# Software Engineering Fundamental (SEF )

by

# Er. Nirdosh Adhikari

# Objective of Course

- This course has intended to make familiar to **software development Life Cycle (SDLC)** and software development practice at the industrial level.
- Make Familiar to the **project management** concept.
- This course helps to make more familiar with **Structural analysis** and less with **Object Oriented Analysis**.

# SEF Book

- Pressman R.S Software Engineering a Practitioner Approach 5th Edition as Text book
- Booch Grady, Rumbaugh James, Jacobson Avar The Unified Modelling Language User Guide as Reference Book

# Object Oriented Concept

1.Class

2.Objects

3.Abstraction

4.Encaptulation

5.Inheritance

6.Polymorphism

# Software Engineering Vs Computer Engineering

**Computer Engineering deals with designing, developing, and developing overall computer system**

**Software Engineering deals with building and maintaining software**

# Software Vs Program

| Program | Software |
| --- | --- |
| 1. Usually small in size | 1.Large in size |
| 2. Single developer | 2. Team of developer |
| 3. Lack of proper user interface | 3.Well-designed interface |
| 4. Lack of Proper Documentation | 4.Proper Documentation |

# Definition of Software

Software consists of :-

- Set of instructions
- Data structures
- Descriptive information – both hard & soft copy

# Types of software

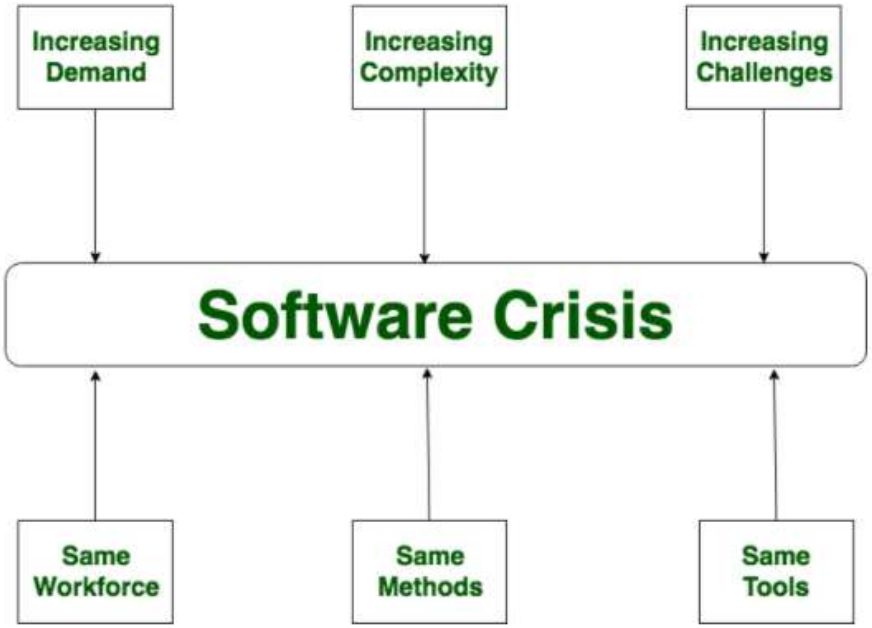
- System Software
- Application Software – package , trailered
- Utility Software
- Artificial Intelligence software
- Engineering/Scientific Software
- Open source software

# Characteristics of software

Some characteristics of software that is considerably different than hardware are:

1. Software is developed; it is not manufactured
2. Software doesn't wear out but it deteriorate
3. Can be custom build

# Software Crisis



# Causes of Software Crisis

- The cost of **owning and maintaining** software was as expensive as developing the software
- At that time **Projects was** running over-time
- At that time Software was very **inefficient**
- The quality of software was **low quality**
- Software often did not **meet requirements**

# Solution to crisis

- **Reduction** in software over-budget
- The **quality** of software must be high
- **Experience** working team member on software project
- Software must be **delivered** on time

# Software Engineering Fundamental

# CHAPTER ONE

- Definition of software
- Software Engineering vs. Computer Science
- Failure curve of  Hardware and software
- Software Myths
- Software development life cycle (SDLC)
- Software Development  methods

- **Software** (set of instruction + design diagrams+ necessary databases)
- **Software Engineering** : Creation and design of the software .
- **Computer Science**: Broad approach to the study of the principles and use of computers that covers both theory and application

# Characteristics of Software

software has characteristics that are considerably different than those of hardware:

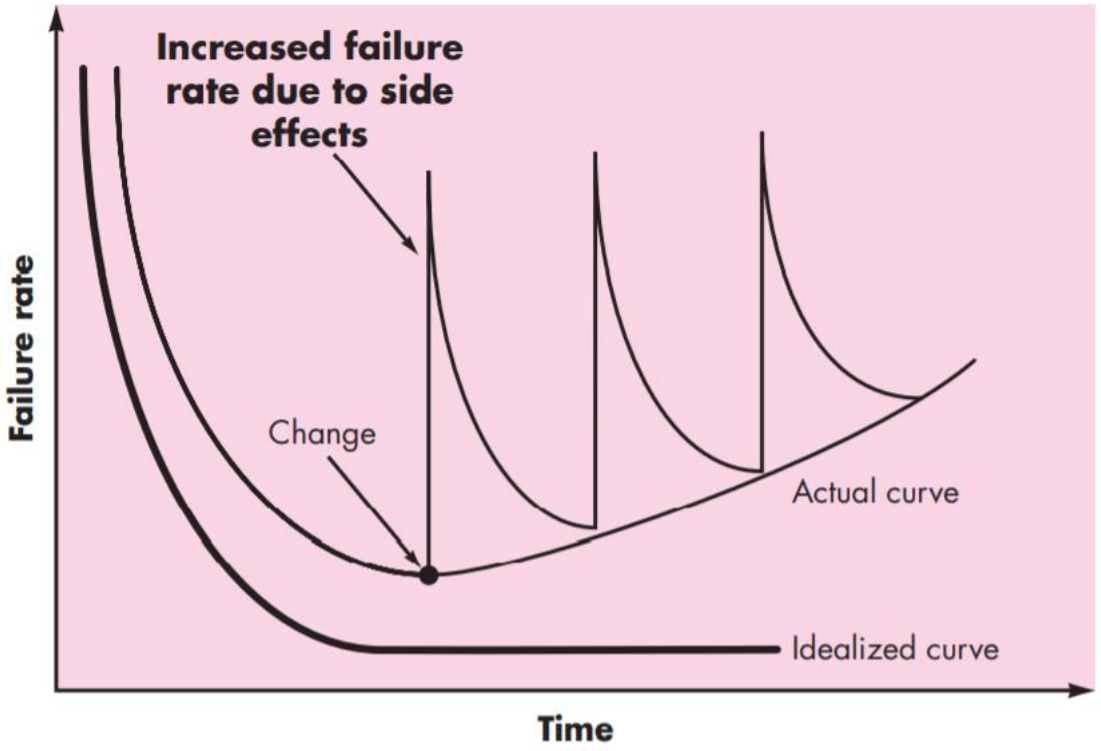
1. Software is developed ; it is not manufactured (*Both process focuses to achieve high quality, for hardware can introduce quality problems that are nonexistent for a software*)
2. Software doesn't "wear out."

3. Software is custom build

# Characteristics of software

**FIGURE 1.2**

Failure curves for software

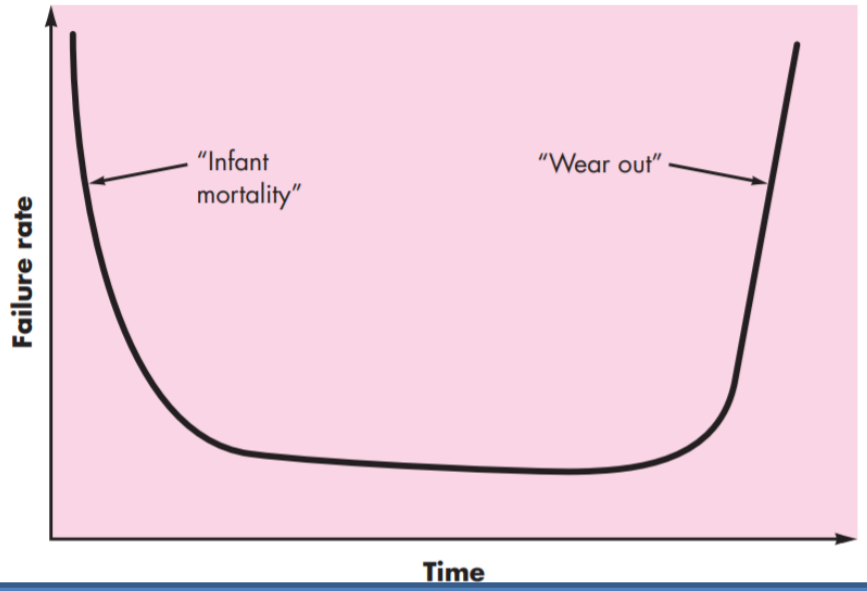



2. Software doesn't wear out but it does deteriorates

# Characteristics of Software



**FIGURE 1.1**
Failure curve for hardware




2. Software doesn't wear out but hardware does

# Software Myths

## Management myths

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: Are software practitioners aware of its existence? Does it reflect modern software engineering practice? Is it complete? Is it adaptable*?

# Software Myths

## Management Myths

Myth :If we get behind schedule, we can add more programmers and catch up

*Reality :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*

# Software Myths

## Management Myth

Myth: 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.*

# Software Myths

**Customer myths**

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

*Reality: an ambiguous "statement of objectives" is a recipe for disaster.*

# Software Myths

**Customer**

Myth: Software requirements continually change, but change can be easily accommodated because software is flexible.

*Reality :When requirements changes are requested early (before design or code has been started), the cost impact is relatively small. However, as time passes, the cost impact grows rapidly*

# Software Myths

Practitioner's myths.

Myth: 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*

# Software Myths

Practitioner's myths

Myth: Until I get the program "running" I have no way of assessing its quality

Reality: Quality focuses from at the time of requirement analysis

# Software Myths

Myth: The only deliverable work product for a successful project is the working program

*Reality: A variety of work products (e.g., models, documents, plans) provide a foundation for successful engineering and, more important, guidance for software support*.

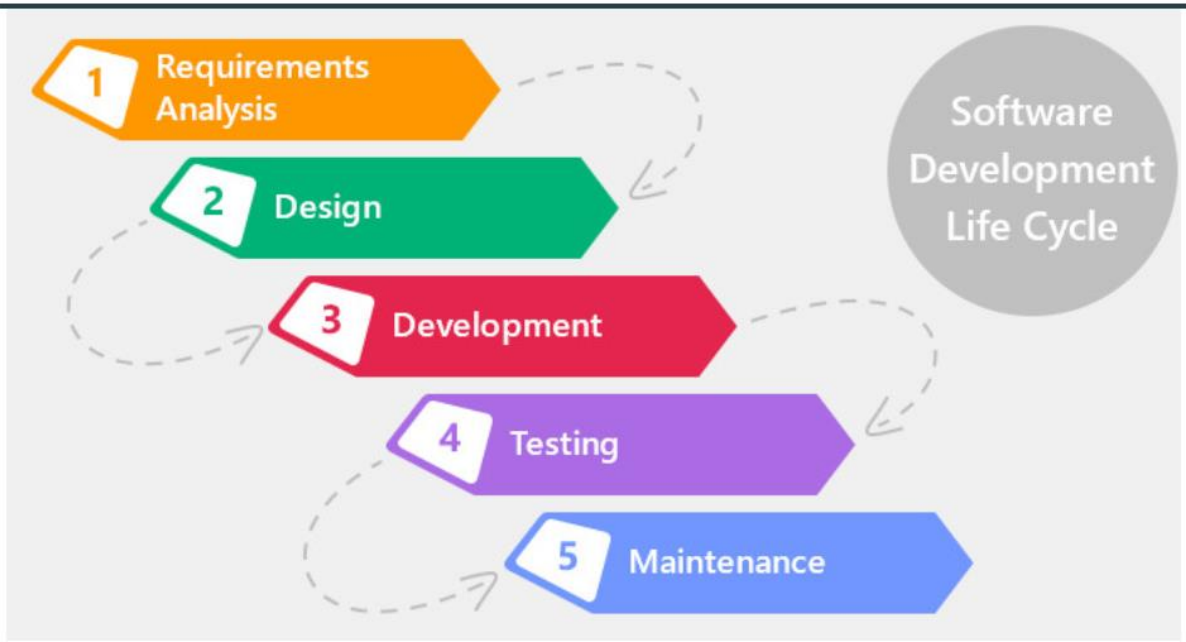
# Software Myths

## Practitioners Myths

Myth: 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. And reduced rework results in faster delivery times.

# Software Development Life Cycle (SDLC)




## What are the Software Development Life Cycle (SDLC) phases?

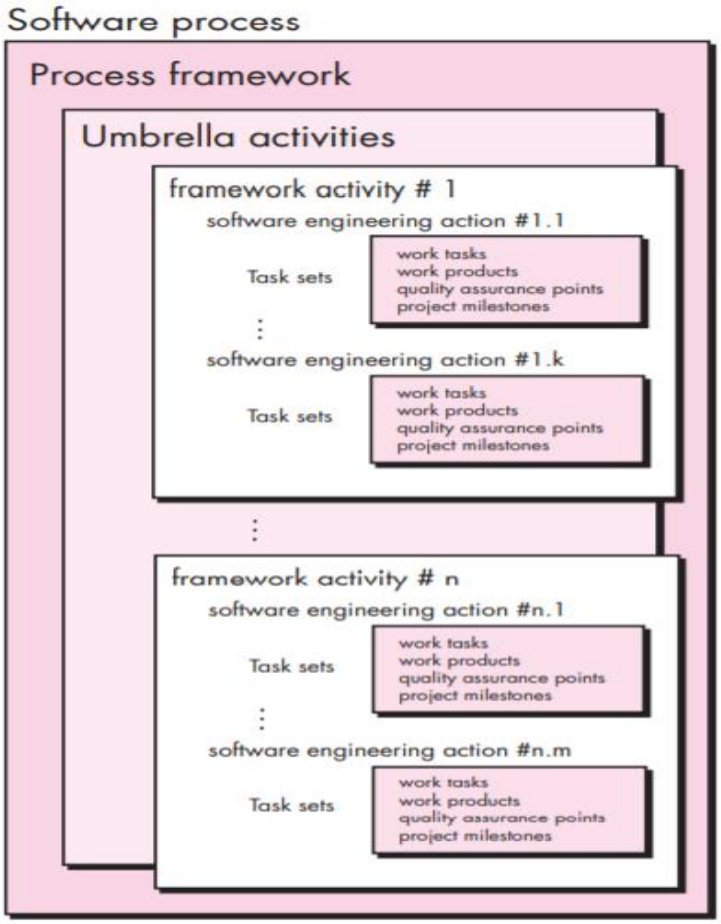
Published on August 23, 2017

# Software Process

## Software Process

- Software process is the **framework** for the **activities, action, and tasks** that are required to build high quality software.

# Process Framework



# Umbrella Activities

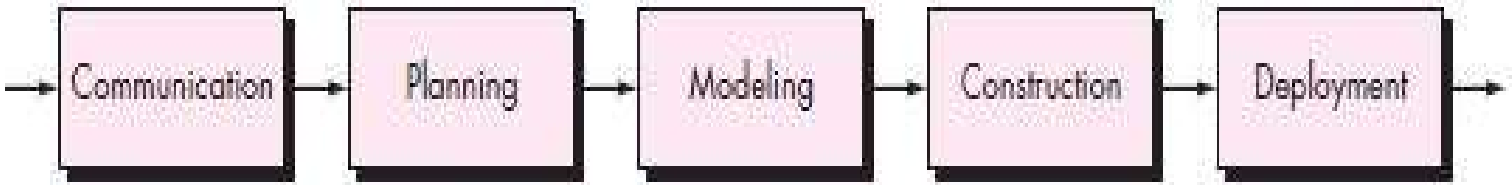
1. Software Project Management
2. Formal Technical Review
3. Software Quality Assurance
4. Software Configuration Management
5. Documentation
6. Reusability Management
7. Measurement
8. Risk Management

# Generic Process Model

- Communication
- Planning
- Modeling
- Construction
- Deployment

# Process Flow

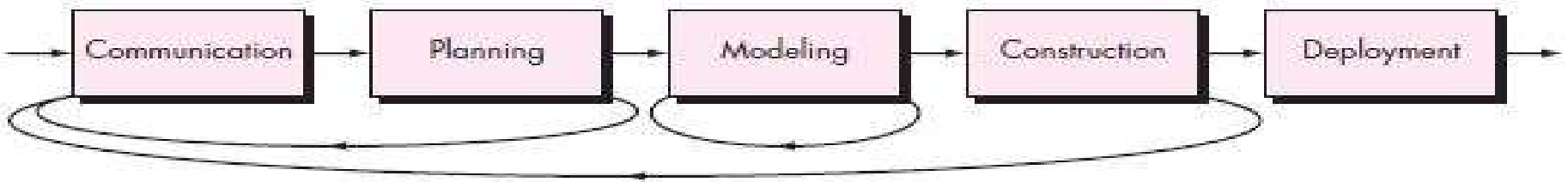
- A **linear process flow** executes each of the five framework activities in sequence, beginning with communication and culminating with deployment.




(a) Linear process flow

# Process Flow

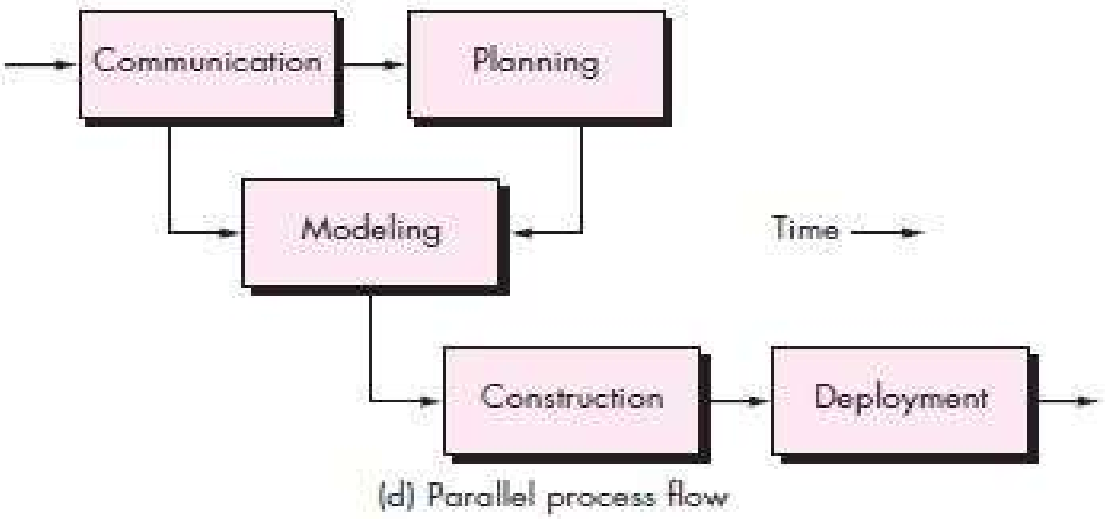
- An **iterative process flow** repeats one or more of the activities before proceeding to the next.




(b) Iterative process flow

# Process Flow

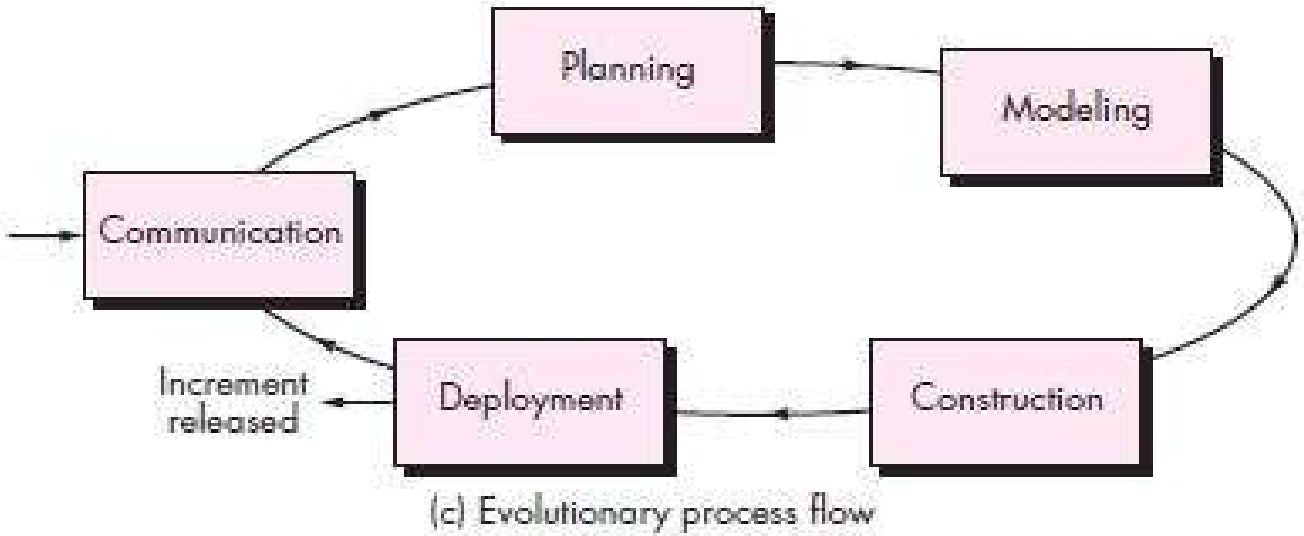
- A **parallel process flow** executes one or more activities in parallel with other activities.



(d) Parallel process flow

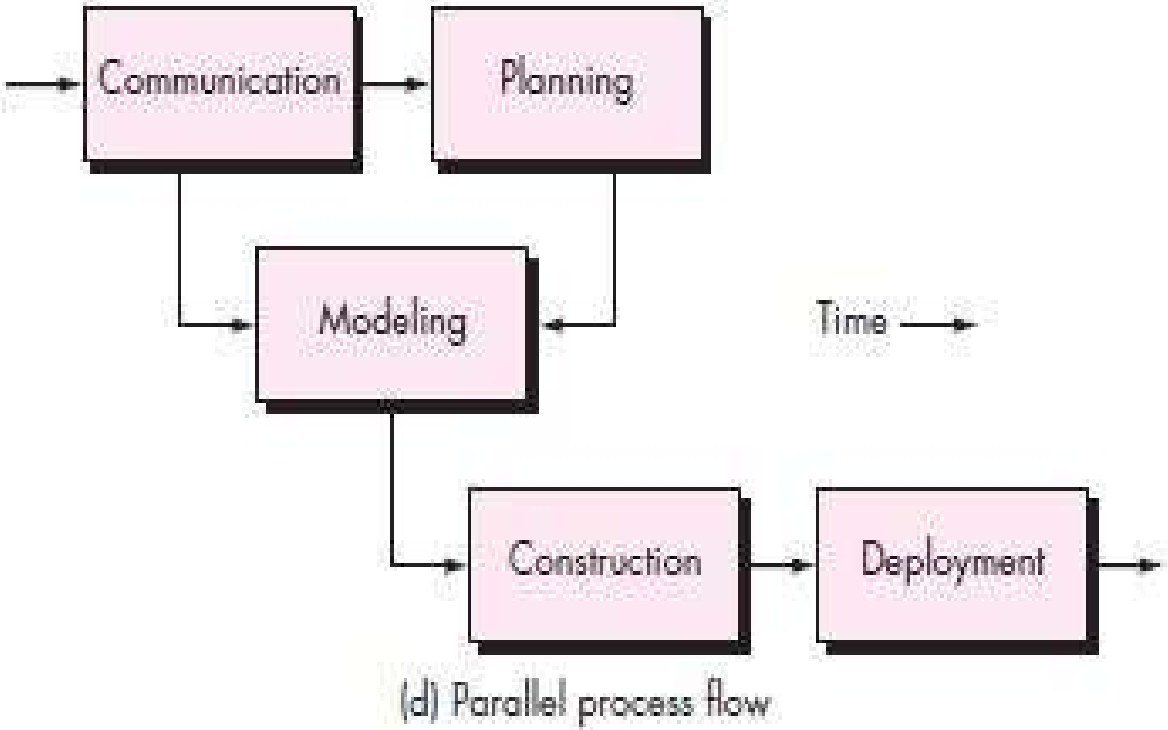
# Process Flow

- An **evolutionary process flow** executes the activities in a “circular” manner.



(c) Evolutionary process flow

# Process Flow



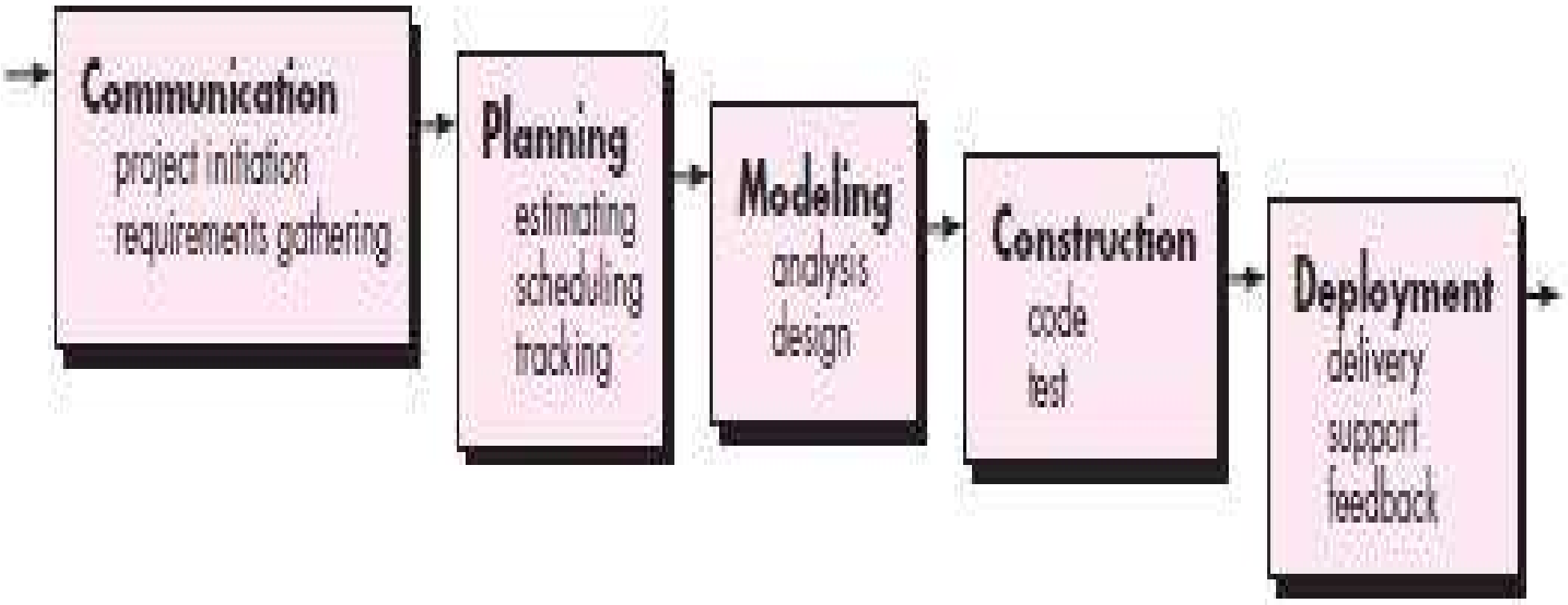

(d) Parallel process flow

# Process Model

- Waterfall Model (Linear Model)
- Incremental Process Model
- Evolutionary Process Model (Prototyping, The Spiral)
- Rapid Application Model (RAD)
- Agile Process
- Unified Process

# Process Model

## Waterfall Model



# Advantages of waterfall model

- This model is simple and easy to understand and use.
- It is easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process.

# Disadvantage

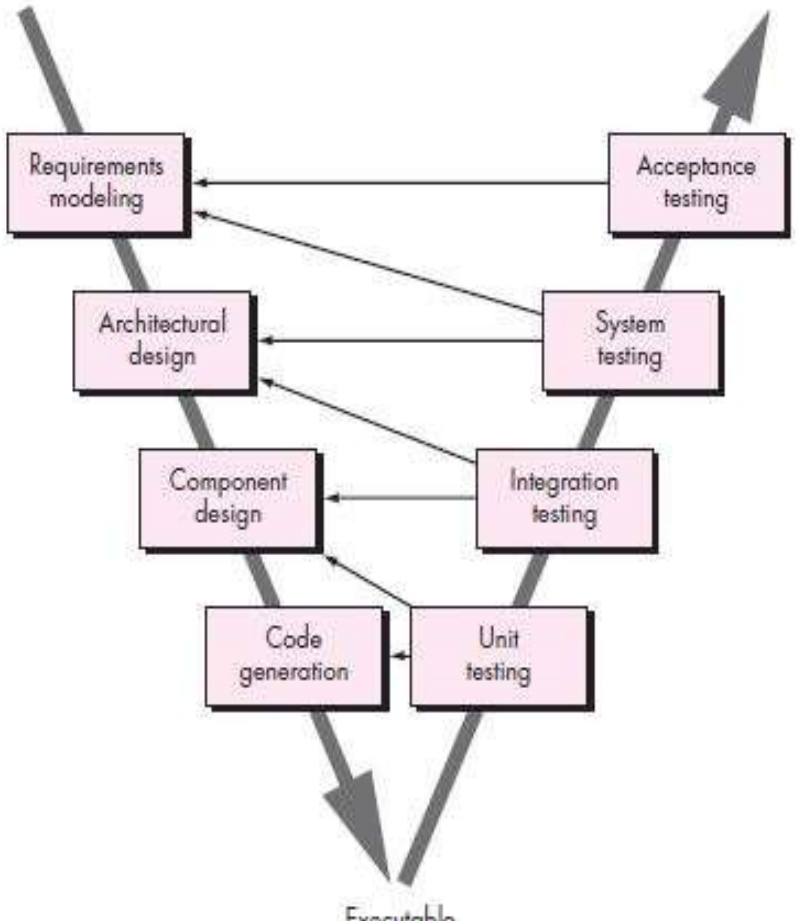
- Once an application is in the **testing** stage, it is very difficult to go back and change something that was not well-thought out in the concept stage.
- Not suitable for the projects where requirements are at a moderate to high risk of changing.

# Waterfall Model

- The waterfall model is the oldest paradigm for software engineering
- When to use Waterfall model?
    a. *Requirement are very well known*
    b. *Production definition is stable*
    c. *Technology is understood*
    d. *New version of existing product*

# V-Model

- Variation in the representation of the water model is called the V-model.

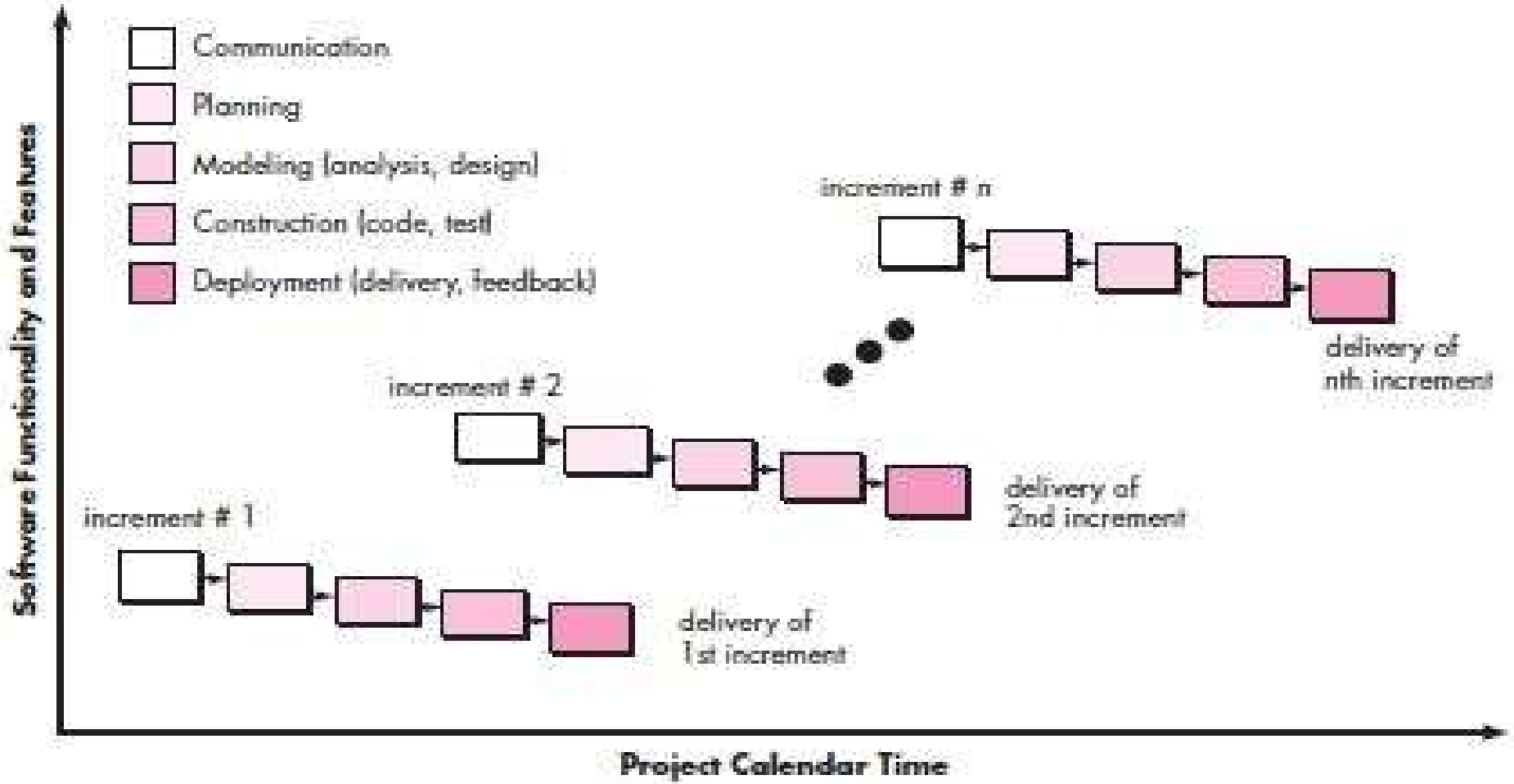


# V-Model

- The V-model depicts the relationship of **quality assurance actions** to the actions associated with the communication, modeling and, early construction activities
- Model **detail and technical representation** of the problem when team move down the left side of the V
- Once the **code** has been generated, team move up the right side of V, performing series of test.

# Incremental Model

- Incremental Process Model



# Incremental Model

- Incremental model delivers a series of releases, called increments, that provide progressively more functionality for customer as each increment delivered.
- The first increment is often a **core product**.
- Basic requirement are addressed but many supplementary features remain undelivered

# Incremental Model

**When to use?**

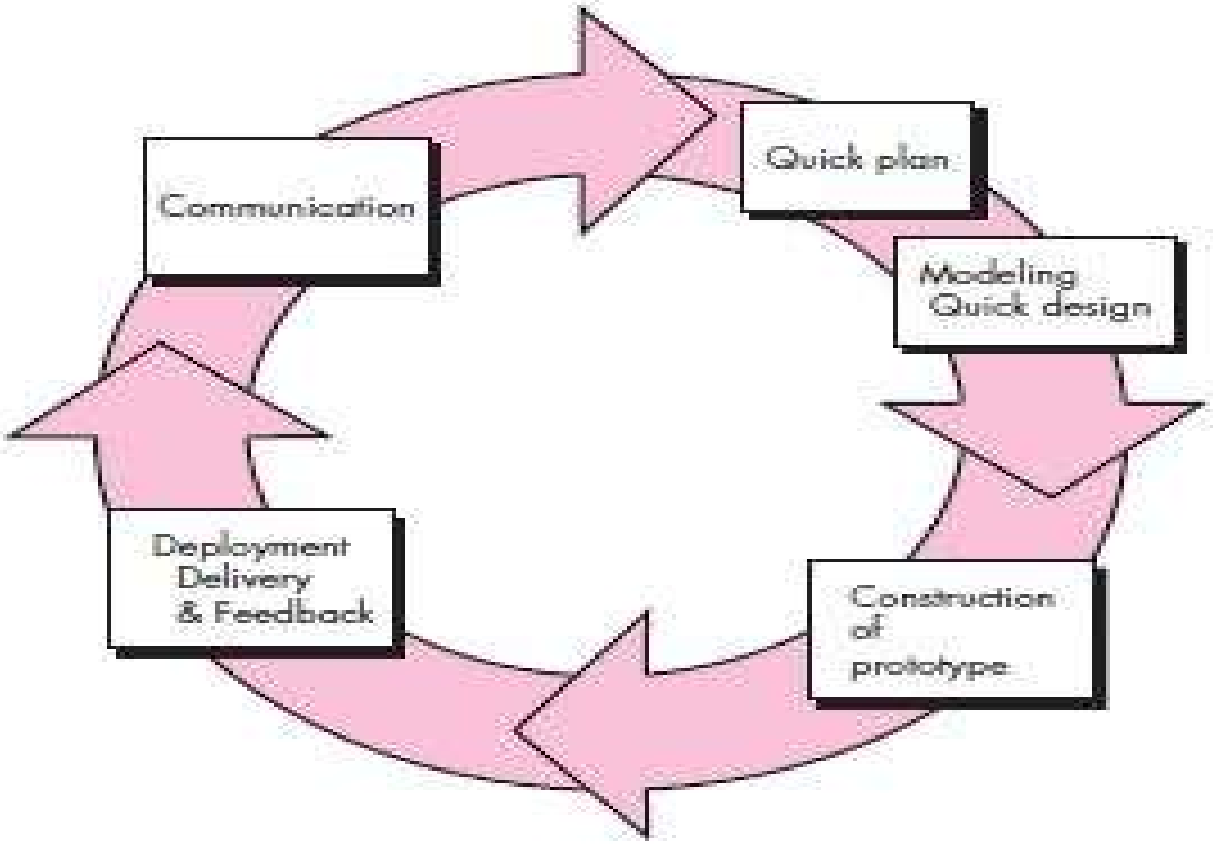
- Useful when staffing unavailable for a complete implementation of project till deadline.
- If the core product is well received, then additional staff (if required) can be added to implement the next increment.
- When software engineering team are not very well skilled or trainee

# Evolutionary Model

- Evolutionary process model produce an increasingly more complete version of the software with each iteration.

# Evolutionary Model

- Prototyping Model



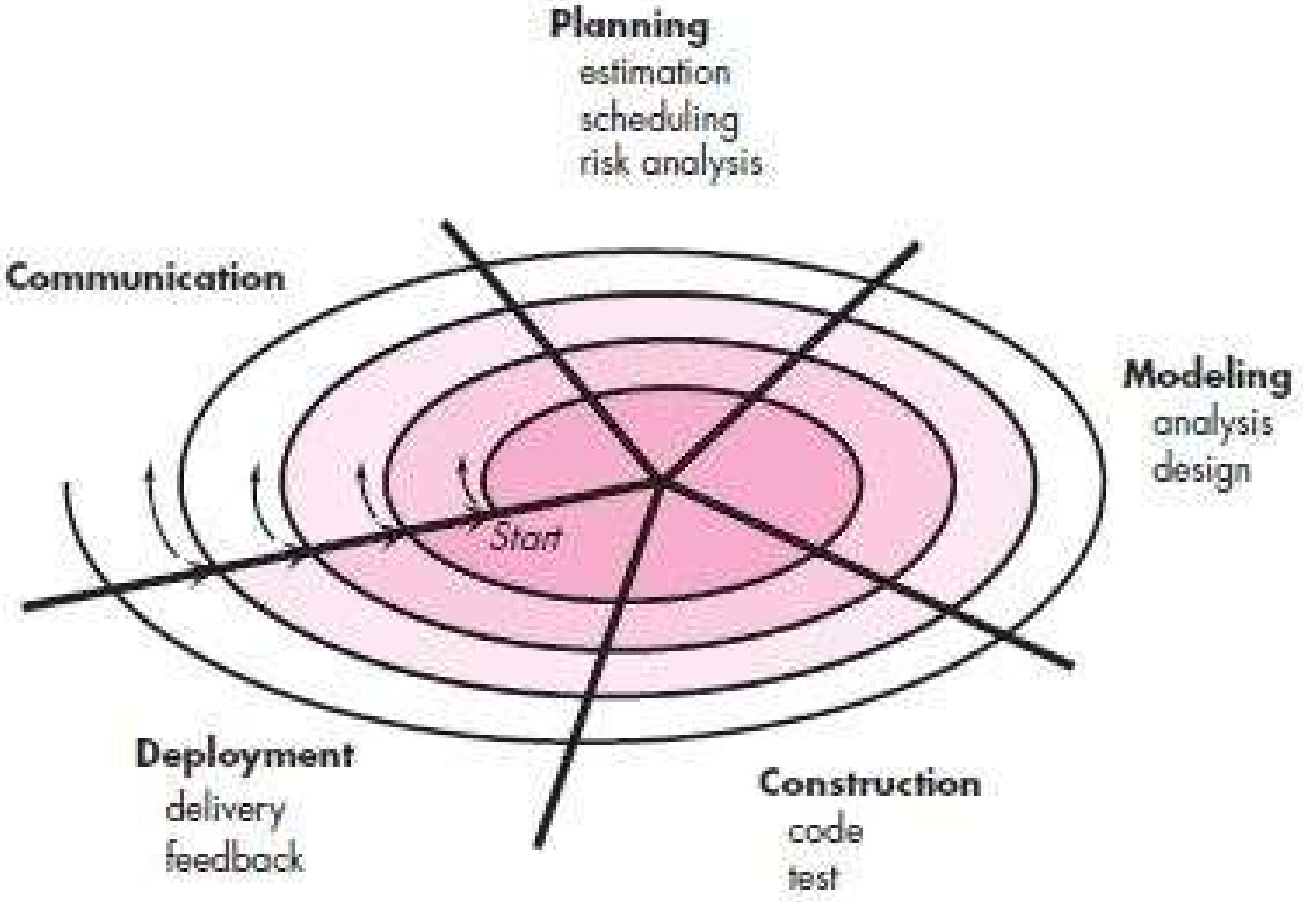
# Prototyping

**When to use?**

- Customer defines a set of general objectives for software, but does not identify detailed requirements for functions and features
- Developer may be unsure of the efficiency of an algorithm, the adaptability of an operating system

# Evolutionary model

## Spiral model

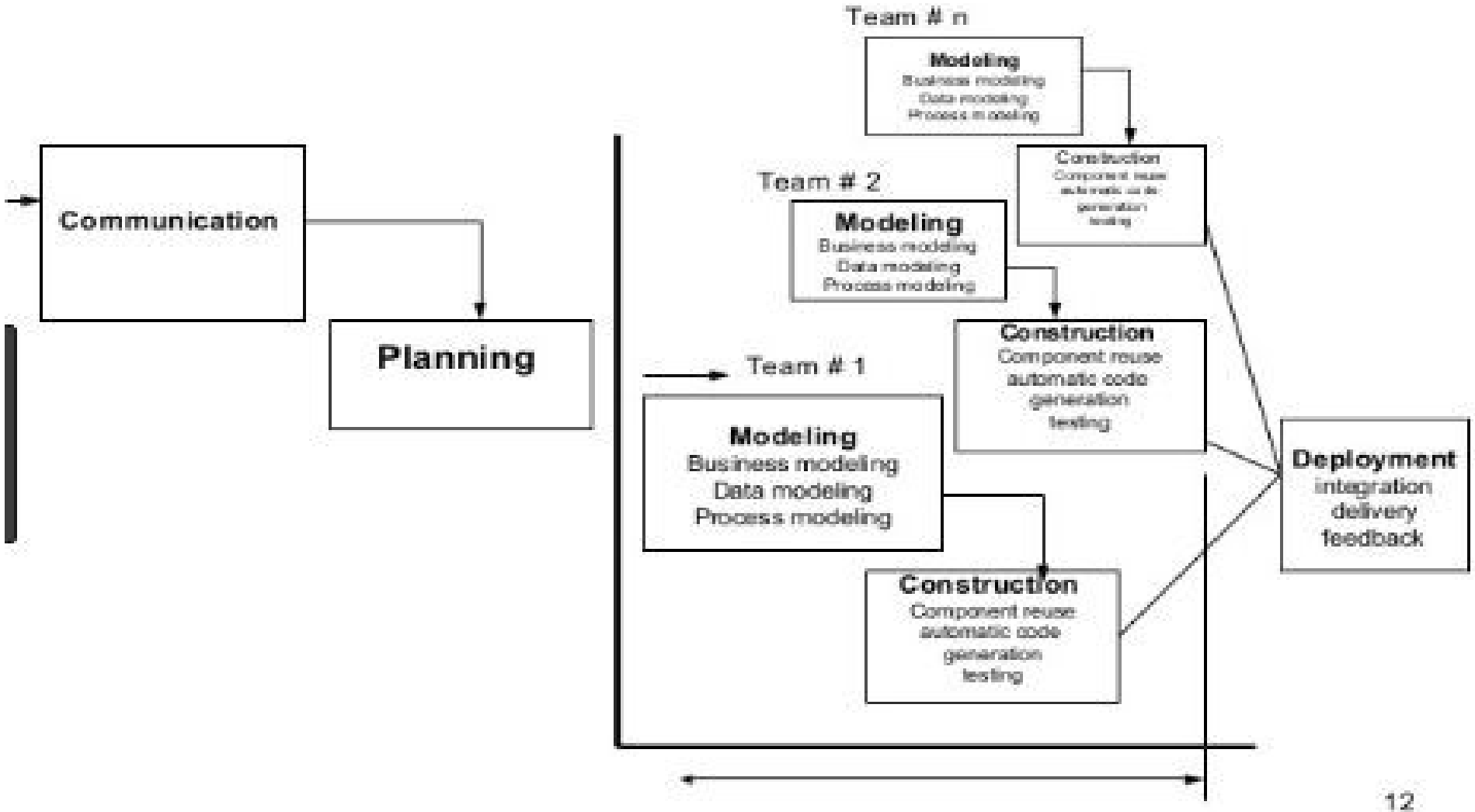


# Spiral Model

- The spiral model is a risk-driven process model.
- Software is developed in series of evolutionary releases
- During early iteration, release might be a model or prototype.
- Later iterations, increasingly complete versions of the product
- The spiral model is the realistic approach to the development of large scale systems and software

# Rapid Application Development(RAD )

## The RAD Model



# RAD

- Multiple software team work in parallel on different functions
- Modeling encompasses three major phases: Business Modeling, Date, Data modeling, Process Modeling
- Construction uses reusable components, automatic code generation and testing

# RAD

## Problems in RAD

- Required number of RAD teams
- Not suited when technical teams are weak
- Requires modularity

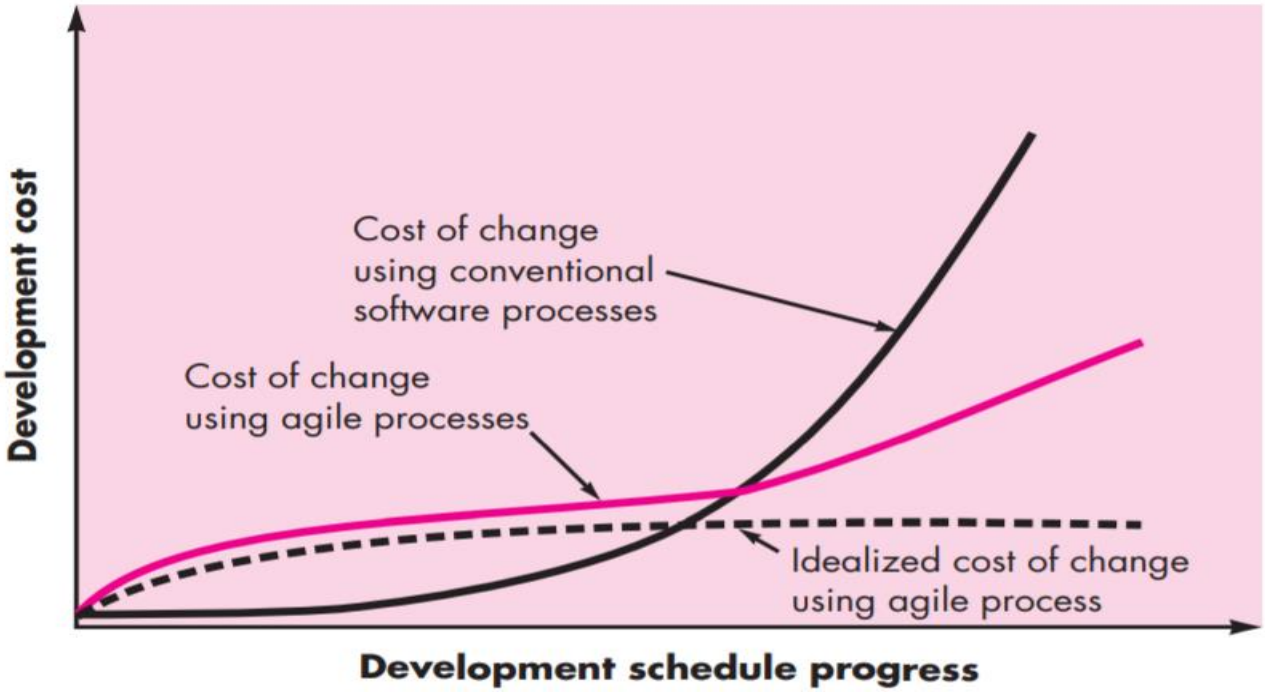
# Software process model

## Agile

- Agility has become today's buzzword when describing a modern software process.
- An agile team quickly respond to changes.
- Change is what software development is very much about.
- Agility has **12 principles** and four guidelines

# Software process model

Agile Process

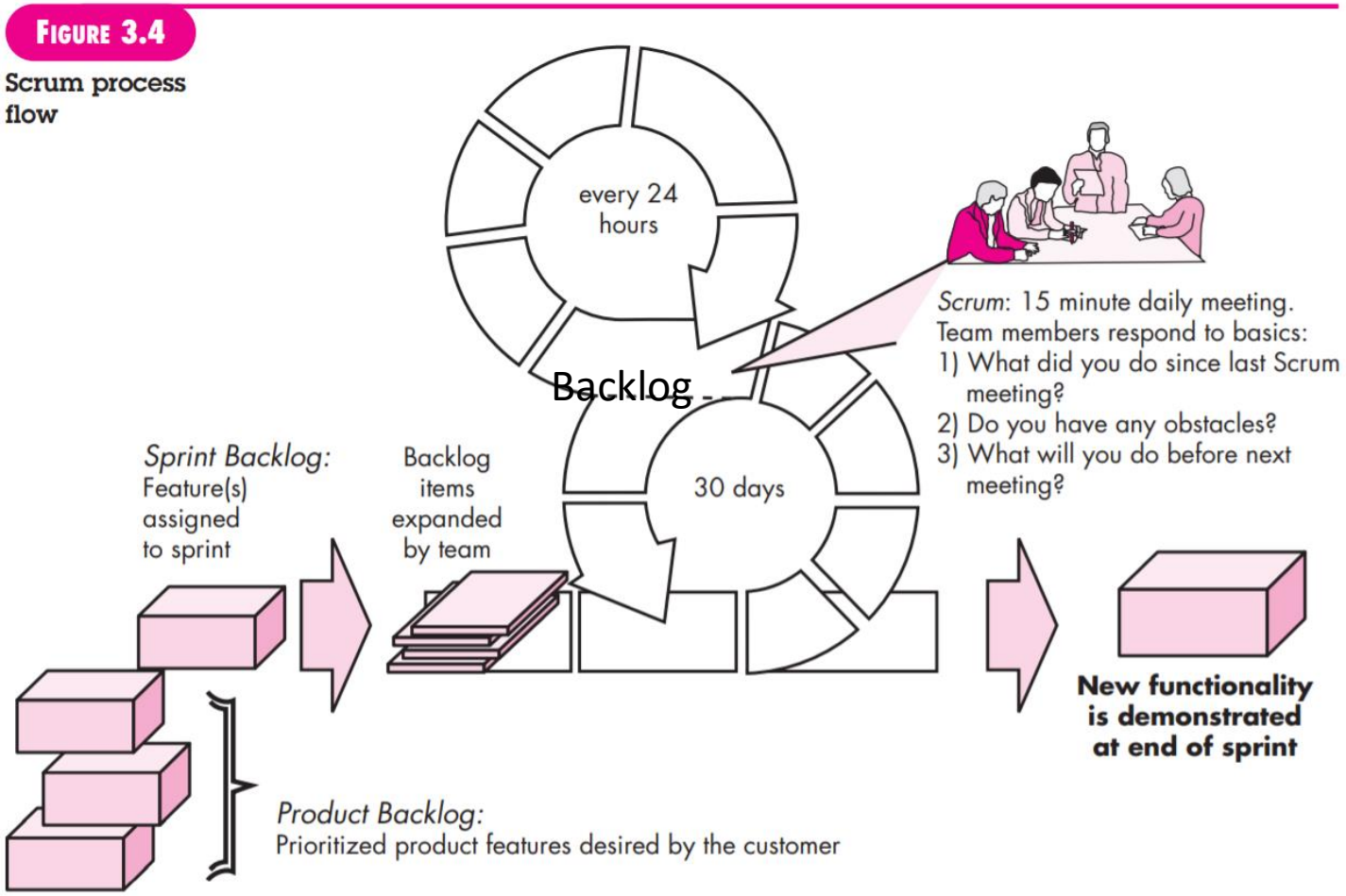


# Software process model

**Scrum**

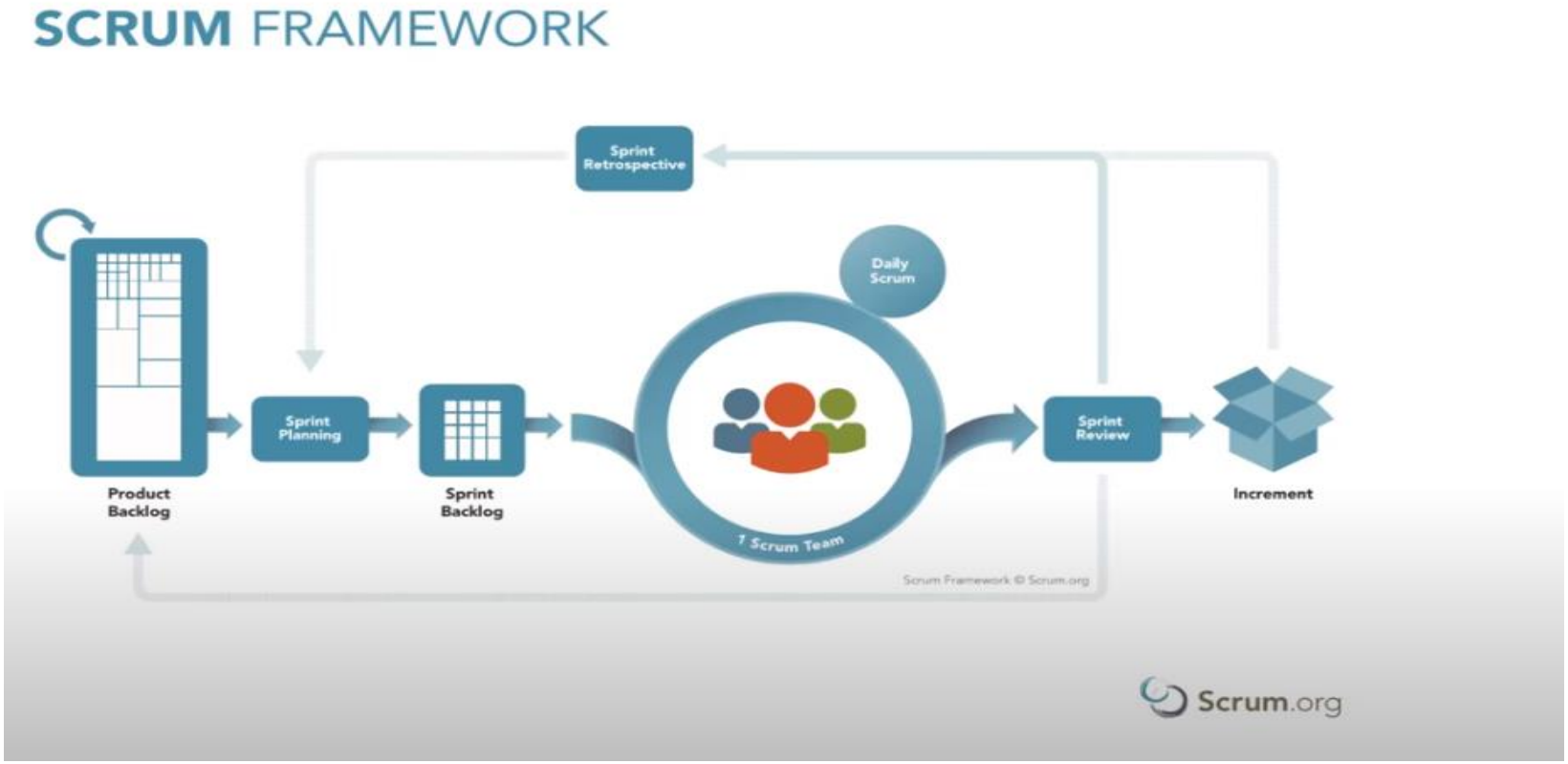
- It is one of the popular framework for the agile software development
- If you go to scrum.org it defines its **Better way Of Building Product.**
- Other popular frameworks are Kanban, Extreme Programming (XP)

# Software process model



# Software process model

Scrum process



# Scrum

## Backlog :

- Items can be added to the backlog at any time (this is how changes are introduced).
- The **product owner** assesses the backlog and updates priorities as required.

## Sprint:

- consist of work units that are required to achieve a requirement defined in the backlog that must be fit into a predefined time-box(typically 30 days).

# Scrum

**Scrum Meeting**

Scrum meetings are short (typically 15 minutes) meetings held daily by the Scrum team.

Three key questions are asked and answered by all team members

- What did you do since the last team meeting?
- What obstacles are you encountering?
- What do you plan to accomplish by the next team meeting?

# SCRUM

- A team leader, **Scrum master**, leads the meeting and assesses the responses from each person.
- The Scrum meeting helps the team to uncover potential problems as early as possible.

**Demos:**

- Deliver the software increment to the customer so that functionality that has been implemented can be demonstrated and evaluated by the customer.
- Demo may not contain all planned functionality, but rather those functions that can be delivered within the time-box that was established

# Process model

Unified process

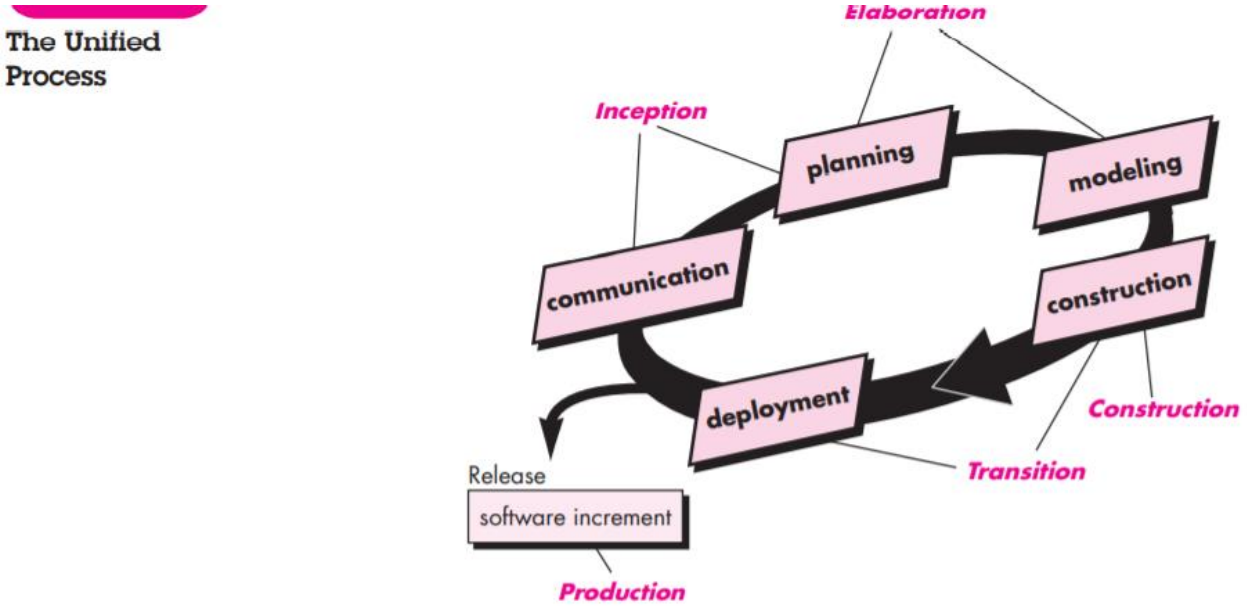
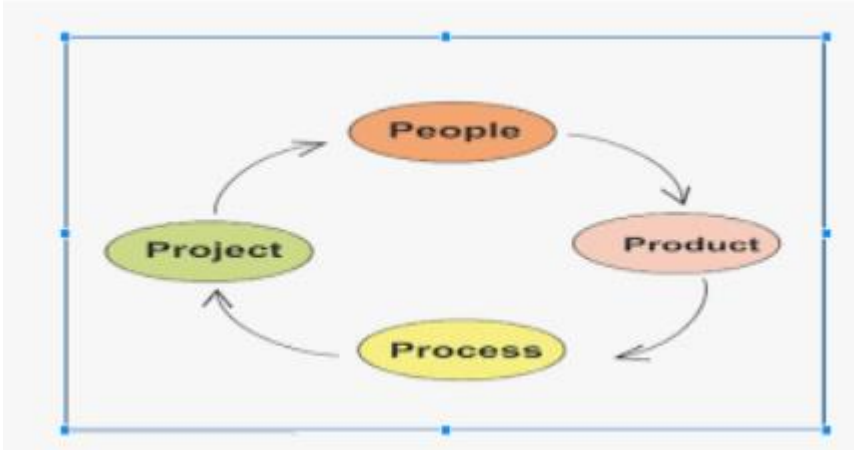



Fig : The  Unified process

# Four Ps in software Engineering



# People

- People are the key factor in the software development
- The following categories of people are involved in the software process.
    - Senior Managers
    - Project Managers
    - Practitioners--Developers
    - Customers
    - End Users

# Product

- Before a project can be planned, product objectives and scope should be established.
- Without product information it is impossible to define reasonable estimates of the cost.

# Process

- A  method provides the framework from that a comprehensive arrange for development is established

# Project

- We conduct planned and controlled software projects to manage complexity of project.
- To avoid project failure a software project manager and software engineers who built the project must :
- Develop a common sense approach for planning, monitoring, and controlling the project