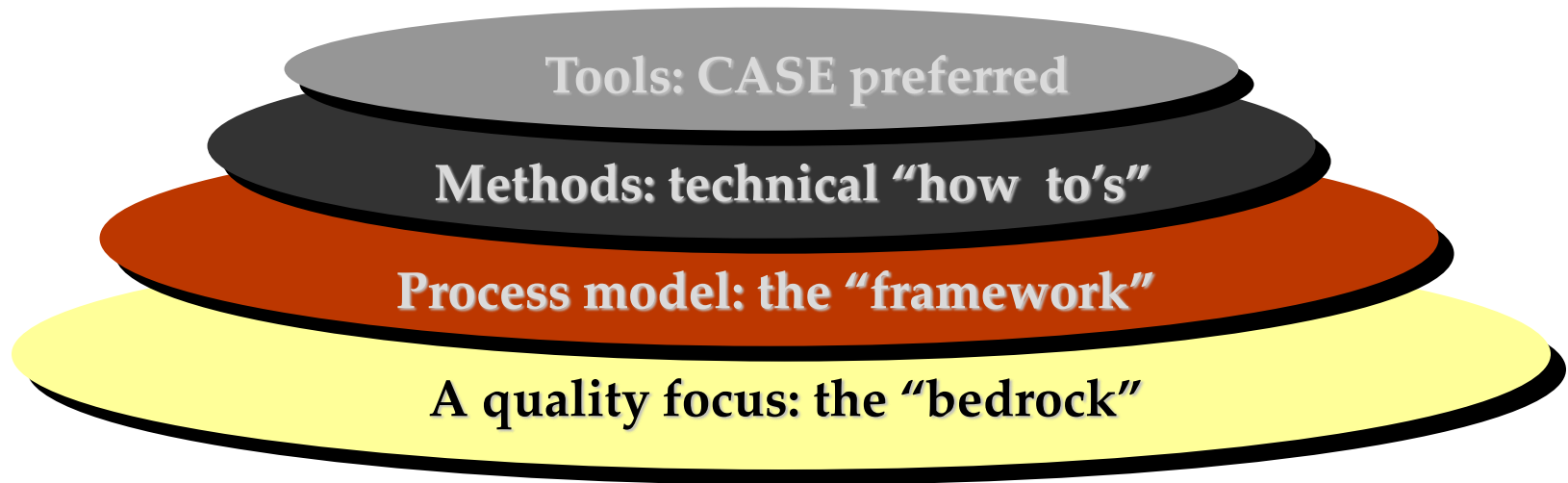
- Its a discipline that is concerned with all aspects of software production.
- Software Engineering is the science and art of building (designing and writing programs) a software systems that are:
  - 1) on time
  - 2) on budget
  - 3) with acceptable performance
  - 4) with correct operation

# What is software engineering?

- Software engineers should adopt systematic and organized approach to their work, Use appropriate tools and techniques depending on the problem to be solved and the development constraints and the resources available.
- Apply Engineering Concepts to developing Software.
- Challenge for Software Engineers is to produce high quality software with finite amount of resources & within a predicted schedule.

# Software Engineering – Layered Technology

## Layered Technology



# A quality Focus

- Every organization is rest on its commitment to quality.
- Total quality management, Six Sigma, or similar continuous improvement culture and it is this culture ultimately leads to development of increasingly more effective approaches to software engineering.
- The bedrock that supports software engineering is a quality focus.

## Process:

- It's a foundation layer for software engineering.
- It's define **framework** for a set of *key process areas* (KRA) for effectively manage and deliver quality software in a cost effective manner
- The processes define the tasks to be performed and the order in which they are to be performed

## Methods:

- It provide the technical **how-to's** for building software.
- Methods encompass a broad array of tasks that include requirements analysis, design, program construction, testing, and support.
- There could be more than one technique to perform a task and different techniques could be used in different situations.

## Tools:

- Provide automated or semi-automated support for the process, methods and quality control.
- 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)*

# Process framework

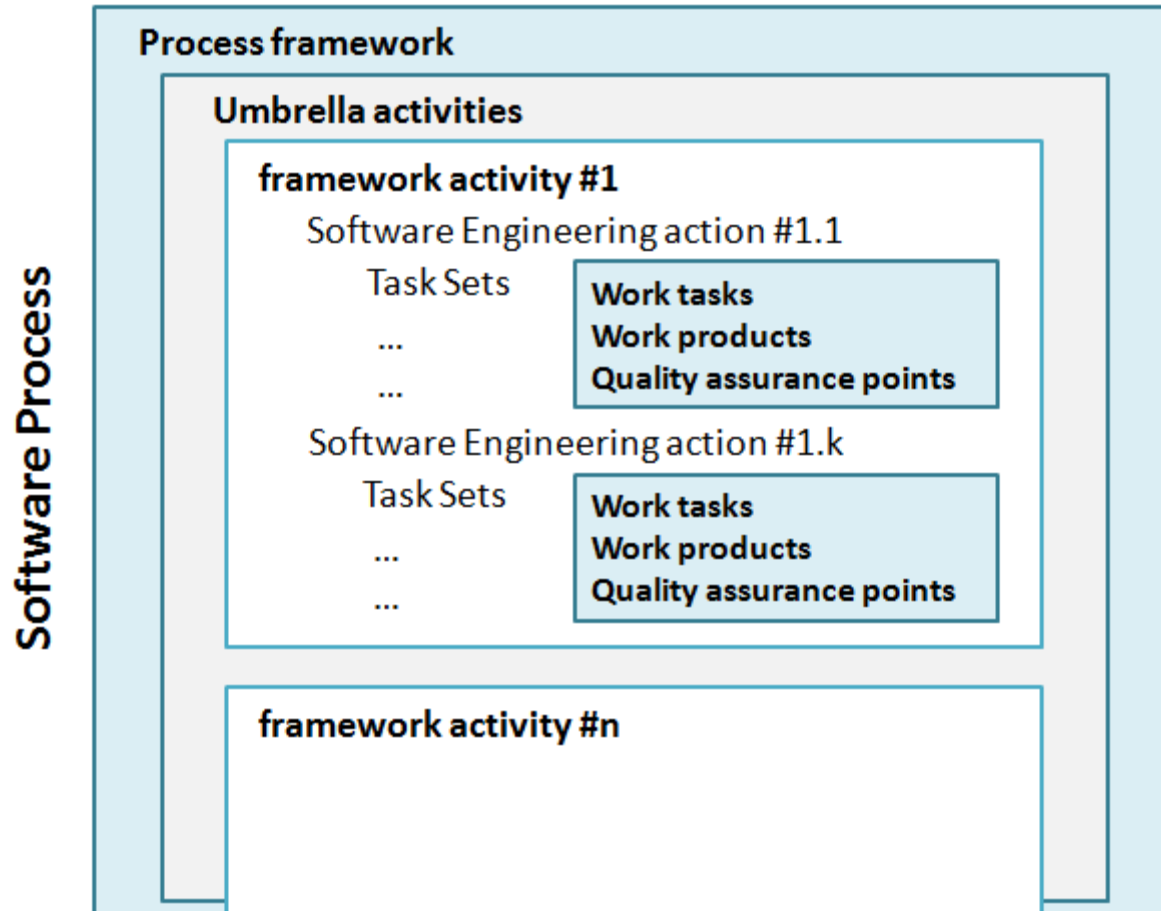
- **Why process :**
- A process defines who is doing what, when and how to reach a certain goal.
- To build complete software process.
- Identified a small number of framework activities that are applicable to all software projects, regardless of their size or complexity.
- It encompasses a set of umbrella activities that are applicable across the entire software process.
- Each framework activities is populated by a set for software engineering actions – a collection of related tasks.
- Each action has individual work task.

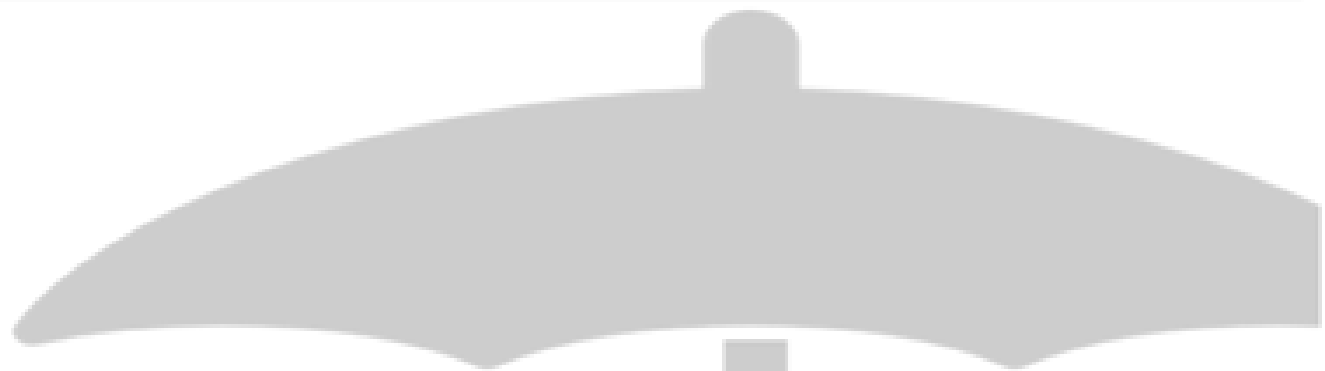


# Generic Process Framework Activities

- Communication:
  - Heavy communication with customers, stakeholders, team
  - Encompasses requirements gathering and related activities
- Planning:
  - Workflow that is to follow
  - Describe technical task, likely risk, resources will require, work products to be produced and a work schedule.
- Modeling:
  - Help developer and customer to understand requirements (Analysis of requirements) & Design of software
- Construction
  - Code generation: either manual or automated or both
  - Testing – to uncover error in the code.
- Deployment:
  - Delivery to the customer for evaluation
  - Customer provide feedback

# Process Framework





**Software project  
Tracking & Control**

**Risk  
Management**

**Software quality  
assurance**

**Technical Reviews**

**Measurement**

**Software Configuration Management**

**Reusability  
Management**

**Work product preparation and production**

# Umbrella Activities

- Software project tracking and control
  - Assessing progress against the project plan.
  - Take adequate action to maintain schedule.
- Formal technical reviews
  - Assessing software work products in an effort to uncover and remove errors before goes into next action or activity.
- Software quality assurance
  - Define and conducts the activities required to ensure software quality.
- Software configuration management
  - Manages the effects of change.
- Document preparation and production
  - Help to create work products such as models, documents, logs, form and list.
- Reusability management
  - Define criteria for work product reuse
  - Mechanisms to achieve reusable components.
- Measurement
  - Define and collects process, project, and product measures
  - Assist the team in delivering software that meets customer's needs.
- Risk management
  - Assesses risks that may effect that outcome of project or quality of product (i.e. software)

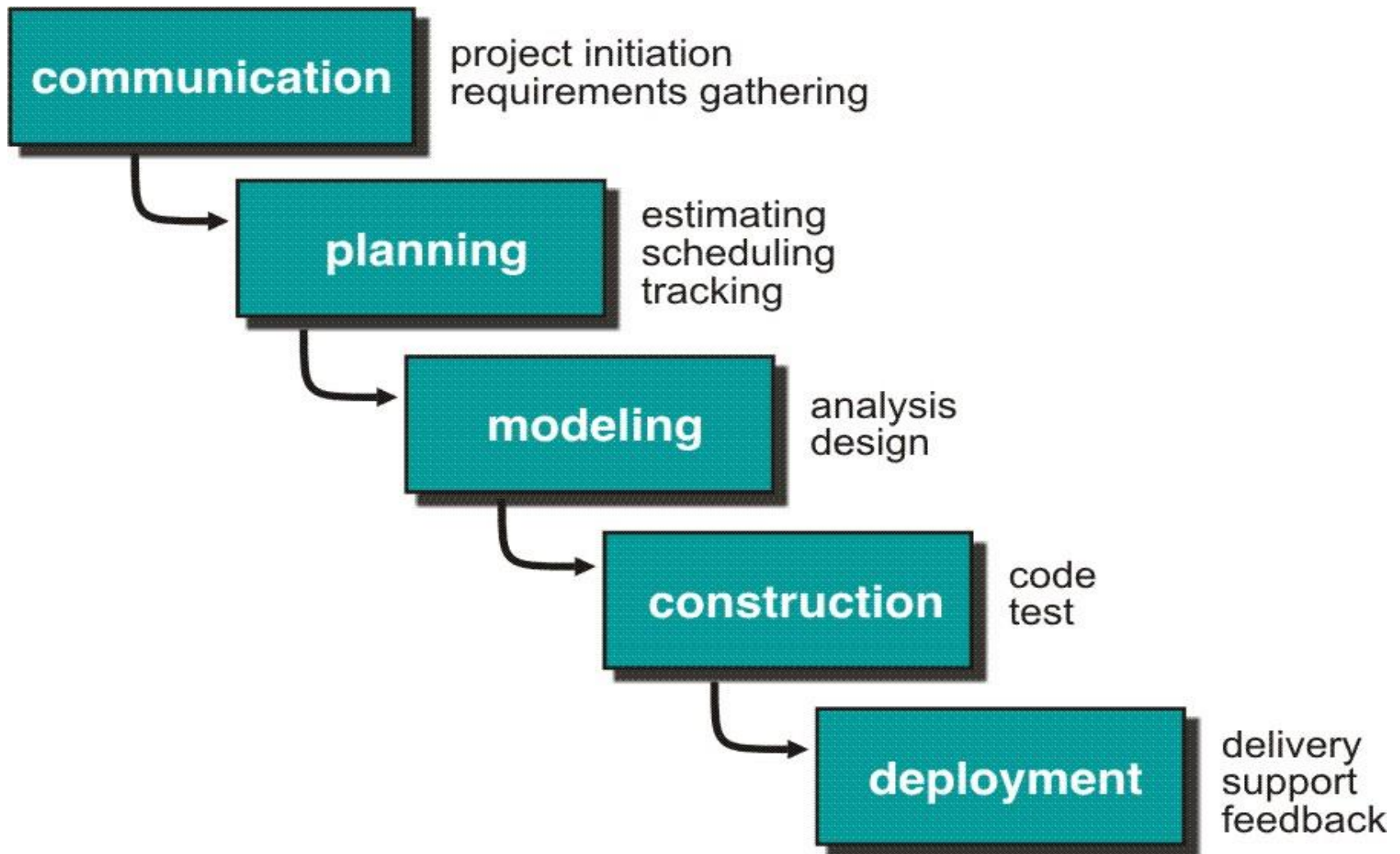
# Process Models

- **Process models prescribe a distinct set of activities, actions, tasks, milestones, and work products required to engineer high quality software.**
- **Process models are not perfect, but provide roadmap for software engineering work.**
- Software models provide stability, control, and organization to a process that if not managed can easily get out of control
- Software process models are adapted to meet the needs of software engineers and managers for a specific project.
- Why Models are needed?
  - Symptoms of inadequacy: the software crisis
    1. scheduled time and cost exceeded
    2. user expectations not met
    3. poor quality

# Different process models

- Waterfall Model (Linear Sequential Model)
- Incremental Process Model
  - 1) Incremental model 2) RAD model
- Evolutionary process model
  - 1) Prototyping Model
  - 2) The Spiral Model
  - 3) Concurrent development model
- Specialized process model
  - 1) Component based development model

# Water fall model (Classic Life Cycle or linear sequential model)



- Requirement Analysis and Definition: What - The systems services, constraints and goals are defined by customers with system users.
- Scheduling tracking -
  - Assessing progress against the project plan.
  - Require action to maintain schedule.
- System and Software Design: How -It establishes and overall system architecture. Software design involves fundamental system abstractions and their relationships.
- Integration and system testing: The individual program unit or programs are integrated and tested as a complete system to ensure that the software requirements have been met. After testing, the software system is delivered to the customer.
- Operation and Maintenance: Normally this is the longest phase of the software life cycle. The system is installed and put into practical use. Maintenance involves correcting errors which were not discovered in earlier stages of the life-cycle.



- When **requirements** for a problems are **well understood** then this model is used in which **work flow** from communication to deployment is **linear**
- This Model also called as the **Classic life cycle** or **linear sequential model**.
- **When to use ?**
- Requirements are very well known, clear and fixed
- Product definition is stable
- Technology is understood
- There are no ambiguous (unclear) requirements
- Ample (sufficient) resources with required expertise are available freely
- Short project

# The Waterfall Model cont.

- **Advantages**
- **Simple to implement** and manage
- **Drawbacks**
- **Unable to accommodate changes** at later stages, that is required in most of the cases.
- **Working version is not available** during development. Which can lead the development with major mistakes.
- **Deadlock can occur** due to delay in any step.
- Not suitable for large projects.

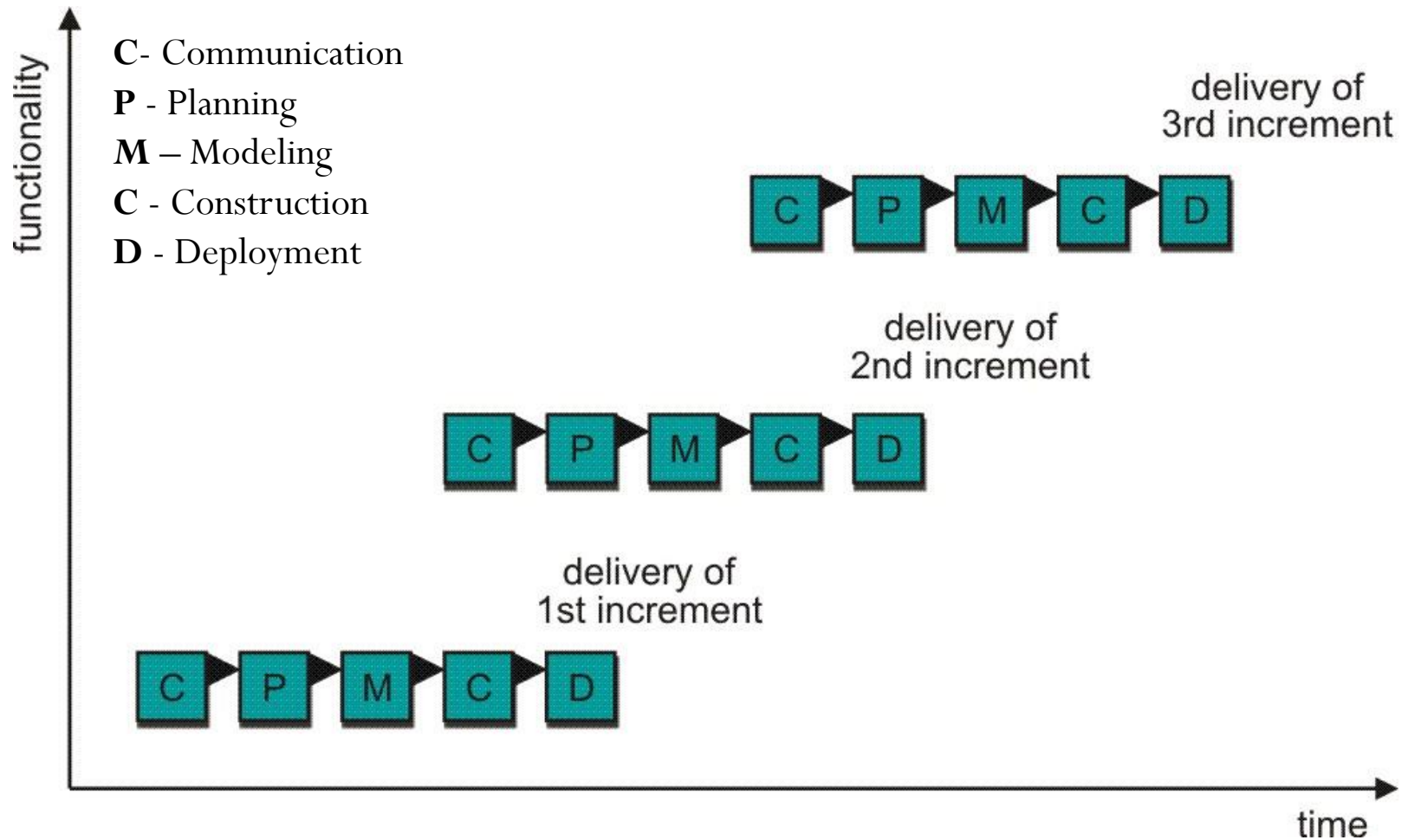
# The Incremental model

- Rather than deliver the system as a single delivery, the development and delivery is broken down into increments with each increment delivering part of the required functionality.
- First Increment is often core product
  - Includes basic requirement
  - Many supplementary features (known & unknown) remain undelivered
- A plan of next increment is prepared
  - Modifications of the first increment
  - Additional features of the first increment
- It is particularly useful when **enough staffing** is not available for the whole project
- Increment can be planned to **manage technical risks.**
- Incremental model focus more on delivery of operation product with each increment.

# The Incremental model

- User requirements are prioritised and the highest priority requirements are included in early increments.
- Once the development of an increment is started, the requirements are frozen though requirements for later increments can continue to evolve.
- Customer value can be delivered with each increment so system functionality is available earlier.
- Early increments act as a **prototype** to help elicit requirements for later increments.
- **Lower risk of overall project failure.**
- **The highest priority system services tend to receive the most testing.**

# The Incremental model



Delivers software in small but usable pieces, each piece builds on pieces already delivered

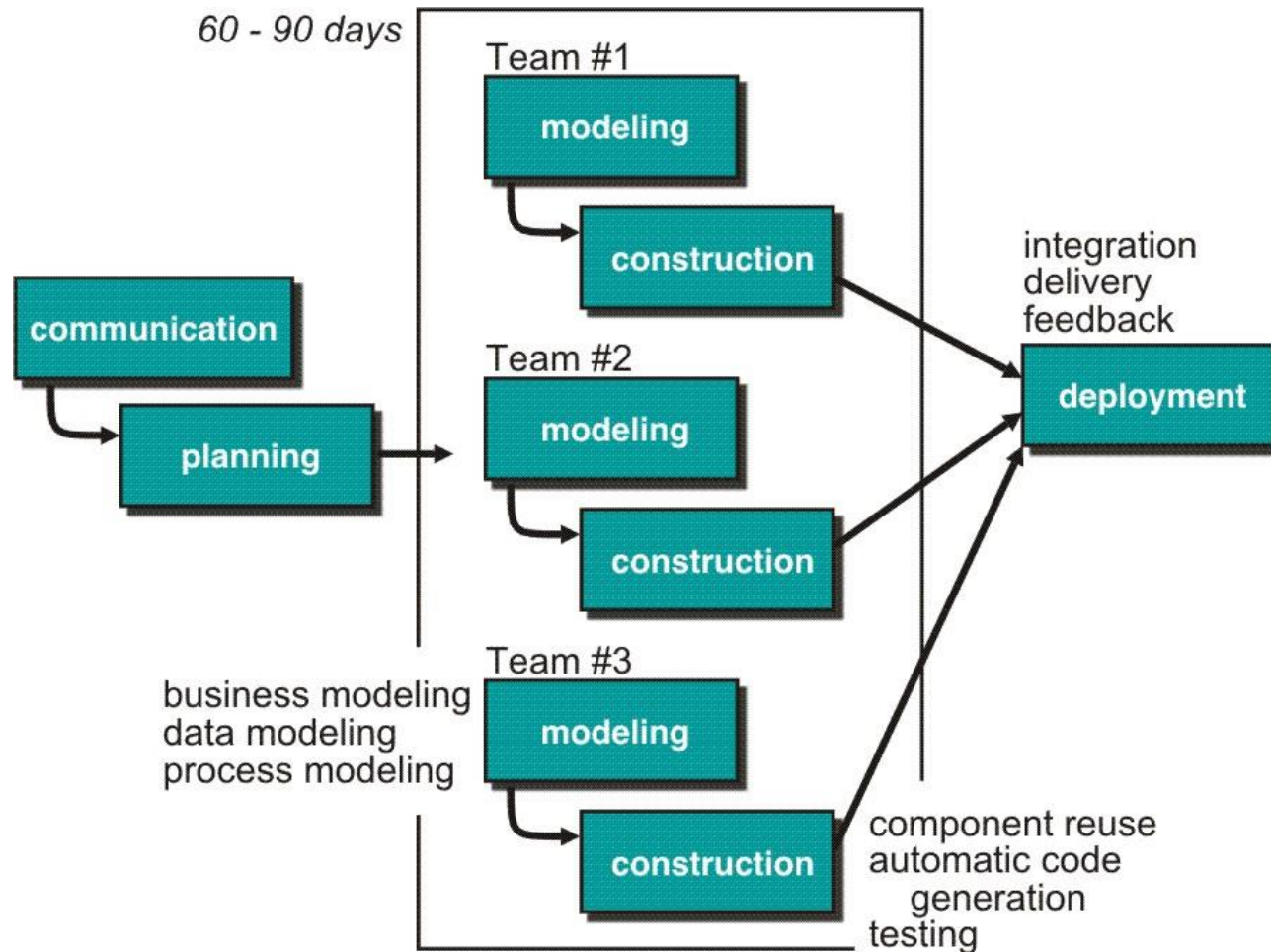
# The Incremental model

- **When to Use ?**
- When the **requirements** of the **complete** system are clearly **defined** and understood but **staffing is unavailable** for a **complete implementation** by the business deadline.
- **Advantages**
- Generates **working software quickly** and early during the software life cycle.
- It is **easier to test** and debug during a smaller iteration.
- **Customer** can **respond** to each built.
- **Lowers initial** delivery **cost**.
- **Easier** to **manage risk** because risky pieces are identified and handled during iteration.

# The Incremental model

- **Disadvantage**
- Final cost may be increase
- After increment if changes are demanded by customers then may cause serious problem in system architecture
- Needs very clear and complete planning before breaking whole system in to small increments

# RAD Model





# RAD Model

- **Communication** – to understand business problem.
- **Planning** – multiple s/w teams works in parallel on diff. system.
- **Modeling** –
  - **Business modeling** – Information flow among business is working.  
Ex. What kind of information drives?  
Who is going to generate information?  
From where information comes and goes?
  - **Data modeling** – Information refine into set of data objects that are needed to support business.
  - **Process modeling** – Data object transforms to information flow necessary to implement business.

# RAD Model

- **Construction** – it highlighting the use of pre-existing software component.
- **Deployment** – Deliver to customer basis for subsequent iteration.
- RAD model emphasize a **short development cycle**.
- **“High speed”** edition of linear sequential model.
- If requirement are well understood and project scope is constrained then it enable development team to create “fully functional system” within a very short time period.

# RAD Model

- If application is modularized (“Scalable Scope”), each major function to be completed in less than three months.
- Each major function can be addressed by a separate team and then integrated to form a whole.

## **Advantages:**

- Short time product can be developed
- Support reusability components
- Encourage customer feedback
- Module integration done at initial stage so resolve number of integral issues.
- Quick initial review possible.

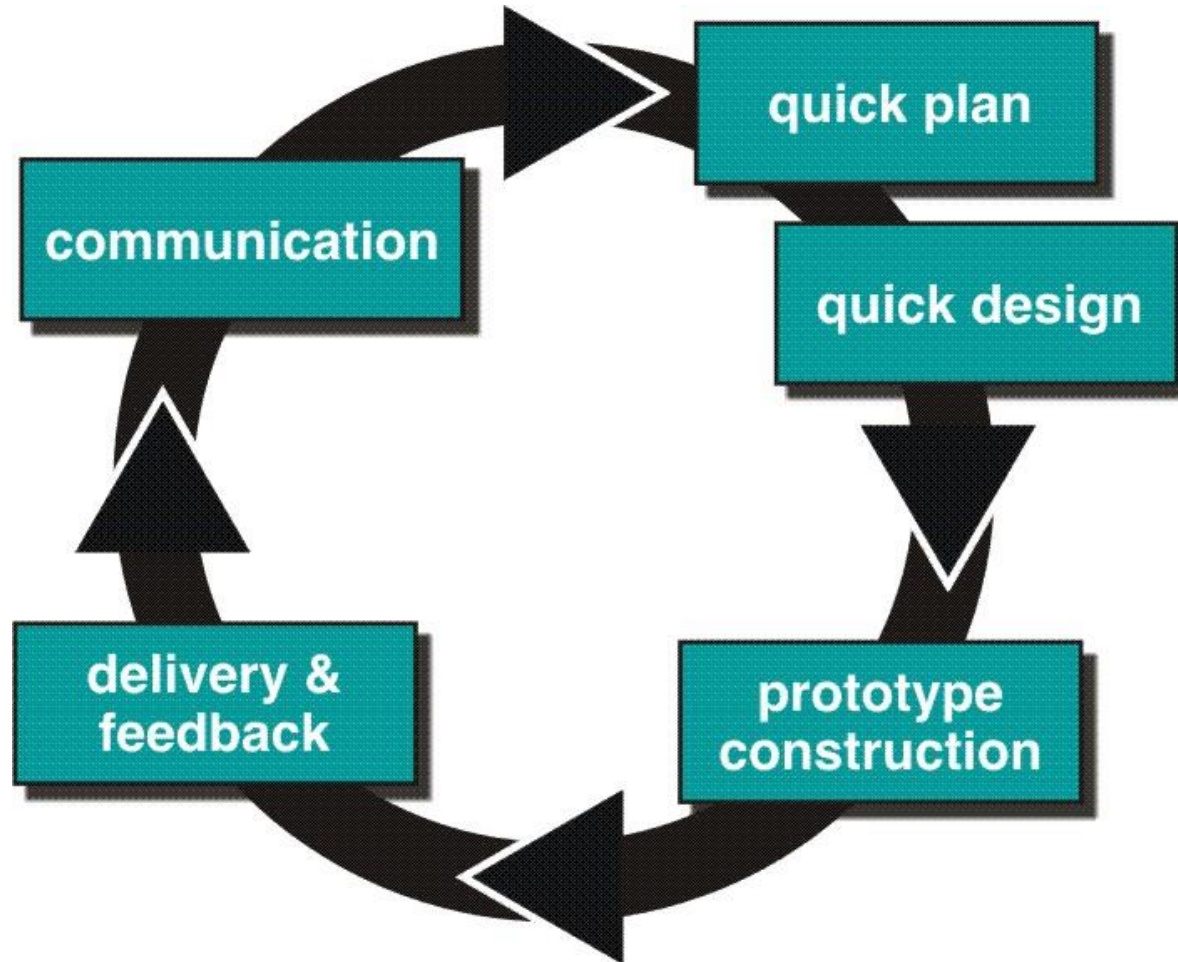
# RAD Model

## **Drawback:**

- For large but scalable projects
  - RAD requires sufficient human resources
- Projects fail if developers and customers are not committed in a much shortened time-frame
- Problematic if system can not be modularized
- Not appropriate when technical risks are high( heavy use of new technology)

# Evolutionary Process Models :

## Prototyping



- **Best approach when:**
  - Objectives defines by customer are general but does not have details like input, processing, or output requirement.
  - Developer may be unsure of the efficiency of an algorithm, operating system or the form that human machine interaction should take.
- It can be used as **standalone process model**.
- Model assist software engineer and customer to better understand what is to be built when **requirement are fuzzy**.
- Prototyping start with communication, between a customer and software engineer to define overall objective, identify requirements and make a boundary.
- Going ahead, planned quickly and modeling (software layout visible to the customers/end-user) occurs.
- Quick design leads to prototype construction.
- Prototype is deployed and evaluated by the customer/user.
- Feedback from customer/end user will refine requirement and that is how iteration occurs during prototype to satisfy the needs of the customer.

- ❑ **Prototype can be serve as “the first system”.**
- ❑ Both customers and developers like the prototyping paradigm.
  - Customer/End user gets a feel for the actual system
  - Developer get to build something immediately.

### **Problem Areas:**

- ❑ Customer cries foul and demand that “a few fixes” be applied to make the prototype a working product, due to that **software quality suffers** as a result.
- ❑ Developer often makes implementation in order to get a prototype working quickly without considering other factors in mind like OS, Programming language, etc.

Customer and developer both must be agree that the prototype is built to serve as a mechanism for defining requirement.

### **Advantages:**

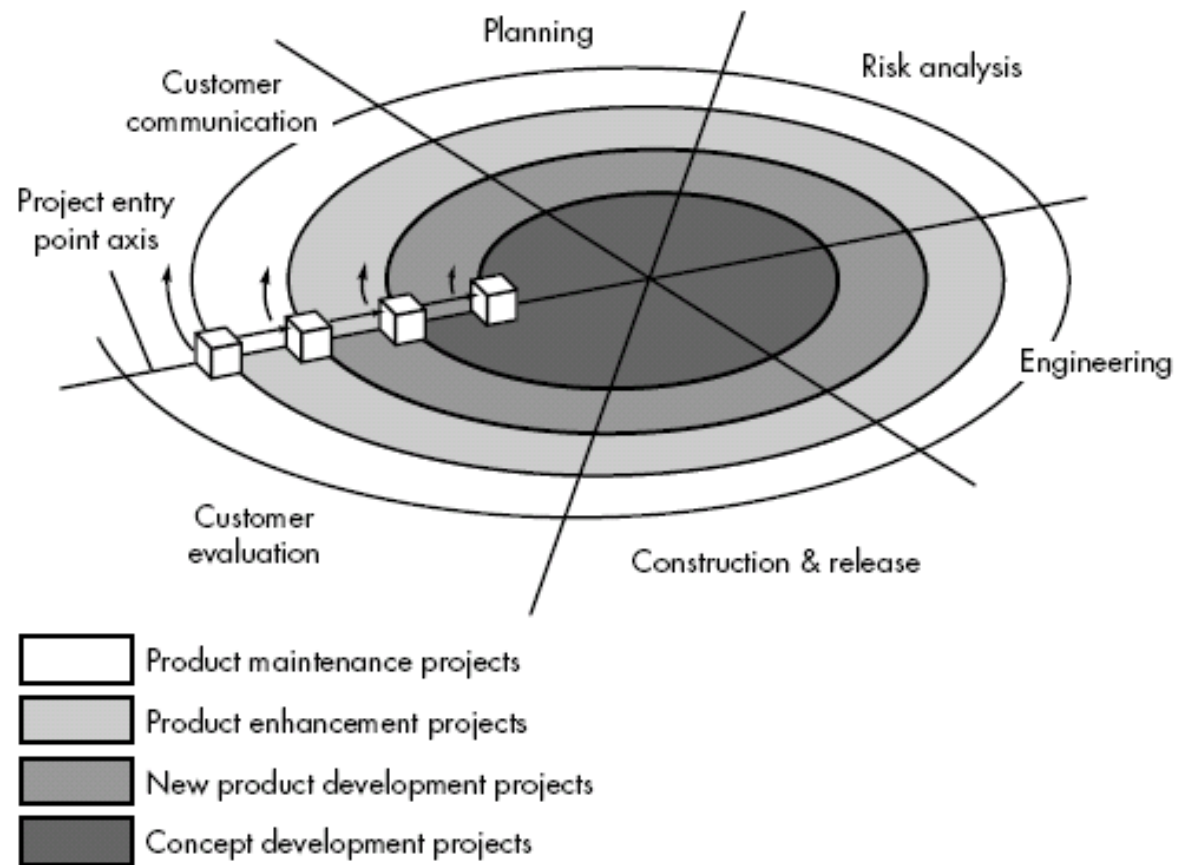
- **Users** are actively **involved** in the **development**
- Since in this methodology a working model of provided, the **users get a better understanding** being developed.
- **Errors** can be **detected** much **earlier**.

# Spiral Model

- ❑ It Couples iterative nature of **prototyping** with the controlled and systematic aspects of the **linear sequential model**.
- It provide potential for rapid development of increasingly more complete version of the software.
- Using spiral, software developed in as series of evolutionary release.
  - Early iteration, release might be on paper or prototype.
  - Later iteration, more complete version of software.
- Divided into framework activities (C,P,M,C,D). Each activity represent one segment.
- Evolutionary process begins in a clockwise direction, beginning at the center risk.
- First circuit around the spiral might result in development of a product specification. Subsequently, develop a prototype and then progressively more sophisticated version of software.
- Unlike other process models that end when software is delivered.
- It can be adapted to apply **throughout the life of software**.



# Spiral Model



# Spiral Model

## **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.

# Spiral Model

- Spiral models uses prototyping as a risk reduction mechanism but, more important, enables the developer to apply the prototyping approach at each stage in the evolution of the product.
- It maintains the systematic stepwise approach suggested by the classic life cycle but also incorporates it into an iterative framework activity.
- If risks cannot be resolved, project is immediately terminated

## Problem Area:

- It may be difficult to convince customers (particularly in contract situations) that the evolutionary approach is controllable.
- If a major risk is not uncovered and managed, problems will undoubtedly occur.

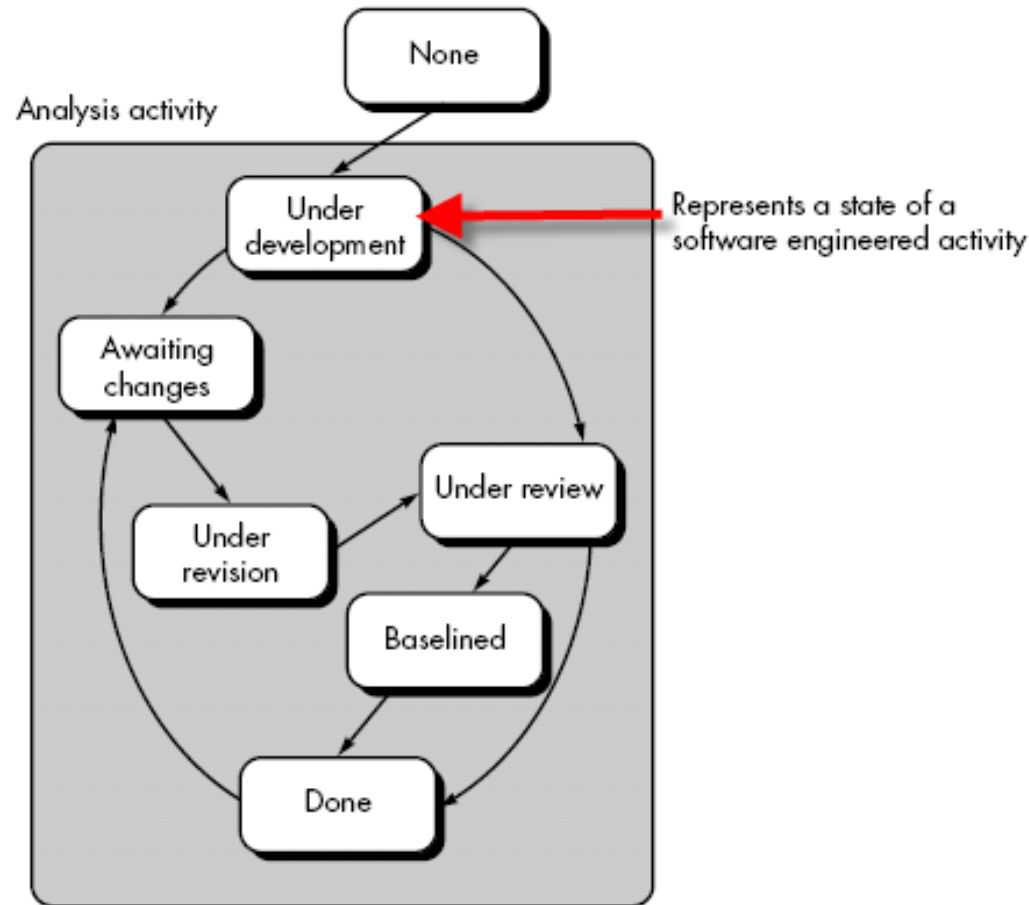
- **Advantages**

- High amount of risk analysis hence, **avoidance of Risk** is enhanced.
- **Strong approval** and **documentation** control.
- **Additional functionality** can be **added** at a later date.
- **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.

# Concurrent Development Model



- It represented schematically as series of major technical activities, tasks, and their associated states.
- It is often more appropriate for system engineering projects where different engineering teams are involved.
- The activity-modeling may be in any one of the states for a given time.
- All activities exist concurrently but reside in different states.

E.g.

- The *analysis* activity (existed in the **none** state while initial customer communication was completed) now makes a transition into the **under development** state.

- *Analysis* activity moves from the **under development** state into the **awaiting changes** state only if customer indicates changes in requirements.
- Series of event will trigger transition from state to state.

E.g. During initial stage there was inconsistency in design which was uncovered. This will triggers the analysis action from the **Done** state into **Awaiting Changes** state.

- Visibility of current state of project
- It define network of activities
- Each activities, actions and tasks on the network exists simultaneously with other activities,actions and tasks.
- Events generated at one point in the process network trigger transitions among the states.

- Advantages:
- Use for all types of software development process.
- Clear picture of current state.
- Easy to understand and use.
- Immediate feedback from testing.
- Disadvantages:
- Require excellent updated communication between team members.
- SRS updated regular interval.
- Longer development cycle then expected compared to planned due to dependent component developments.

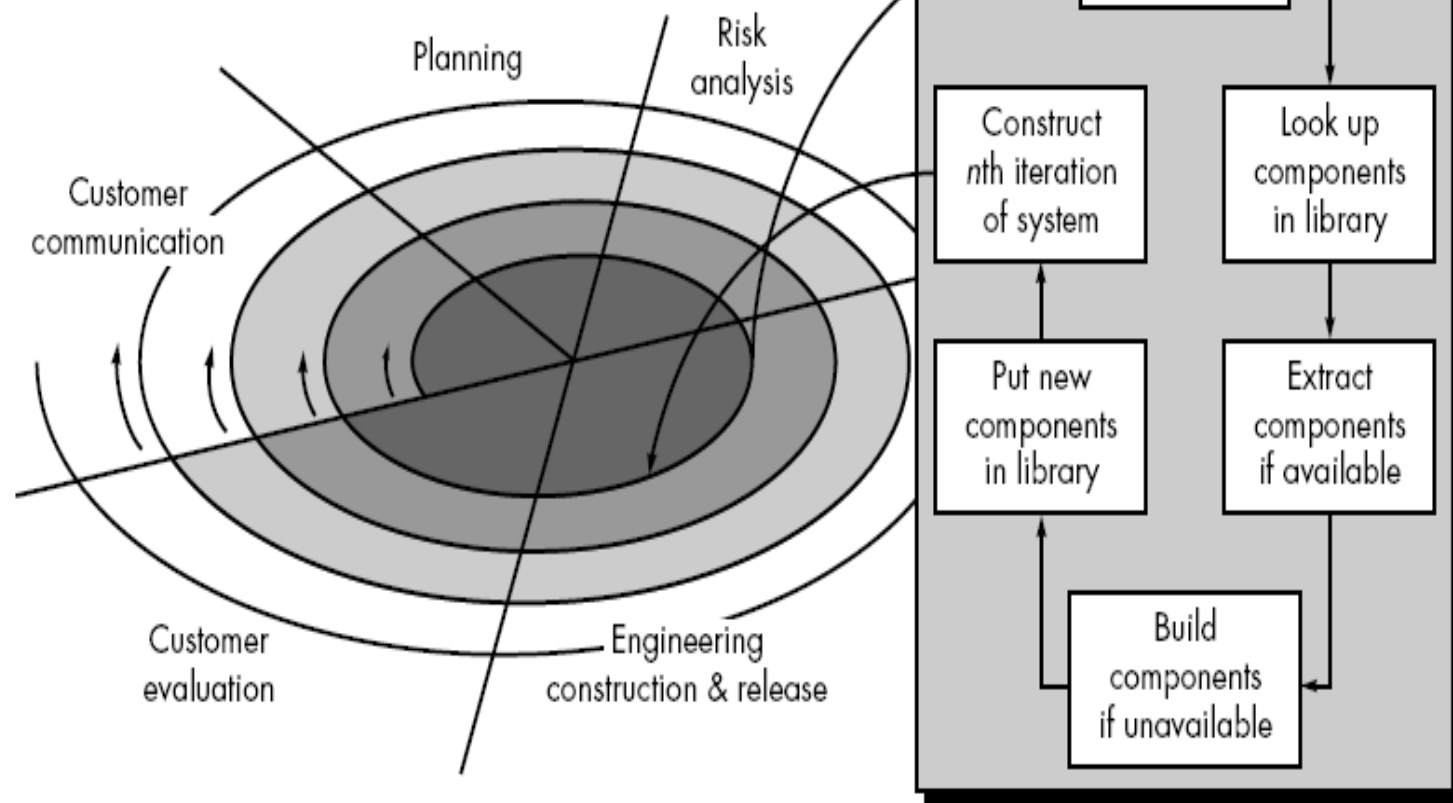


# Component Based Development

- component-based development (CBD) model incorporates many of the characteristics of the spiral model.
- It is evolutionary by nature and iterative approach to create software.
- CBD model creates applications from prepackaged software components (called *classes*).

**FIGURE 2.11**

Component-based development



- Modeling and construction activities begin with identification of candidate components.
- Classes created in past software engineering projects are stored in a class library or repository.
- Once candidate classes are identified, the class library is searched to determine if these classes already exist.
- If class is already available in library extract and reuse it.
- If class is not available in library, it is engineered or developed using object-oriented methods.
- Any new classes built to meet the unique needs of the application.
- Now process flow return to the spiral activity.

- Advantages:
- CBD model leads to software reusability.
- Based on studies, CBD model leads to 70 % reduction in development cycle time.
- 84% reduction in project cost.
- Productivity is very high.
- Disadvantages:
- Difficult to find which component should develop in particular function.
- Late discovery of the errors due to architecture mismatch.

# Difference between incremental and waterfall

<b>WATERFALL MODEL</b>	<b>INCREMENTAL MODEL</b>
Need of Detailed Documentation in waterfall model is Necessary.	Need of Detailed Documentation in incremental model is Necessary but not too much.
There is long waiting time for running software in waterfall model.	There is short waiting time for running software in incremental model.
Flexibility to change in waterfall model is Difficult.	Flexibility to change in incremental model is Easy.
Testing is done in waterfall model after completion of all coding phase.	Testing is done in incremental model after every iteration of phase.
Returning to previous stage/phase in waterfall model is not possible.	Returning to previous stage/phase in incremental model is possible.

<b>WATERFALL MODEL</b>	<b>INCREMENTAL MODEL</b>
In waterfall model large team is required.	In incremental model large team is not required.
In waterfall model overlapping of phases is not possible.	In incremental model overlapping of phases is possible.
There is only one cycle in waterfall model.	There is multiple development cycles take place in incremental model.

# Difference between prototype and waterfall

Waterfall	Prototype
Client can only preview the system only after the final version of the software is developed because there is no feed back loop.	Client have a preview of the system from the "quick design" and the prototype developed early at the of the process.
Developers encounter a freezing requirement where they are not allow to modify the requirements or specification of the previous phase until the next iteration.	Developers can refine or add requirements and specification to the system after the prototype is built.
The complexity of an error increases because of the nature of the model; each phase is sequential of the other.	The complexity of an error is low because the prototype enables the developer to detect any deficiency early at the process.
This model is easy to understand.	This model is not easy to understand.
Complete program analysis.	Incomplete program Analysis.

# Differentiate Prototype & RAD

Prototype Model	RAD Model
Using Prototype model, a product cannot be developed in a short period of time as the requirements are refined in later prototypes.	Using the RAD approach, a product can be developed within a very short period of time.
It can be used, even if the number of staff is less in the beginning.	It can be used, if a sufficient number of staff is available.
In this approach, quick initial reviews are not possible.	In this approach, quick initial reviews are possible.
The prototype model approach is suitable for high-risk projects.	The RAD model is not suitable when technical risks are high.
It doesn't support automatic code generation.	It results in minimal code writing as it supports automatic code generation.
User Involvement High.	User Involvement Only at the beginning.



# Difference between spiral and prototype

BASIS OF COMPARISON	SPIRAL MODEL	PROTOTYPE MODEL
<b>Description</b>	Spiral model is a risk-driven software development process model. Based on the unique risk patterns of a given project, the spiral model guides a team to adopt elements of one or more process models such as incremental, waterfall or evolutionary prototyping.	A prototyping model is a systems development method in which a prototype (an initial approximation of a final system or product) is built, tested and then reworked as necessary until an acceptable prototype is finally achieved from which the complete system or product can now be developed.

BASIS OF COMPARISON	SPIRAL MODEL	PROTOTYPE MODEL
<b>Cost</b>	Cost effective quality improvement is not possible in spiral model.	In prototype model, cost effective quality improvement is very much possible.
<b>Quality Improvement Cost</b>	Improvement of quality of the project may increase the cost of the product.	Improvement of quality does not have effects on cost of the product.
<b>Suitability</b>	It is suitable when the customer specification requirements are clear.	It is a trial-and-error kind of model, only suitable when the specification requirement of the customer are not clear and are supposed to change.

BASIS OF COMPARISON	SPIRAL MODEL	PROTOTYPE MODEL
<b>Alternative Name</b>	Spiral model is also referred to as meta model.	Prototype mode can also be referred to as rapid or closed ended prototyping.
<b>Risk Analysis</b>	In spiral model, thorough risk analysis of risk and alternative solution is undertaken.	Prototype mode does not give due emphasis on risk analysis and alternative solutions.
<b>Customer Evaluation</b>	In spiral model there is no continuous customer interaction. Customer interaction comes at the tail end of the project.	In prototype model, customer interaction is continuous until the prototype is approved.

BASIS FOR COMPARISON	WATERFALL MODEL	SPIRAL MODEL
Basic	Model follows a sequential approach.	Model is based on an evolutionary approach.
Process involves	Stages or phases	Task regions
Requirement	Needs a thorough understanding of requirements from the beginning.	Requirements can be added up in the new iterations if required.
Risk management	Risks can only be detected during the completion of the model.	Risks are identified and rectified earlier.
Working model of the product	Can only be generated in the end.	Each iteration produces a working model.
Preferred by	Customers	Developers
Suitable for	Small systems	Large systems

# Assignment

SR NO	Questions
1	What is Software Engineering? What is the role of software engineer? Compare Hardware and Software product characteristics.
2	Explain Software Engineering as a Layered Technology.
3	Explain software myth for customers, practitioners and project manager. What is reality in each case?
4	Describe generic view of software engineering or List and explain very briefly various activities of software engineering process framework or Explain Umbrella activities and its role in SDLC.
5	Distinguish between a program and a software product.
6	What do you mean by software model? Explain each model in detail.
7	Differentiation between incremental process model and waterfall model.
8	Differentiation between prototype model and waterfall model.
9	Differentiation between prototype model and RAD model.
10	Differentiation between prototype model and spiral model.
11	Explain Spiral model with suitable example. Also explain how it differs from Software Prototyping model. Or Using example describes the spiral model.
12	Explain the process model which is used in situations where the requirements are well defined.
13	Explain the process model which is normally suits for development of large-scale software system.

# Extra

- introductions
- <https://nptel.ac.in/courses/106/101/106101061/>
- Models:
- <https://www.youtube.com/watch?v=kwsKr1MObxs>