# Software Reliability

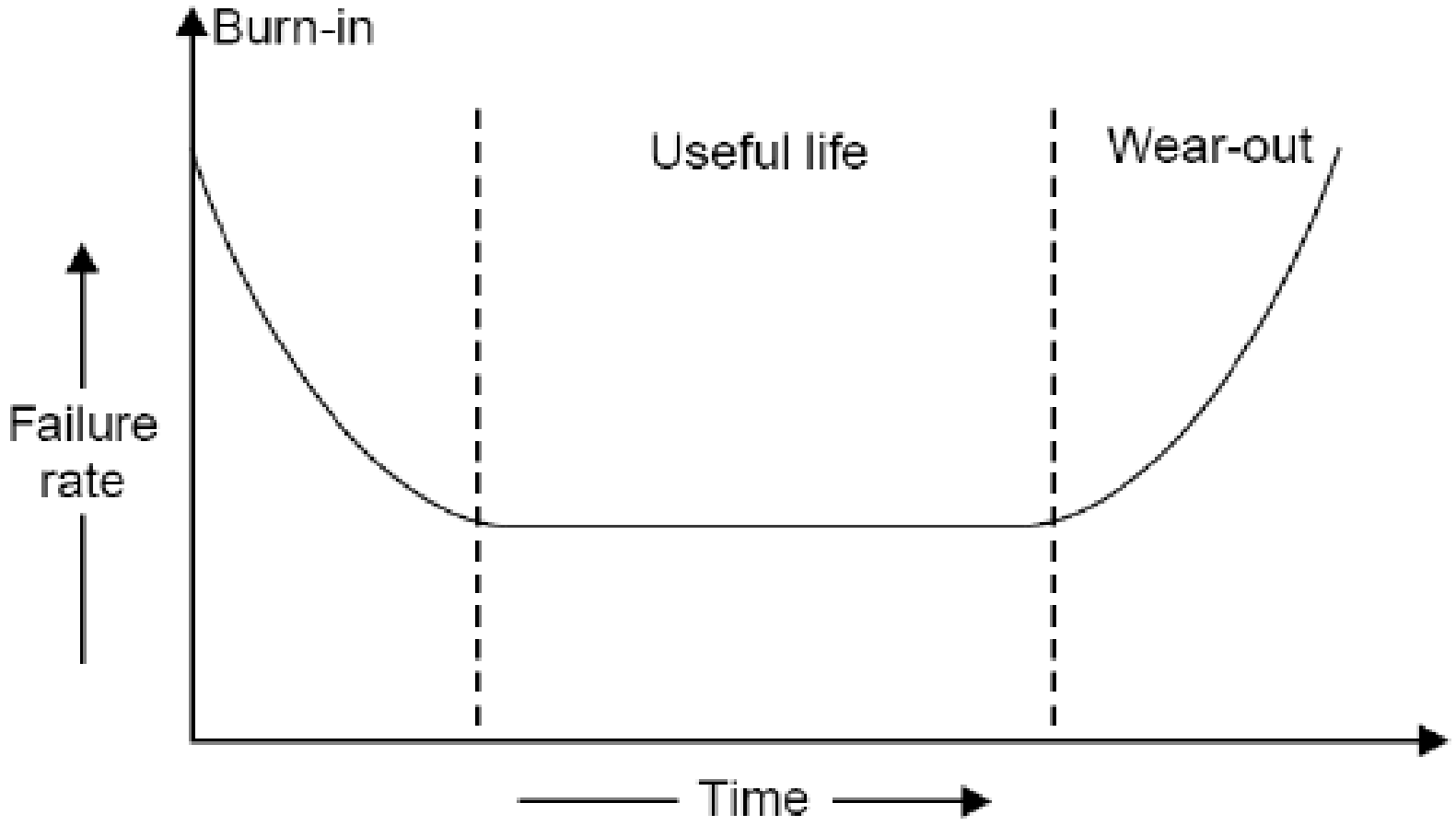
## Basic Concepts

There are three phases in the life of any hardware component i.e., burn-in, useful life & wear-out.

In **burn-in phase,** failure rate is quite high initially, and it starts decreasing gradually as the time progresses.

During **useful life period**, failure rate is approximately constant.

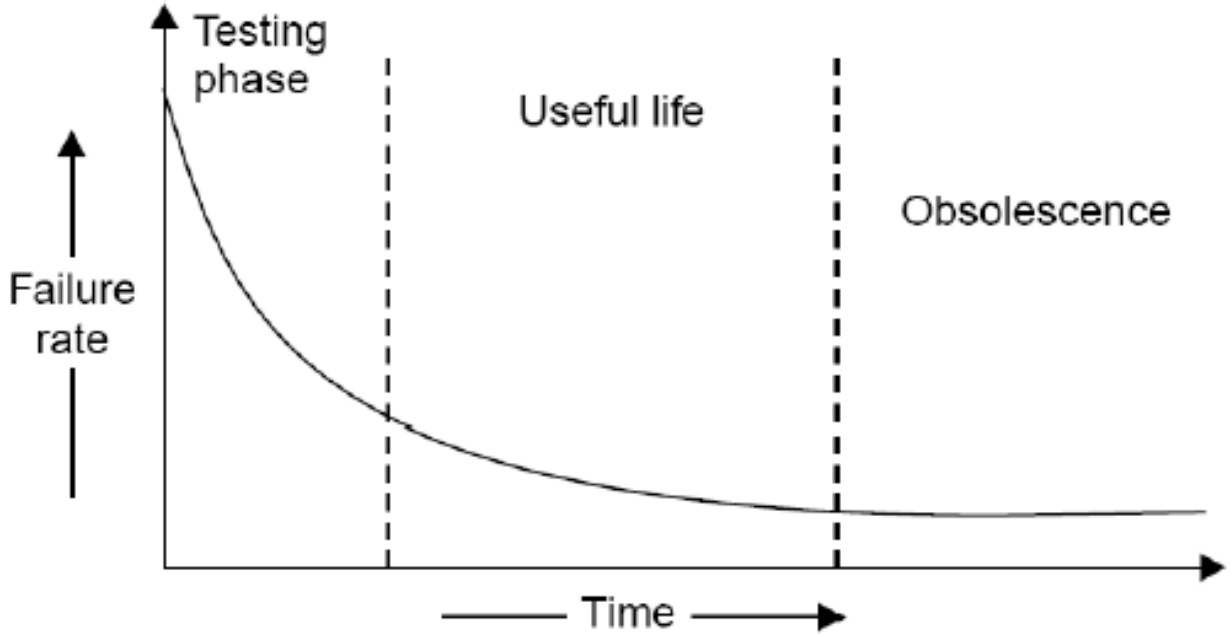
Failure rate increase in **wear-out phase** due to wearing out/aging of components. The best period is useful life period. The shape of this curve is like a "bath tub" and that is why it is known as bath tub curve. The "bath tub curve" is given in Fig.




Bath tub curve of hardware reliability.

# Software Reliability

We do not have wear out phase in software. The expected curve for software is given in fig. 7.2.




Software reliability curve (failure rate versus time)

# *Software Reliability*

Software may be retired only if it becomes obsolete. Some of contributing factors are given below:

- ✓ change in environment
- ✓ change in infrastructure/technology
- ✓ major change in requirements
- ✓ increase in complexity
- ✓ extremely difficult to maintain
- ✓ deterioration in structure of the code
- ✓ slow execution speed
- ✓ poor graphical user interfaces

# *Software Reliability*

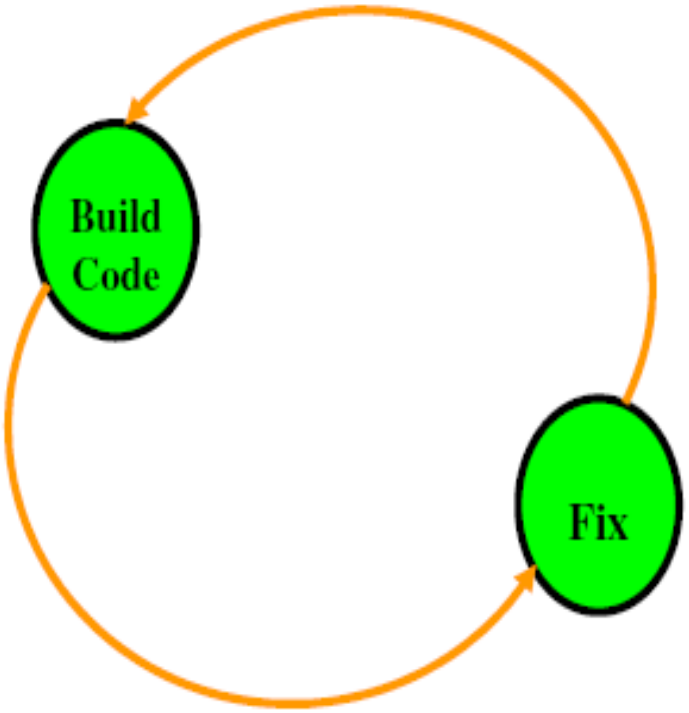
## What is Software Reliability?

"Software reliability means operational reliability. Who cares how many bugs are in the program?

As per IEEE standard: "Software reliability is defined as the ability of a system or component to perform its required functions under stated conditions for a specified period of time".

# Build & Fix Model

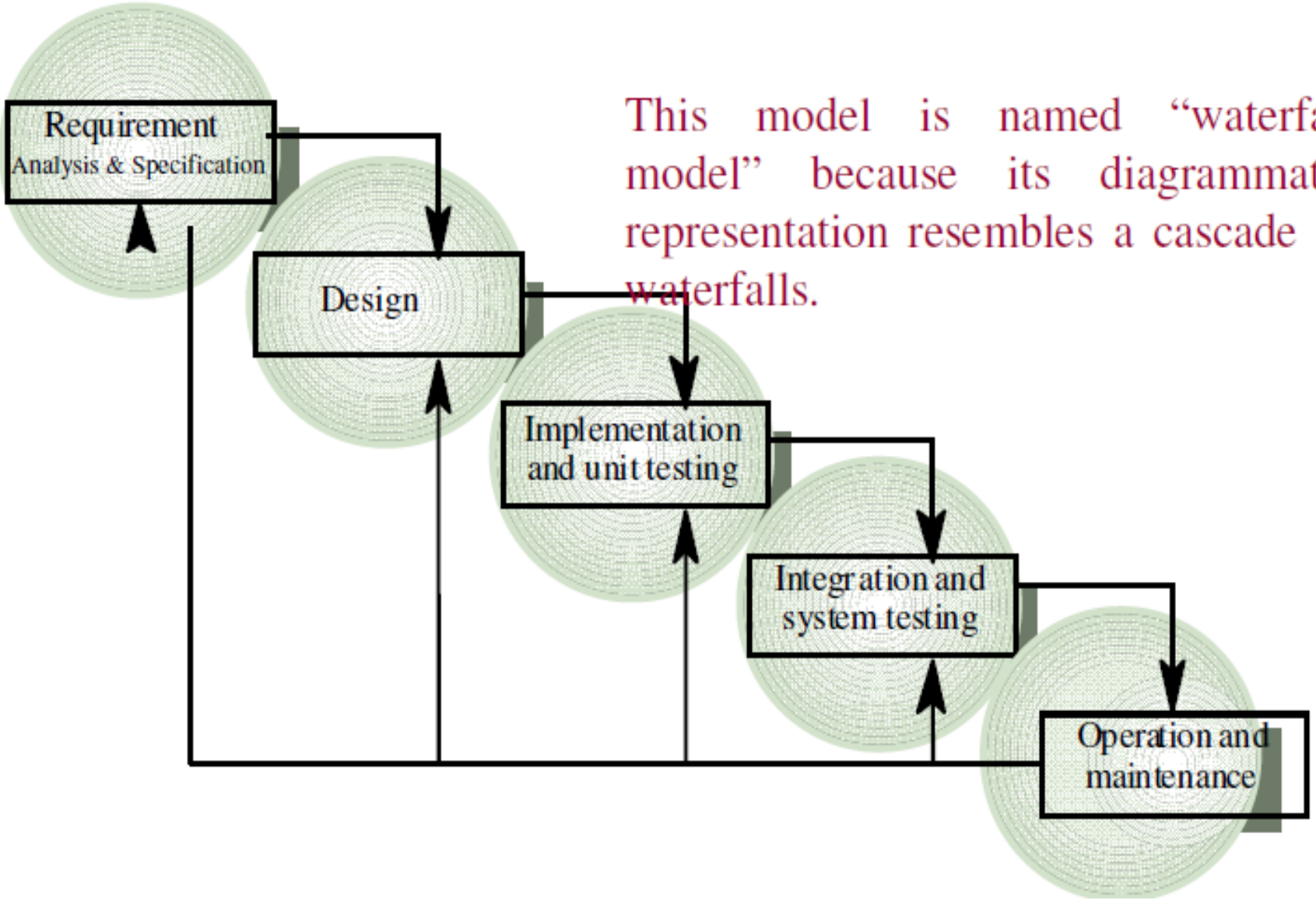
- Product is constructed without specifications or any attempt at design
- Adhoc approach and not well defined
- Simple two phase model



# Build & Fix Model

- Suitable for small programming exercises of 100 or 200 lines
- Unsatisfactory for software for any reasonable size
- Code soon becomes unfixable & unenhanceable
- No room for structured design
- Maintenance is practically not possible

# Waterfall Model




This model is named "waterfall model" because its diagrammatic representation resembles a cascade of waterfalls.

# *Waterfall Model*

This model is easy to understand and reinforces the notion of "define before design" and "design before code".

The model expects complete & accurate requirements early in the process, which is unrealistic

# Waterfall Model

Problems of waterfall model

i. It is difficult to define all requirements at the beginning of a project

ii. This model is not suitable for accommodating any change

iii. A working version of the system is not seen until late in the project's life

iv. It does not scale up well to large projects.

v. Real projects are rarely sequential.

# Incremental Process Models

They are effective in the situations where requirements are defined precisely and there is no confusion about the functionality of the final product.

After every cycle a useable product is given to the customer.

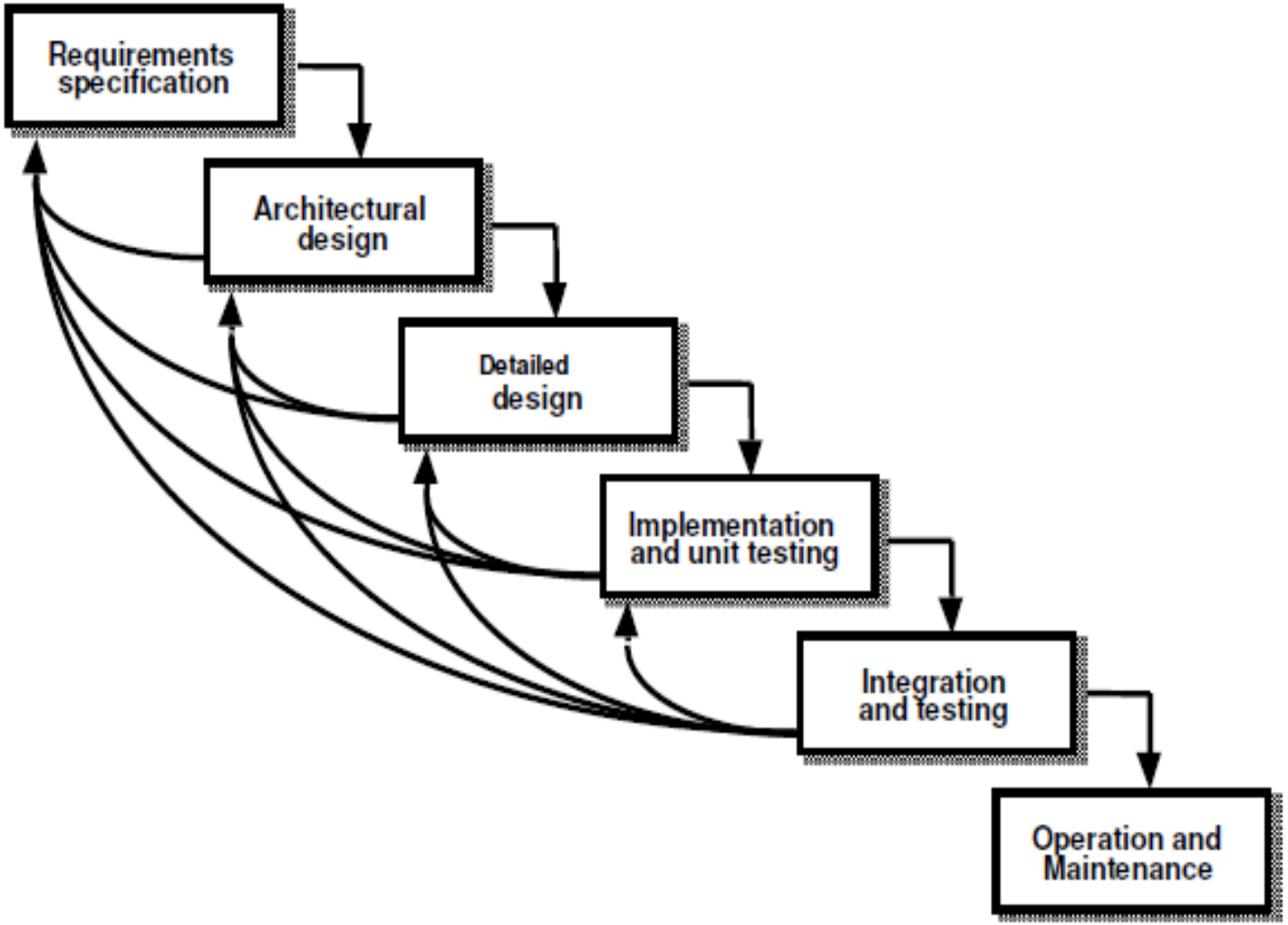
Popular particularly when we have to quickly deliver a limited functionality system.

# *Iterative Enhancement Model*

This model has the same phases as the waterfall model, but with fewer restrictions. Generally the phases occur in the same order as in the waterfall model, but they may be conducted in several cycles. Useable product is released at the end of the each cycle, with each release providing additional functionality.

- ✓ Customers and developers specify as many requirements as possible and prepare a SRS document.
- ✓ Developers and customers then prioritize these requirements
- ✓ Developers implement the specified requirements in one or more cycles of design, implementation and test based on the defined priorities.

# Iterative Enhancement Model

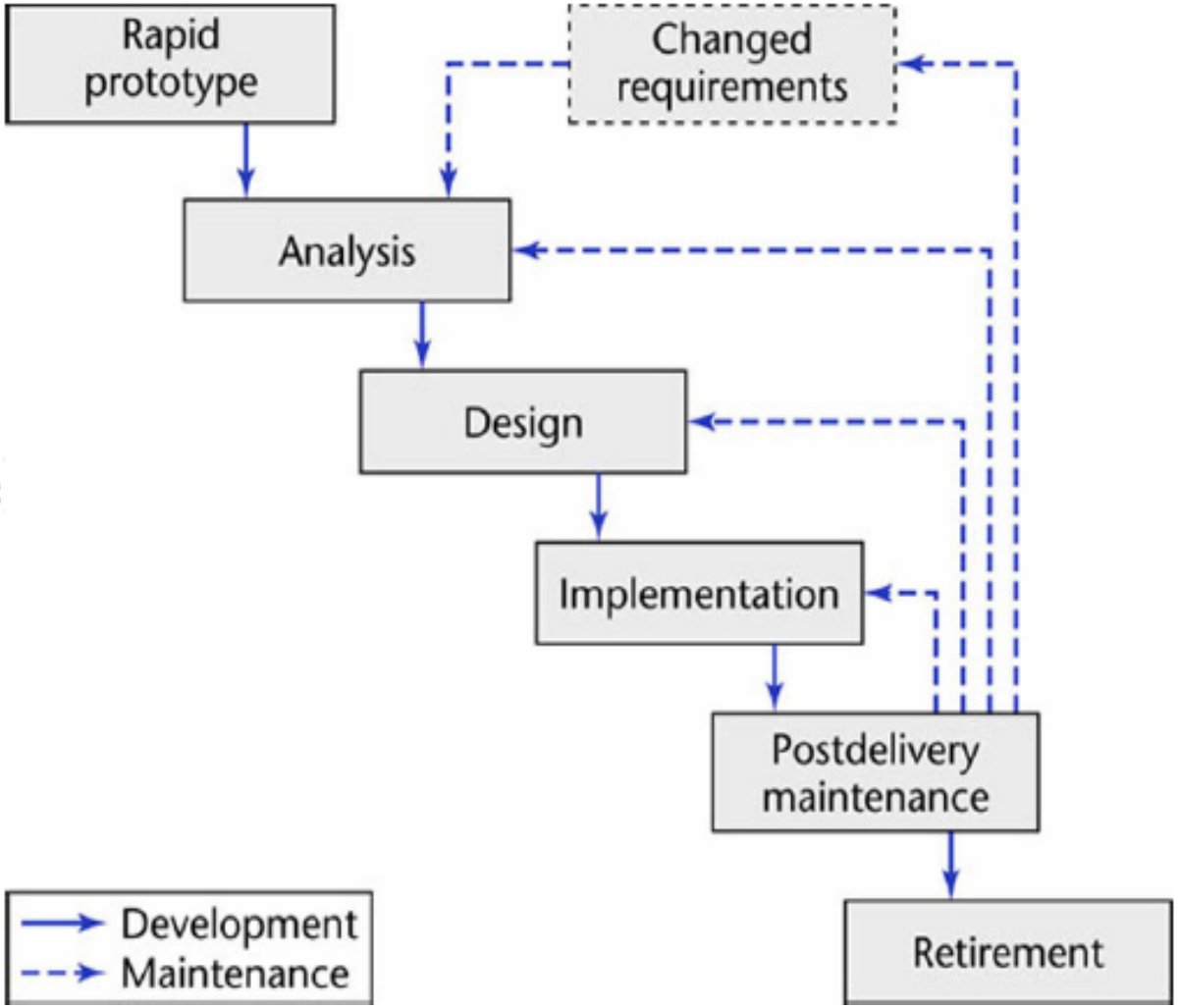


# Prototyping Model

- The prototype may be a usable program but is not suitable as the final software product.
- The code for the prototype is thrown away. However experience gathered helps in developing the actual system.
- The development of a prototype might involve extra cost, but overall cost might turnout to be lower than that of an equivalent system developed using the waterfall model.

# Prototyping Model

- Linear mode
- "Rapid"



# *Spiral Model*

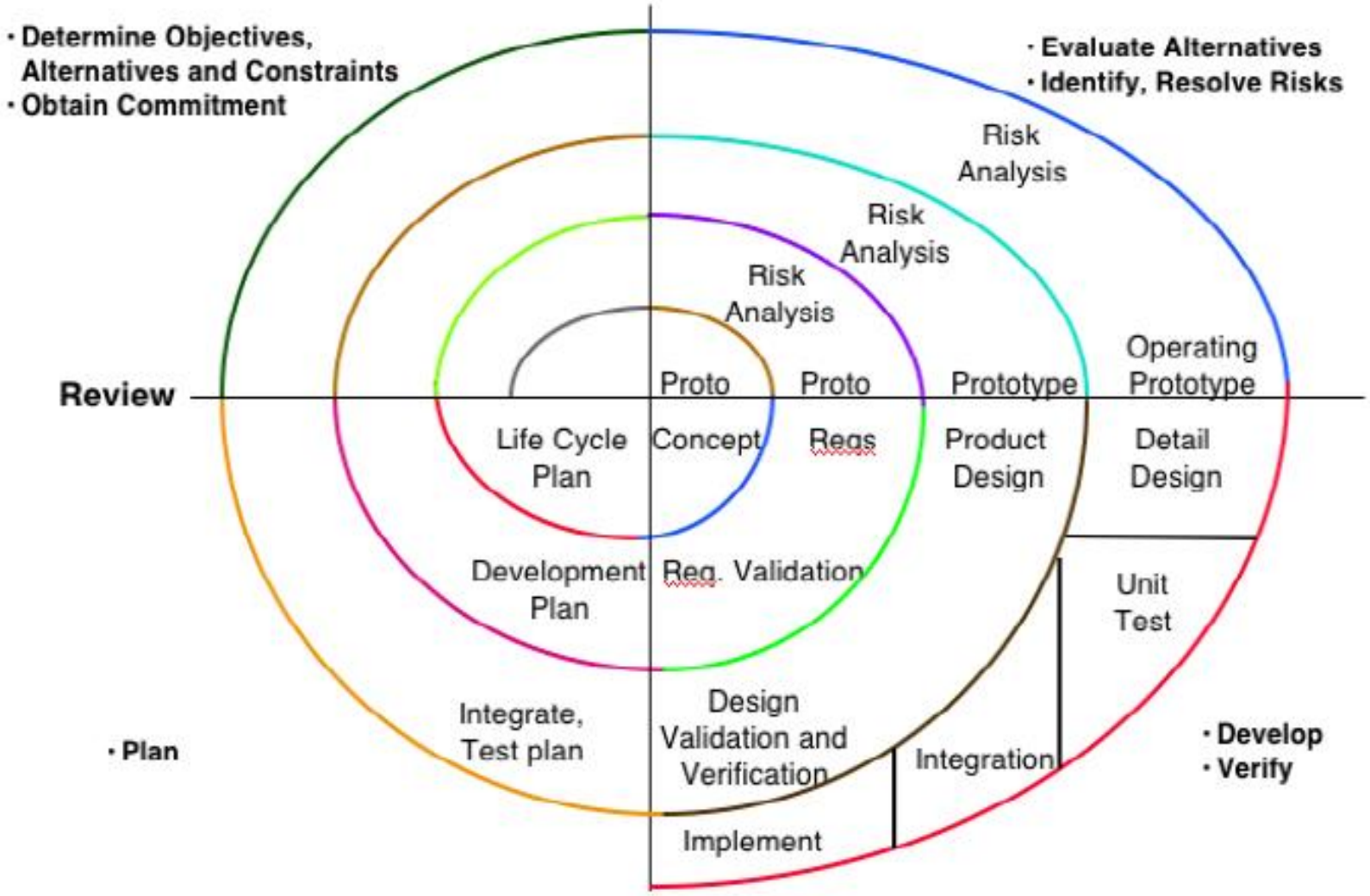
Models do not deal with uncertainly which is inherent to software projects.

Important software projects have failed because project risks were neglected & nobody was prepared when something unforeseen happened.

Barry Boehm recognized this and tired to incorporate the "project risk" factor into a life cycle model.

The result is the spiral model, which was presented in 1986.

# Spiral Model



# *Spiral Model*

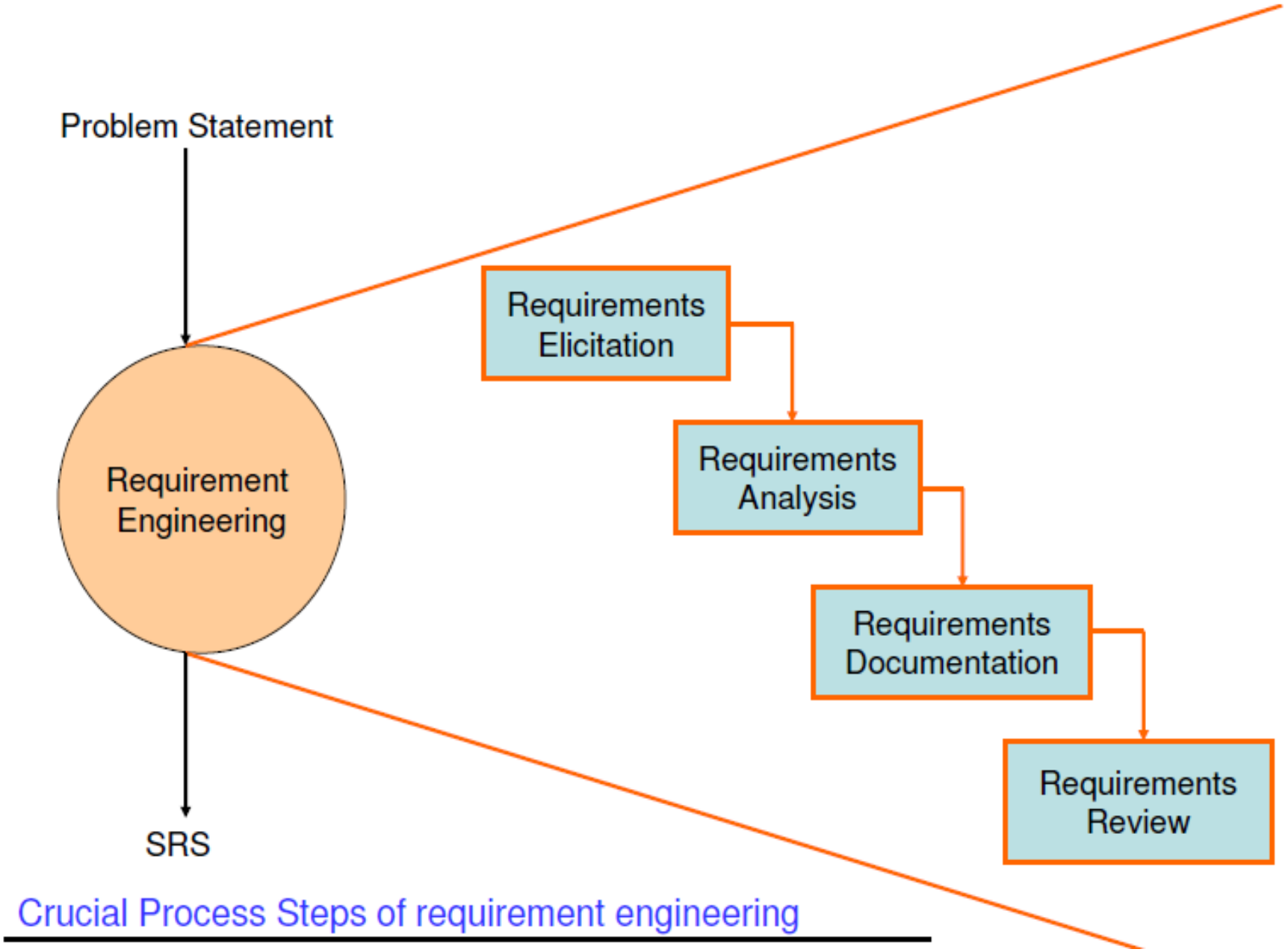
The radial dimension of the model represents the cumulative costs. Each path around the spiral is indicative of increased costs. The angular dimension represents the progress made in completing each cycle. Each loop of the spiral from X-axis clockwise through 360º represents one phase. One phase is split roughly into four sectors of major activities.

- **Planning:** Determination of objectives, alternatives & constraints.
- **Risk Analysis:** Analyze alternatives and attempts to identify and resolve the risks involved.
- **Development:** Product development and testing product.
- **Assessment:** Customer evaluation

# Spiral Model

- An important feature of the spiral model is that each phase is completed with a review by the people concerned with the project (designers and programmers)
- The advantage of this model is the wide range of options to accommodate the good features of other life cycle models.
- It becomes equivalent to another life cycle model in appropriate situations.

The spiral model has some difficulties that need to be resolved before it can be a universally applied life cycle model. These difficulties include lack of explicit process guidance in determining objectives, constraints, alternatives; relying on risk assessment expertise; and provides more flexibility than required for many applications.




Crucial Process Steps of requirement engineering

# Requirement Engineering Process

## It is a four step process, which includes:

- Requirement Elicitation
- Requirements Analysis
- Requirement Documentations
- Requirements Review

# Requirement Elicitation

It is the process to find out the requirements for an intended software system by communicating with client, end users, system users and others who have a stake in the software system development.

# Requirement Analysis

- It is a process to determine the expectation of user.
- Requirement must be detailed and relevant
- These requirements are also called functional specification of a software.

# Requirement Documentations

- Generally we can say that software requirement document is a description of features and functionalities of targeted application/software

- Software requirement documents specify that what the software/application will do.

# Requirement Review

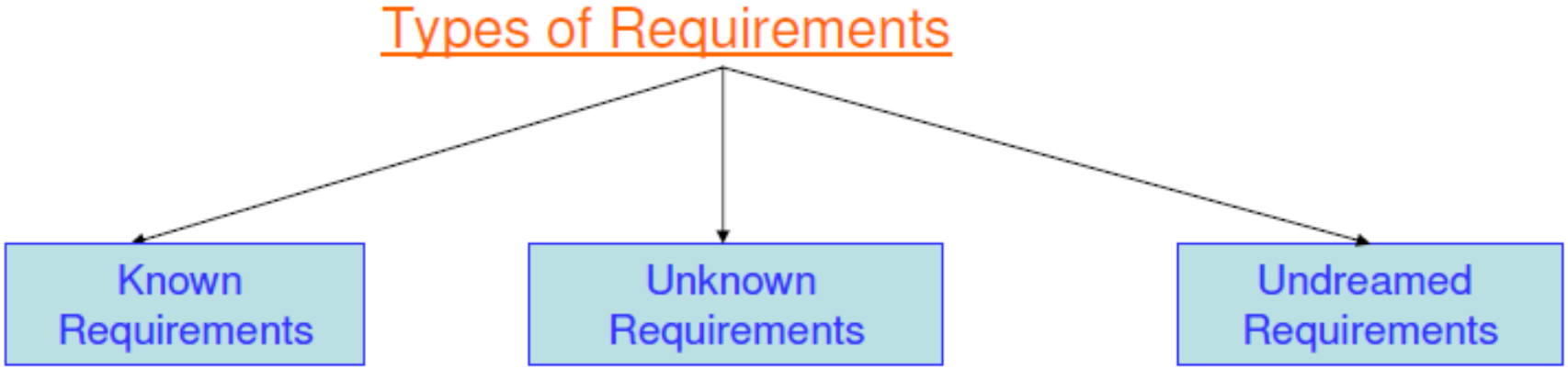
- It is a review process to ensure that all requirements are identified properly.
- It is a process in which both parties involve.

# *Requirement Engineering*

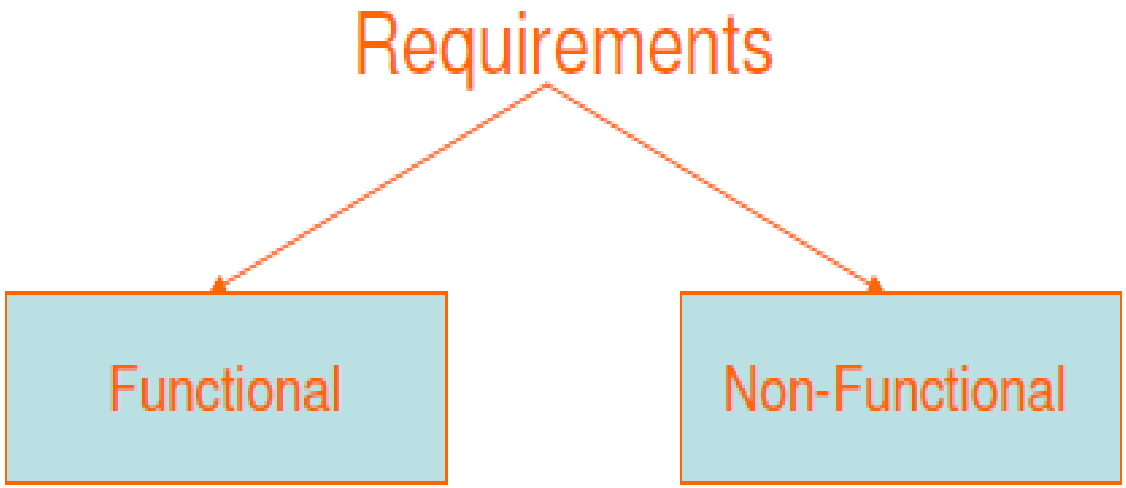
Requirement Engineering is the disciplined application of proven principles, methods, tools, and notations to describe a proposed system's intended behavior and its associated constraints.

SRS may act as a contract between developer and customer.

# Types of Requirements




Stakeholder: Anyone who should have some direct or indirect influence on the system requirements.



# *Types of Requirements*

Functional requirements describe what the software has to do. They are often called product features.

Non Functional requirements are mostly quality requirements. That stipulate how well the software does, what it has to do.

Availability
Reliability
Usability
Flexibility

} For Users

Maintainability
Portability
Testability

} For Developers

# Types of Requirements

## User and system requirements

- User requirement are written for the users and include functional and non functional requirement.
- System requirement are derived from user requirement.
- The user  system requirements are the parts of software requirement and specification (SRS) document.

# Use Cases

A use case is initiated by a user with a particular goal in mind, and completes successfully when that goal is satisfied.

Use Case captures who (actor) does what (interaction) with the system, for what purpose (goal), without dealing with system internals.

Jacobson & others proposed a template for writing Use cases as shown below:

1. Introduction
Describe a quick background of the use case.

2.Actors
List the actors that interact and participate in the use cases.

3.Pre Conditions
Pre conditions that need to be satisfied for the use case to perform.

4. Post Conditions
Define the different states in which we expect the system to be in, after the use case executes.

5. Flow of events

5.1 Basic Flow
List the primary events that will occur when this use case is executed.

5.2 Alternative Flows
Any Subsidiary events that can occur in the use case should be separately listed. List each such event as an alternative flow. A use case can have many alternative flows as required.

6.Special Requirements

Business rules should be listed for basic & information flows as special requirements in the use case narration .These rules will also be used for writing test cases. Both success and failures scenarios should be described.

7.Use Case relationships

For Complex systems it is recommended to document the relationships between use cases. Listing the relationships between use cases also provides a mechanism for traceability

# Use Case Guidelines

1. Identify all users
2. Create a user profile for each category of users including all roles of the users play that are relevant to the system.
3. Create a use case for each goal, following the use case template maintain the same level of abstraction throughout the use case. Steps in higher level use cases may be treated as goals for lower level (i.e. more detailed), sub-use cases.
4. Structure the use case
5. Review and validate with users.

# Use case Diagrams

-- represents what happens when actor interacts with a system.

-- captures functional aspect of the system.

Actor

Use Case

Relationship between actors and use case and/or between the use cases.

-- Actors appear outside the rectangle.

--Use cases within rectangle providing functionality.

--Relationship association is a solid line between actor & use cases.

*Use cases should not be used to capture all the details of the system.

*Only significant aspects of the required functionality

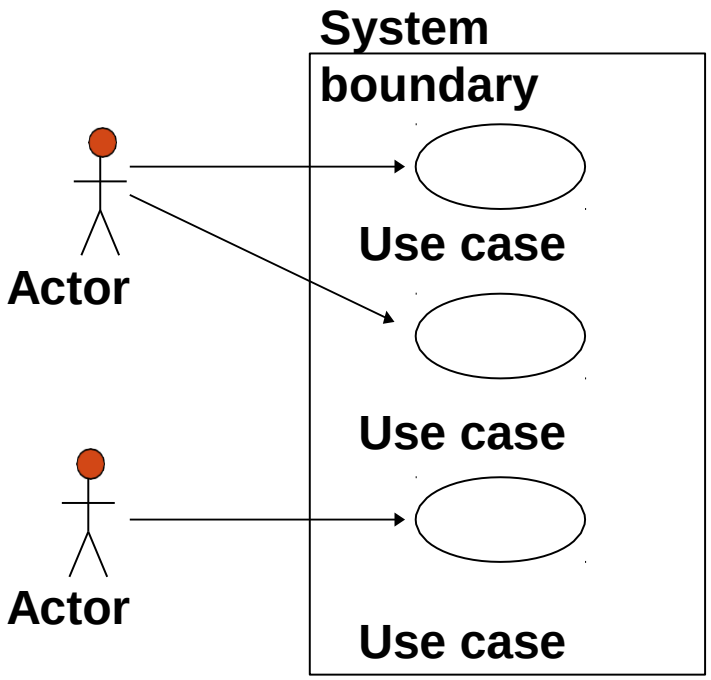
*No design issues

*Use Cases are for “what” the system is , not “how” the system will be designed

* Free of design characteristics

# Components of use case diagram

- **Actor** : Who interacts with the system
- **Use case :** Functionality and services provided by the system
- **Relationship :** Relation between actor and use cases
- **System boundary**




-- Actors appear outside the rectangle.

-- Rectangular box represent the System

--Use cases within rectangle providing functionality.

--Relationship association is a solid line between actor & use cases.

# Actor

An actor or external agent, lies outside the system model, but interacts with it in some way.

**Actor------->** Person/Machine/Information System

Actor may be distinguished as primary actor and secondary actor

**Primary Actor:** Primary actor is one having a goal requiring the assistance of the system.

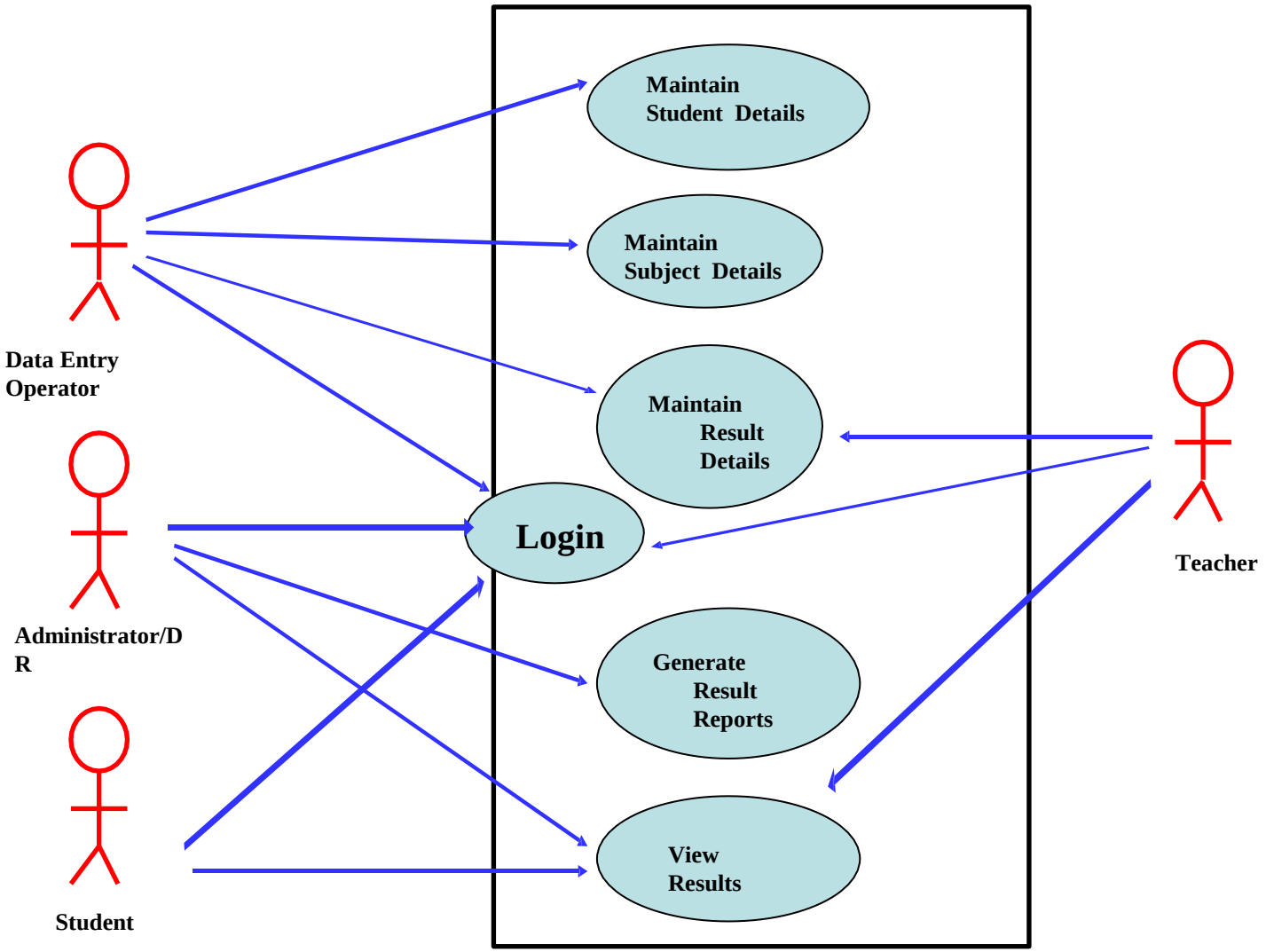
**Secondary Actor:** Secondary actor is one from which system needs assistance.

# Actor

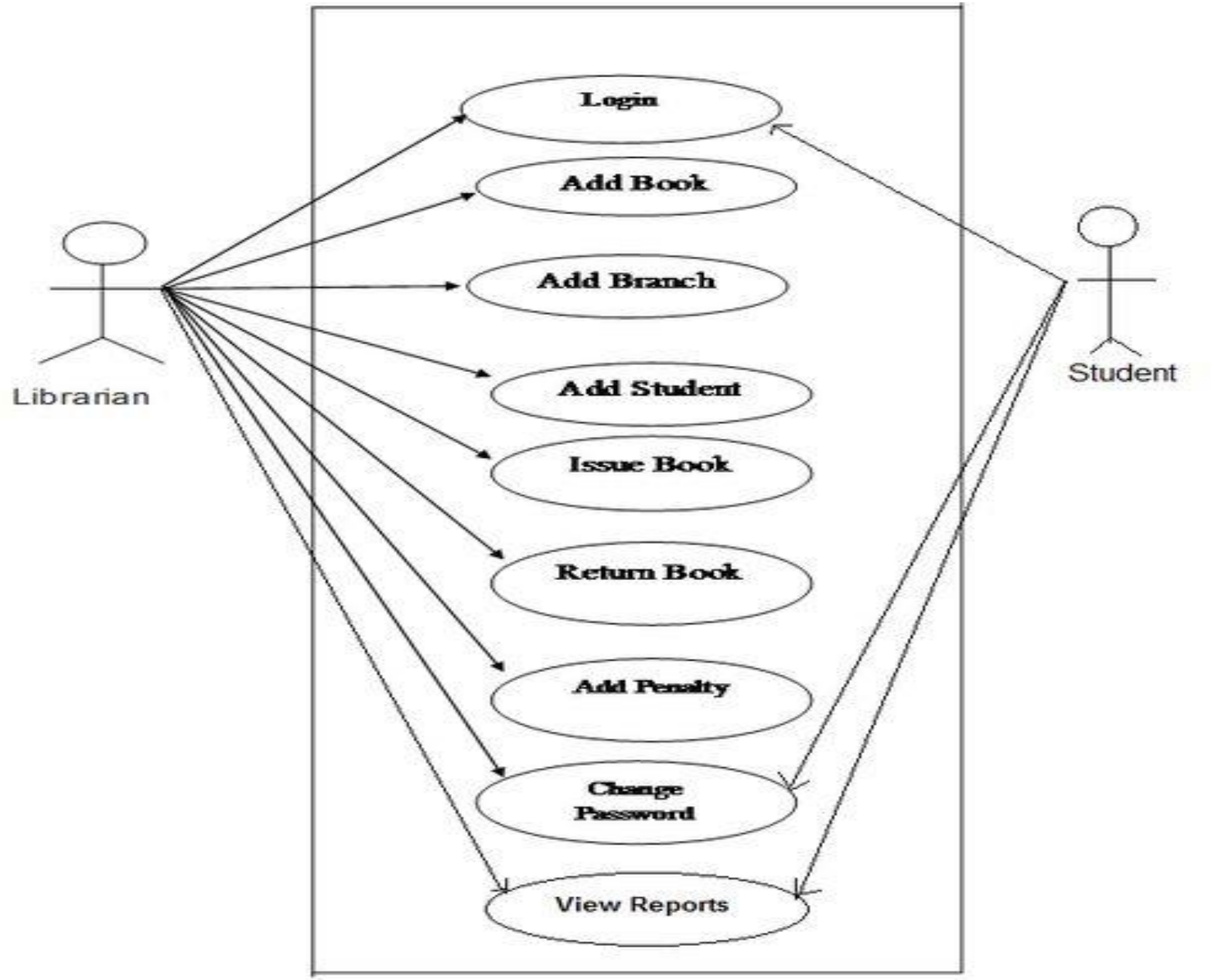
Actor has responsibility toward the system (inputs) and actor has expectations from the system (outputs).

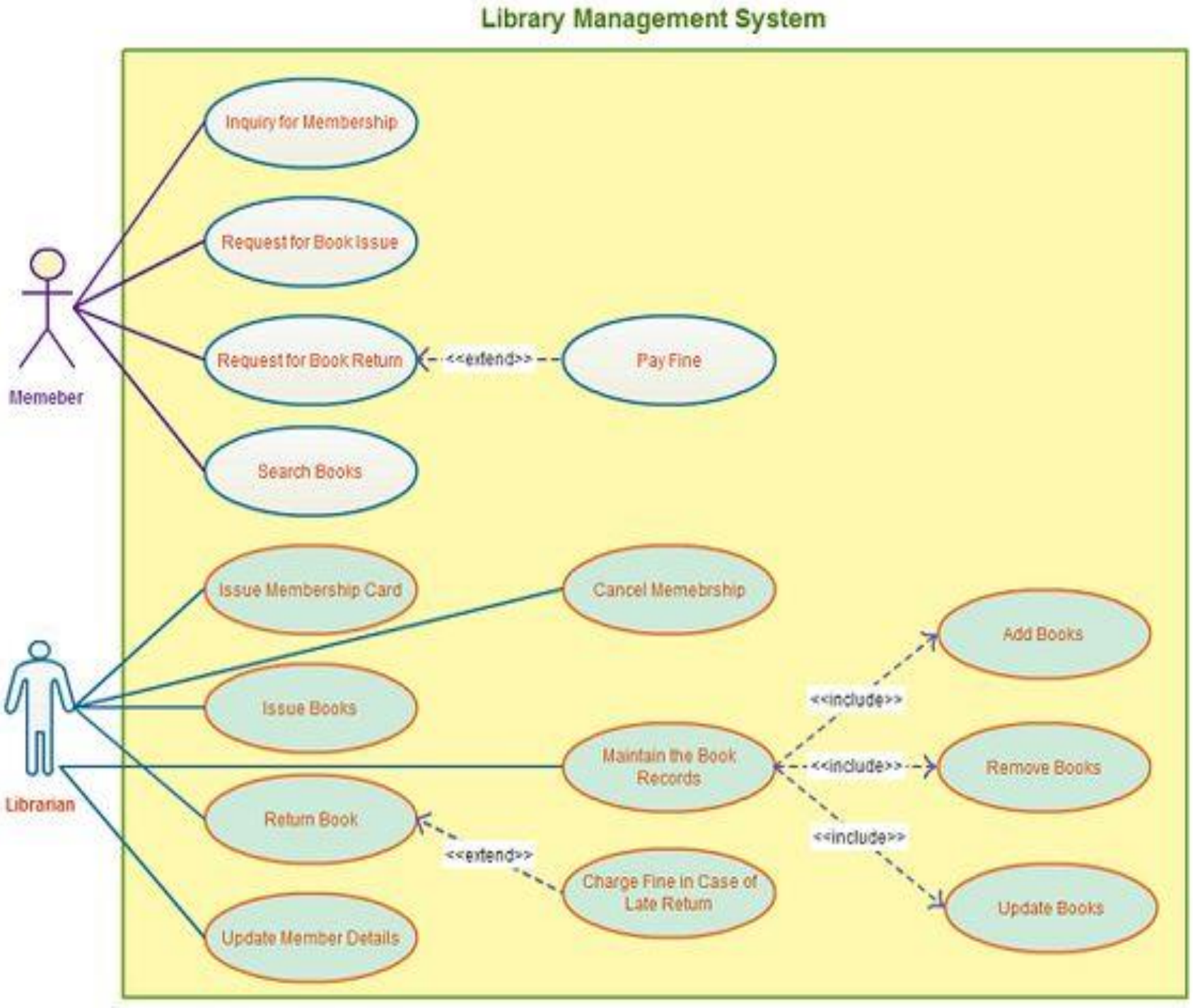
Actors can be a human user, some internal applications, or may be some external applications.

# Use case diagram for Result Management System



# Use case diagram for Library Management System




Library Management System
Inquiry for Membership
Request for Book issue
Request for Book Return
<<extend>>
Pay Fine
Search Books
Memeber
Issue Membership Card
Cancel Memebrship
Issue Books
Add Books
<<include>>
Maintain the Book Records
<<include>>
Remove Books
Librarian
Return Book
<<include>>
<<extend>>
Charge Fine in Case of Late Return
Update Books
Update Member Details

# Example

**Here we will discuss the case for 'Login' to a 'School Management System'.**

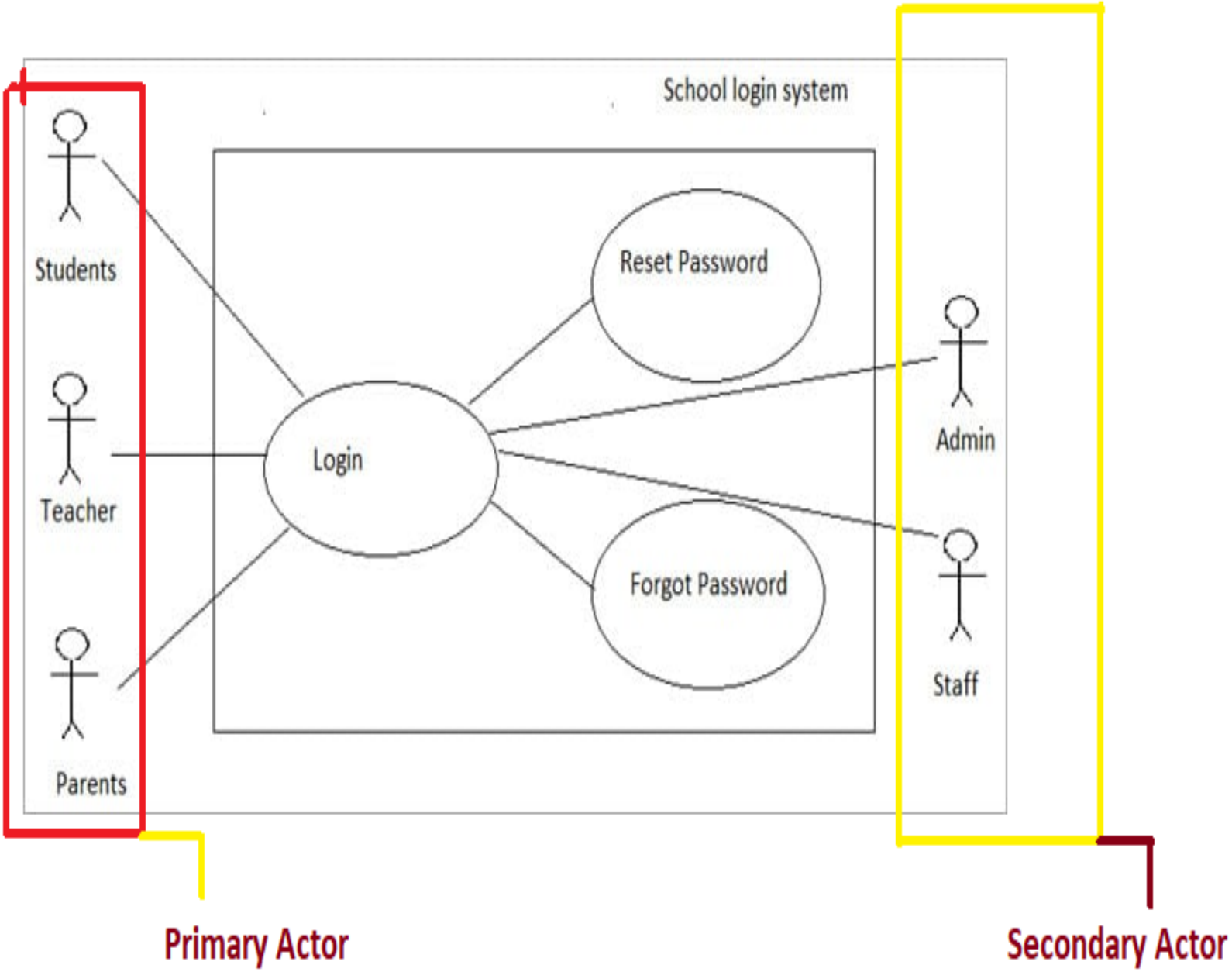
**Use Case Name ----------** **Login**

**Use case Description------** **A user login to System to access the functionality of the system.**

**Actors----------Parents, Students, Teacher, Admin**

**Pre-Condition--------System must be connected to the network.**

**Post –Condition-------** **After a successful login a notification mail is sent to the User mail id**


School login system
Students
Teacher
Parents
Login
Reset Password
Forgot Password
Admin
Staff
Primary Actor
Secondary Actor

Consider a project with the following functional units:

Number of user inputs = 50

Number of user outputs = 40

Number of user enquiries = 35

Number of user files = 06

Number of external interfaces = 04

Assume all complexity adjustment factors and weighting factors are average. Compute the function points for the project.

## Solution

We know

$$UFP = \sum_{i=1}^{5} \sum_{J=1}^{3} Z_{ij} w_{ij}$$

UFP = 50 x 4 + 40 x 5 + 35 x 4 + 6 x 10 + 4 x 7

= 200 + 200 + 140 + 60 + 28 = 628

CAF = $(0.65 + 0.01\ \Sigma F_i)$

= (0.65 + 0.01 (14 x 3)) = 0.65 + 0.42 = 1.07

FP = UFP x CAF

= 628 x 1.07 = 672

### Counting function points

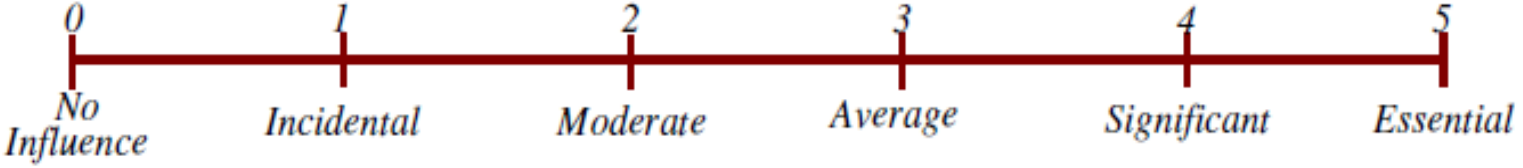
| Functional Units | Weighting factors | | |
|---|---|---|---|
| | Low | Average | High |
| External Inputs (EI) | 3 | 4 | 6 |
| External Output (EO) | 4 | 5 | 7 |
| External Inquiries (EQ) | 3 | 4 | 6 |
| External logical files (ILF) | 7 | 10 | 15 |
| External Interface files (EIF) | 5 | 7 | 10 |

Where CAF is complexity adjustment factor and is equal to [0.65 + 0.01 x $\Sigma F_i$]. The $F_i$ ($i$=1 to 14) are the degree of influence

# Software Project Planning

Computing function points.

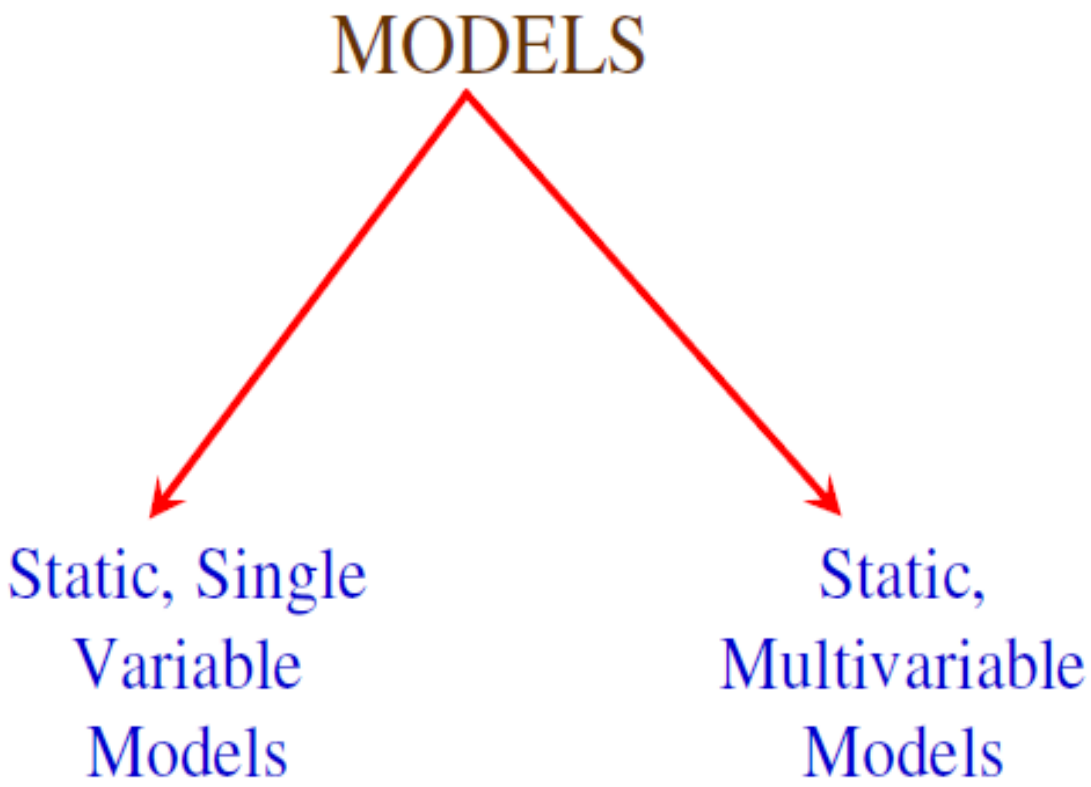
Rate each factor on a scale of 0 to 5.




Number of factors considered ( $F_i$ )

1. Does the system require reliable backup and recovery ?
2. Is data communication required ?
3. Are there distributed processing functions ?
4. Is performance critical ?
5. Will the system run in an existing heavily utilized operational environment ?
6. Does the system require on line data entry ?
7. Does the on line data entry require the input transaction to be built over multiple screens or operations ?
8. Are the master files updated on line ?
9. Is the inputs, outputs, files, or inquiries complex ?
10. Is the internal processing complex ?
11. Is the code designed to be reusable ?
12. Are conversion and installation included in the design ?
13. Is the system designed for multiple installations in different organizations ?
14. Is the application designed to facilitate change and ease of use by the user ?

# *Software Project Planning*



## Static, Single Variable Models

Methods using this model use an equation to estimate the desired values such as cost, time, effort, etc. They all depend on the same variable used as predictor (say, size). An example of the most common equations is :

$$C = a\,L^{b} \qquad \text{(i)}$$

C is the cost, L is the size and a,b are constants

$E = 1.4\,L^{0.93}$

$DOC = 30.4\,L^{0.90}$

$D = 4.6\,L^{0.26}$

Effort (E in Person-months), documentation (DOC, in number of pages) and duration (D, in months) are calculated from the number of lines of code (L, in thousands of lines) used as a predictor.

**The Software Engineering Laboratory established a model called SEL model, for estimating its software production. This model is an example of the static, single variable model.**

## Static, Multivariable Models

These models are often based on equation (i), they actually depend on several variables representing various aspects of the software development environment, for example method used, user participation, customer oriented changes, memory constraints, etc.

$$E = 5.2\ L^{0.91}$$

$$D = 4.1\ L^{0.36}$$

The productivity index uses 29 variables which are found to be highly correlated to productivity as follows:

$$I = \sum_{i=1}^{29} W_i X_i$$

Where $\mathbf{W_i}$ is the weight factor for the $\mathbf{i^{th}}$variable and $\mathbf{X_i}$**={-1,0,+1}** the estimator gives $\mathbf{X_i}$one of the values **-1, 0 or +1** depending on the variable decreases, has no effect or increases the productivity.

WALSTON and FELIX develop the models at IBM.

Compare the Walston-Felix model with the SEL model on a software development expected to involve 8 person-years of effort.

(a)Calculate the number of lines of source code that can be produced.

(b)Calculate the duration of the development.

(c)Calculate the productivity in LOC/PY

(d)Calculate the average manning

## Solution

The amount of manpower involved = 8 PY = 96 person-months

**(a)** Number of lines of source code can be obtained by reversing equation to give:

$$L = (E/a)^{1/b}$$

Then

$$L(SEL) = (96/1.4)^{1/0.93} = 94264 \text{ LOC}$$

$$L(SEL) = (96/5.2)^{1/0.91} = 24632 \text{ LOC.}$$

**(b)** Duration in months can be calculated by means of equation

$$D(SEL) = 4.6\,(L)^{0.26}$$

$$= 4.6\,(94.264)^{0.26} = 15 \text{ months}$$

$$D(W\text{-}F) = 4.1\,L^{0.36}$$

$$= 4.1(24.632)^{0.36} = 13 \text{ months}$$

**(c)** Productivity is the lines of code produced per person/month (year)

$$P(SEL) = \frac{94264}{8} = 11783\,LOC / Person - Years$$

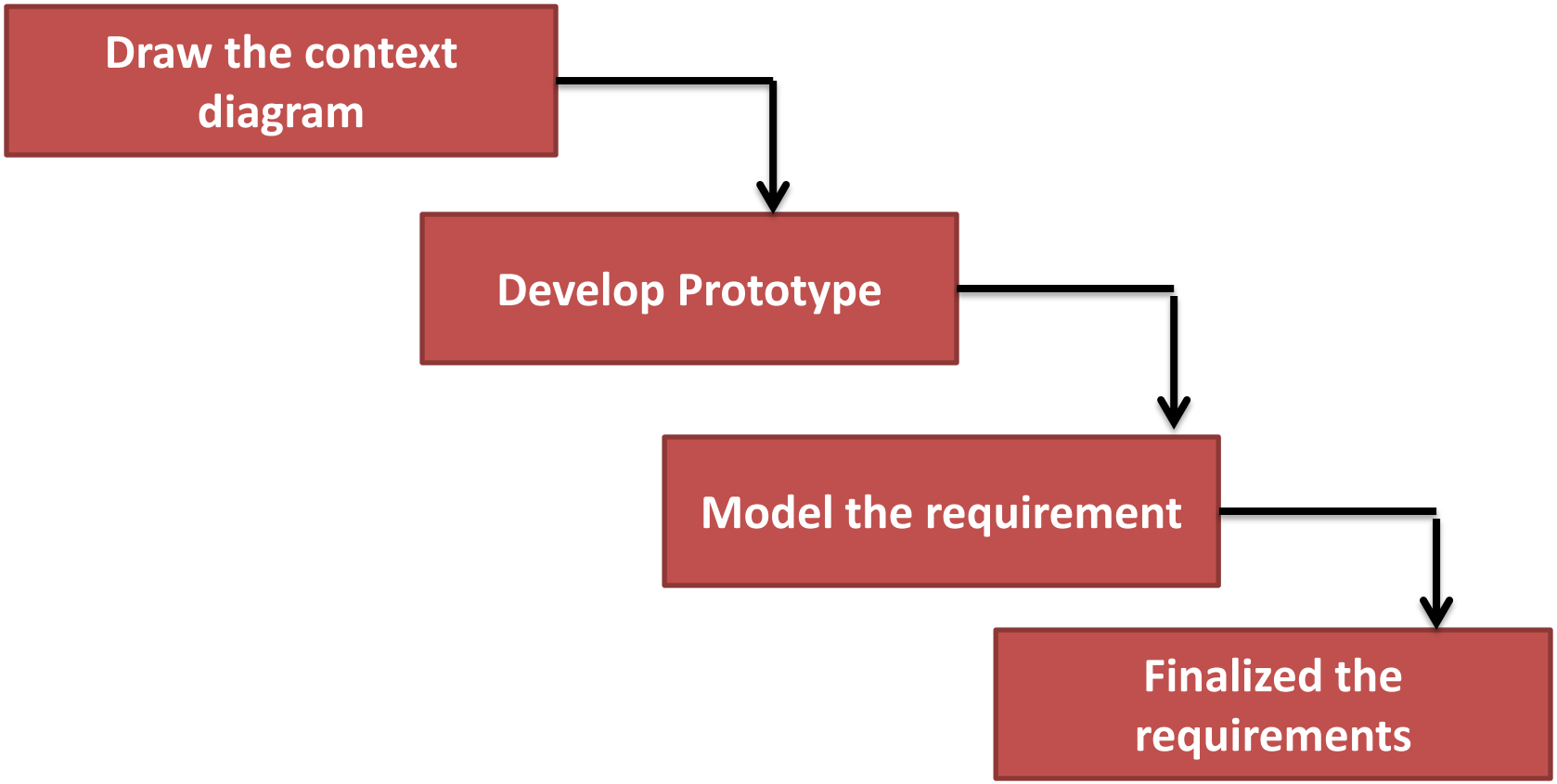
$$P(W - F) = \frac{24632}{8} = 3079\,LOC / Person - Years$$

(**d**) Average manning is the average number of persons required per month in the project.

$$M(SEL) = \frac{96P - M}{15M} = 6.4 Persons$$

$$M(W - F) = \frac{96P - M}{13M} = 7.4 Persons$$

# Requirement Analysis Steps



# Requirement Analysis Steps(cont..)

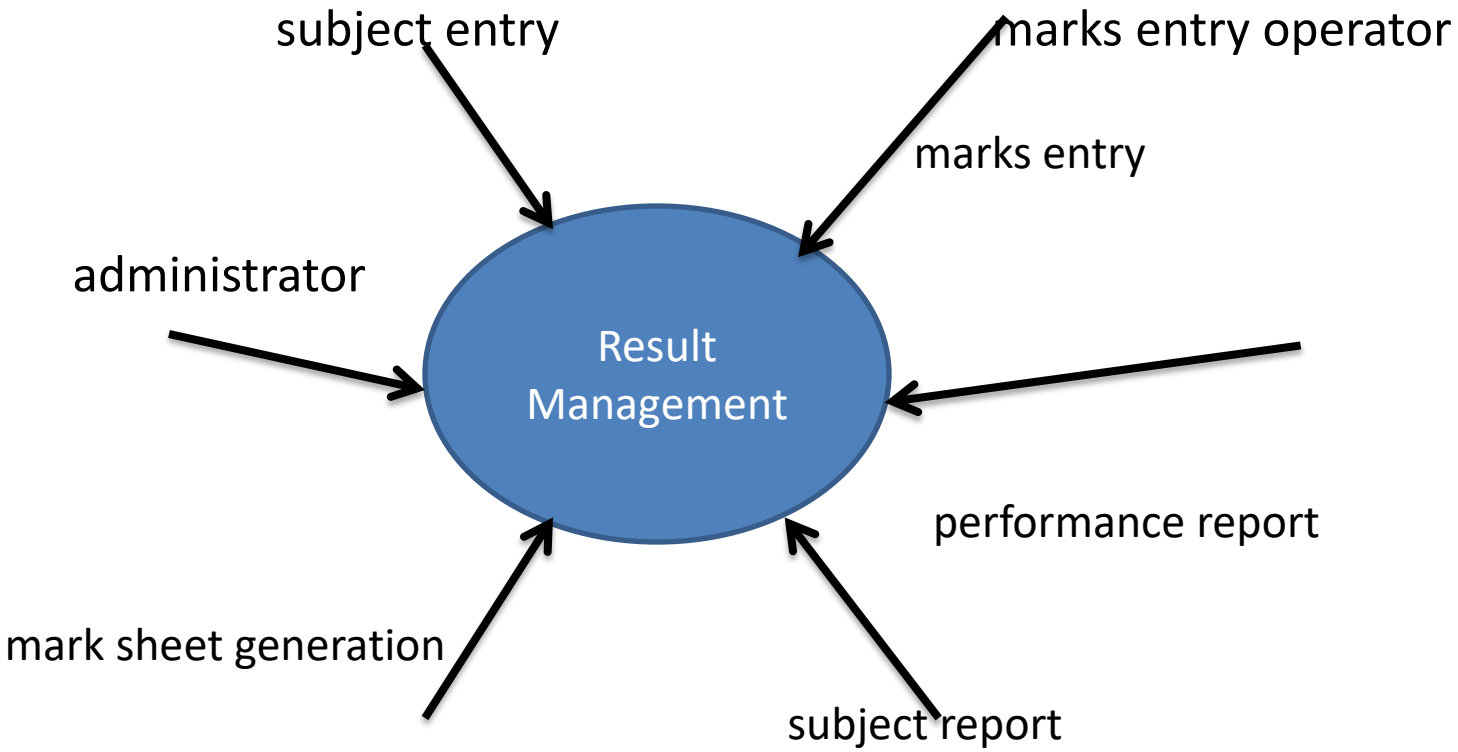
**Draw the context diagram**

It is a simple diagram.

It defines interface and boundaries of system with external environment.

It identifies the  entities outside the proposed system, which interact with the system

# Context diagram....



# Develop Prototype

One effective way to find out the customer's expectation is to construct a prototype.

Customer's feedback can be used to modify the prototype until the customer is satisfied

In case where developers and users are not sure about some of the elements, a prototype may help both the parties to take a final decision.

# Model the requirement

This process generally consists of various graphical representations of the functions, data entities, external entities, and the relationships between them.

# Finalized the requirements

After framing the requirements, we will have a better understanding of the system behavior.

The inconsistencies and ambiguities have been identified and corrected.

The flow of data amongst various modules has been analyzed.

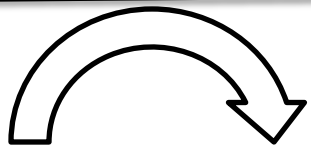
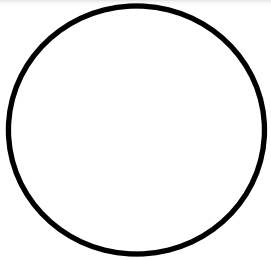
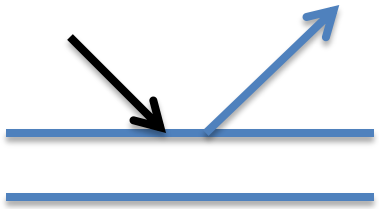
Elicitation and analyze activities have provided better insight into the system.

After that we finalize the analyzed requirements, and the next step is to prepare documentation in proper way.

# Data Flow Diagram

- It shows the flow of data through system.
- All names should be unique
- It shows how the data enter and leaves the system and where data is stored.
- The objective of DFD is to show the scope and boundaries of a system as whole.
- It is also called data flow graph or bubble chart.

# Standard symbols of Data flow diagram

| Symbol | Name | Function |
| --- | --- | --- |
|  | **Data Flow** | It is used to connect processes with each other |
|  | **Process** | Perform transformation of its input data to output data |
|  | **Source or sink** | A source of system inputs or sink of system outputs |
|  | **Data Store** | A repository of data a arrow head indicate net input and net output of data |

# Levels in Data Flow Diagrams (DFD)

## 0-level DFDM

- It is also known as fundamental system model, or context diagram represents the entire software requirement as a single bubble with input and output data denoted by incoming and outgoing arrows.
- Then the system is decomposed and described as a DFD with multiple bubbles. Parts of the system represented by each of these bubbles are then decomposed and documented as more and more detailed DFDs.

# 0-level DFDM(continue)

This process may be repeated at as many levels as necessary until the program at hand is well understood.

It is essential to preserve the number of inputs and outputs between levels, this concept is called leveling by DeMacro. Thus, if bubble "A" has two inputs $x_1$ and $x_2$ and one output y, then the expanded DFD, that represents "A" should have exactly two external inputs and one external output as shown in fig:

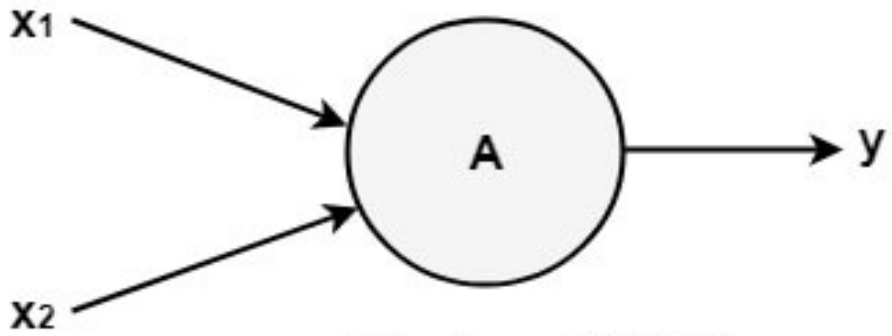



Fig: Level-0 DFD.

The Level-0 DFD, also called context diagram.

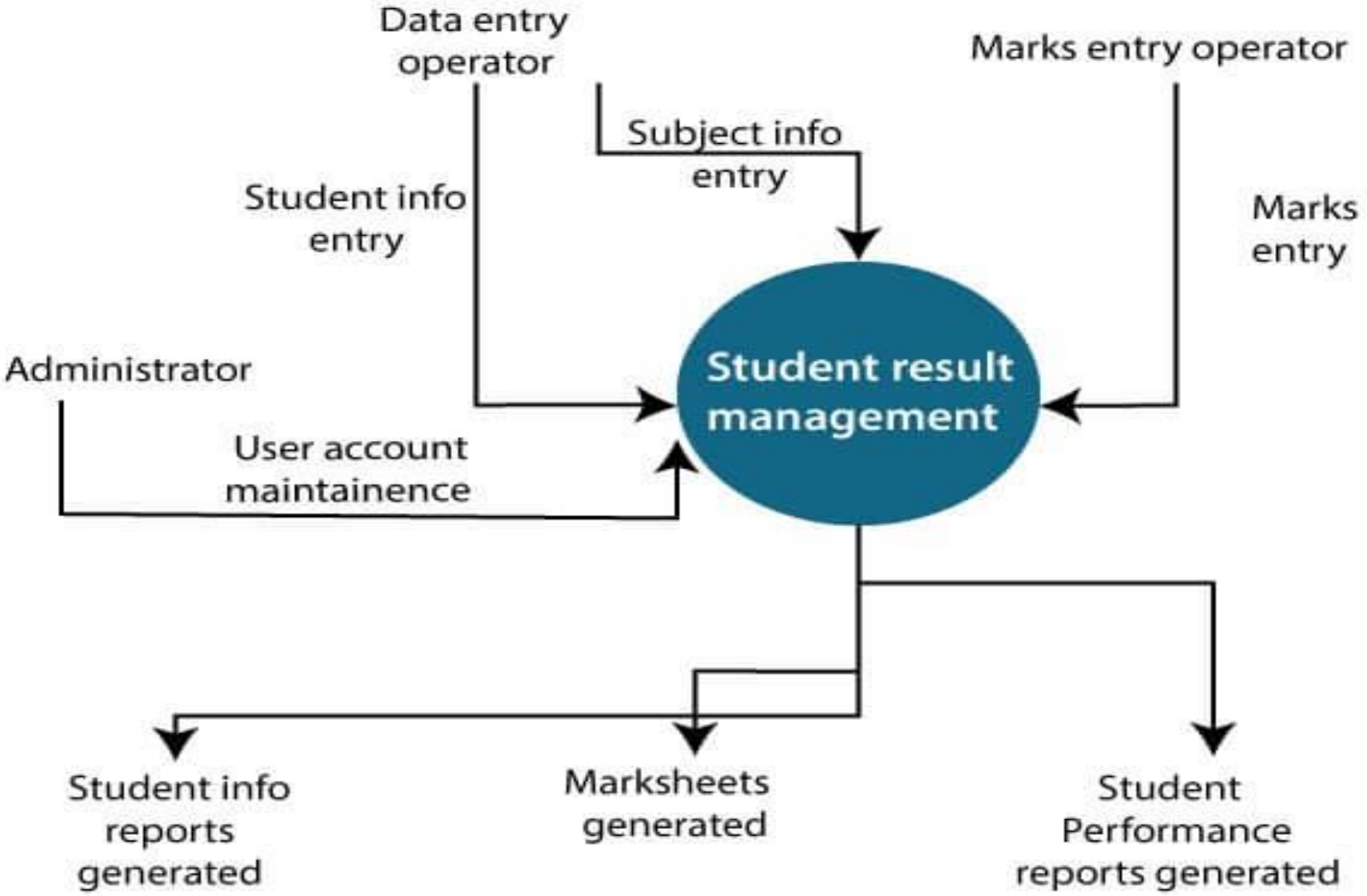
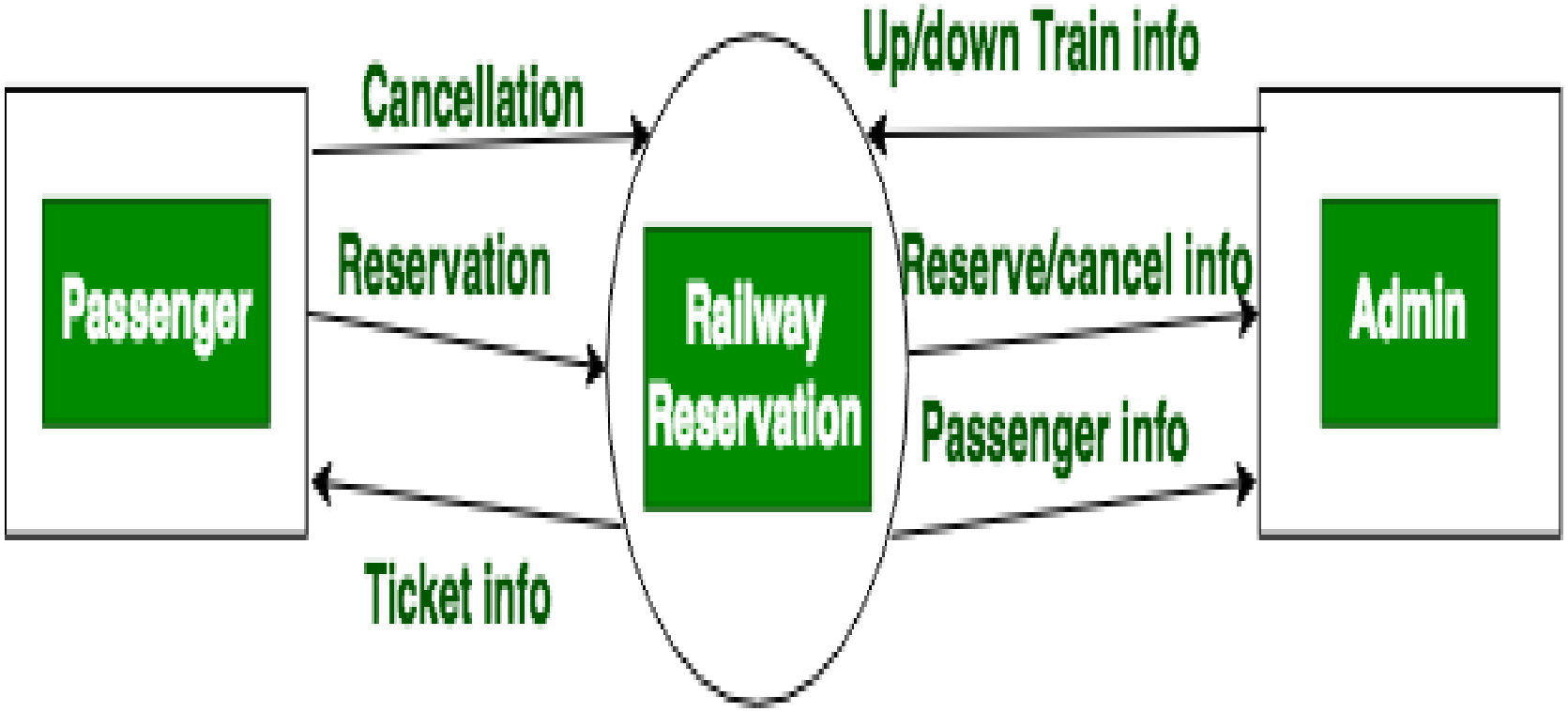
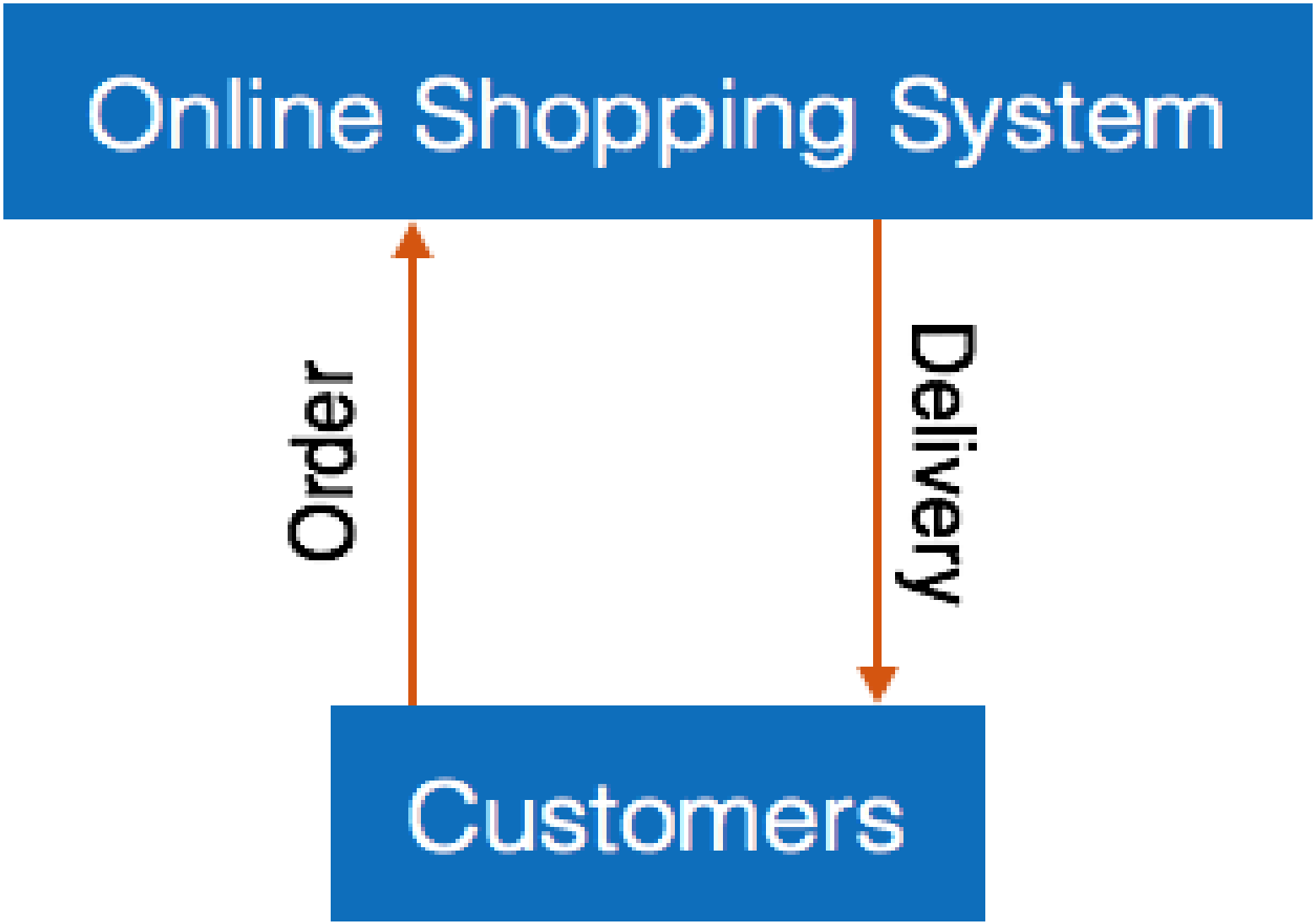



**Fig: Level-0 DFD of result management system**


Up/down Train info
Cancellation
Passenger
Reservation
Railway Reservation
Reserve/cancel info
Admin
Passenger info
Ticket info

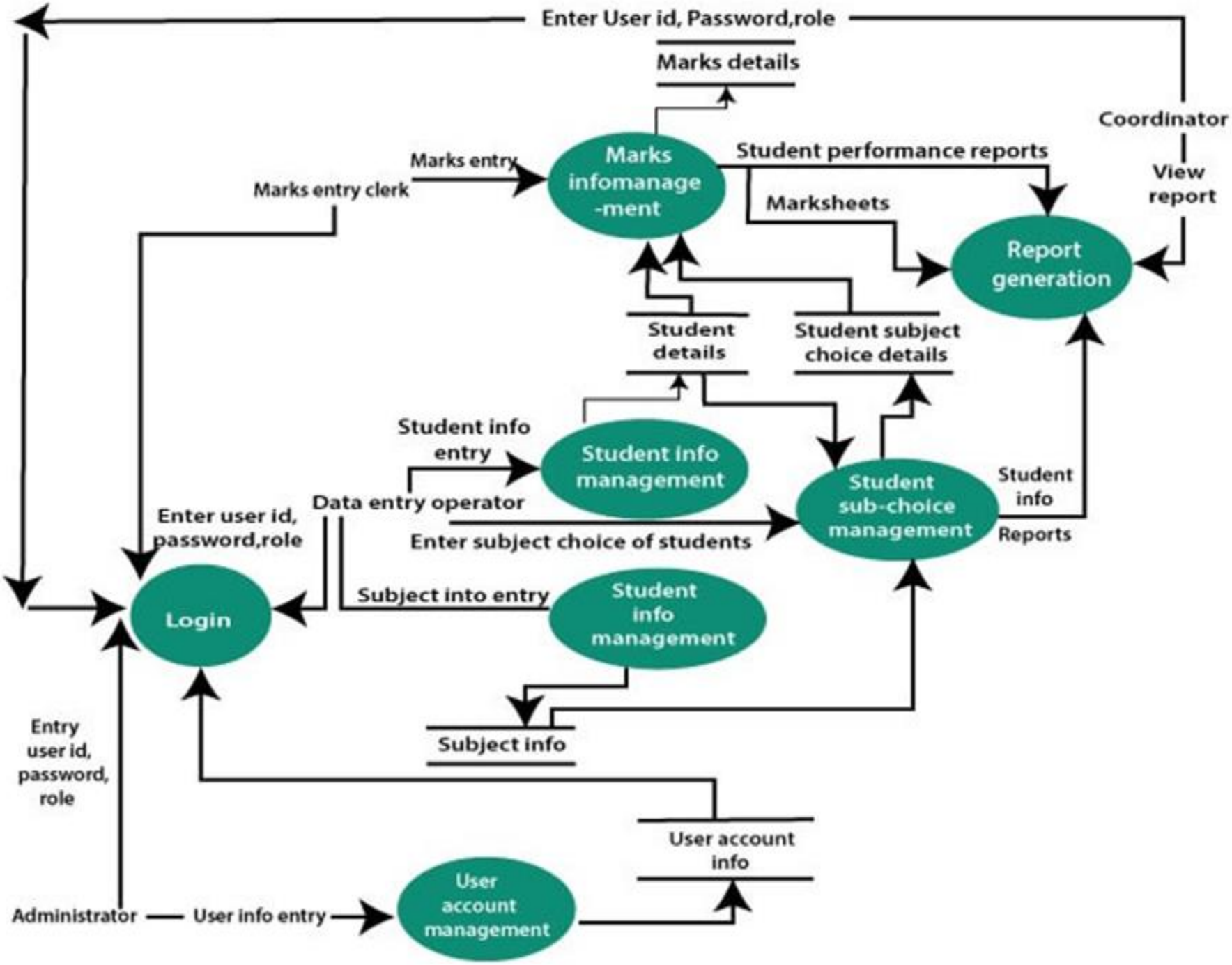
# 0-LEVEL DFD

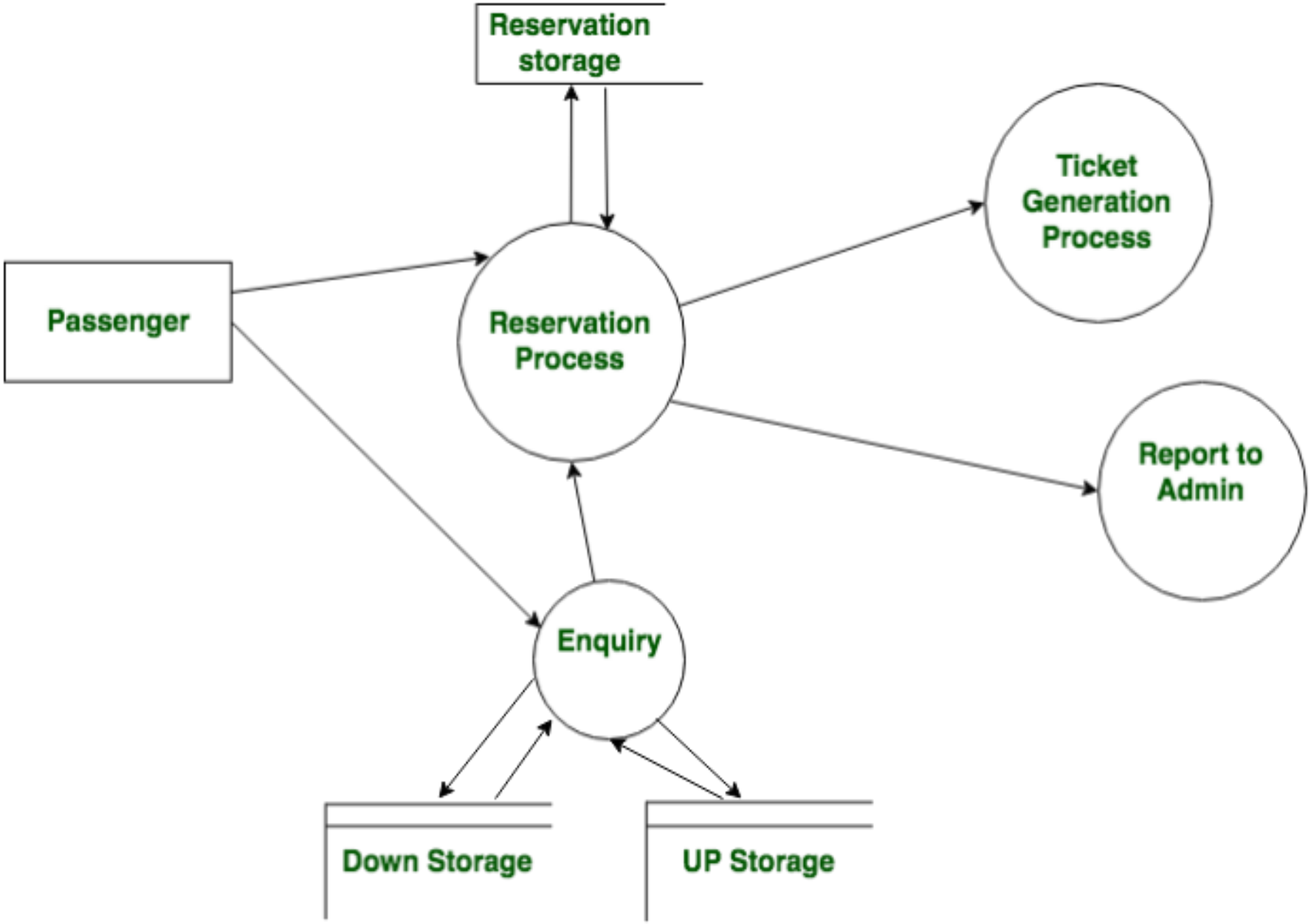

Online Shopping System
Order
Delivery
Customers

# 1-level DFD

- In 1-level DFD, a context diagram is decomposed into multiple bubbles/processes. In this level,

- we highlight the main objectives of the system and breakdown the high-level process of 0-level DFD into sub processes.

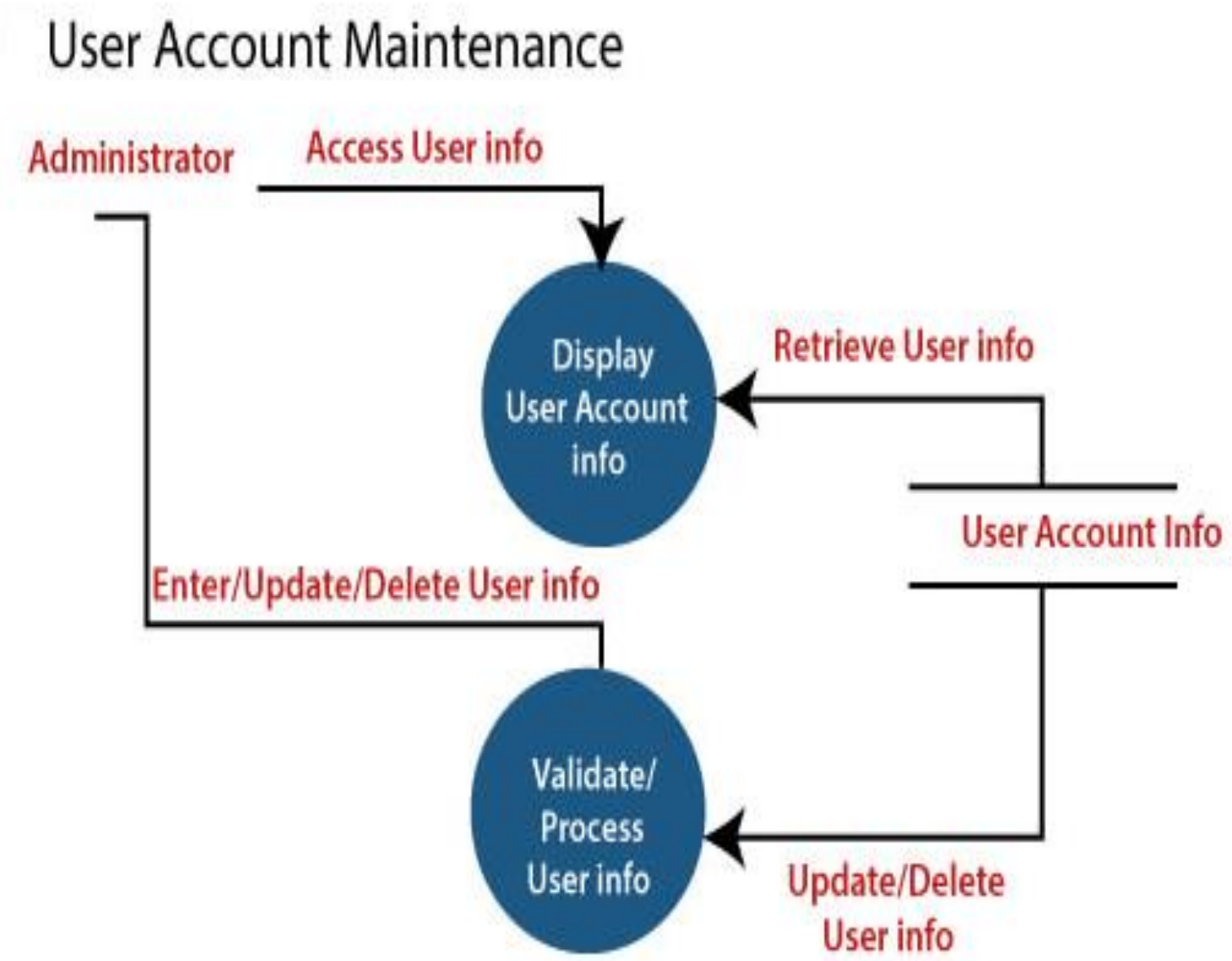

Enter User id, Password,role
Marks details
Coordinator
Marks entry
Marks entry clerk
Marks infomanage -ment
Student performance reports
View report
Marksheets
Report generation
Student details
Student subject choice details
Student info entry
Student info management
Data entry operator
Student sub-choice management
Student info
Reports
Enter user id, password,role
Enter subject choice of students
Login
Subject into entry
Student info management
Entry user id, password, role
Subject info
User account info
Administrator
User info entry
User account management


Reservation storage
Ticket Generation Process
Passenger
Reservation Process
Report to Admin
Enquiry
Down Storage
UP Storage

# 1-LEVEL DFD

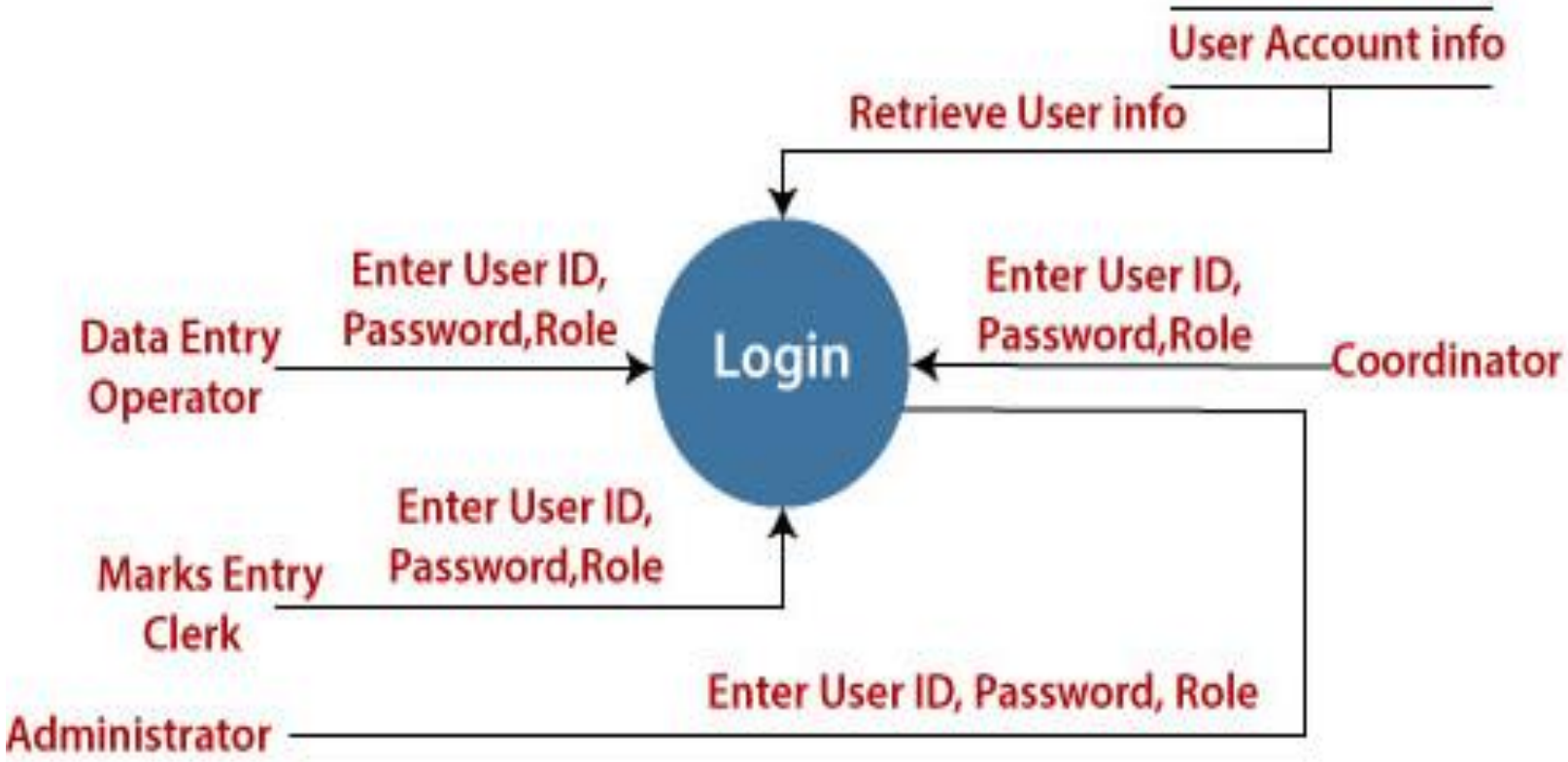
# 2-Level DFD

- 2-level DFD goes one process deeper into parts of 1-level DFD. It can be used to project or record the specific/necessary detail about the system's functioning.


User Account Maintenance
Administrator
Access User info
Display
User Account
info
Retrieve User info
User Account Info
Enter/Update/Delete User info
Validate/
Process
User info
Update/Delete
User info

# Login

The level 2 DFD of this process is given below:

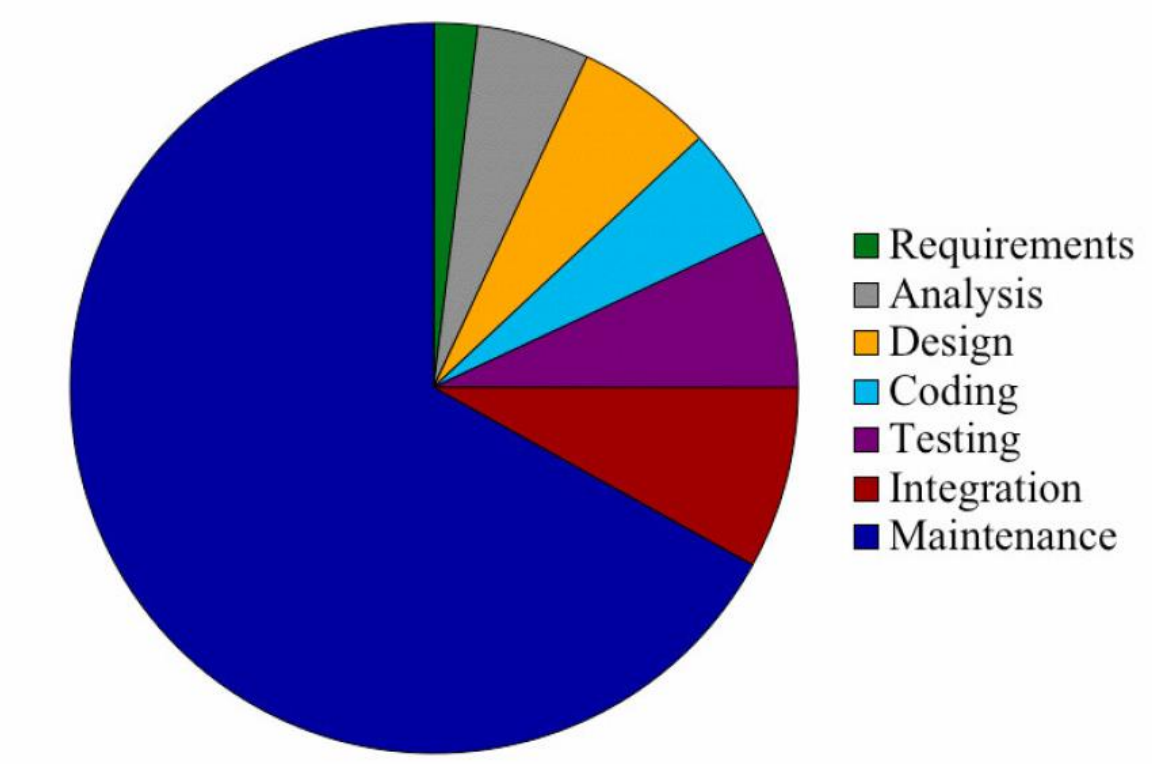



# Software Engineering

## Software Project Planning
## **Cost Estimation**

# Software Project Planning

## Relative Cost of Software Phases:

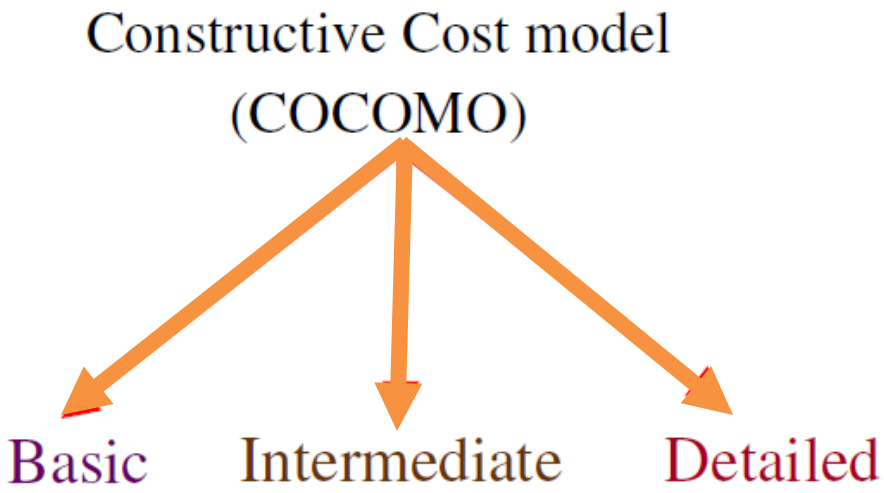



Source: Software Engineering (3rd ed.), By K.K Aggarwal & Yogesh Singh


# Software Engineering

## Software Project Planning
## COCOMO MODEL

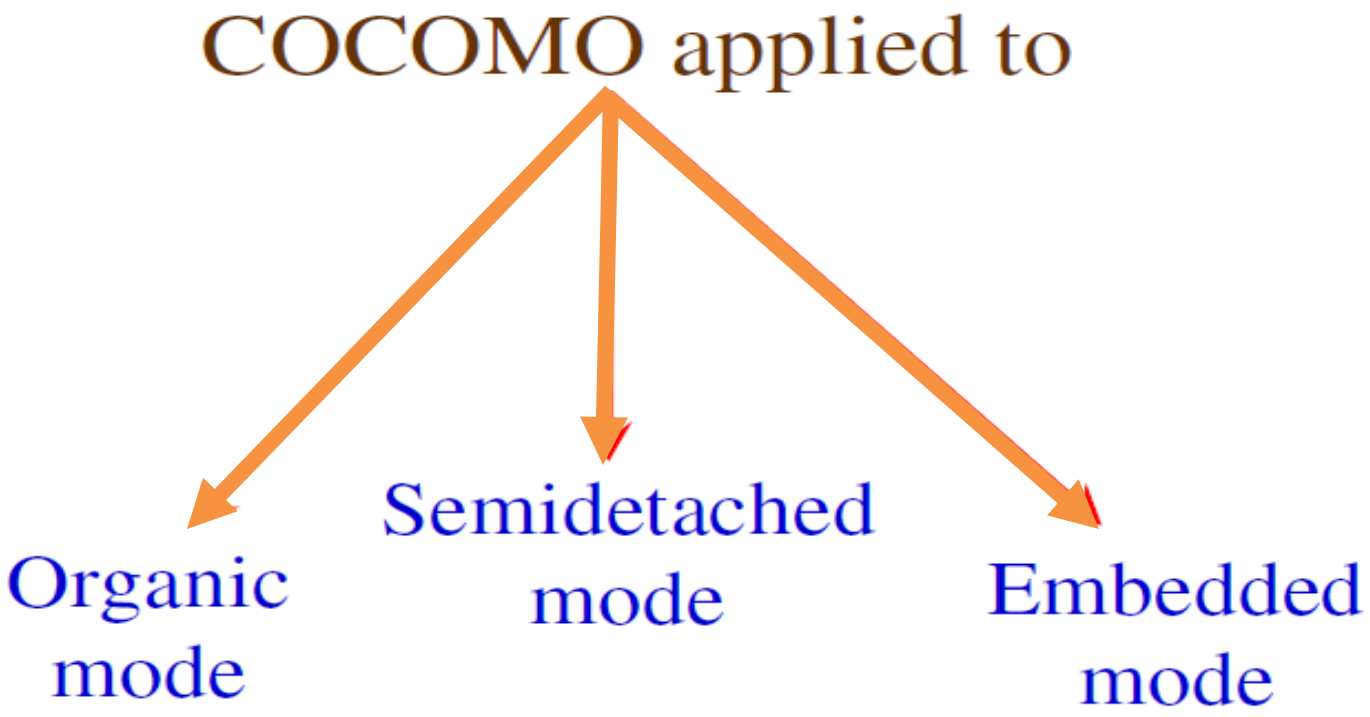
# Software Project Planning




Software Engineering Economics in 1981

Source: Software Engineering, By K.K Aggarwal & Yogesh Singh

# Software Project Planning




Source: Software Engineering , By K.K Aggarwal & Yogesh Singh

# Software Project Planning

| Mode | Project size | Nature of Project | Innovation | Deadline of the project | Development Environment |
|---|---|---|---|---|---|
| Organic | Typically 2-50 KLOC | Small size project, experienced developers in the familiar environment. For example, pay roll, inventory projects etc. | Little | Not tight | Familiar & In house |
| Semi detached | Typically 50-300 KLOC | Medium size project, Medium size team, Average previous experience on similar project. For example: Utility systems like compilers, database systems, editors etc. | Medium | Medium | Medium |
| Embedded | Typically over 300 KLOC | Large project, Real time systems, Complex interfaces, Very little previous experience. For example: ATMs, Air Traffic Control etc. | Significant | Tight | Complex Hardware/ customer Interfaces required |

The comparison of three COCOMO modes

Source: Software Engineering , By K.K Aggarwal & Yogesh Singh

# Software Project Planning

## Basic Model

Basic COCOMO model takes the form

$$E = a_b (KLOC)^{b_b}$$

$$D = c_b (E)^{d_b}$$

where E is effort applied in Person-Months, and D is the development time in months. The coefficients $a_b$, $b_b$, $c_b$ and $d_b$ are given in table

Source: Software Engineering, By K.K Aggarwal & Yogesh Singh

# Software Project Planning

| Software Project | $a_b$ | $b_b$ | $c_b$ | $d_b$ |
|---|---|---|---|---|
| Organic | 2.4 | 1.05 | 2.5 | 0.38 |
| Semidetached | 3.0 | 1.12 | 2.5 | 0.35 |
| Embedded | 3.6 | 1.20 | 2.5 | 0.32 |

Basic COCOMO coefficients

Source: Software Engineering, By K.K Aggarwal & Yogesh Singh

# Software Project Planning

When effort and development time are known, the average staff size to complete the project may be calculated as:

$$\text{Average staff size } (SS) = \frac{E}{D} Persons$$

When project size is known, the productivity level may be calculated as:

$$\text{Productivity } (P) = \frac{KLOC}{E} KLOC / PM$$

Source: Software Engineering, By K.K Aggarwal & Yogesh Singh

# Software Project Planning

Suppose that a project was estimated to be 400 KLOC. Calculate the effort and development time for each of the three modes i.e., organic, semidetached and embedded.

Source: Software Engineering, By K.K Aggarwal & Yogesh Singh

# Software Project Planning

## Solution

The basic COCOMO equation take the form:

$$E = a_b (KLOC)^{b_b}$$

$$D = c_b (E)^{d_b}$$

Estimated size of the project = 400 KLOC

(i) Organic mode

$$E = 2.4(400)^{1.05} = 1295.31 \text{ PM}$$

$$D = 2.5(1295.31)^{0.38} = 38.07 \text{ M}$$

| Software Project | $a_b$ | $b_b$ | $c_b$ | $d_b$ |
|---|---|---|---|---|
| Organic | 2.4 | 1.05 | 2.5 | 0.38 |
| Semidetached | 3.0 | 1.12 | 2.5 | 0.35 |
| Embedded | 3.6 | 1.20 | 2.5 | 0.32 |

Source: Software Engineering, By K.K Aggarwal & Yogesh Singh

# Software Project Planning

| Software Project | $a_b$ | $b_b$ | $c_b$ | $d_b$ |
|---|---|---|---|---|
| Organic | 2.4 | 1.05 | 2.5 | 0.38 |
| Semidetached | 3.0 | 1.12 | 2.5 | 0.35 |
| Embedded | 3.6 | 1.20 | 2.5 | 0.32 |

## (ii) Semidetached mode

$$E = 3.0(400)^{1.12} = 2462.79 \text{ PM}$$

$$D = 2.5(2462.79)^{0.35} = 38.45 \quad M$$

## (iii) Embedded mode

$$E = 3.6(400)^{1.20} = 4772.81 \text{ PM}$$

$$D = 2.5(4772.8)^{0.32} = 38 \quad M$$

| Software Project | $a_b$ | $b_b$ | $c_b$ | $d_b$ |
|---|---|---|---|---|
| Organic | 2.4 | 1.05 | 2.5 | 0.38 |
| Semidetached | 3.0 | 1.12 | 2.5 | 0.35 |
| Embedded | 3.6 | 1.20 | 2.5 | 0.32 |

Source: Software Engineering, By K.K Aggarwal & Yogesh Singh

# Software Project Planning

A project size of 200 KLOC is to be developed. Software development team has average experience on similar type of projects. The project schedule is not very tight. Calculate the effort, development time, average staff size and productivity of the project.

Source: Software Engineering, By K.K Aggarwal & Yogesh Singh

# Software Project Planning

## Solution

The semi-detached mode is the most appropriate mode; keeping in view the size, schedule and experience of the development team.

Hence $E = 3.0(200)^{1.12} = 1133.12\ PM$

$D = 2.5(1133.12)^{0.35} = 29.3\ 'M$

$$\text{Average staff size } (SS) = \frac{E}{D} Persons$$

$$= \frac{1133.12}{29.3} = 38.67\ Persons$$

| Software Project | $a_b$ | $b_b$ | $c_b$ | $d_b$ |
|---|---|---|---|---|
| Organic | 2.4 | 1.05 | 2.5 | 0.38 |
| Semidetached | 3.0 | 1.12 | 2.5 | 0.35 |
| Embedded | 3.6 | 1.20 | 2.5 | 0.32 |

**Table 4(a):** Basic COCOMO coefficients

Source: Software Engineering, By K.K Aggarwal & Yogesh Singh

# Software Project Planning

$$\text{Productivity} = \frac{KLOC}{E} = \frac{200}{1133.12} = 0.1765\ KLOC / PM$$

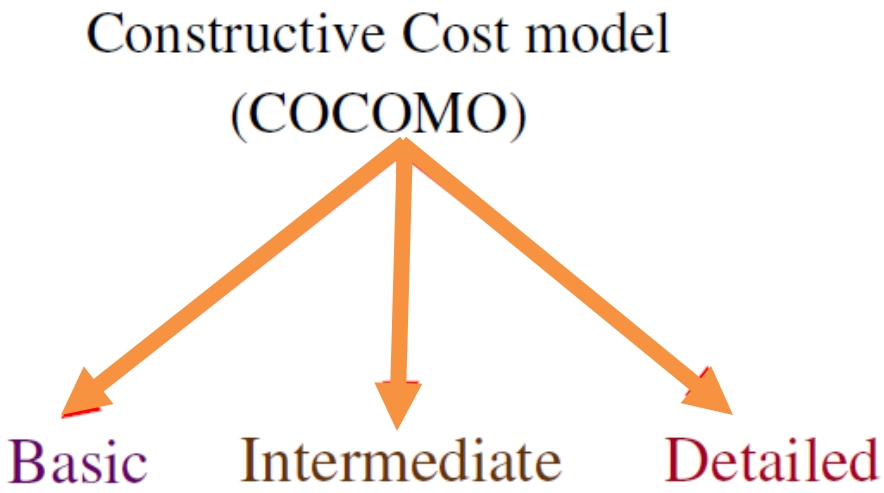
$$P = 176\ LOC / PM$$

Source: Software Engineering, By K.K Aggarwal & Yogesh Singh


# Software Engineering

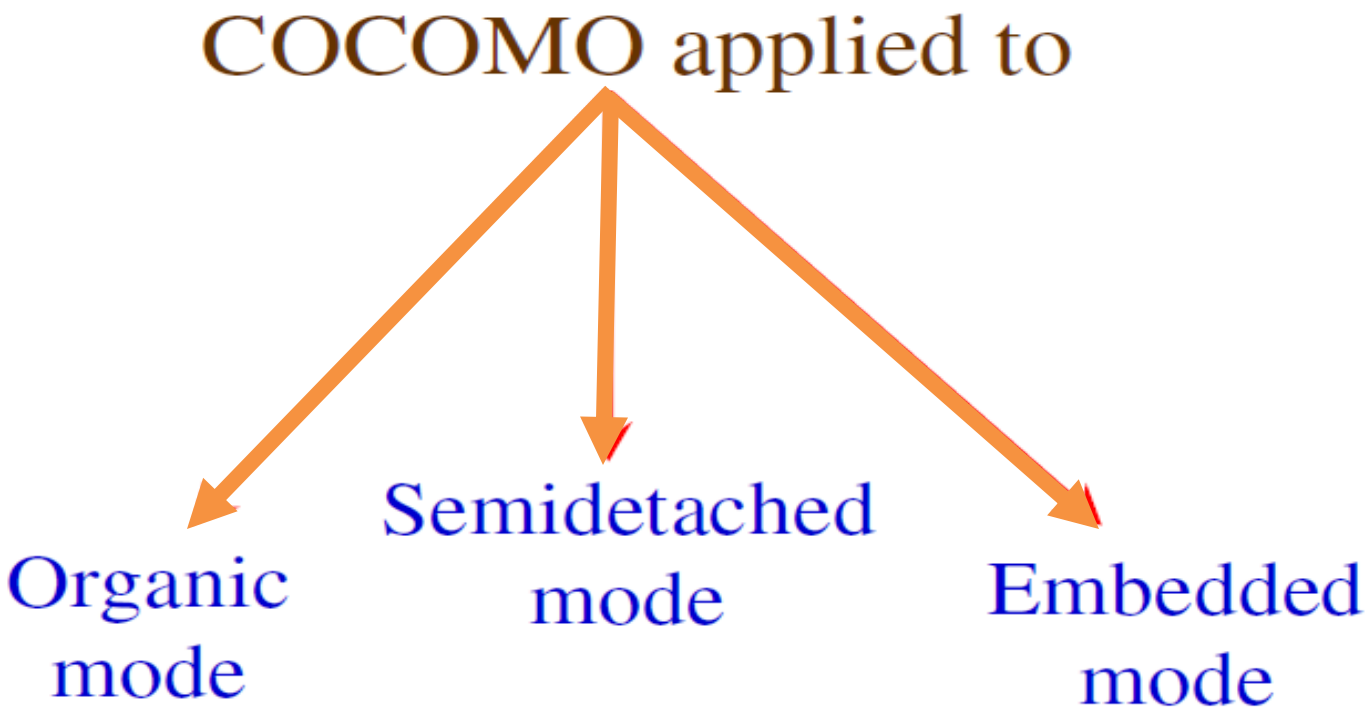
## Software Project Planning
## COCOMO MODEL

# Software Project Planning




Source: Software Engineering, By K.K Aggarwal & Yogesh Singh

# Software Project Planning




Source: Software Engineering , By K.K Aggarwal & Yogesh Singh

# Software Project Planning

## Intermediate Model

Cost drivers

(*i*) Product Attributes

- Required s/w reliability
- Size of application database
- Complexity of the product

(*ii*) Hardware Attributes

- Run time performance constraints
- Memory constraints
- Virtual machine volatility
- Turnaround time

Source: Software Engineering, By K.K Aggarwal & Yogesh Singh

# Software Project Planning

(*iii*) Personal Attributes

- Analyst capability
- Programmer capability
- Application experience
- Virtual m/c experience
- Programming language experience

(*iv*) Project Attributes

- Modern programming practices
- Use of software tools
- Required development Schedule

Source: Software Engineering, By K.K Aggarwal & Yogesh Singh

# Software Project Planning

Multipliers of different cost drivers

| Cost Drivers | RATINGS | | | | | |
|---|---|---|---|---|---|---|
| | Very low | Low | Nominal | High | Very high | Extra high |
| **Product Attributes** | | | | | | |
| RELY | 0.75 | 0.88 | 1.00 | 1.15 | 1.40 | -- |
| DATA | -- | 0.94 | 1.00 | 1.08 | 1.16 | -- |
| CPLX | 0.70 | 0.85 | 1.00 | 1.15 | 1.30 | 1.65 |
| **Computer Attributes** | | | | | | |
| TIME | -- | -- | 1.00 | 1.11 | 1.30 | 1.66 |
| STOR | -- | -- | 1.00 | 1.06 | 1.21 | 1.56 |
| VIRT | -- | 0.87 | 1.00 | 1.15 | 1.30 | -- |
| TURN | -- | 0.87 | 1.00 | 1.07 | 1.15 | -- |

Source: Software Engineering, By K.K Aggarwal & Yogesh Singh

# Software Project Planning

| Cost Drivers | RATINGS | | | | | |
|---|---|---|---|---|---|---|
| | Very low | Low | Nominal | High | Very high | Extra high |
| **Personnel Attributes** | | | | | | |
| ACAP | 1.46 | 1.19 | 1.00 | 0.86 | 0.71 | -- |
| AEXP | 1.29 | 1.13 | 1.00 | 0.91 | 0.82 | -- |
| PCAP | 1.42 | 1.17 | 1.00 | 0.86 | 0.70 | -- |
| VEXP | 1.21 | 1.10 | 1.00 | 0.90 | -- | -- |
| LEXP | 1.14 | 1.07 | 1.00 | 0.95 | -- | -- |
| **Project Attributes** | | | | | | |
| MODP | 1.24 | 1.10 | 1.00 | 0.91 | 0.82 | -- |
| TOOL | 1.24 | 1.10 | 1.00 | 0.91 | 0.83 | -- |
| SCED | 1.23 | 1.08 | 1.00 | 1.04 | 1.10 | -- |

Multiplier values for effort calculations

Source: Software Engineering, By K.K Aggarwal & Yogesh Singh

# Software Project Planning

## Intermediate COCOMO equations

$$E = a_i (KLOC)^{b_i} * EAF$$ **(Effort Adjustment Factor)**

$$D = c_i (E)^{d_i}$$

| Project | $a_i$ | $b_i$ | $c_i$ | $d_i$ |
|---|---|---|---|---|
| Organic | 3.2 | 1.05 | 2.5 | 0.38 |
| Semidetached | 3.0 | 1.12 | 2.5 | 0.35 |
| Embedded | 2.8 | 1.20 | 2.5 | 0.32 |

Coefficients for intermediate COCOMO

Source: Software Engineering, By K.K Aggarwal & Yogesh Singh

# Software Project Planning

A new project with estimated 400 KLOC embedded system has to be developed. Project manager has a choice of hiring from two pools of developers: Very highly capable with very little experience in the programming language being used

**OR**

Developers of low quality but a lot of experience with the programming language. What is the impact of hiring all developers from one or the other pool ?

| Cost Drivers | RATINGS | | | | | |
|---|---|---|---|---|---|---|
| | Very low | Low | Nominal | High | Very high | Extra high |
| **Personnel Attributes** | | | | | | |
| ACAP | 1.46 | 1.19 | 1.00 | 0.86 | 0.71 | -- |
| AEXP | 1.29 | 1.13 | 1.00 | 0.91 | 0.82 | -- |
| PCAP | 1.42 | 1.17 | 1.00 | 0.86 | 0.70 | -- |
| VEXP | 1.21 | 1.10 | 1.00 | 0.90 | -- | -- |
| LEXP | 1.14 | 1.07 | 1.00 | 0.95 | -- | -- |
| **Project Attributes** | | | | | | |
| MODP | 1.24 | 1.10 | 1.00 | 0.91 | 0.82 | -- |
| TOOL | 1.24 | 1.10 | 1.00 | 0.91 | 0.83 | -- |
| SCED | 1.23 | 1.08 | 1.00 | 1.04 | 1.10 | -- |

Multiplier values for effort calculations

Source: Software Engineering, By K.K Aggarwal & Yogesh Singh

# Software Project Planning

## Solution

This is the case of embedded mode and model is intermediate COCOMO.

Hence $$E = a_i (KLOC)^{d_i}$$

$$= 2.8\ (400)^{1.20} = 3712\ PM$$

| Project | $a_i$ | $b_i$ | $c_i$ | $d_i$ |
|---|---|---|---|---|
| Organic | 3.2 | 1.05 | 2.5 | 0.38 |
| Semidetached | 3.0 | 1.12 | 2.5 | 0.35 |
| Embedded | 2.8 | 1.20 | 2.5 | 0.32 |

**Case I:** Developers are very highly capable with very little experience in the programming being used.

EAF = 0.82 x 1.14 = 0.9348

E = 3712 x .9348 = 3470 PM

D = $2.5\ (3470)^{0.32}$ = 33.9 M

| Cost Drivers | RATINGS | | | | | |
|---|---|---|---|---|---|---|
| | Very low | Low | Nominal | High | Very high | Extra high |
| Personnel Attributes | | | | | | |
| ACAP | 1.46 | 1.19 | 1.00 | 0.86 | 0.71 | -- |
| AEXP | 1.29 | 1.13 | 1.00 | 0.91 | 0.82 | -- |
| PCAP | 1.42 | 1.17 | 1.00 | 0.86 | 0.70 | -- |
| VEXP | 1.21 | 1.10 | 1.00 | 0.90 | -- | -- |
| LEXP | 1.14 | 1.07 | 1.00 | 0.95 | -- | -- |

Source: Software Engineering, By K.K Aggarwal & Yogesh Singh

# Software Project Planning

**Case II:** Developers are of low quality but lot of experience with the programming language being used.

EAF = 1.29 x 0.95 = 1.22

E = 3712 x 1.22 = 4528 PM

D = $2.5\ (4528)^{0.32}$ = 36.9 M

| Cost Drivers | RATINGS | | | | | |
|---|---|---|---|---|---|---|
| | Very low | Low | Nominal | High | Very high | Extra high |
| Personnel Attributes | | | | | | |
| ACAP | 1.46 | 1.19 | 1.00 | 0.86 | 0.71 | -- |
| AEXP | 1.29 | 1.13 | 1.00 | 0.91 | 0.82 | -- |
| PCAP | 1.42 | 1.17 | 1.00 | 0.86 | 0.70 | -- |
| VEXP | 1.21 | 1.10 | 1.00 | 0.90 | -- | -- |
| LEXP | 1.14 | 1.07 | 1.00 | 0.95 | -- | -- |

Case II requires more effort and time. Hence, low quality developers with lot of programming language experience could not match with the performance of very highly capable developers with very litter experience.

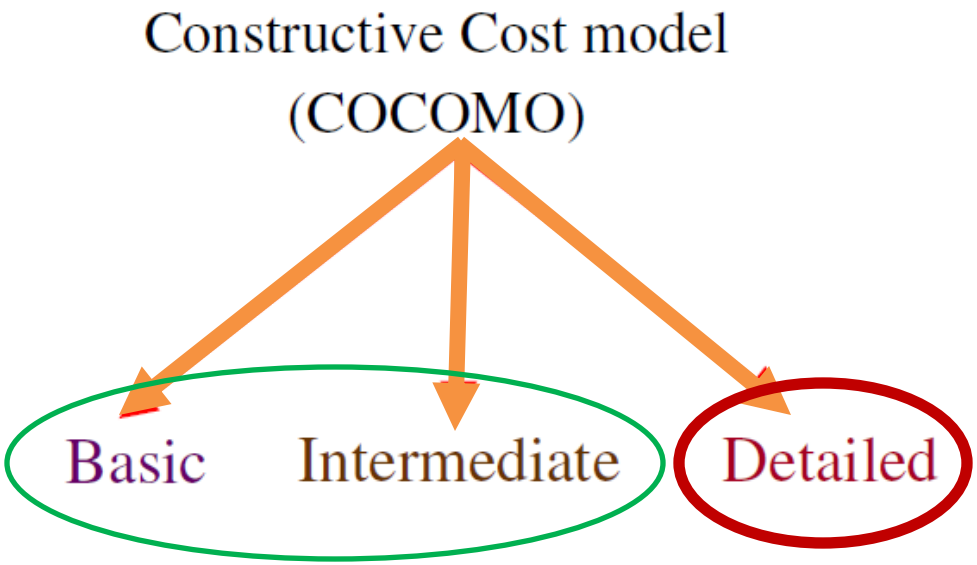
| Project | $a_i$ | $b_i$ | $c_i$ | $d_i$ |
|---|---|---|---|---|
| Organic | 3.2 | 1.05 | 2.5 | 0.38 |
| Semidetached | 3.0 | 1.12 | 2.5 | 0.35 |
| Embedded | 2.8 | 1.20 | 2.5 | 0.32 |

Source: Software Engineering, By K.K Aggarwal & Yogesh Singh


# Software Engineering

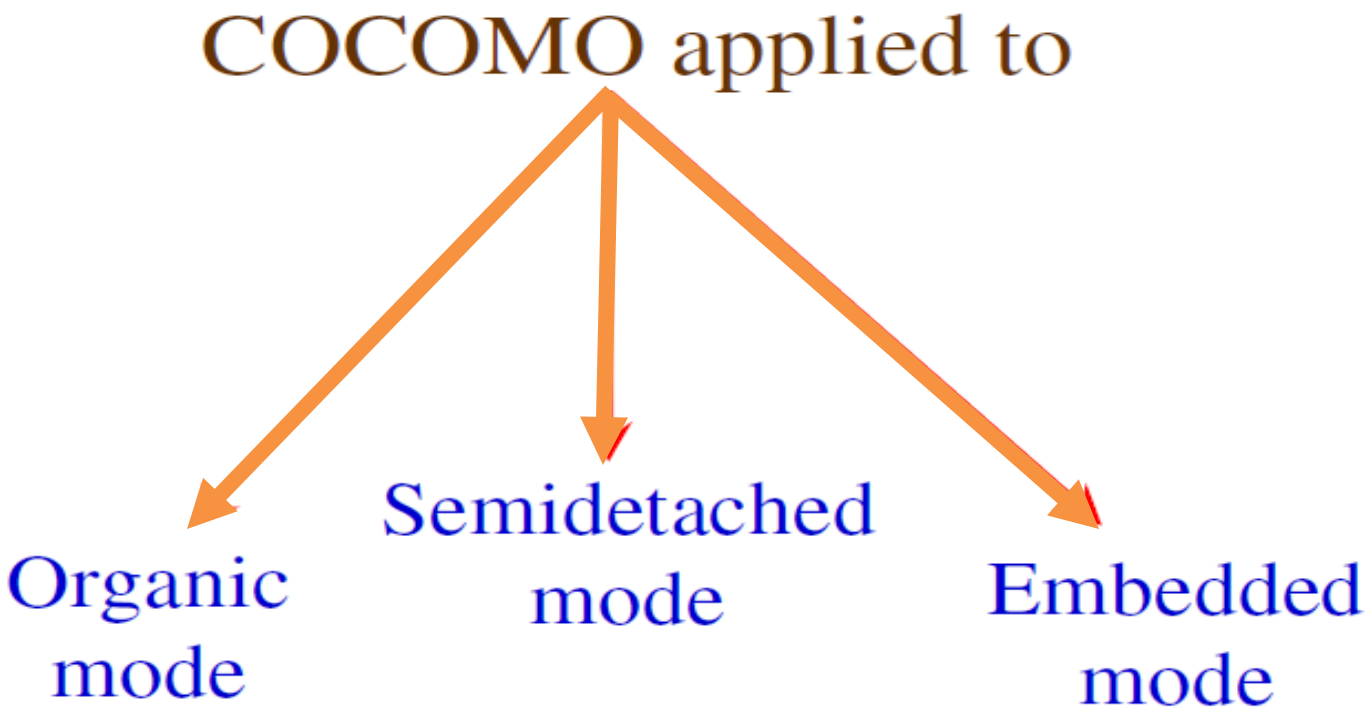
## Software Project Planning
## COCOMO MODEL

# Software Project Planning




Source: Software Engineering, By K.K Aggarwal & Yogesh Singh

# Software Project Planning




Source: Software Engineering , By K.K Aggarwal & Yogesh Singh

# Software Project Planning

## Development Phase

Plan / Requirements

| | | |
|---|---|---|
| EFFORT | : | 6% to 8% |
| DEVELOPMENT TIME | : | 10% to 40% |

% depend on mode & size

Source: Software Engineering, By K.K Aggarwal & Yogesh Singh

# Software Project Planning

## Design

Effort : 16% to 18%

Time : 19% to 38%

## Programming

Effort : 48% to 68%

Time : 24% to 64%

## Integration & Test

Effort : 16% to 34%

Time : 18% to 34%

Source: Software Engineering, By K.K Aggarwal & Yogesh Singh

# Software Project Planning

## Detailed Model Cost Estimation

$$E_p = \mu_p E \rightarrow E = a_i (KLOC)^{bi} \times EAF$$

$$D_p = \tau_p D \rightarrow D = C_i (E)^{di}$$

Lifecycle Phase Values of $\mu_p$

| Mode & Code Size | Plan & Requirements | System Design | Detailed Design | Module Code & Test | Integration & Test |
|---|---|---|---|---|---|
| Organic Small S≈2 | 0.06 | 0.16 | 0.26 | 0.42 | 0.16 |
| Organic medium S≈32 | 0.06 | 0.16 | 0.24 | 0.38 | 0.22 |
| Semidetached medium S≈32 | 0.07 | 0.17 | 0.25 | 0.33 | 0.25 |
| Semidetached large S≈128 | 0.07 | 0.17 | 0.24 | 0.31 | 0.28 |
| Embedded large S≈128 | 0.08 | 0.18 | 0.25 | 0.26 | 0.31 |
| Embedded extra large S≈320 | 0.08 | 0.18 | 0.24 | 0.24 | 0.34 |

Effort and schedule fractions occurring in each phase of the lifecycle

Lifecycle Phase Values of $\tau_p$

| Mode & Code Size | Plan & Requirements | System Design | Detailed Design | Module Code & Test | Integration & Test |
|---|---|---|---|---|---|
| Organic Small S≈2 | 0.10 | 0.19 | 0.24 | 0.39 | 0.18 |
| Organic medium S≈32 | 0.12 | 0.19 | 0.21 | 0.34 | 0.26 |
| Semidetached medium S≈32 | 0.20 | 0.26 | 0.21 | 0.27 | 0.26 |
| Semidetached large S≈128 | 0.22 | 0.27 | 0.19 | 0.25 | 0.29 |
| Embedded large S≈128 | 0.36 | 0.36 | 0.18 | 0.18 | 0.28 |
| Embedded extra large S≈320 | 0.40 | 0.38 | 0.16 | 0.16 | 0.30 |

Effort and schedule fractions occurring in each phase of the lifecycle

Source: Software Engineering, By K.K Aggarwal & Yogesh Singh

# Software Project Planning

## Distribution of software life cycle:

1. Requirement and product design
   (a) Plans and requirements
   (b) System design
2. Detailed Design
   (a) Detailed design
3. Code & Unit test
   (a) Module code & test
4. Integrate and Test
   (a) Integrate & Test

Source: Software Engineering, By K.K Aggarwal & Yogesh Singh

# Software Project Planning

Consider a project to develop a full screen editor. The major components identified are:

I. Screen edit

II. Command Language Interpreter

III. File Input & Output

IV. Cursor Movement

V. Screen Movement

The size of these are estimated to be 4k, 2k, 1k, 2k and 3k delivered source code lines. Use COCOMO to determine

1. Overall cost and schedule estimates (assume values for different cost drivers, with at least three of them being different from 1.0)
2. Cost & Schedule estimates for different phases.

Source: Software Engineering, By K.K Aggarwal & Yogesh Singh

# Software Project Planning



## Solution

Size of five modules are:

| | |
|---|---|
| Screen edit | = 4 KLOC |
| Command language interpreter | = 2 KLOC |
| File input and output | = 1 KLOC |
| Cursor movement | = 2 KLOC |
| Screen movement | = 3 KLOC |
| **Total** | **= 12 KLOC** |

Source: Software Engineering, By K.K Aggarwal & Yogesh Singh

# Software Project Planning

Let us assume that significant cost drivers are

i. Required software reliability is high, i.e.,1.15

ii. Product complexity is high, i.e.,1.15

iii. Analyst capability is high, i.e.,0.86

iv. Programming language experience is low,i.e.,1.07

v. All other drivers are nominal

EAF = 1.15x1.15x0.86x1.07 = 1.2169

Multipliers of different cost drivers

| Cost Drivers | RATINGS | | | | | |
|---|---|---|---|---|---|---|
| | Very low | Low | Nominal | High | Very high | Extra high |
| **Product Attributes** | | | | | | |
| RELY | 0.75 | 0.88 | 1.00 | 1.15 | 1.40 | -- |
| DATA | -- | 0.94 | 1.00 | 1.08 | 1.16 | -- |
| CPLX | 0.70 | 0.85 | 1.00 | 1.15 | 1.30 | 1.65 |

| Cost Drivers | RATINGS | | | | | |
|---|---|---|---|---|---|---|
| | Very low | Low | Nominal | High | Very high | Extra high |
| **Personnel Attributes** | | | | | | |
| ACAP | 1.46 | 1.19 | 1.00 | 0.86 | 0.71 | -- |
| AEXP | 1.29 | 1.13 | 1.00 | 0.91 | 0.82 | -- |
| PCAP | 1.42 | 1.17 | 1.00 | 0.86 | 0.70 | -- |
| VEXP | 1.21 | 1.10 | 1.00 | 0.90 | -- | -- |
| LEXP | 1.14 | 1.07 | 1.00 | 0.95 | -- | -- |

Source: Software Engineering, By K.K Aggarwal & Yogesh Singh

# Software Project Planning

(a) The initial effort estimate for the project is obtained from the following equation

$$E = a_i (KLOC)^{bi} \times EAF$$

$$= 3.2(12)^{1.05} \times 1.2169 = 52.91 \text{ PM}$$

$$D = C_i(E)^{di}$$

$$= 2.5(52.91)^{0.38} = 11.29 \text{ M}$$

(b) Using the following equations and referring Table phase wise cost and schedule estimates can be calculated.

$$E_p = \mu_p E$$

$$D_p = \tau_p D$$

**Table** $\mu_p$

| Mode & Code Size | Plan & Requirements | System Design | Detailed Design | Module Code & Test | Integration & Test |
|---|---|---|---|---|---|
| Organic Small S≈2 | 0.06 | 0.16 | 0.26 | 0.42 | 0.16 |
| Organic medium S≈32 | 0.06 | 0.16 | 0.24 | 0.38 | 0.22 |

**Table** $\tau_p$

| Mode & Code Size | Plan & Requirements | System Design | Detailed Design | Module Code & Test | Integration & Test |
|---|---|---|---|---|---|
| Organic Small S≈2 | 0.10 | 0.19 | 0.24 | 0.39 | 0.18 |
| Organic medium S≈32 | 0.12 | 0.19 | 0.21 | 0.34 | 0.26 |

Source: Software Engineering, By K.K Aggarwal & Yogesh Singh

# Software Project Planning

Since size is only 12 KLOC, it is an organic small model. Phase wise effort distribution is given below:

| | |
|---|---|
| System Design | = 0.16 x 52.91 = 8.465 PM |
| Detailed Design | = 0.26 x 52.91 = 13.756 PM |
| Module Code & Test | = 0.42 x 52.91 = 22.222 PM |
| Integration & Test | = 0.16 x 52.91 = 8.465 Pm |

Now Phase wise development time duration is

| | |
|---|---|
| System Design | = 0.19 x 11.29 = 2.145 M |
| Detailed Design | = 0.24 x 11.29 = 2.709 M |
| Module Code & Test | = 0.39 x 11.29 = 4.403 M |
| Integration & Test | = 0.18 x 11.29 = 2.032 M |

Source: Software Engineering, By K.K Aggarwal & Yogesh Singh