



# Software Engineering

## Unit I

Part I Software Engineering Fundamentals

Part 2 Software Analysis

**part I**

# **Software Engineering Fundamentals**

# Software

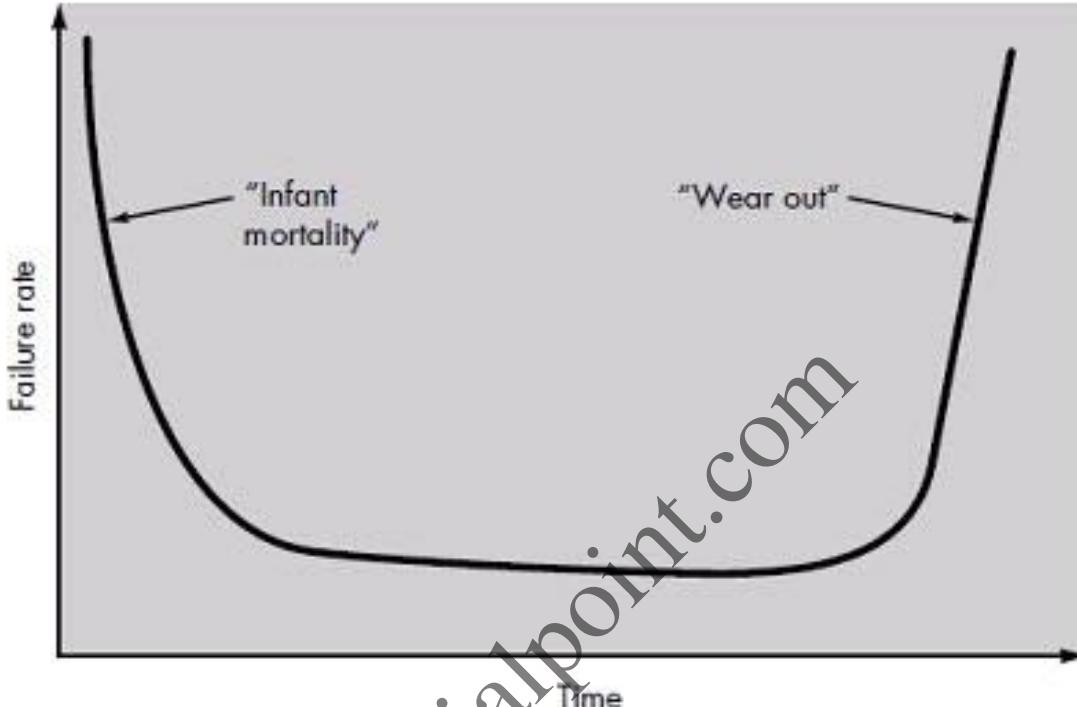
- (1) Instructions (computer programs) that when executed provide desired features, function, and performance
- (2) Data structures that enable the programs to adequately manipulate information
- (3) Descriptive information in both hard copy and virtual forms that describes the operation and use of the programs.

# Software Product

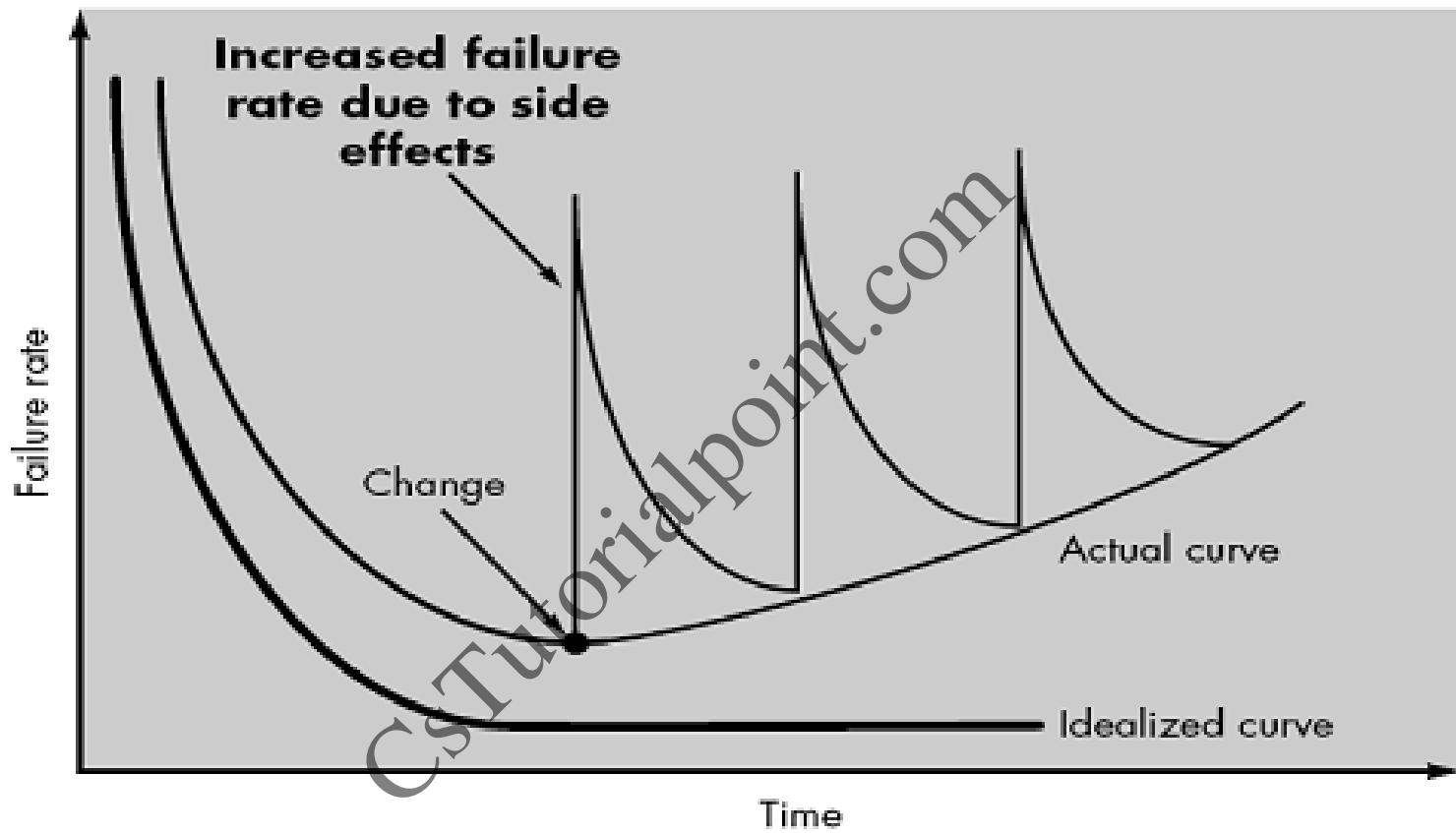
- IEEE defines “Software is the collection of computer programs, procedure rules and associated documentation and data.”
- Software is considered to be collection of executable programming code, associated libraries and documentations. Software, when made for a specific requirement is called **software product**.



# **Characteristics of software**



- software has one fundamental characteristic that makes it considerably different from hardware: **Software doesn't “wear out.”**



- **software will undergo change.**

As changes made, it is likely that errors will be introduced, causing the failure rate curve to spike as shown in the “actual curve” Before the curve can return to the original steady-state failure rate, another change is requested, causing the curve to spike again. Slowly, the minimum failure rate level begins to rise—the software is deteriorating due to change.

# **Software engineering**

- **Software engineering** is an engineering branch associated with development of software product using well-defined scientific principles, methods and procedures. The outcome of software engineering is an efficient and reliable software product.

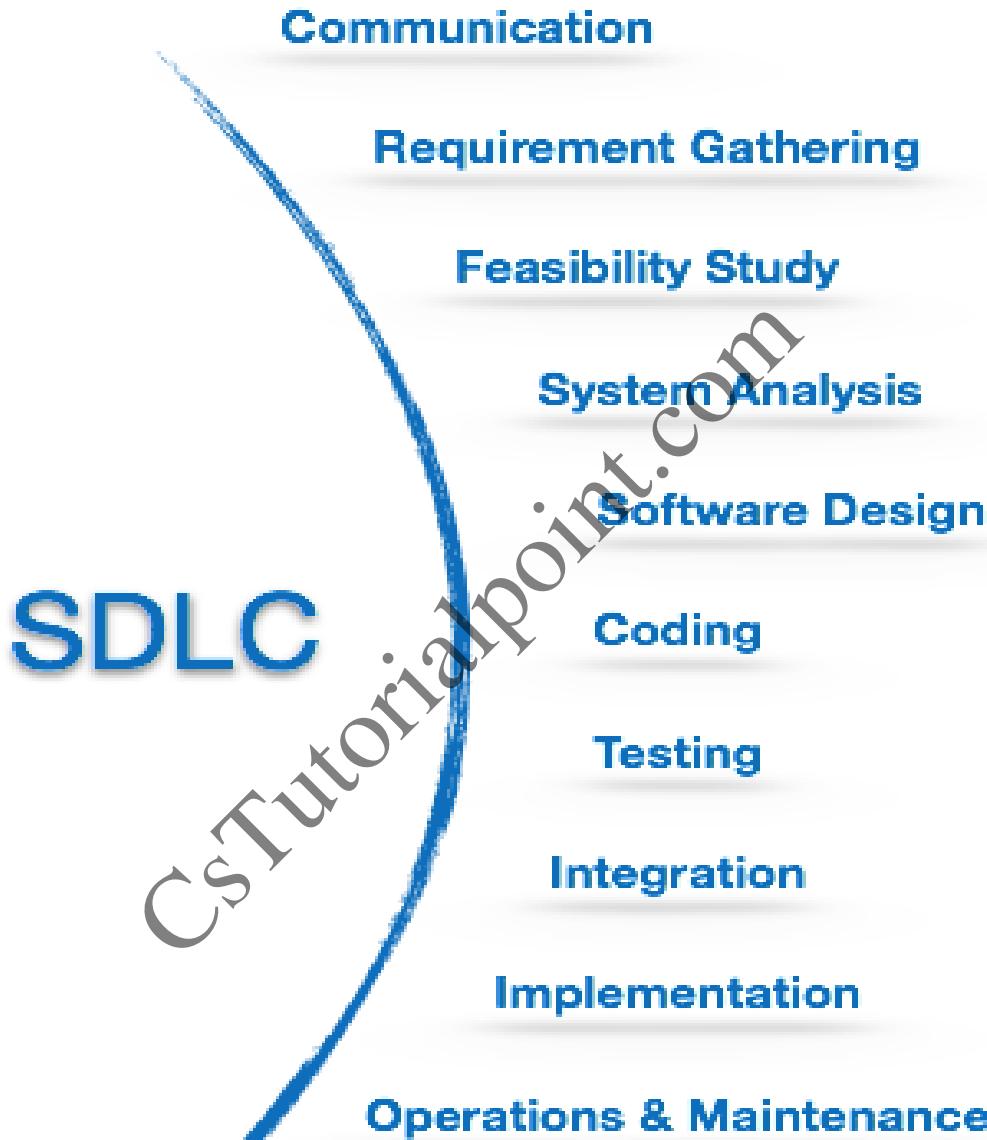
# Definition

**IEEE defines software engineering as:**

- (1) The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; that is, the application of engineering to software.
- (2) The study of approaches as in (1).

# SDLC

- Software Development Life Cycle, SDLC for short, is a well-defined, structured sequence of stages in software engineering to develop the intended software product.



# Software Development Life Cycle



# Communication

- This is the first step where the user initiates the request for a desired software product. He contacts the service provider and tries to negotiate the terms. He submits his request to the service providing organization in writing.

# Requirement Gathering

- The requirements are collected using a number of practices as given -
- studying the existing or obsolete system and software,
- conducting interviews of users and developers,
- referring to the database or
- collecting answers from the questionnaires.

# Feasibility Study

- After requirement gathering, the team comes up with a rough plan of software process. At this step the team analyzes if a software can be made to fulfill all requirements of the user and if there is any possibility of software being no more useful. It is found out, if the project is financially, practically and technologically feasible for the organization to take up. There are many algorithms available, which help the developers to conclude the feasibility of a software project.

# System Analysis

- At this step the developers decide a roadmap of their plan and try to bring up the best software model suitable for the project. System analysis includes Understanding of software product limitations, learning system related problems or changes to be done in existing systems beforehand, identifying and addressing the impact of project on organization and personnel etc. The project team analyzes the scope of the project and plans the schedule and resources accordingly.

# Software Design

- Next step is to bring down whole knowledge of requirements and analysis on the desk and design the software product. The inputs from users and information gathered in requirement gathering phase are the inputs of this step. The output of this step comes in the form of two designs; logical design and physical design. Engineers produce meta-data and data dictionaries, logical diagrams, data-flow diagrams and in some cases pseudo codes.

# Coding

- The goal of coding phase is to translate the design of the system into code in a given programming language.

# Testing

- The basic function of testing is to detect errors in the software
- The goal of testing is to uncover requirement ,design and coding errors in the programs.

# Implementation

- This means installing the software on user machines. At times, software needs post-installation configurations at user end. Software is tested for portability and adaptability and integration related issues are solved during implementation.

# Maintenance

- It is an important part of SDLC
- If there is any error or change is needed in the system then it is part of maintenance.
- The cost of maintenance is more than the cost of development.

# Types of Maintenance

- Corrective maintenance
- Adaptive maintenance
- Perfective maintenance
- Preventive maintenance

# Software Development Paradigm

- The software development paradigm helps developer to select a strategy to develop the software. A software development paradigm has its own set of tools, methods and procedures, which are expressed clearly and defines software development life cycle. A few of software development paradigms or process models are defined as follows:

# Process Model

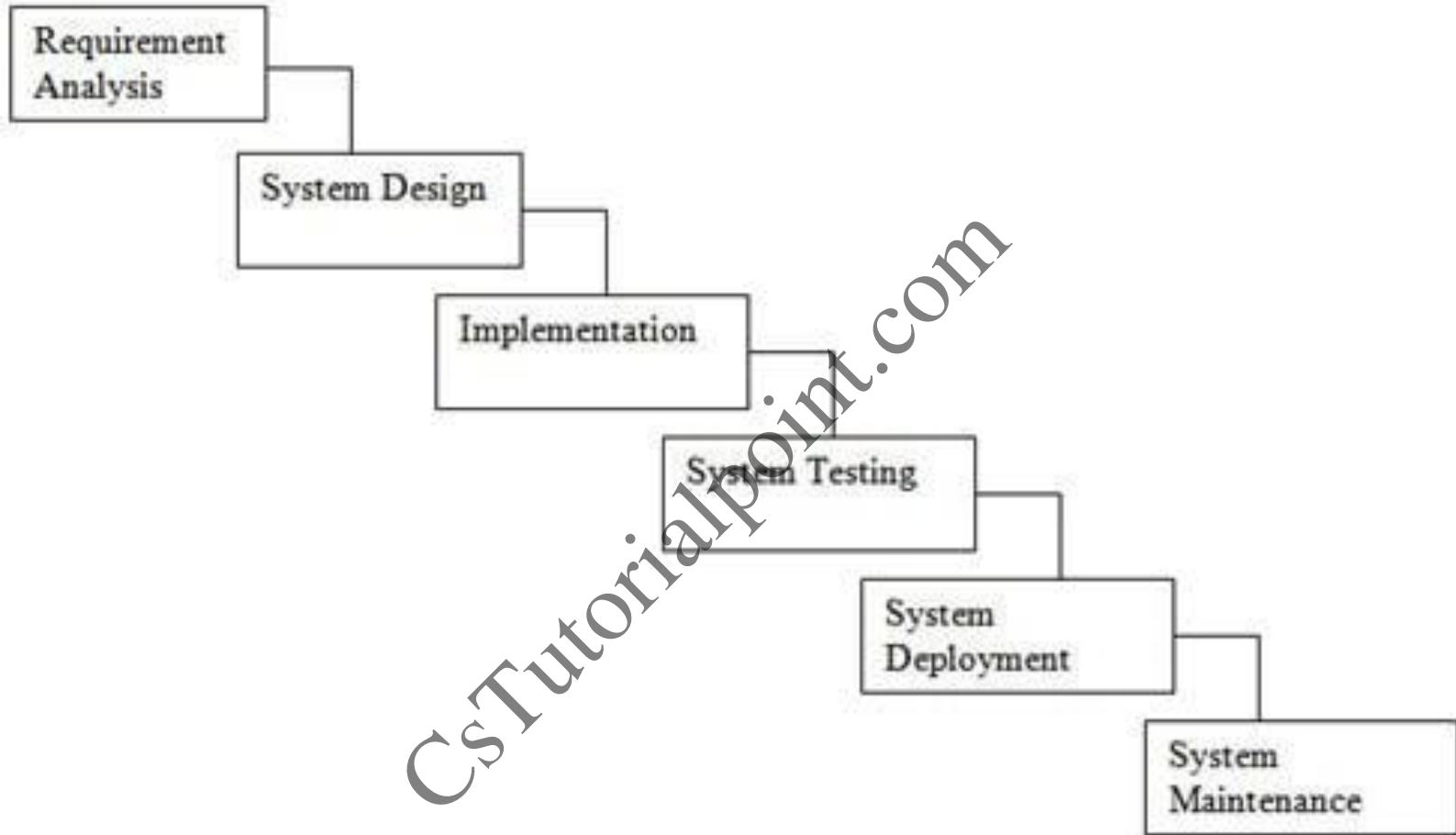
- To solve actual problem in an industry a team of software engineers must incorporate a development strategy that encompasses the process, methods, tools this strategy is called process model.
- S/W process Model is an abstract representation of S/W process.
- S/W processes are categorized into three phases: Definition phase, Development phase and support phase.
- Process model depends on the nature of software.

# Process models

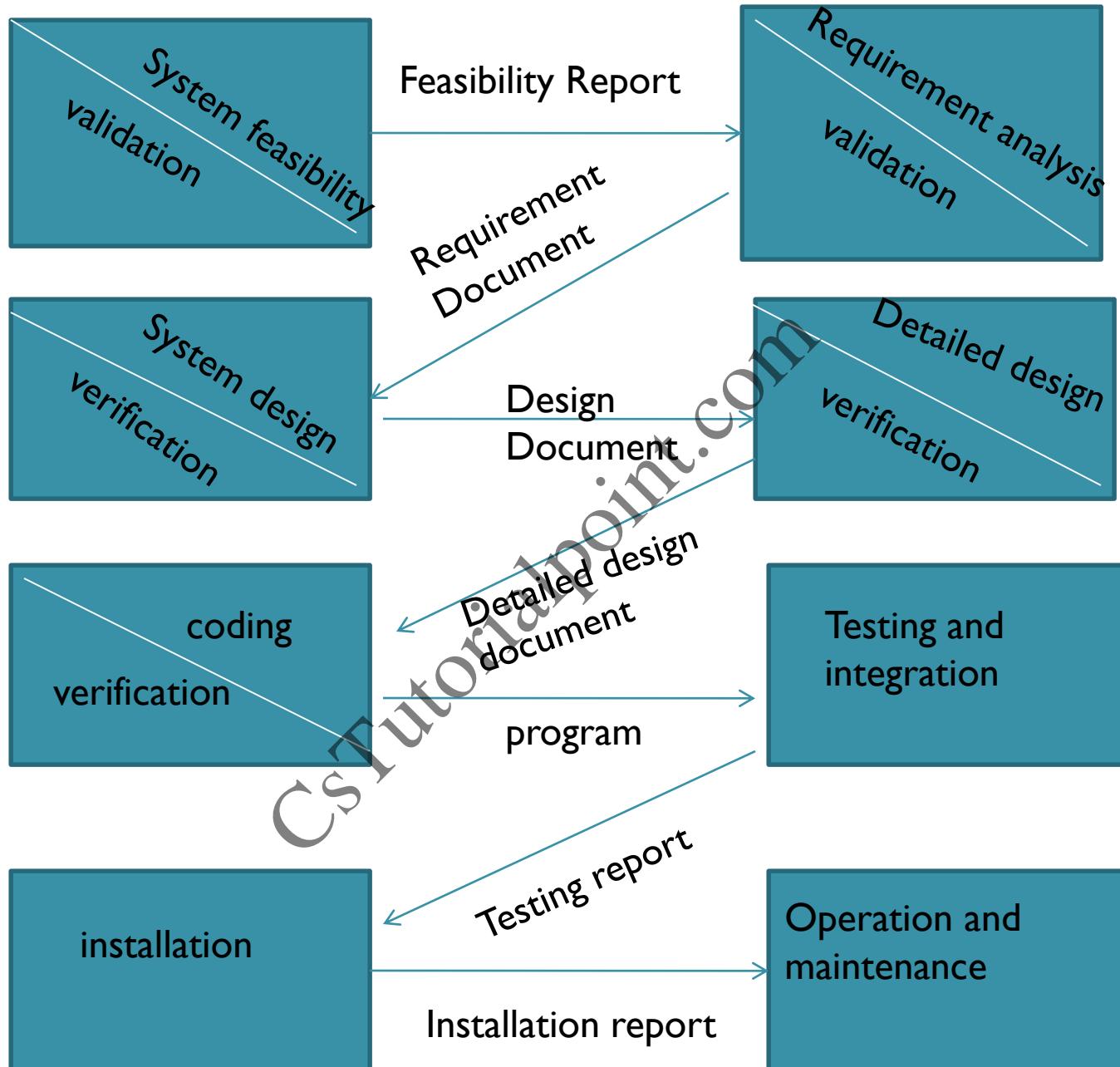
- Waterfall model
- Evolutionary model
  - 1. Prototype model
  - 2. Spiral model
- Iterative model
- Incremental model

# Waterfall model/Linear sequential/Classic life cycle

- Waterfall model is the simplest process model .
- Proposed by ROYCE.
- It consists of all the phases of SDLC.
- Phases organized in linear order that's why it is called linear sequential model.



Waterfall Model - © www.SoftwareTestingHelp.com



# When to use

- Requirements are very well known, clear and fixed.
- Product definition is stable.
- Technology is understood.
- There are no ambiguous requirements.
- Ample resources are available.
- Project is short.

# Advantages

- It is the simplest model.
- Best for straight line development.
- It encompasses all stages of sdlc.
- It divides the task into clearly defined phases.
- Costly and slow but satisfies all the requirements of user.

# Disadvantages

- Customer needs a lot of patience.
- It is difficult for a customer to state all the requirements explicitly.
- Real project rarely follow this model.
- Costly and slow.

Example: automobile companies who make cars or bikes.

# Prototype model

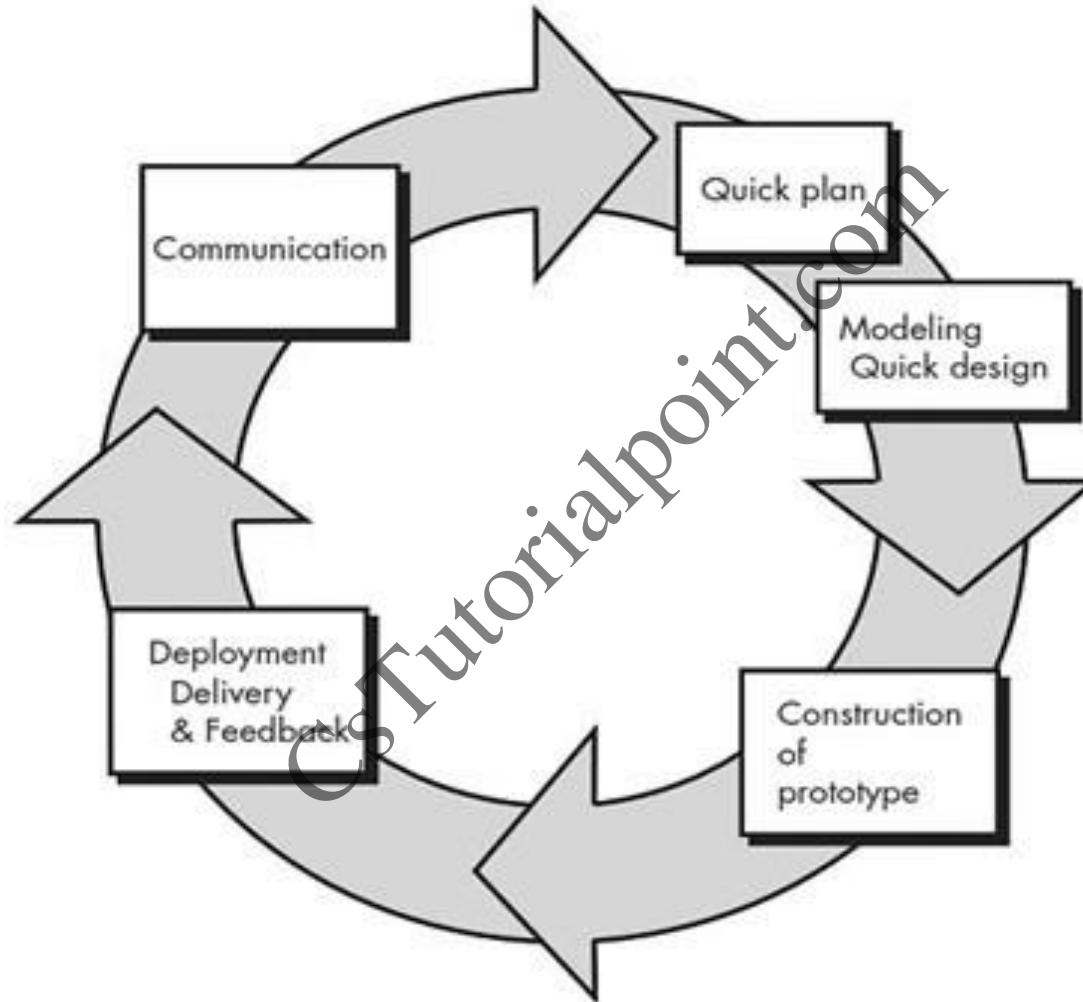


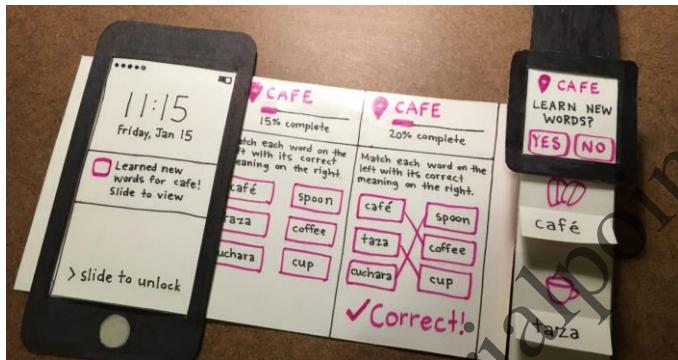
Figure: Prototype Model

# Definition

- The goal of prototype model is to build a prototype that help to understand the requirements.
- Prototype is build on the basis of current requirement.
- Client can get an actual feel of the system by using prototype.
- It gives better idea to understand the requirements of the desired system.
- Generally used for large and complicated system.

# Types of prototype model

- Throwaway prototype



- Evolutionary prototype

# When to use

- When customer is clueless about detailed requirements of the system.
- When it requires a lot of interaction with user.
- Customer is freely available to provide feedback of the prototype.

# Advantages

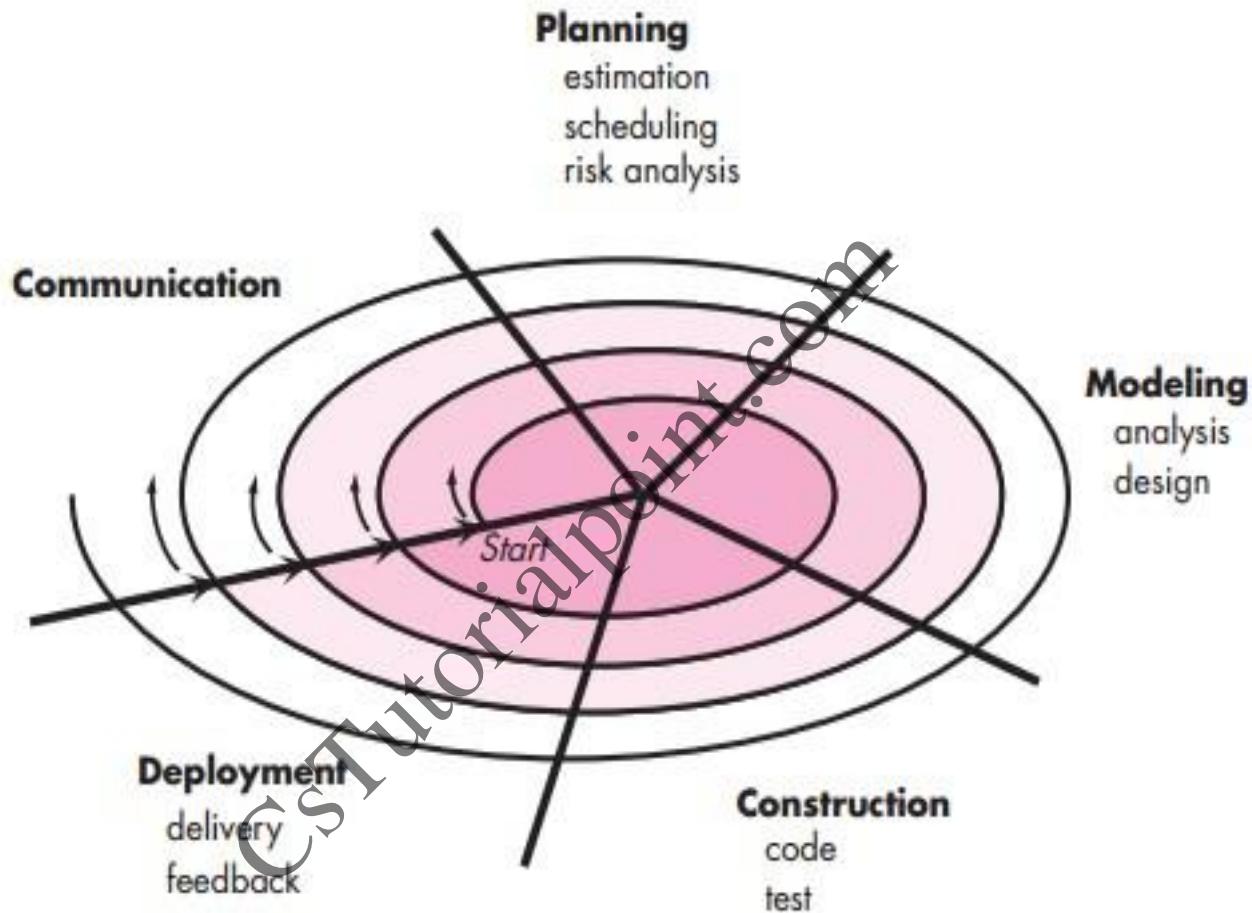
- It does not require complete set of specification or requirement before the development of software begins.
- When the customer is clueless this model is perfect to develop the quality software.

# Disadvantages

- The developer makes implementation compromise.
- Needs a lot of interaction between client and developer.
- This model may increase the complexity of the system as scope of the system may expand beyond original plan.

# Spiral model

- Proposed by Barry Boehm.
- Spiral model is divided into a set of framework activities defined by s/w engineering team.
- These activities are called task region.



# When to use?

- When cost and risk evaluation is important.
- For medium to high risk projects.
- Users are unsure about their needs.
- Requirements are complex.
- New product line.
- Significant changes are expected.

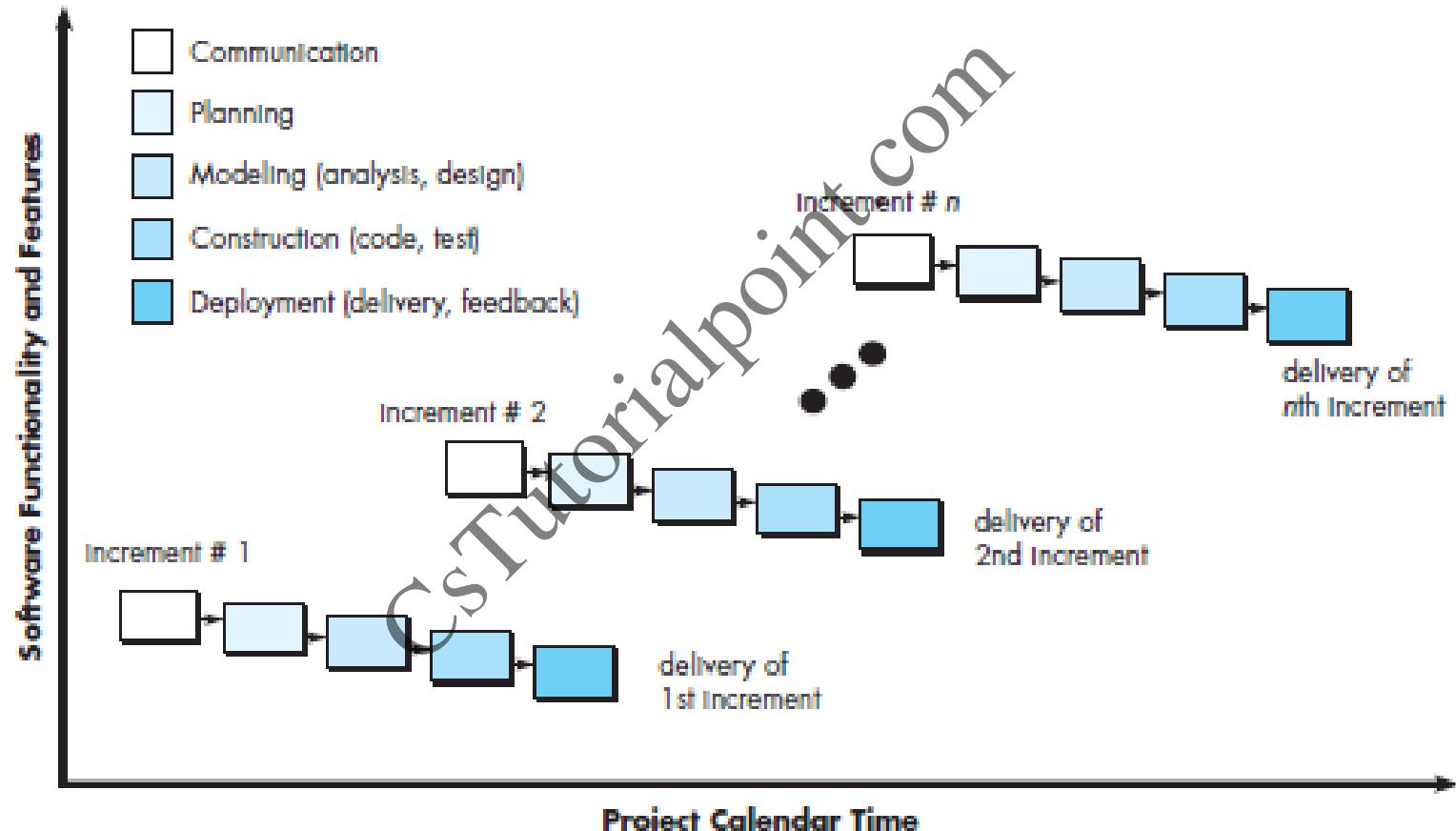
# Advantage

- Realistic approach.
- Customer and developer better understand and reacts to the risk at each evolutionary level.
- Use prototype as risk reduction mechanism.
- It maintains the systematic approach like waterfall model as well as incorporating iterative framework that are realistically reflects the real world.
- No distinction between development and maintenance.

# Disadvantage

- Time consuming approach.
- Needs greater communication between developer and customer.

# Incremental model



# When to use?

- Requirements are clearly understood.
- Major requirements are known some are evolve over time.
- There is a need to get a product in market early.
- New technology being used.
- Resources with needed skills are not available.

# Advantages

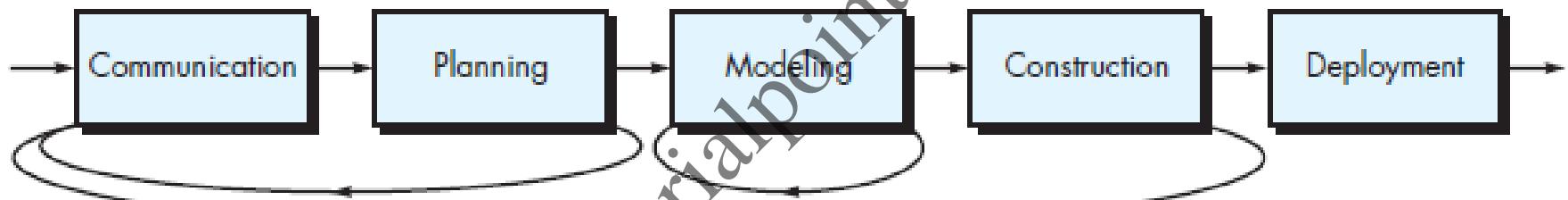
- Flexible and less costly.
- Easier to test and debug.
- Customer respond to each built.

# Disadvantage

- Needs good planning and design.
- Total cost is high.

CSTutorialpoint.com

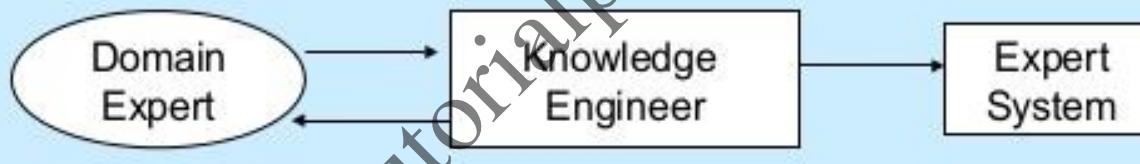
# Iterative model



(b) Iterative process flow

# Knowledge Engineering

- Incremental Development



- Include End User from beginning
- Provides choices

- Knowledge engineering is a field of artificial intelligence(AI) that creates rules to apply to data to imitate the thought process of a human expert. It looks at the structure of a task or a decision to identify how a conclusion is reached.
- In its initial form, knowledge engineering focused on the transfer process; transferring the expertise of a problem-solving human into a program that could take the same data and make the same conclusions.

# End user development

- **End-user development (EUD) or end-user programming (EUP)** refers to activities and tools that allow end-users – people who are not professional software developers – to program computers. People who are not professional developers can use EUD tools to create or modify *software artifacts* (descriptions of automated behavior) and complex data objects without significant knowledge of a programming language. Examples include natural language programming, spreadsheets.

# Software Requirements

- A requirement is a feature of the system or a description of something the system is capable of doing in order to fulfill the system's purpose.

## Types of requirement

- Those that should be absolutely met.
- Those that are highly desirable but not necessary.
- Those that are possible but could be eliminated.

# Software Requirements

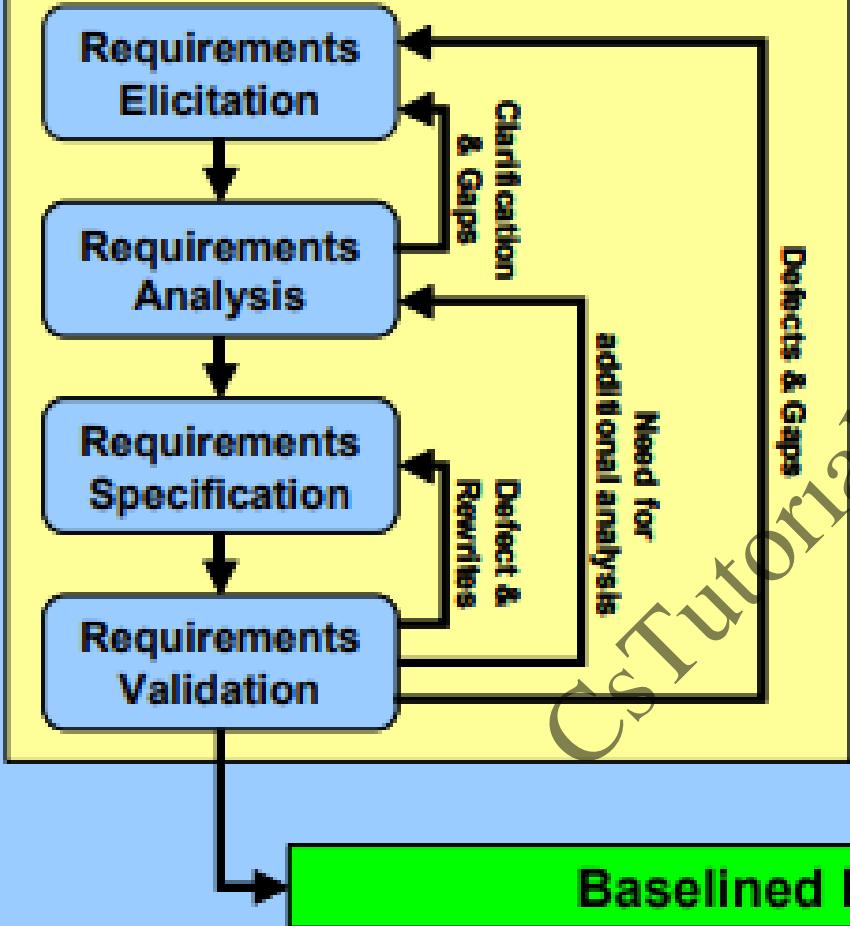
Broadly software requirements should be categorized in two categories:

1. **Functional Requirements**: Requirements, which are related to functional aspect of software fall into this category. Like I/O format, storage structure ,computational capabilities, timing and synchronization
  
2. **Non-Functional Requirements**: Requirements, which are not related to functional aspect of software, fall into this category. They are implicit or expected characteristics of software, which users make assumption of.

- Non-functional requirements include -
- Security
- Portability
- Performance
- Interoperability
- Flexibility
- Accessibility
- Usability
- Efficiency
- Reliability

# Requirements Engineering

## Requirements Development



## Requirements Management

- Establish & maintain an agreement with the customers & users on the requirements
- Control the baselined requirements
- Process proposed changes to the requirements
- Keep requirements consistent with plans & work products
- Negotiate new commitments based on impact of approved changes

Current Requirements

Revised Requirements

Baselined Requirements

# Software analysis

- Software requirement specification(SRS):  
IEEE defines a requirement as
  - “(1)A condition of capability needed by a user to solve a problem or achieve an objective.
  - (2) A condition or a capability that must be met or possessed by a system to satisfy a contract,standard,specification or other formally imposed document”
- The goal of requirement activity is to produce the SRS that describes *what* the proposed software should do without describing *how* the software will do it.

# Need of SRS

- Basic purpose of SRS is to bridge the communication gap between client and developer.

## Advantage of SRS

1. It is an agreement between user and developer.
2. It provides reference for validation of the final product.
3. High quality SRS is prerequisite to high quality S/W.
4. High quality SRS reduces the development cost.

# Characteristics of an SRS

1. Correct
2. Complete
3. Unambiguous
4. Verifiable
5. Consistent
6. Ranked for importance
7. Modifiable
8. Traceable

# Components of SRS

- Functional requirements :- Functional requirements specify what output should be produced from the given inputs. So they basically describe the connectivity between the input and output of the system.
- Performance requirements :- Performance requirements are typically expressed as processed transactions per second or response time from the system for a user event or screen refresh time or a combination of these.

- Design constraints: The client environment may restrict the designer to include some design constraints that must be followed. The various design constraints are standard compliance, resource limits, operating environment, reliability and security requirements and policies that may have an impact on the design of the system. An SRS should identify and specify all such constraints.
- External interface requirements: All the possible interactions of the software with people hardware and other software should be clearly specified.



## Software Requirements Specification Template

A software requirements specification (SRS) is a work product that is created when a detailed description of all aspects of the software to be built must be specified before the project is to commence. It is important to note that a formal SRS is not always written. In fact, there are many instances in which effort expended on an SRS might be better spent in other software engineering activities. However, when software is to be developed by a third party, when a lack of specification would create severe business issues, or when a system is extremely complex or business critical, an SRS may be justified.

Karl Wiegers [Wie03] of Process Impact Inc. has developed a worthwhile template (available at [www.processimpact.com/process\\_assets/srs\\_template.doc](http://www.processimpact.com/process_assets/srs_template.doc)) that can serve as a guideline for those who must create a complete SRS. A topic outline follows:

### Table of Contents

### Revision History

#### 1. Introduction

- 1.1 Purpose
- 1.2 Document Conventions
- 1.3 Intended Audience and Reading Suggestions
- 1.4 Project Scope
- 1.5 References

#### 2. Overall Description

- 2.1 Product Perspective
- 2.2 Product Features
- 2.3 User Classes and Characteristics
- 2.4 Operating Environment
- 2.5 Design and Implementation Constraints
- 2.6 User Documentation
- 2.7 Assumptions and Dependencies

#### 3. System Features

- 3.1 System Feature 1
- 3.2 System Feature 2 (and so on)

#### External Interface Requirements

- 4.1 User Interfaces
- 4.2 Hardware Interfaces
- 4.3 Software Interfaces
- 4.4 Communications Interfaces

#### 5. Other Nonfunctional Requirements

- 5.1 Performance Requirements
- 5.2 Safety Requirements
- 5.3 Security Requirements
- 5.4 Software Quality Attributes

#### 6. Other Requirements

##### Appendix A: Glossary

##### Appendix B: Analysis Models

##### Appendix C: Issues List

A detailed description of each SRS topic can be obtained by downloading the SRS template at the URL noted in this sidebar.

# Specification Tools

- Flow based- DFD
- Data based

CSTutorialpoint.com

# Data Flow Diagram

- Data flow diagram is graphical representation of flow of data in an information system. It is capable of mentioning of depicting incoming data flow, outgoing data flow and stored data. There is a prominent difference between DFD and Flowchart. The flowchart depicts flow of control in program modules. DFDs depict flow of data in the system at various levels. DFD does not contain any control or branch elements.

# Types of DFD

Data Flow Diagrams are either Logical or Physical.

- Logical DFD : This type of DFD concentrates on the system process, and flow of data in between the system . For example in a Banking software system, how data is moved between different entities.
- Physical DFD: This type of DFD shows how the data flow is actually implemented in the system. It is more specific and close to the implementation.

# DFD Components

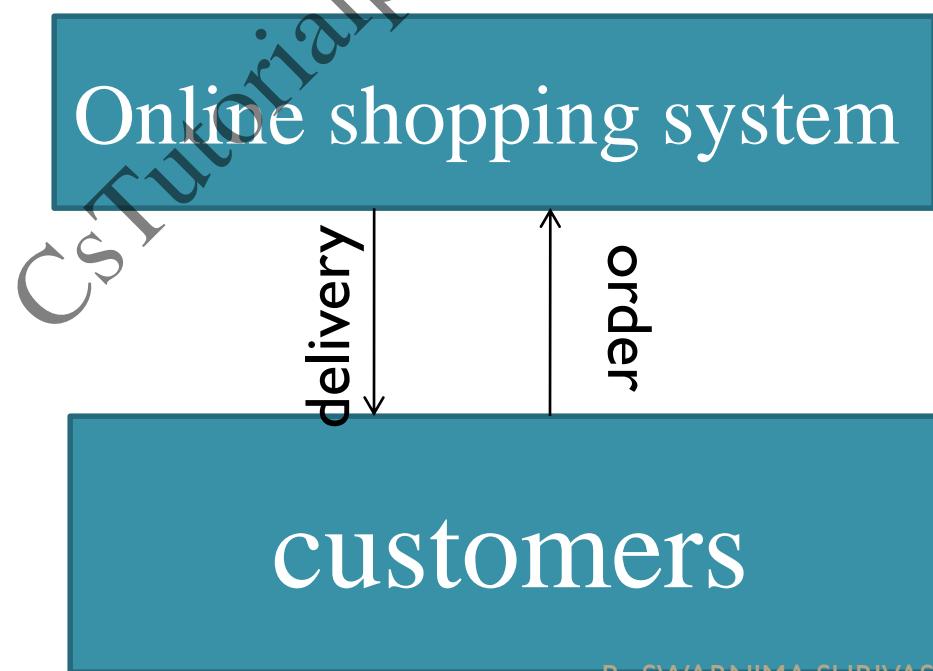


- **Entities** - Entities are source and destination of information data. Entities are represented by a rectangles with their respective names.
- **Process** - Activities and action taken on the data are represented by Circle or Round-edged rectangles.

- **Data Storage** - There are two variants of data storage - it can either be represented as a rectangle with absence of both smaller sides or as an open-sided rectangle with only one side missing.
- **Data Flow** - Movement of data is shown by pointed arrows. Data movement is shown from the base of arrow as its source towards head of the arrow as destination.

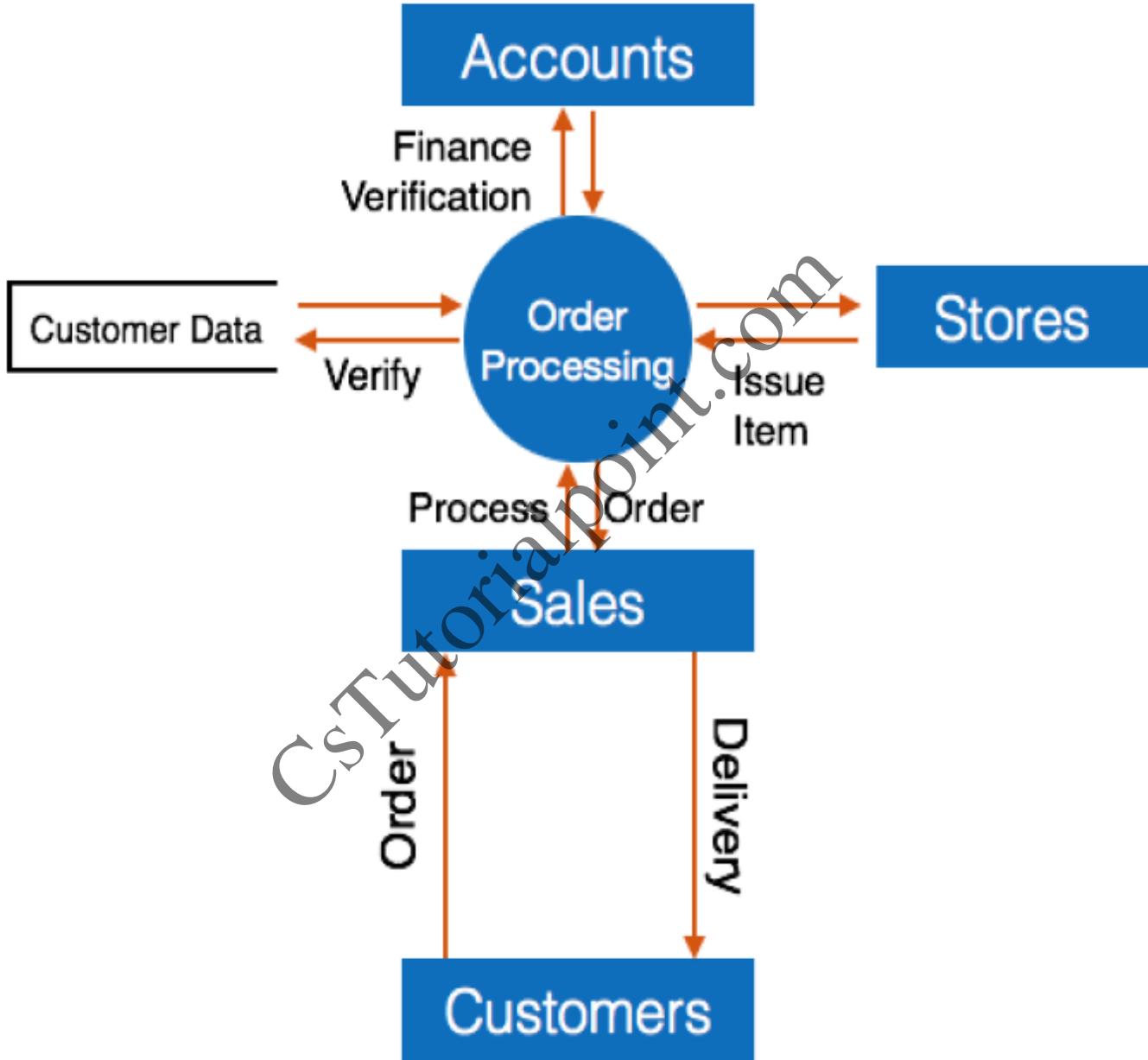
# Levels of DFD

- Level 0- Highest abstraction level DFD is known as Level 0 DFD, which depicts the entire information system as one diagram concealing all the underlying details. Level 0 DFDs are also known as context level DFDs.



# **Level I DFD**

- The Level 0 DFD is broken down into more specific, Level 1 DFD. Level 1 DFD depicts basic modules in the system and flow of data among various modules. Level 1 DFD also mentions basic processes and sources of information.



# Data Dictionary

- The data dictionary is centralized repository of information about such a meaning relationship to other data, origin, usage and format. Data dictionary is read only set of tables that provide information about the data base. Data dictionary manages metadata(database about database).
- Data Dictionary contains:-
  - 1) Definition of all schema object in the database(table, views, indexes, clusters, synonyms, procedure, functions, triggers)

- 2) How much space has been allocated for and is currently used by schema object.
- 3) Default value for columns.
- 4) Integrity constraint information.
- 5) Names of users.
- 6) Privileges and roles each user has been granted.
- 7) Auditing information such as who has accessed.

# Data Dictionary files

## I) Field's File

FIELD NAME	TYPE	SIZE
Rollno	Number	5
Name	Text	10
Courseid	Number	5

## 2) File's File

FIELD NAME	SIZE
Student	200
Course	300

# Data dictionary are of two types

- **Active data dictionary**:- Managed automatically by the DBMS . Maintained by system itself.
- **Passive data dictionary**:-Also called non integrated data dictionary used only for documentation purpose . Managed by users of the system and modified whenever the db changed.

# OOAD - Object Oriented Analysis

- In the system analysis or object-oriented analysis phase of software development, the system requirements are determined, the classes are identified and the relationships among classes are identified.
- The three analysis techniques that are used in conjunction with each other for object-oriented analysis are object modelling, dynamic modelling, and functional modelling.

# Object modeling

Object modeling develops the static structure of the software system in terms of objects. It identifies the objects, the classes into which the objects can be grouped into and the relationships between the objects. It also identifies the main attributes and operations that characterize each class.

The process of object modeling can be visualized in the following steps –

- Identify objects and group into classes
- Identify the relationships among classes
- Create user object model diagram
- Define user object attributes
- Define the operations that should be performed on the classes.

# Dynamic Modeling

Dynamic Modeling can be defined as “a way of describing how an individual object responds to events, either internal events triggered by other objects, or external events triggered by the outside world”.

The process of dynamic modeling can be visualized in the following steps –

- Identify states of each object.
- Identify events and analyze the applicability of actions.
- Construct dynamic model diagram, comprising of state transition diagrams.
- Express each state in terms of object attributes.
- Validate the state–transition diagrams drawn.

# Functional Modeling

- Functional Modeling is the final component of object-oriented analysis. The functional model shows the processes that are performed within an object and how the data changes as it moves between methods. It specifies the meaning of the operations of object modeling and the actions of dynamic modeling. The functional model corresponds to the data flow diagram of traditional structured analysis.

The process of functional modeling can be visualized in the following steps –

- Identify all the inputs and outputs.
- Construct data flow diagrams showing functional dependencies.
- State the purpose of each function.
- Identify constraints.
- Specify optimization criteria.