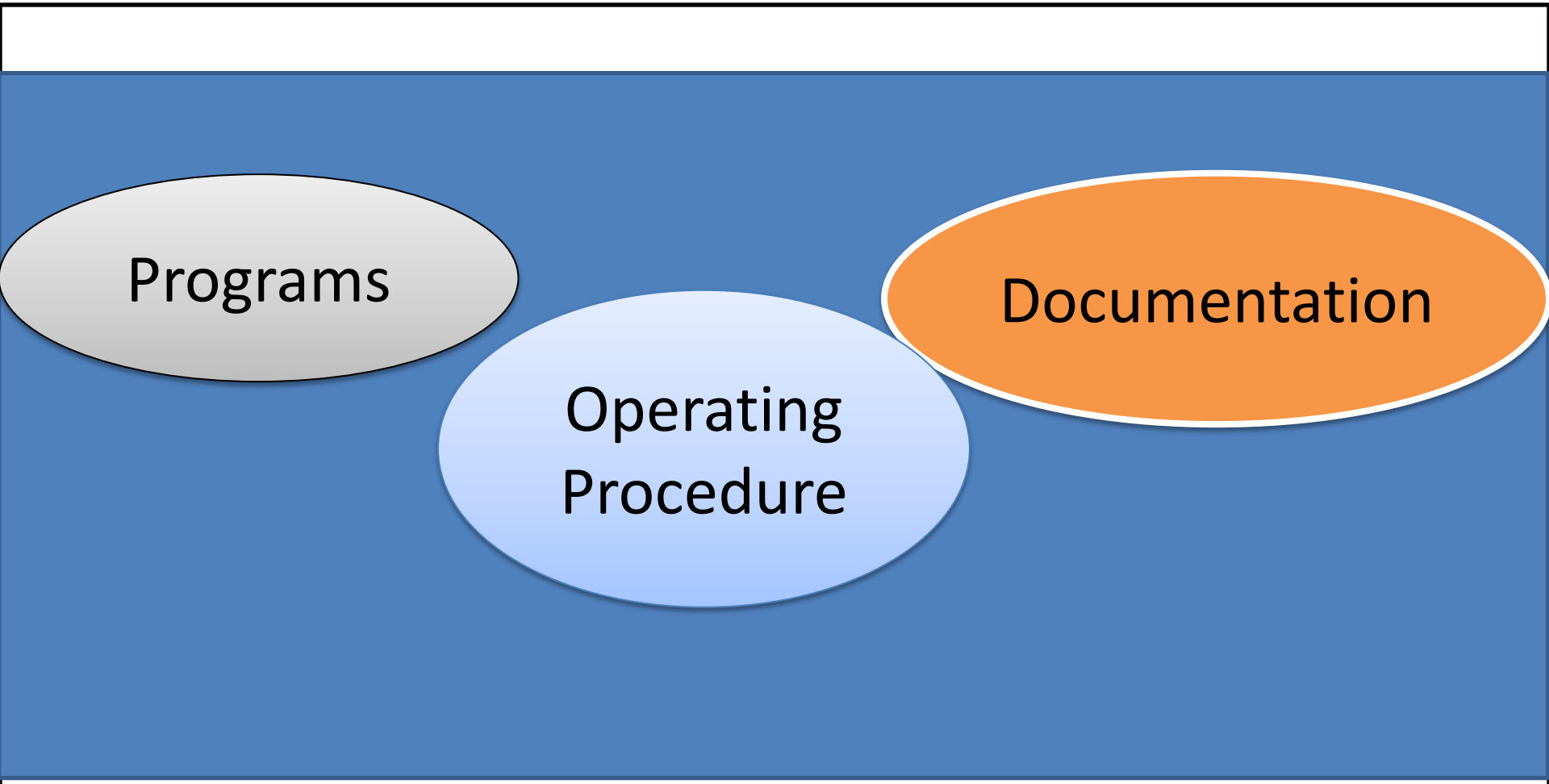
Subject Name: Software Engineering

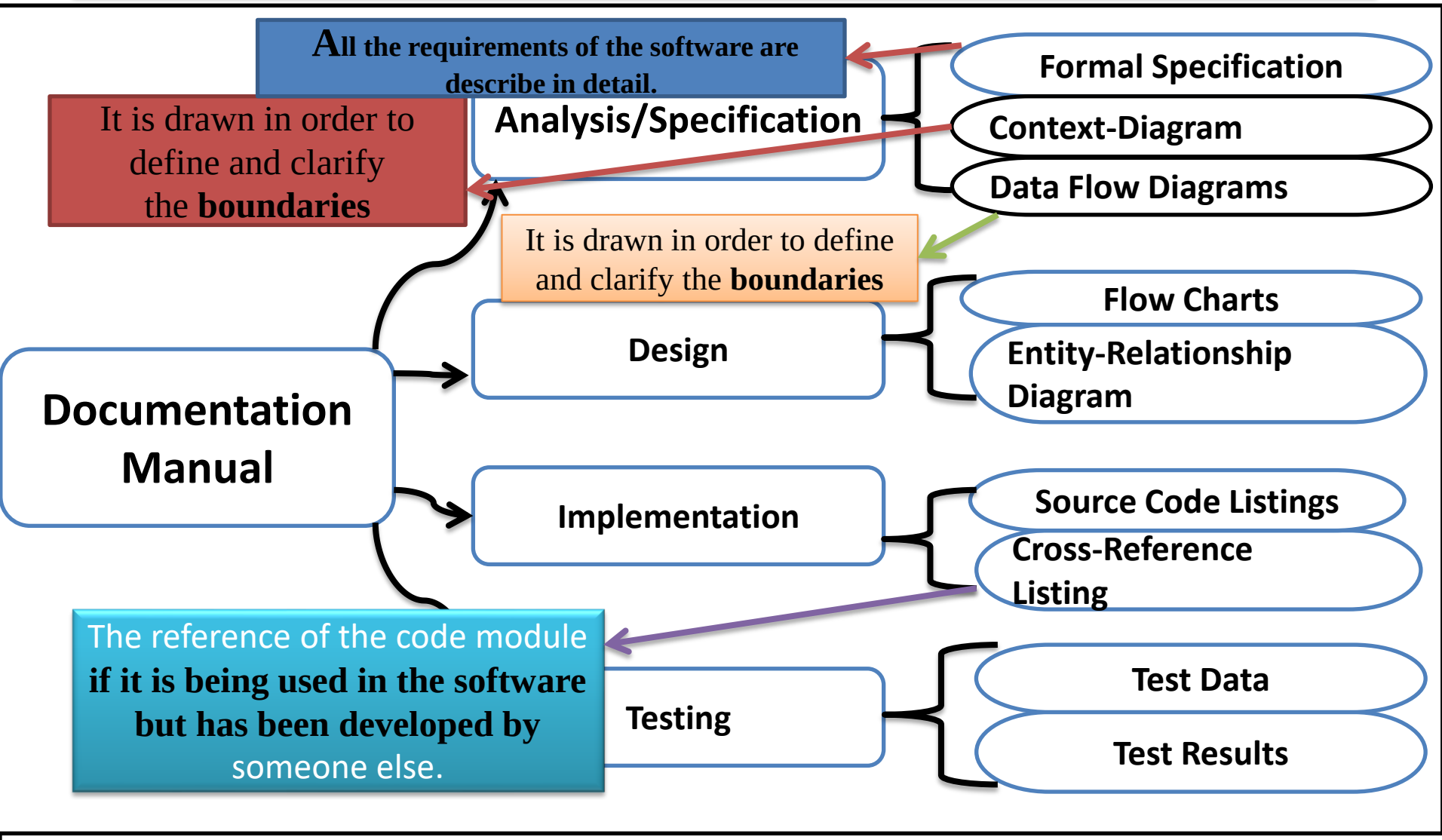
Subject Code: (BCSC-0009)

Topic Name: Software Engineering

Software



Manuals in Documentation



Software Engineering

- The term **software engineering** is the product of two words, **software**, and **engineering**.
- The **software** is a collection of integrated programs.
- **Engineering** is the application of scientific and practical knowledge to create, design and maintain processes.
- It is a process of analyzing the requirements of user and then designing, building, and testing software application which will satisfy these requirements.

IEEE Definition of Software Engineering

Software engineering as the application of a systematic, disciplined, which is a computable approach for the development, operation, and maintenance of software.

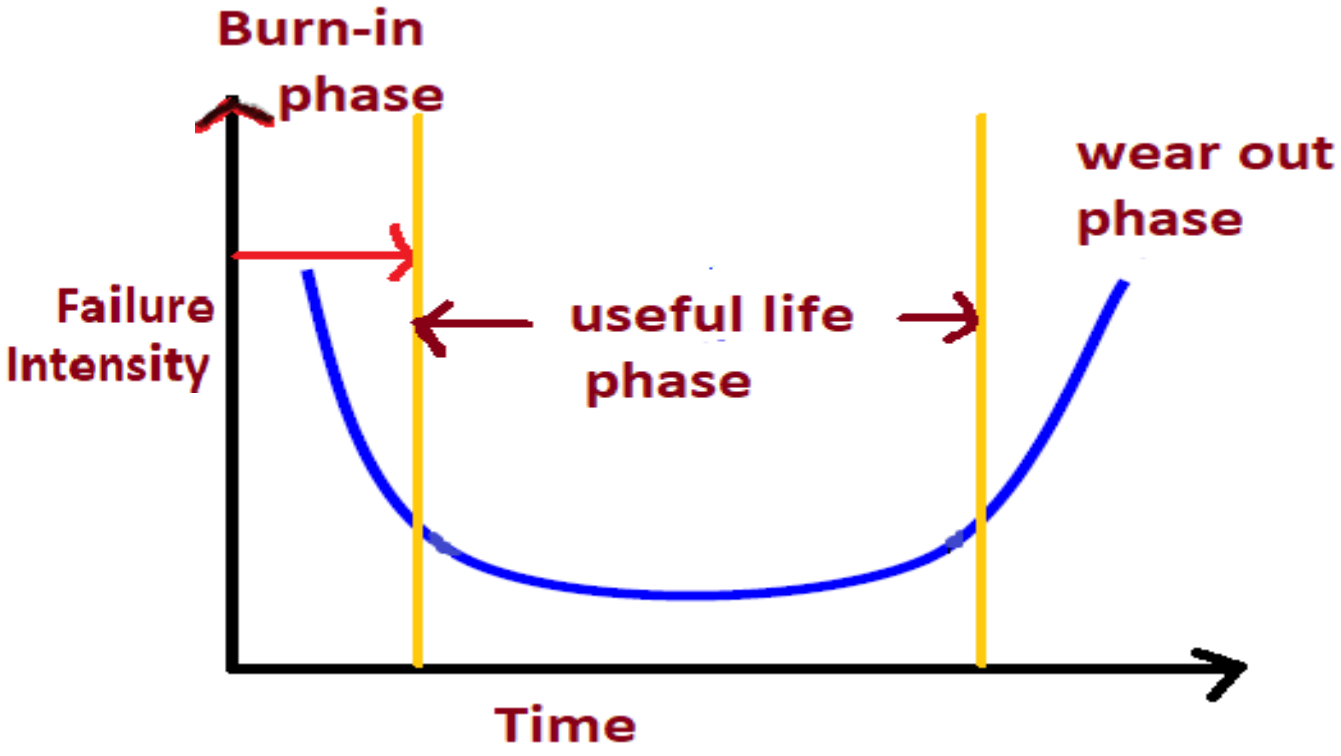
Boehm's Definition of Software Engineering

The practical application of scientific knowledge to the creative design and building of computer programs.

It also includes associated documentation needed for developing, operating, and maintaining them.

Characteristics of Software

Software Does not wear out



Characteristics of Software

Portability

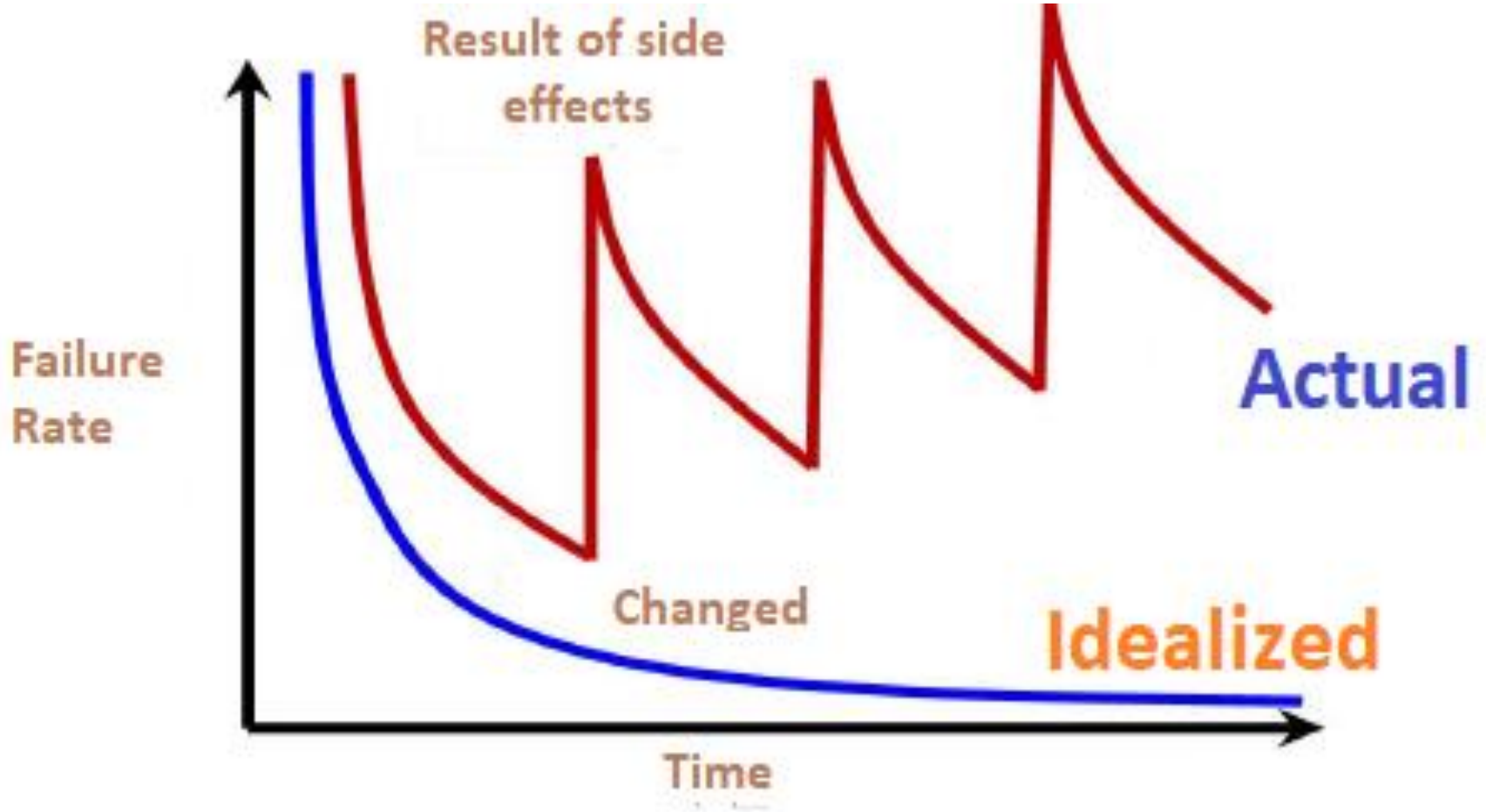
A set of attributes that bears on the ability of the software to be transferred from one environment to another, without or minimum changes.

Software Failure

If there is any bug or any error in a program or a system then we will get incorrect or unpredicted result.

Due to error the behave of system will be inappropriate.

Failure Curve of Software



Open Source Software

Software in which source code is available is known as Open Source Software.

Ex: PHP, Linux, MySQL etc.

Need of Software Engineering

Reduce Complexities

Software engineering divides big problems into various small issues and then start solving each small problem one by one.

Need of Software Engineering(cont..)

DecreaseTime

Anything that is not made according to the project always wastes time.

And if you are making software/application, then you may need to run many codes to get final running code.

if you are making your software according to the software engineering method, then it will decrease a lot of time.

Characteristics of Software

Functionality

It refers to the degree of performance of the software against its intended purpose. It basically means are the required functions.

Characteristics of Software

Reliability

A set of attribute that Bear on the capability of software to maintain its level of performances understated conditions for a stated period of time.

Characteristics of Software

Efficiency

It refers to the ability of the software to use System Resources in the most Effective and Efficient Manner. The software should make effective use of storage space and executive commands as per desired timing requirement.

Characteristics of Software

Usability

It refers to the extent to which the software can be used with ease. Or the amount of effort or time required to learn how to use the software should be less.

Characteristics of Software

Maintainability

Refers to the ease with which the modifications can be made in a software system to extend its functionality, improvement, performance or correct errors.

Characteristics of Software

Portability

A set of attributes that bears on the ability of the software to be transferred from one environment to another, without or minimum changes.

Software Failure

If there is any bug or any error in a program or a system then we will get incorrect or unpredicted result.

Due to error the behave of system will be inappropriate.

Program

Program can be defined as the set of instruction, that perform specific task.

Actually, program is developed by an individual or a group of programmers for their own use.

Further classification of a program is not possible.

There is no need to use SDLC(Software Development life cycle in program).

There is no need of well defined/dedicated user interface in program.

Software

Software is a collection of programs.

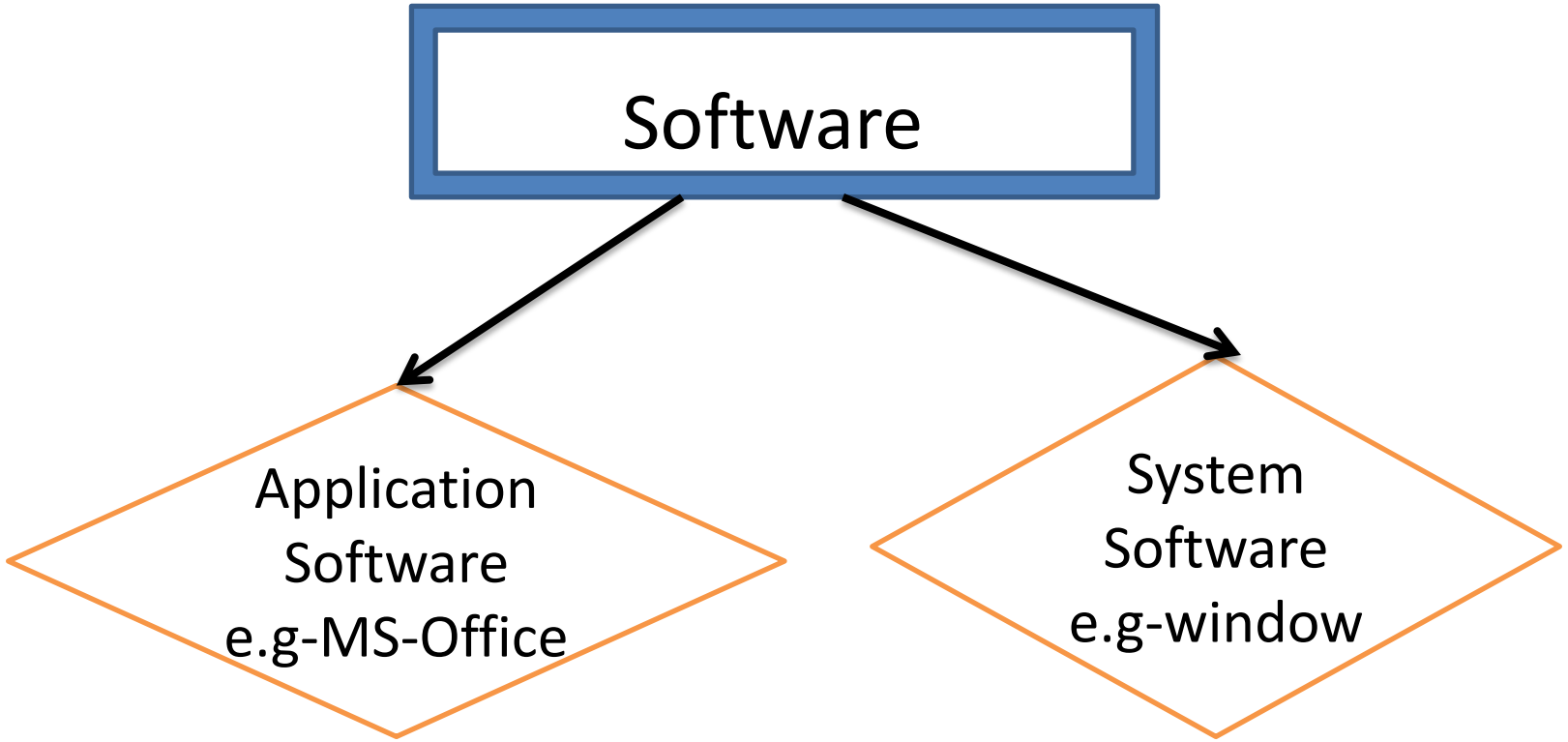
There may be bundles of programs and data files in software.

There is need to use SDLC(Software Development life cycle) in software.

There is a well defined/dedicated user interface in software.

Software(cont..)

Further software is classified into two categories



SDLC

- SDLC stands for **Software Development Life Cycle** and is also referred to as the Application Development life-cycle
- It offers a basis for project planning, scheduling, and estimating
- Provides a framework for a standard set of activities and deliverables
- It is a mechanism for project tracking and control
- Increases visibility of project planning to all involved stakeholders of the development process
- Increased and enhance development speed
- Improved client relations
- Helps you to decrease project risk and project management plan overhead

SDLC Phases

Requirement Analysis

Feasibility Study

Design

Coding

Testing

Install Deploy

Maintenance

Phase 1: Requirement collection and analysis

- The requirement is the first stage in the SDLC process. It is conducted by the senior team members with inputs from all the stakeholders and domain experts in the industry.
- This stage gives a clear picture of the scope of the entire project and the anticipated issues, opportunities, and directives which triggered the project.

- Requirements Gathering stage need teams to get detailed and precise requirements. This helps companies to finalize the necessary timeline to finish the work of that system.

Phase 2: Feasibility study

- **There are mainly five types of feasibilities checks:**
- **Economic:** Can we complete the project within the budget or not?
- **Legal:** Can we handle this project as cyber law and other regulatory framework/compliances.
- **Operation feasibility:** Can we create operations which is expected by the client?
- **Technical:** Need to check whether the current computer system can support the software
- **Schedule:** Decide that the project can be completed within the given schedule or not.

Phase 3: Design

- In this third phase, the system and software design documents are prepared as per the requirement specification document. This helps define overall system architecture.
- This design phase serves as input for the next phase of the model.

There are two kinds of design documents developed in this phase:

High-Level Design (HLD):

- Brief description and name of each module
- An outline about the functionality of every module
- Interface relationship and dependencies between modules
- Database tables identified along with their key elements
- Complete architecture diagrams along with technology details

Low-Level Design(LLD)

- Functional logic of the modules
- Database tables, which include type and size
- Complete detail of the interface
- Addresses all types of dependency issues
- Listing of error messages
- Complete input and outputs for every module

Phase 4: Coding

Once the system design phase is over, the next phase is coding. In this phase, developers start build the entire system by writing code using the chosen programming language. In the coding phase, tasks are divided into units or modules and assigned to the various developers. It is the longest phase of the Software Development Life Cycle process.

In this phase, Developer needs to follow certain predefined coding guidelines. They also need to use programming tools like compiler, interpreters, debugger to generate and implement the code.

Phase 5: Testing

During this phase, QA and testing team may find some bugs/defects which they communicate to developers. The development team fixes the bug and send back to QA for a re-test. This process continues until the software is bug-free, stable, and working according to the business needs of that system.

Phase 6: Installation/Deployment

Once the software testing phase is over and no bugs or errors left in the system then the final deployment process starts. Based on the feedback given by the project manager, the final software is released and checked for deployment issues if any.

Phase 7: Maintenance

- Once the system is deployed, and customers start using the developed system, following 3 activities occur
- Bug fixing - bugs are reported because of some scenarios which are not tested at all
- Upgrade - Upgrading the application to the newer versions of the Software
- Enhancement - Adding some new features into the existing software

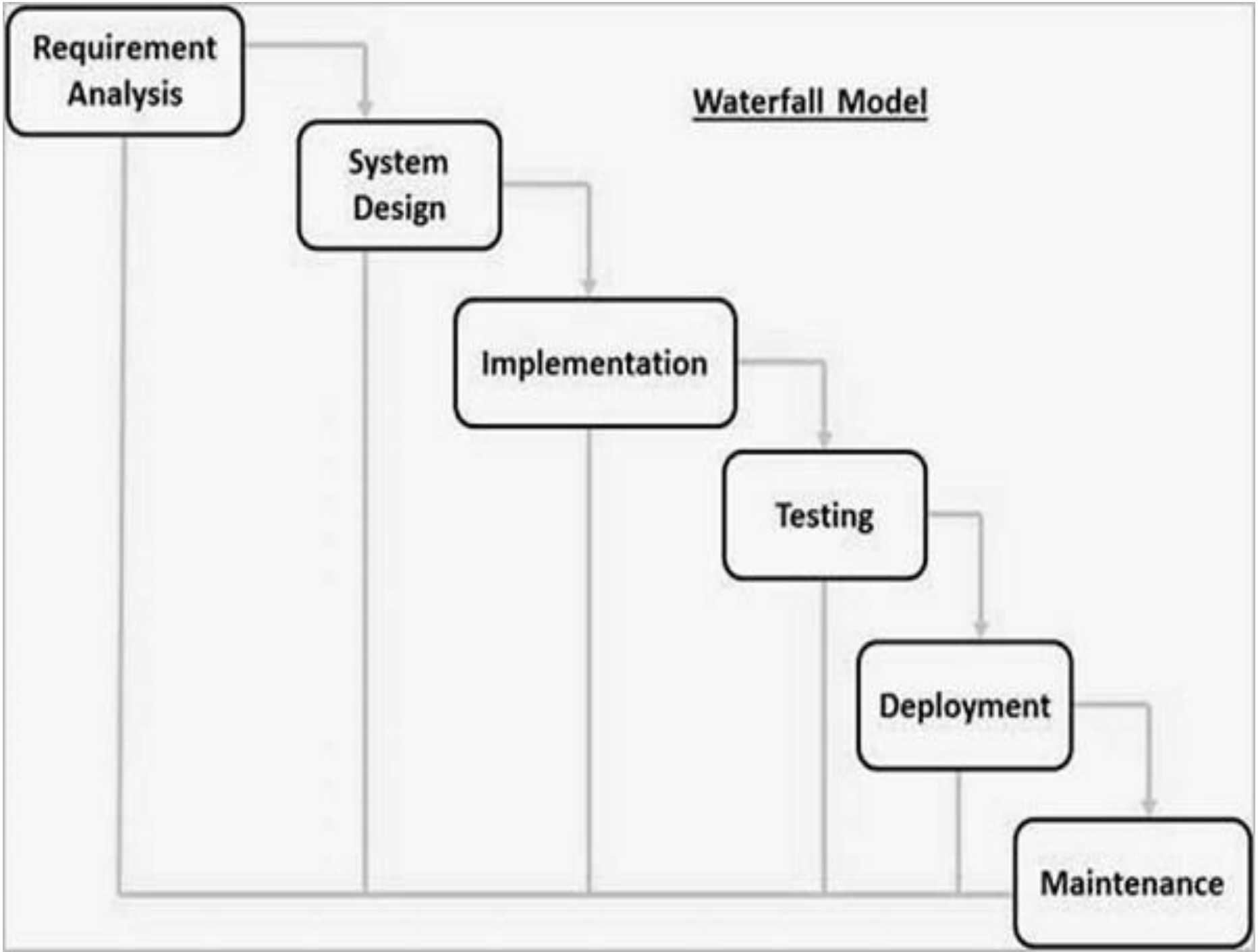
Waterfall Model

The Waterfall Model was the first Process Model to be introduced. It is also referred to as a **linear-sequential life cycle model**.

Waterfall approach was first SDLC Model to be used widely in Software Engineering to ensure success of the project.

In "The Waterfall" approach, the whole process of software development is divided into separate phases.

In this Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.


Requirement Analysis
Waterfall Model
System Design
Implementation
Testing
Deployment
Maintenance

Applications of Waterfall Model

- Requirements are very well documented, clear and fixed.
- Product definition is stable.
- Technology is understood and is not dynamic.
- There are no ambiguous requirements.
- Resources with required expertise are available to support the product.
- The project is short.

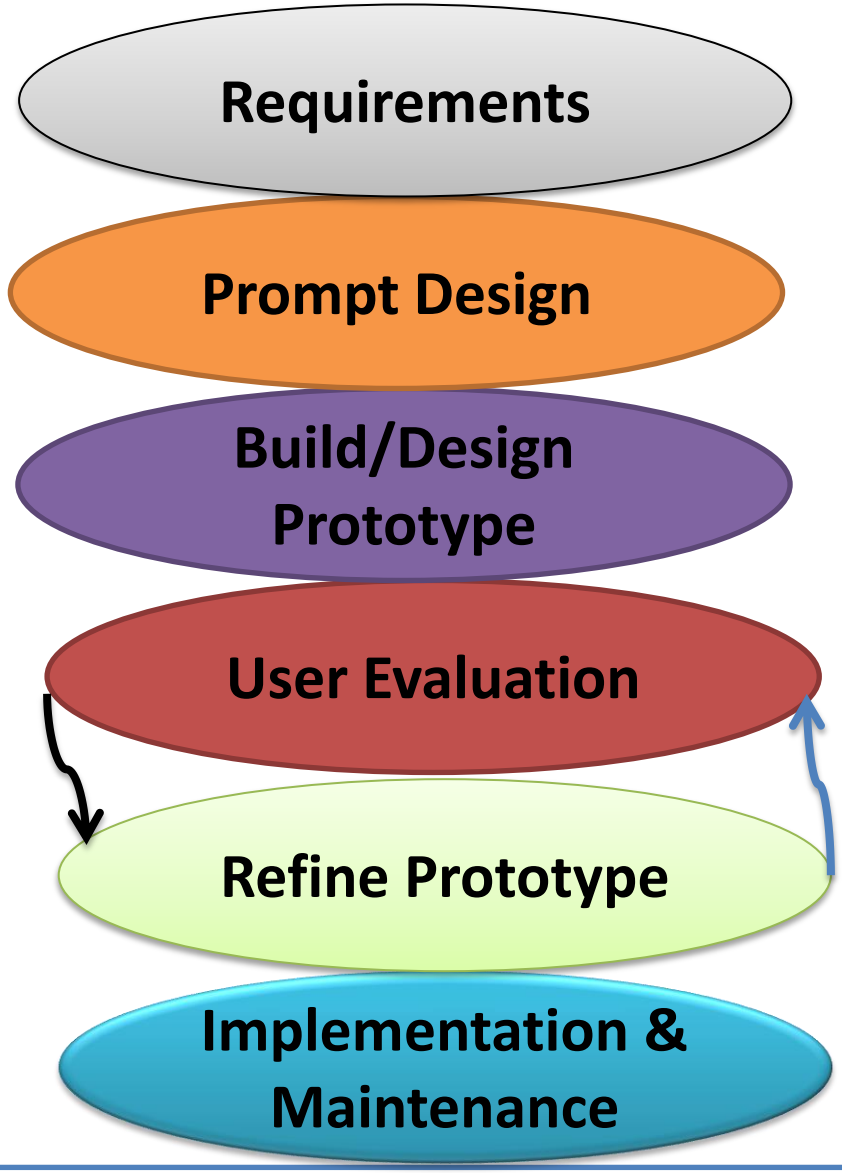
Waterfall Model - Advantages

- Simple and easy to understand and use
- Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.
- Phases are processed and completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Clearly defined stages.
- Well understood milestones.
- Easy to arrange tasks.
- Process and results are well documented.

Waterfall Model - Disadvantages

- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing. So, risk and uncertainty is high with this process model.
- It is difficult to measure progress within stages.
- Cannot accommodate changing requirements.

Phases in Prototype Model



Phases of Prototyping Model

- **Requirement**: In this phase requirement is gathered. In this phase the interaction with user is done to know his/her expectation from system.

- **Prompt Design:** This is very second phase of model. In this phase a simple quick design is formed. This is similar to actual design so that user would be able to get the brief idea about system.

- **Design/Build of Prototype:** This is very important phase because in this phase an actual prototype is build that is totally based on the information collected from prompt design.

Phases of Prototyping Model(Cont..)

- **Initial user evaluation**

In this stage, the proposed system is presented to the client for an initial evaluation. It helps to find out the strength and weakness of the working model. Comment and suggestion are collected from the customer and provided to the developer.

- **Refining prototype**

If the user is not happy with the current prototype, you need to refine the prototype according to the user's feedback and suggestions.

- **Implement Product and Maintain**

Once the final system is developed based on the final prototype, it is thoroughly tested and deployed to production. The system undergoes routine maintenance for minimizing downtime and prevent large-scale failures.

Build & Fix Model

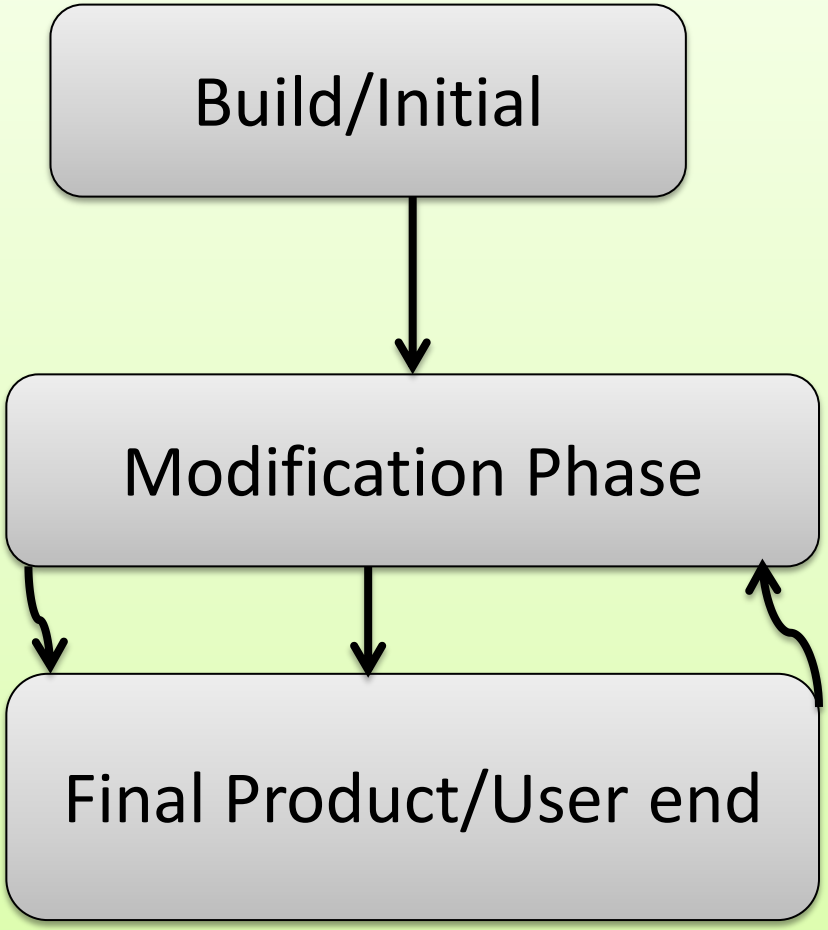
This model is also known as Ad-Hoc Model

In it software is developed without any specified design.

Or

Initial build up is made and necessary corrections are made until desire or expected output does not meet.

Build & Fix Model(cont…)



Phases in Build & Fix Model

Build Phase

This is very initial phase, in it software code is create/developed and moved to another phase for verification.

Fix Phase

This is very important phase. All errors are fixed in this phase to meet expected output/result.

Advantages of Phases in Build & Fix Model

There is no need of high project planning.

Suitable model for small project.

No large management activities are required.

Less management experience is required

Disadvantages of Phases in Build & Fix Model

Required more time.

Required high cost.

Unplanned model.

No documentation is required.

Maintenance is also tough.

Iterative Enhancement Model

1:It is also known as increment model.

2:It comprises the features of waterfall model in iterative manner.

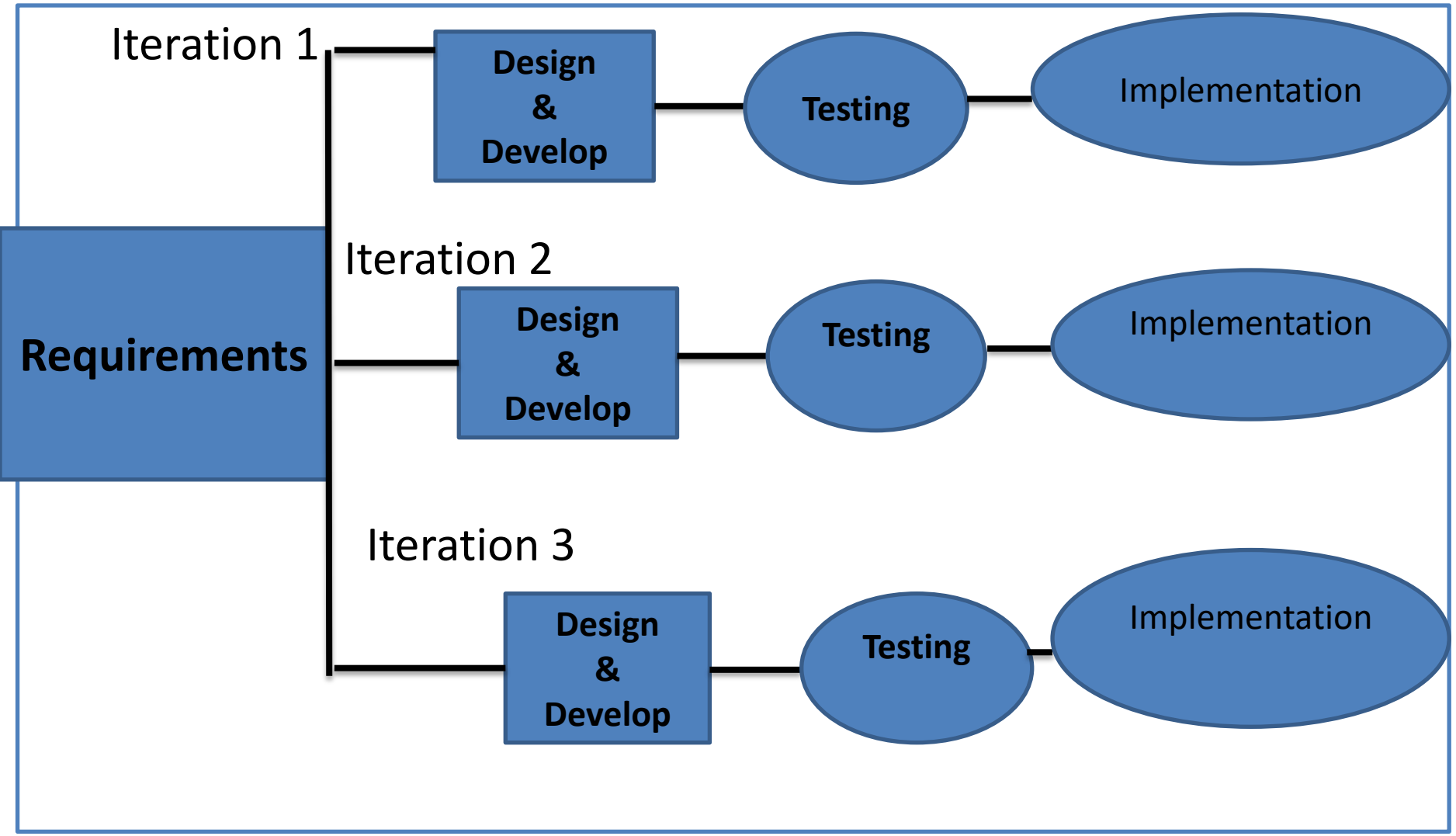
3:At each iteration, modifications in design are made and new functionalities are added.

4:The basic idea is to develop a product in iterative and incremental manner.

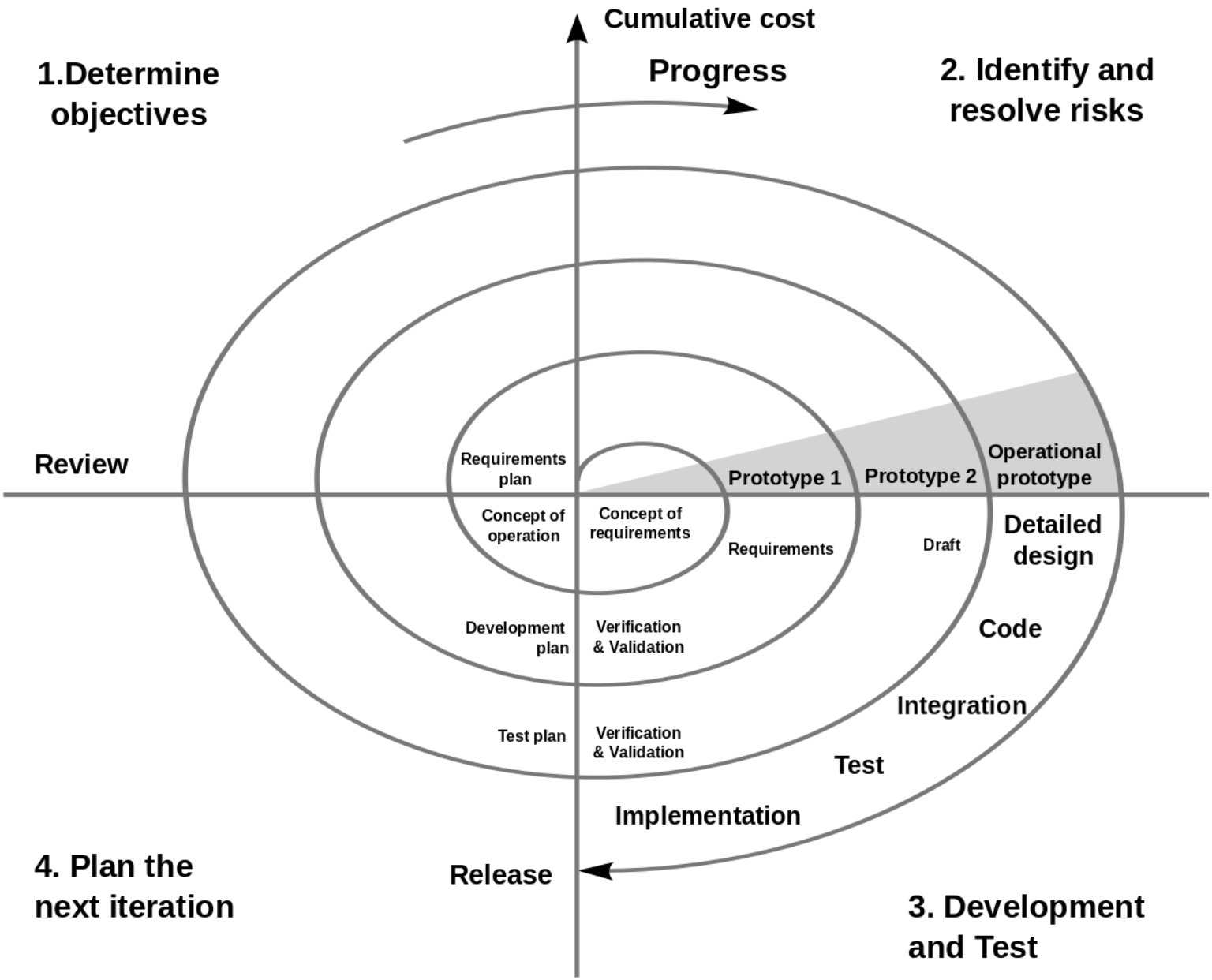
Iterative: Repeated Cycles

Incremental: Smaller portion at a time.

Iterative Enhancement Model



Spiral Model



Spiral Model

1:Determination of Objectives and identification of solution: All the requirements are collected from users and the objectives are identified. Analysis of objective is done at each phase.

2:Identification the solution of risk: In this quadrant the main task is to collect all possible solutions and select optimal solution among those. Now next task is to evaluate the risk associated with that solution. If risk persist then it is solved with the best method.

3:Development and Test: In this quadrant, all the features which have been identified are now tested for verification purpose.

4:Planning of next phase: In this phase product is reviewed and planning for next phase is started.

Spiral model is also called Meta Model

- It subsumes all other SDLC.
- It incorporates the features of waterfall model.

- It incorporates the features of prototype model.
- It incorporates the features of iterative model.

Spiral Model-Advantages

1:In spiral model risk handling and risk analysis is possible at every phase, because there is continuous and repeated development process.

2:If additional functionalities are required then these can be added or done at later stage.

3:Development process is fast.

4:Customer can observe the development of product at early stage.

5:It is suitable for large product.

Spiral Model-Disadvantages

1:If we compare this model with other SDLC models than it is complex.

2:Due to its complex nature expert resources are required which lead to high cost. So we can say that it is not suitable for small product.

It has so many intermediate phases so there are large documents in it.

Estimation of time is not so easy at the starting of this model due to unknown number of phases.

Agile Software Development Model

It is combination of iterative and incremental process

Basically, the main motive of agile is that every project must be dealt with different approaches/methods.

In agile task is broken into small iterations.

Plan for each iteration is clearly defined in advance.

The motive to divide project in small part is to minimize risk and to deliver the project on time.

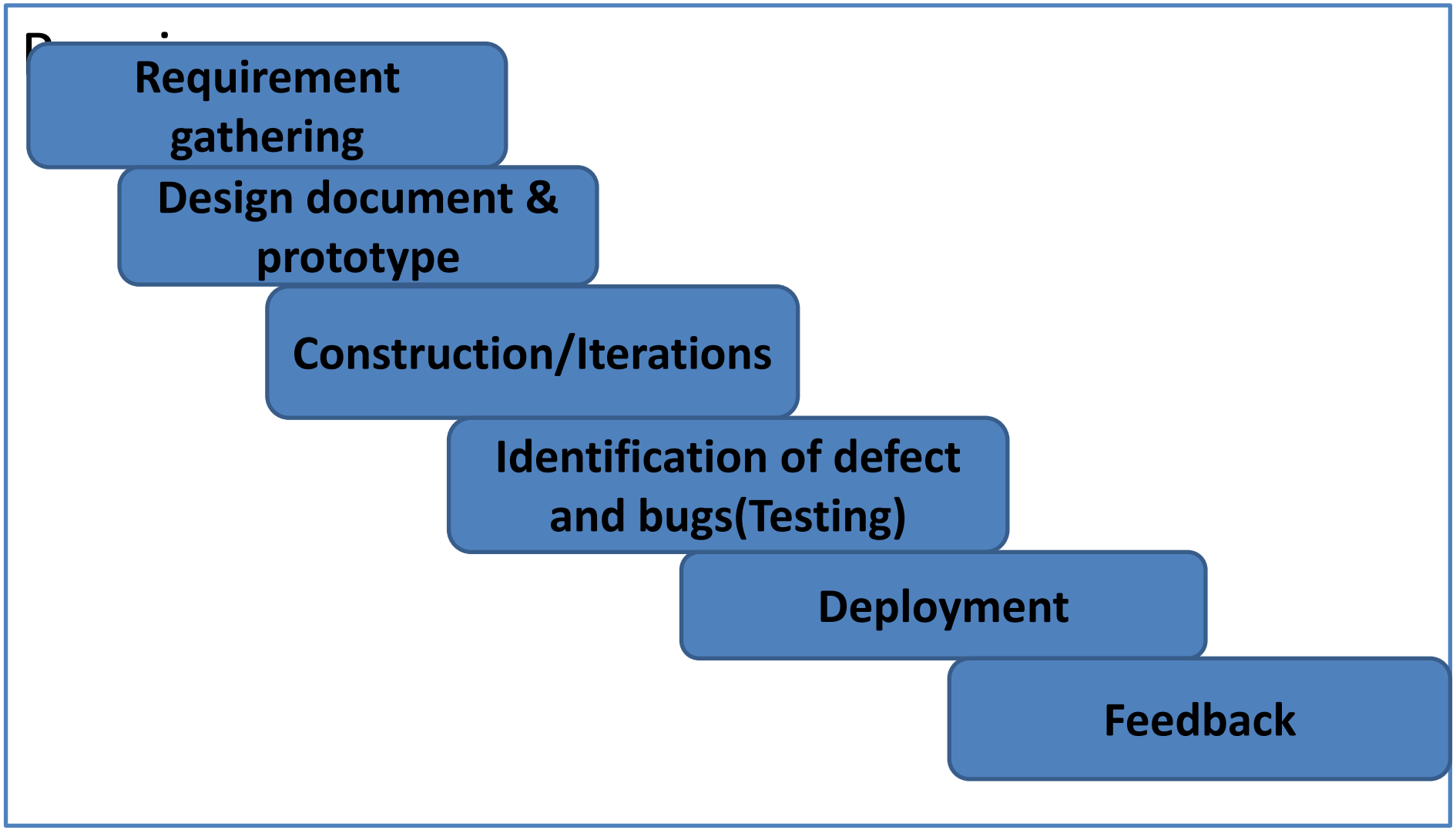
Agile Software Development Model

As development process is iterative, so the project may be executive in very short time period and no long planning is required.

There is a testing phase with each iteration.

Both testers and developers work together.

Phases in agile Model



Agile Software Development Model

1: Requirements gathering: This is most important phase of any model. All the future requirements of product is collected ,evaluated .Based on this information the feasibility of project is evaluated.

2:Design the requirements: After requirement gathering design of requirement is started. This design can be achieved using various data flow diagram.

3:Construction/ iteration: In this phase team members start working on assigned module or part.

4:Testing: This is the phase related to quality assurance. This is done to check the working capability of the product and to make a comparison between expected and actual output.

Agile Software Development Model

5: Deployment: In this phase product is issued to work in user's environment.

6:Feedback: This is last step in which team receive feedback from user and work on that.

Scrum

This is a agile methodology or we can say agile framework.

It is widely used by software development team.

Its main focus is on time business delivery(two to three week time).

Basically scrum is used where requirement is rapidly change, because due to its flexibility its quickly adopt changes.

Features of Scrum

- Light Weighted
- Self-organized
- Easy to understand

Scrum(cont…)

The meaning of light weighted is that the overhead of the process is kept as small as possible.

Self organization principles to complete work and to coordinate with other teams.

Scrum(cont…)

Scrum is a framework used to manage product development.

Scrum is run on time boxes of one month or less.

Scrum team consist of **product owner**, **development team** and **scrum master.**

Product owner: They are responsible for maintaining the product backlog and ensuring the entire scrum team understands the overall vision for the product

Development team: It is a team who has every thing to deliver the product on time without a dependency on outsider team. It is a self-organizing team.

Scrum(cont…)

Scrum master:

Responsible to ensure scrum process.

Responsible to train the team as and when required.

Responsible to remove all barriers.

Responsible for the practice of defined rules.

Scrum Events and Artifacts

Scrum Event: In scrum all events are time box events. Every event has maximum duration.

Scrum Artifact: It may also said as “work of art”.It provides information to be aware about the product development.

What activities have done?

What activities planned?

What activities to be?

List of Scrum Events and Artifacts

Scrum Event	Scrum Artifacts
Sprint	Product backlog
Sprint planning	Sprint backlog
Daily Scrum	Increments
Sprint review	
Sprint retrospective	

Sprint: Scrum project is divided into small iterations. Normally the duration of these iterations is one to three week. This is called Sprint.

Sprint Planning: Meaning of Sprint planning is to give answer.

What,we are doing?

How, we are doing?

Daily Scrum: Daily Scrum is a 15-minute time-boxed event for the Development Team to synchronize activities and create a plan for the next 24 hours.

Sprint Review: The sprint review is an informal meeting with the development team, the scrum master, the product owner and the stakeholders will attend.

Sprint Retrospective: Sprint Retrospective is an opportunity for the **Scrum** Team to inspect itself and create a plan for improvements to be enacted during the next **Sprint**. The **Sprint Retrospective** occurs after the **Sprint** Review and prior to the next **Sprint** Planning.

Product Backlog: **A product backlog** is a list of the new features, changes to existing features, bug fixes, infrastructure changes or other activities that a team may deliver in order to achieve a specific outcome.

Sprint Backlog: The **sprint backlog** is a list of tasks identified by the **Scrum** team to be completed during the **Scrum sprint.**

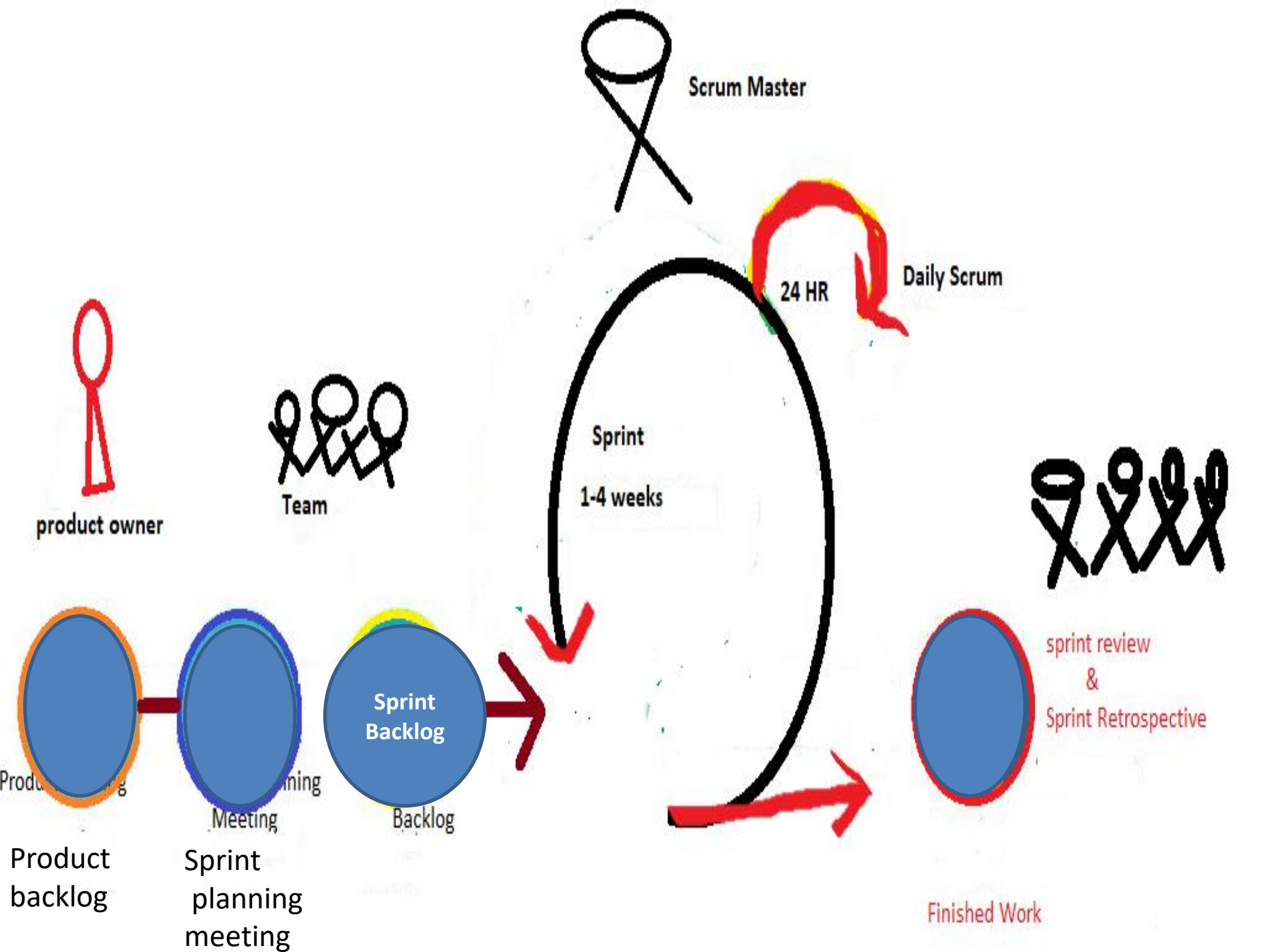
Increment: The **Increment** is the sum of all the Product Backlog items completed during a Sprint and the value of the **increments** of all previous Sprints

Sprint

Scrum project is divided into small iterations. Normally the duration of these iterations is one to three week. This is called Sprint.

It makes project more manageable.

Due to its flexibility it adopt changes easily.



Product backlog

Sprint planning meeting

Sprint Planning

Sprint planning is a collaborative event. Which can be done with the collaborative effort of whole Scrum team.

It is a time boxed to a maximum eight hours for a one month Sprint or Sprint planning should be constrained no more than two hours for each week of the sprint.

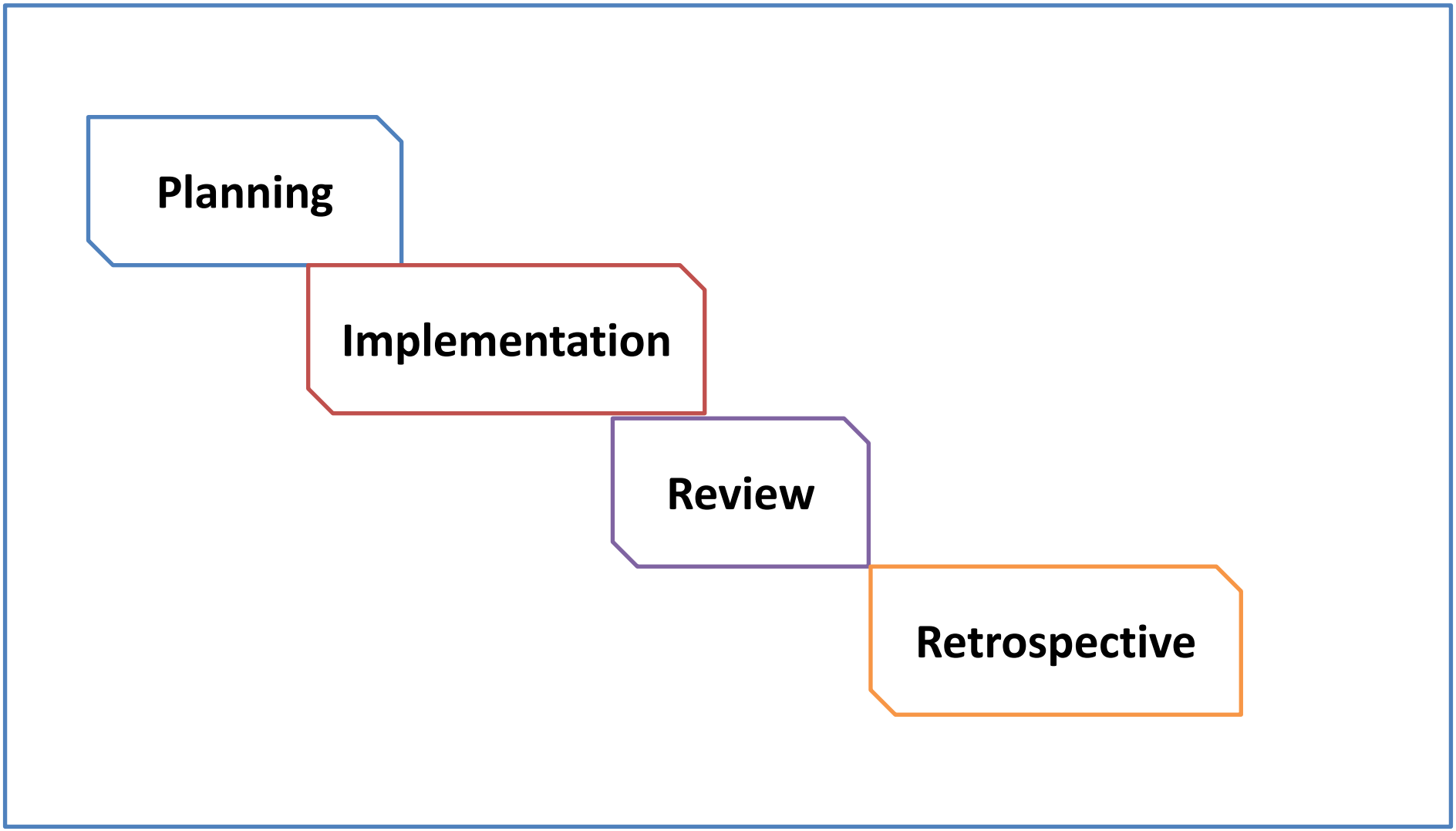
It provides some flexibility to developer regarding the functionality implemented within Sprint.

Meaning of Sprint planning is to give answer.

What are we doing?

How are we doing?

Sprint Lifecycle



Sprint Planning Meeting

The main motive of Sprint planning meeting is to provide structure, guidelines, criteria, expectations and define backlog for forthcoming backlog.

There are following people involved in Sprint planning meeting.

Product Owner: This is most demanding position. Product owner is responsible for managing the product backlog and product backlog visibility. product owner looks at the project from the customer's perspective. product owners are responsible for managing product backlogs

Scrum Master: Scrum Master is responsible to motivate the team in tough time and also responsible for timely completion of project. Scrum Master mainly focus on team's operation.

Scrum master improve the quality of product.

Product Backlog

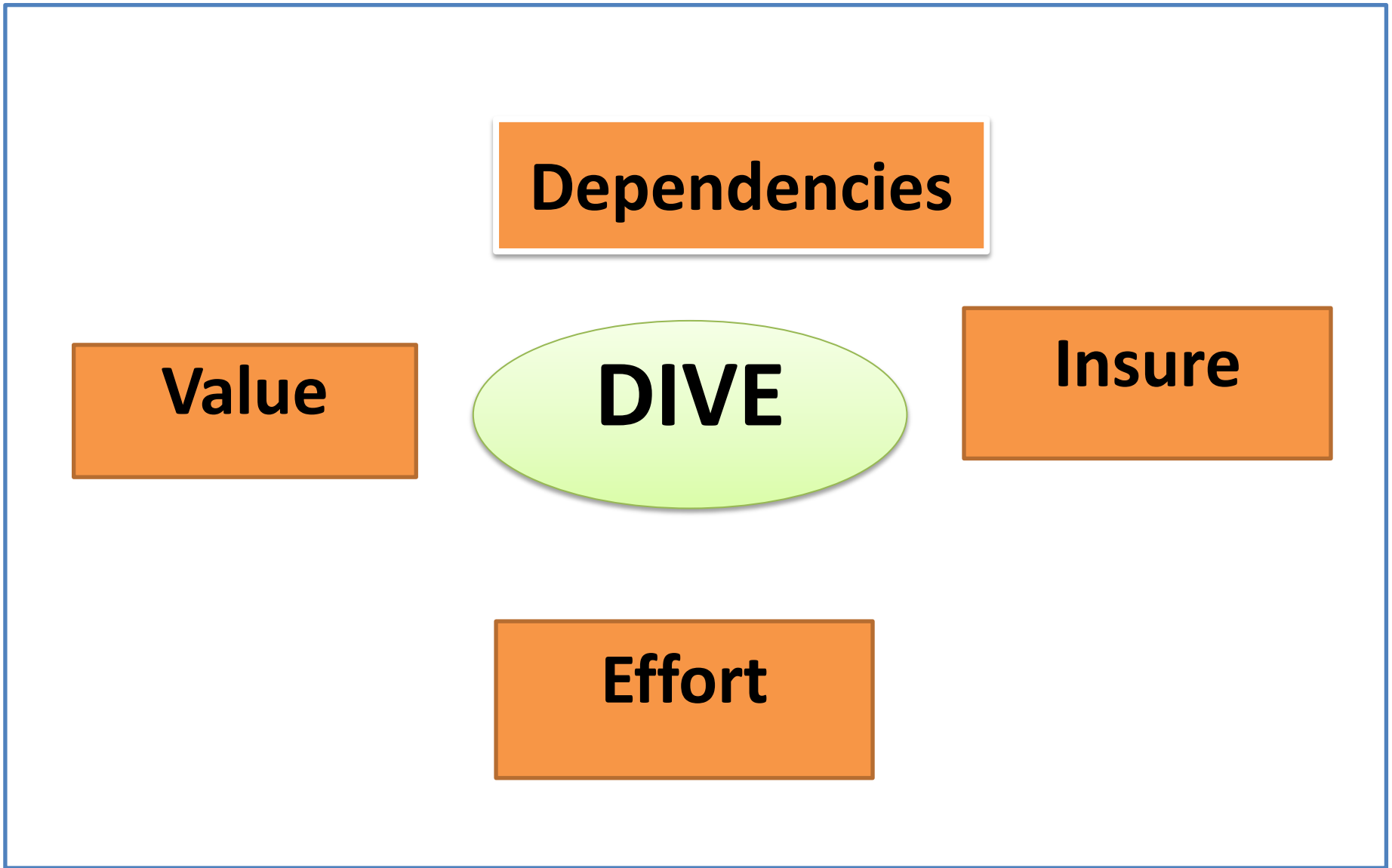
It is compiled of all the things that must be done to complete the whole project.

Product owner owns product backlog.

The product backlog acts as an input to the sprint backlog when comes to functionality

There are also bugs/issues, epic, user stories and themes are included in the product backlog

Techniques to maintain Product Backlog



Techniques to maintain Product Backlog

DIVE

Product backlog items are prioritized in linearly ordered based upon DIVE criteria

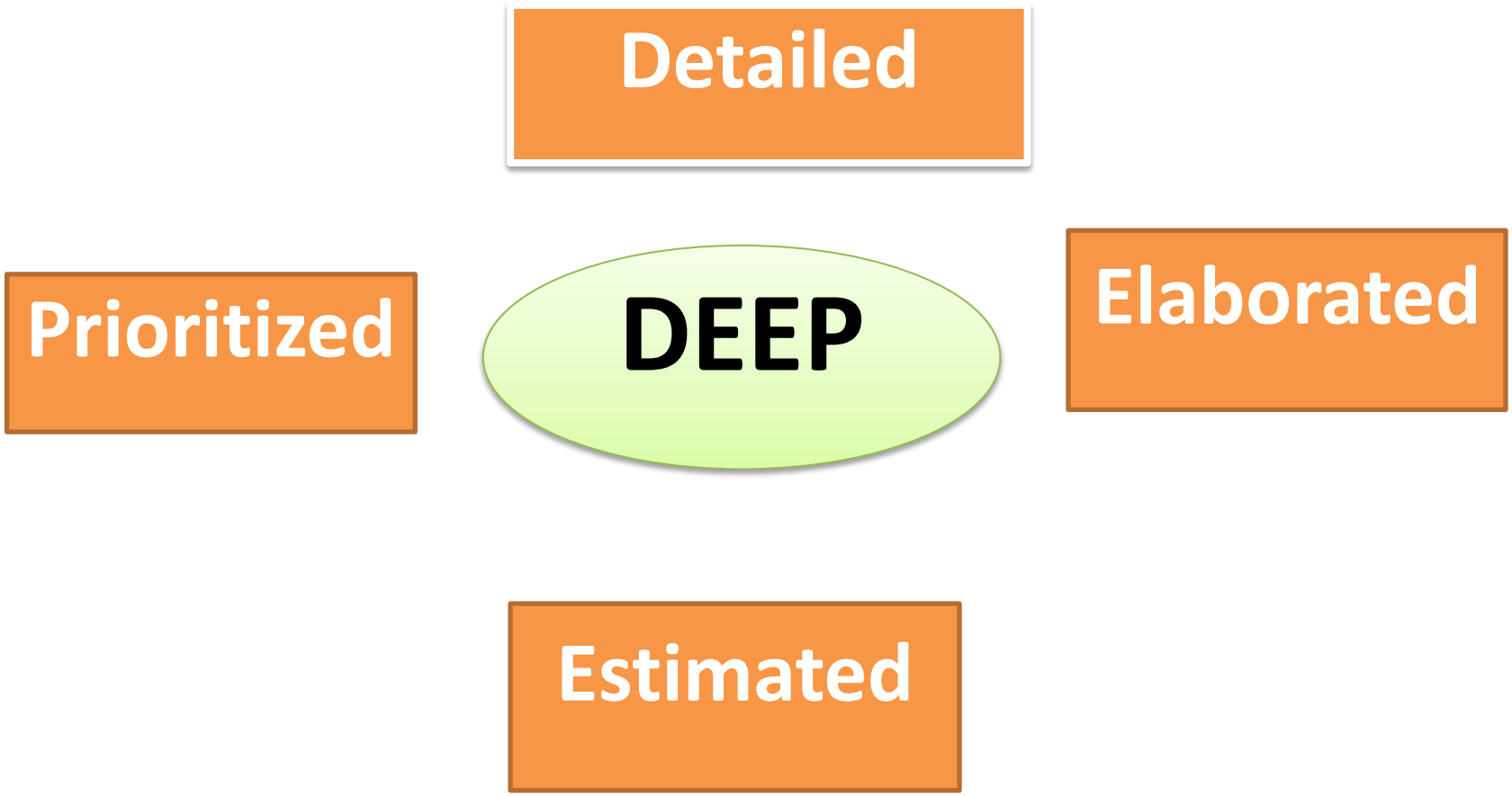
Dependencies – keep it linear with fewer dependencies with other user stories, epic or themes. It's OK to have horizontal dependencies.

Insure against risk (business and technical)

Value(Business)

Effort(Estimated)

Techniques to maintain Product Backlog



Techniques to maintain Product Backlog

INVEST

Independent : The user story should be self-contained, in a way that there is no inherent dependency on another user story.

Negotiable: User stories, up until they are part of an iteration, can always be changed and rewritten.

Valuable: A user story must deliver value to the end user.

Estimable: You must always be able to estimate the size of a user story. Small User stories should not be so big as to become impossible to plan/task/prioritize with a certain level of certainty.

Testable: The user story or its related description must provide the necessary information to make test development possible.

Sprint Backlog

Sprint backlog is the subset of the product backlog.

Product Owner set the sprint’s goal for the team, scrum team pick the user stories from product backlog fulfilling those goals.

Characteristic of a Sprint Backlog

Sprint backlog is dynamic in nature.

Sprint backlog is a subset of product backlog

Sprint backlog is an output of a sprint planning meeting.

In Sprint backlog, scrum team works on how the user stories would be implemented in a sprint by dividing it further into tasks and estimating it.

Characteristic of a Sprint Backlog

Assuming Product Backlog has stories: 1, 2, 3, 4 , 5 and 6. The team decides to do stories 1,2 and 4.

As during sprint planning team realized that there still some question which is not answered well by the Product Owner, so they decided not to include the user story no: 3 and jump to user story no:4, which was defined well.

Sprint backlog is owned by the Development Team and contains what and how it get’s delivered

GLA UNIVERSITY
MATHURA
Recognised by UGC Under Section 2(f)
Accredited with A Grade by NAAC

Subject Name

Software Engineering

Topic : Requirement Engineering

Department of Computer Engineering & Applications

Requirement Engineering

Requirement Engineering is mainly focused on discovering -

- what is to be developed?
- How it should be developed?

Main aspects are:

- What does the customer want?
- What does the customer require in order to use the system?
- What will be the software's impact on user be?

What requirement process ensure?

It ensures that our software must meet the user expectations and ending up with a high quality software.

It’s a critical stage of the software process as errors at this stage will reflect later on the next stages, which definitely will cause you a higher costs.

GLA UNIVERSITY MATHURA
Recognised by UGC Under Section 2(f)
Accredited with A Grade by NAAC

What should we do when we are going to develop a software?

To develop the software system we should have clear understanding of Software system.

To achieve this we need to continuous communication with customers to gather all requirements.

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.

GLA UNIVERSITY MATHURA
Recognised by UGC Under Section 2(f)
Accredited with A Grade by NAAC

Types of requirements

Known Requirements

Something on which the stakeholder believes to be implemented

Unknown Requirements

Forgotten by stakeholders as they are not needed right now.

Undreamed Requirements

Stakeholder may not be able to think due to lack of domain knowledge

Feasibility study

What is feasibility study?

It is a analysis that accounts all the factors, i.e economic, technical, legal and others with are necessary for the completion of a project successfully.

Feasibility study focuses on

- Is the concept of project is workable
- Should we proceed with the proposed project idea
- To find out the estimated cost and schedule of project
- Identification of all the major risks
- Stability of requirements
- Identification of new possibilities through investigative process

References

1:K.K. Aggarwal &Yogesh Singh

2:https://www.tutorialspoint.com/

Software Requirement Specification

It is a description of a software system to be developed.

A **software requirements specification** (SRS) is a detailed description of a software system to be developed with its functional and non-functional requirements.

The SRS is developed based the agreement between customer and contractors. It may include the use cases of how user is going to interact with software system.

The software requirement specification document consistent of all necessary requirements required for project development.

To develop the software system we should have clear understanding of Software system.

To achieve this we need to continuous communication with customers to gather all requirements.

GLA UNIVERSITY MATHURA
Recognised by UGC Under Section 2(f)
Accredited with A Grade by NAAC

Requirement Elicitation

Requirements Elicitation 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.

GLA UNIVERSITY MATHURA
Recognised by UGC Under Section 2(f)
Accredited with A Grade by NAAC

Requirement Elicitation Process

- Requirement gathering
- Requirement Organization
- Negotiation &Discussion
- Requirement Specification

Requirement Elicitation Process

- **Requirements gathering**

The developers discuss with the client and end users and know their expectations from the software.

- **Organizing Requirements**

The developers prioritize and arrange the requirements in order of importance, urgency and convenience.

Requirement Elicitation Process(cont..)

- **Negotiation & discussion**

If requirements are ambiguous or there are some conflicts in requirements of various stakeholders, if they are, it is then negotiated and discussed with stakeholders. Requirements may then be prioritized and reasonably compromised

- **Requirement Specification**

Software requirement specification(SRS)is a detailed description of a software system with its functional and non functional requirements.

Functional Requirements

- It specify the behavior of a system.
- What software system will do.

Non-Functional Requirements

- It is related to quality of a system.
- How software system will do .

GLA UNIVERSITY MATHURA
Recognised by UGC Under Section 2(f)
Accredited with A Grade by NAAC

Non-Functional Requirements(Contd..)

Generally Non-functional requirements fall into many areas some of them are as given below.

- **Efficiency**
- **Availability**
- **Flexibility**
- **Usability**
- **Portability**
- **Maintainability**

GLA UNIVERSITY MATHURA
Recognised by UGC Under Section 2(f)
Accredited with A Grade by NAAC

Requirement Elicitation Techniques

Interviews

- Interviews are strong medium to collect requirements. Organization may conduct several types of interviews
  - **Structured (closed) interviews**
- Where every single information to gather is decided in advance
- Follow pattern and matter of discussion.

  - **Non-structured (open) interviews**
- Where information to gather is not decided in advance These are more flexible and less biased.

Interview(cont..)

- Oral interviews
- Written interviews
- One-to-one interviews

which are held between two persons across the table.

- Group interviews

Held between groups of participants.

Help to uncover any missing requirement as numerous people are involved.

Requirement Elicitation Techniques

Brainstorming

- It is a group technique
- So many peoples are involved in this activity so lots of new ideas are shared
- Every idea/view is documented properly so that every one would be able to see these ideas.

Requirement Elicitation Techniques

Brainstorming

- A detailed report of all is prepared which consist list of requirements and their priority.
- As it's a group activity so there are chances of group conflict and bias. To handle such circumstances a highly trained facilitator is required.

Requirement Elicitation Techniques

FAST

- It stands for Facilitated Application Specification Technique.

- Due team oriented nature of this approach it is somewhere similar brainstorming.
- Main motive of this approach is to fill the gap between the customer and developer.
- Each participants has to prepare his/her list.
- All different lists are combined and redundant lists are removed.
- Sub teams are made for mini specifictions.

Requirement Elicitation Techniques

QFD

- It stands for **Quality Function Deployment.**
- Main priority of this technique is customer satisfaction.
- It is obvious ,if customer requirements are fulfilled then customer will be satisfied.

Requirement Elicitation Techniques

QFD(Cont..)

There may be various types of customer requirements.

Normal Requirement

In it all the requirements of software (which is to be developed) are discussed with customer.

Expected requirements

Requirements which are not explicitly stated by customer.(e.g:Security from unauthorized access)

Requirement Elicitation Techniques

QFD(Cont..)

Exciting Requirement

These are beyond the expectation of user.

In it developer adds some unexpected features to make customer more satisfied.

References

1:K.K. Aggarwal &Yogesh Singh

2:https://www.tutorialspoint.com/

Use case diagram

- Use case diagrams are graphical representations
- Use case diagram is the primary form of system/software requirements for a new software program underdeveloped.
- Use cases specify the expected behavior (what), and not the exact method of making it happen (how).
- Use cases once specified can be denoted both textual and visual representation (i.e. use case diagram).
- A key concept of use case modeling is that it helps us design a system from the end user's perspective.
- It is an effective technique for communicating system behavior in the user's terms.

Use case diagram(cont..)

- Use case diagrams are typically developed in the early stage of development and people often apply use case modeling for the following purposes:

- Specify the context of a system
- Capture the requirements of a system
- Validate a systems architecture
- Drive implementation and generate test cases
- Developed by analysts together with domain experts

Components of use case approach

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.

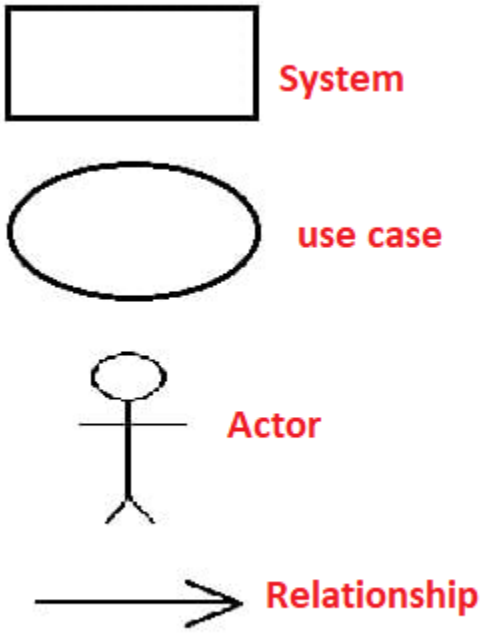
Secondary Actor: Secondary actor is one from which system needs assistance.

Actor

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

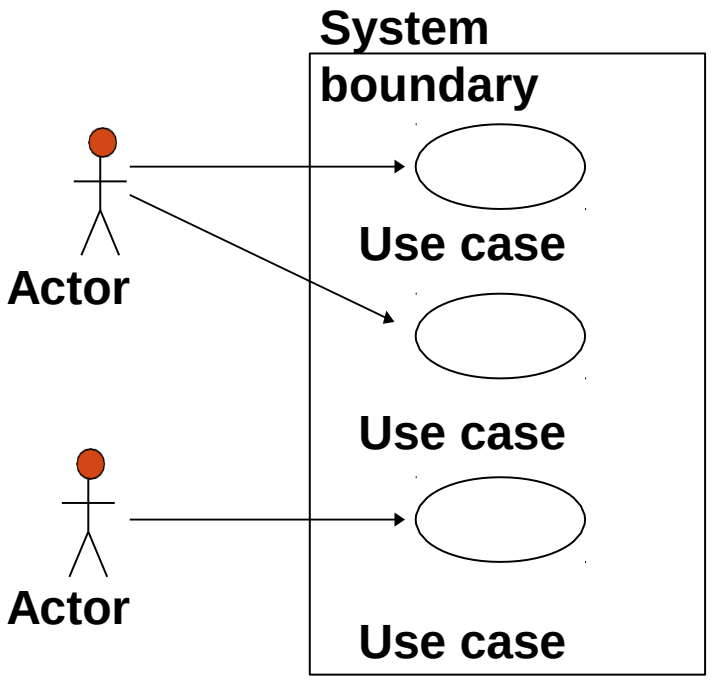
Actors can be a human user, some internal applications, or may be some external applications.

Notations



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.

Components of use case approach(cont..)

Use Case

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

It describes the sequence of interactions between actors and the system necessary to deliver the services that satisfied the goal.

So, we can say that

A use case captures who(actor) does what (interacting) with the system, for what purpose(goal),without dealing with system internals.

Components of use case approach(cont..)

Communication link

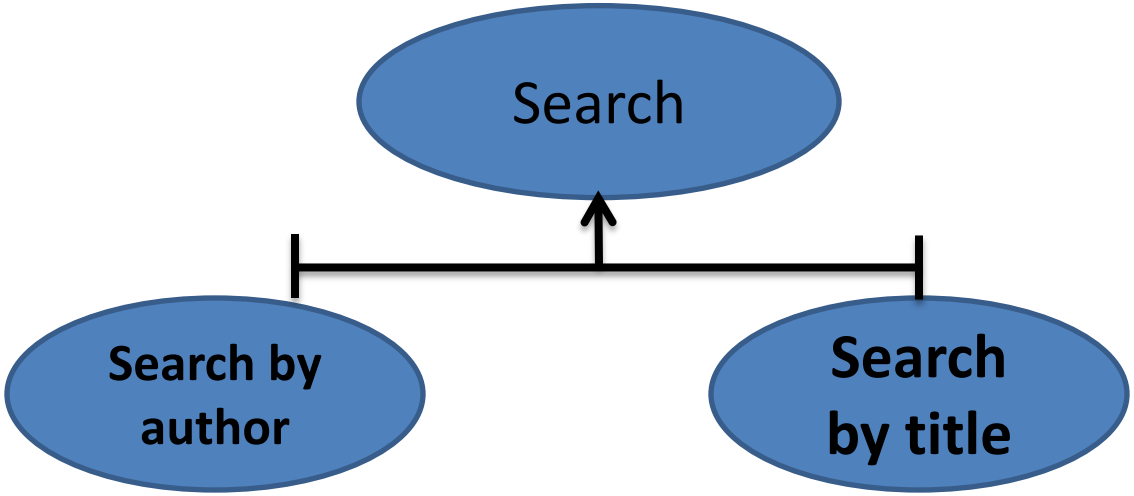
The participation of an actor in a use case is shown by connecting an actor to a use case by a solid link.

Boundary of the system

The system boundary is an entire system.

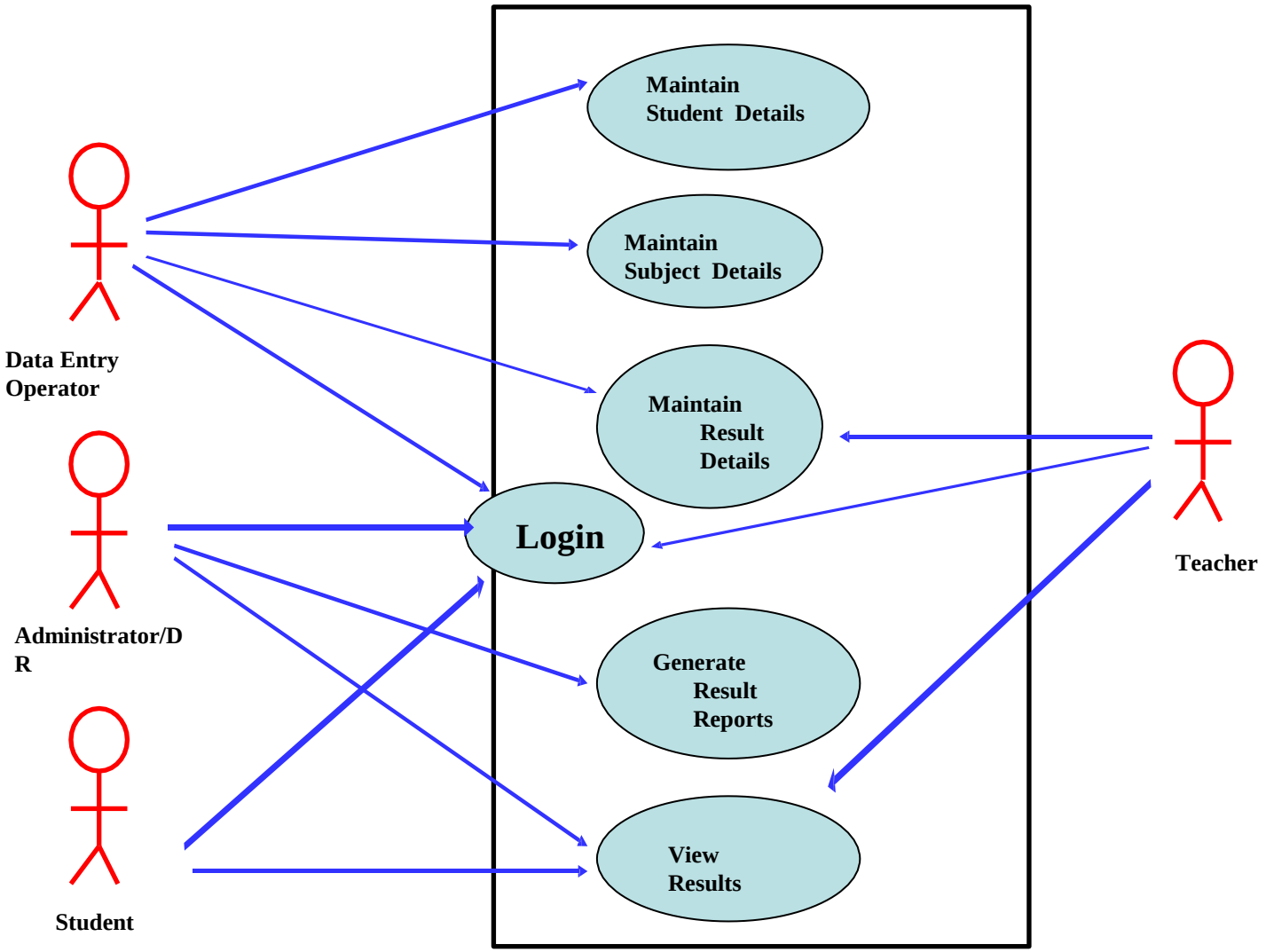
Use Case Example

Generalization Relationship

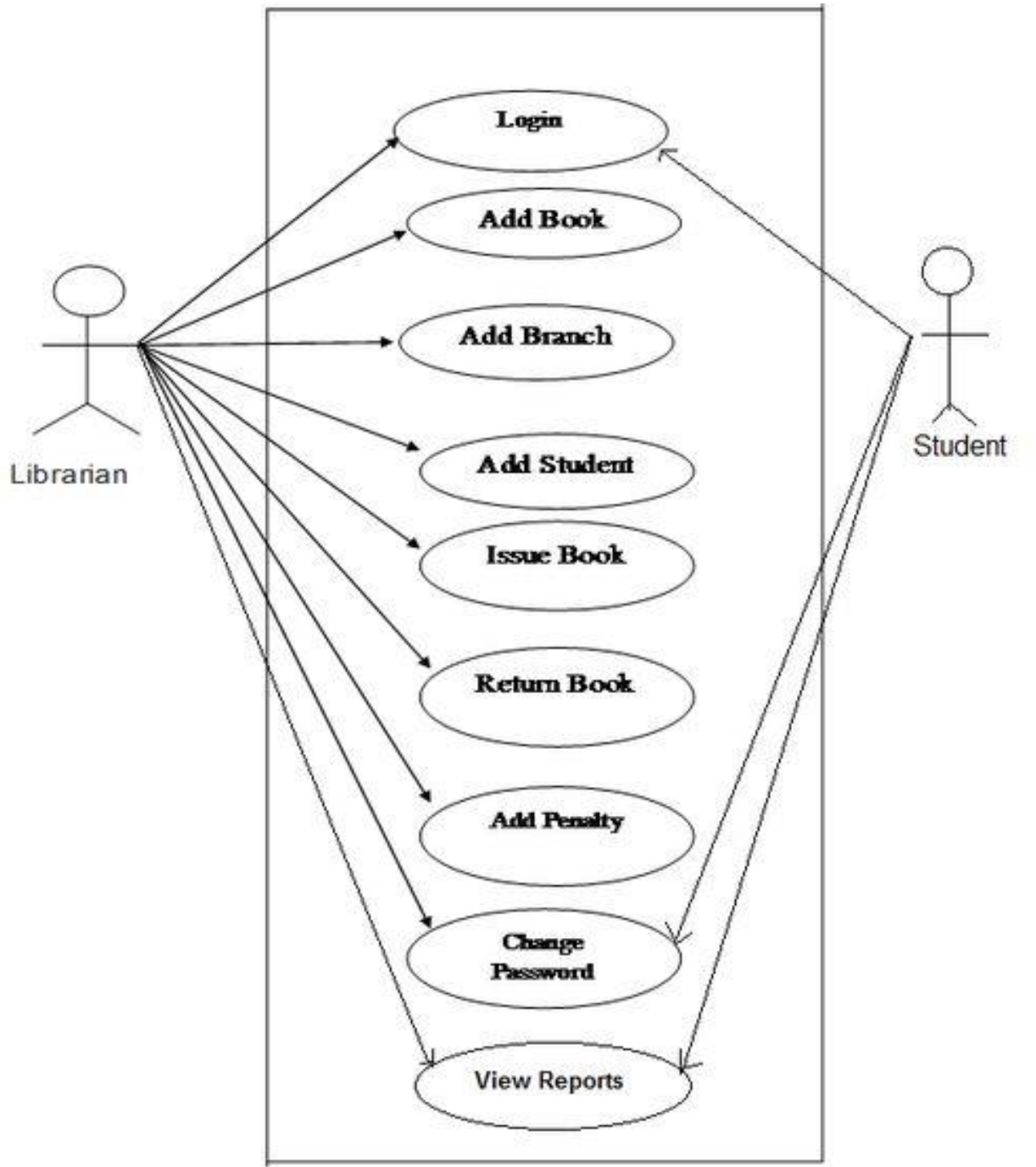


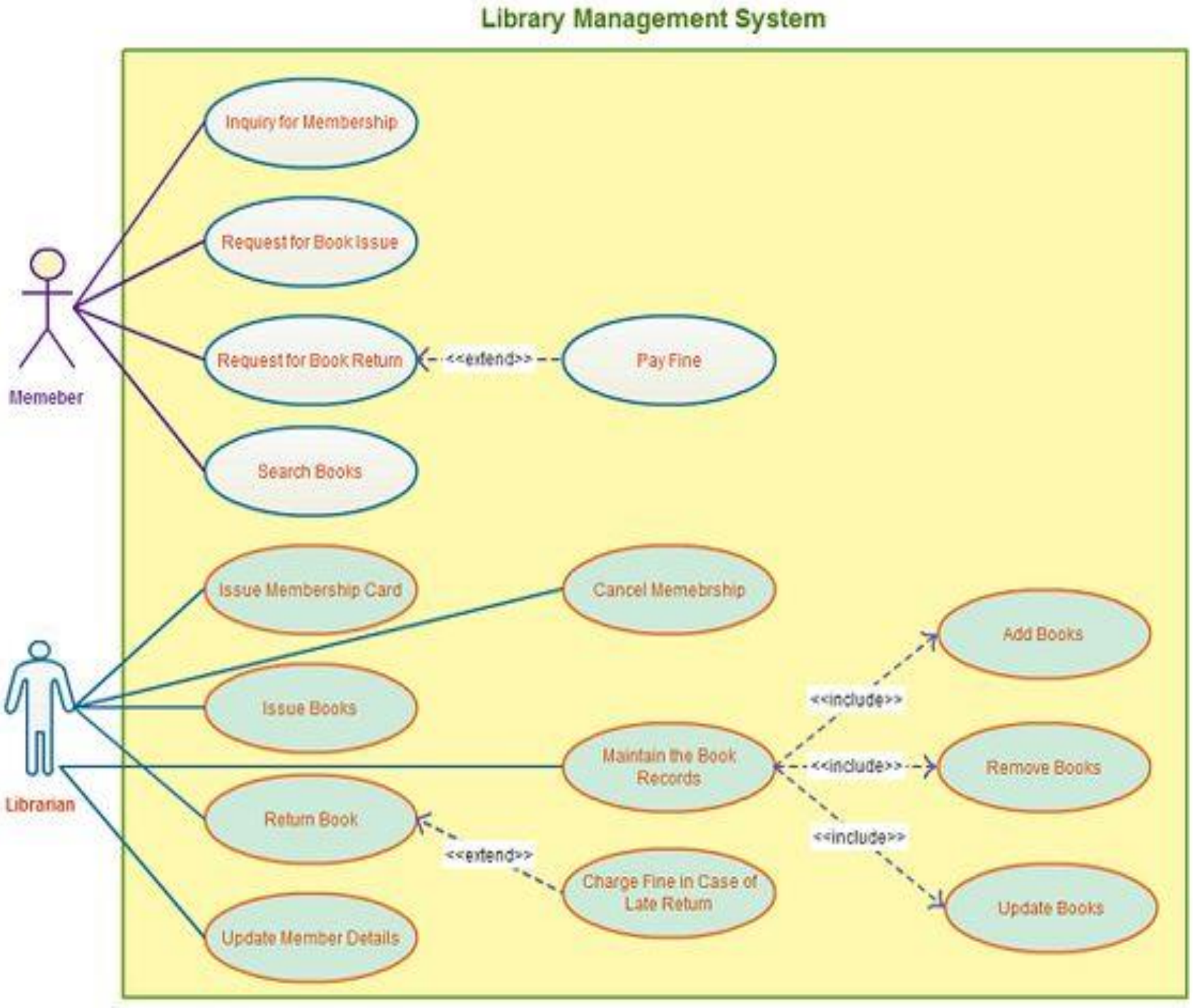
A generalization relationship means that a child use case inherits the behavior and meaning of the parent use case.

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

How to Identify Actor

Who uses the system?

Who installs the system?

Who starts up the system?

Who maintains the system?

Who shuts down the system?

Who gets information from this system?

Who provides information to the system?

How to Identify Use Cases?

What functions will the actor want from the system?

Does the system store information? What actors will create, read, update or delete this information?

Does the system need to notify an actor about changes in the internal state?

Are there any external events the system must know about? What actor informs the system of those events?

Representation

A case is often represented in a plain text or a diagram. Due to the simplicity of the use case diagram, it is considered to be optional by any organization

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

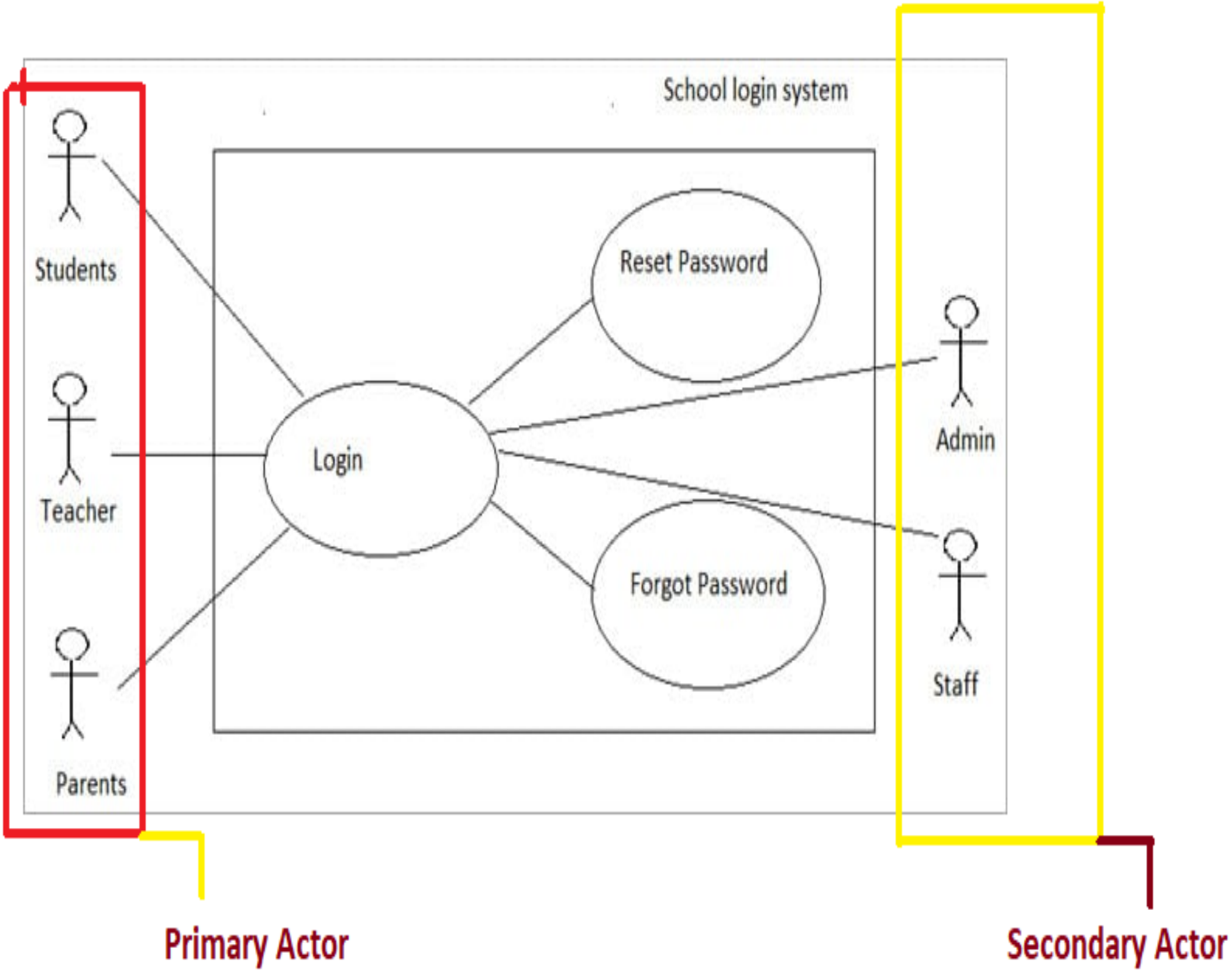
Pre-Condition--------System must be connected to the network.

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

Use Case Testing

It comes under the Functional Black Box testing technique. As it is a black box testing, there won't be any inspection of the codes.

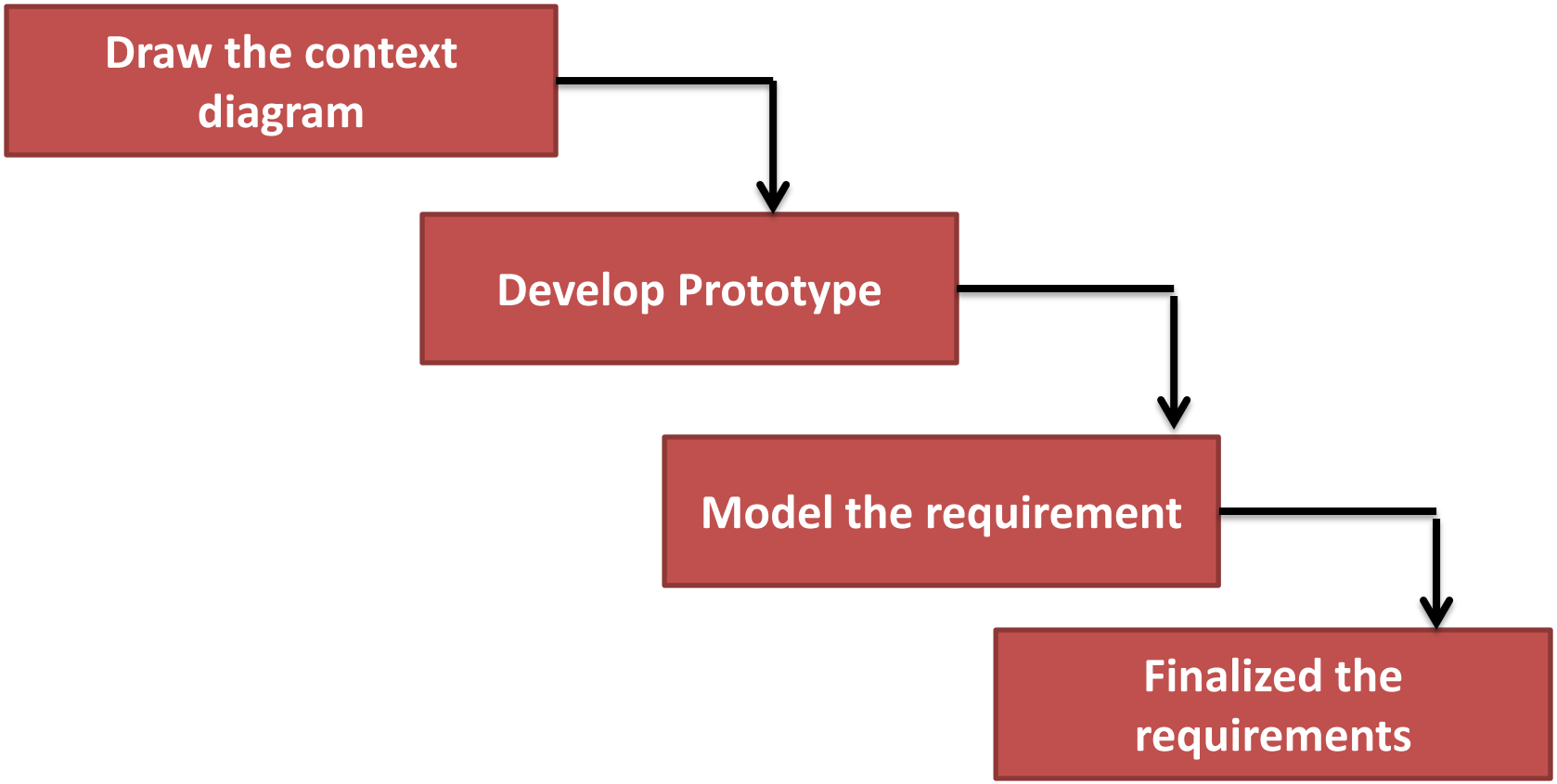
It ensures if the path used by the user is working as intended or not. It makes sure that the user can accomplish the task successfully.


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

Test Case Example

Test Cases	Steps	Expected Result
1	Enter Student Name	User can enter Student name
2	Enter Student ID	User can enter Student ID
3	Click on View Mark	System displays Student Marksfor ‘Show Student Marks’ case

Requirement Analysis Steps



Requirement Analysis Steps(cont..)

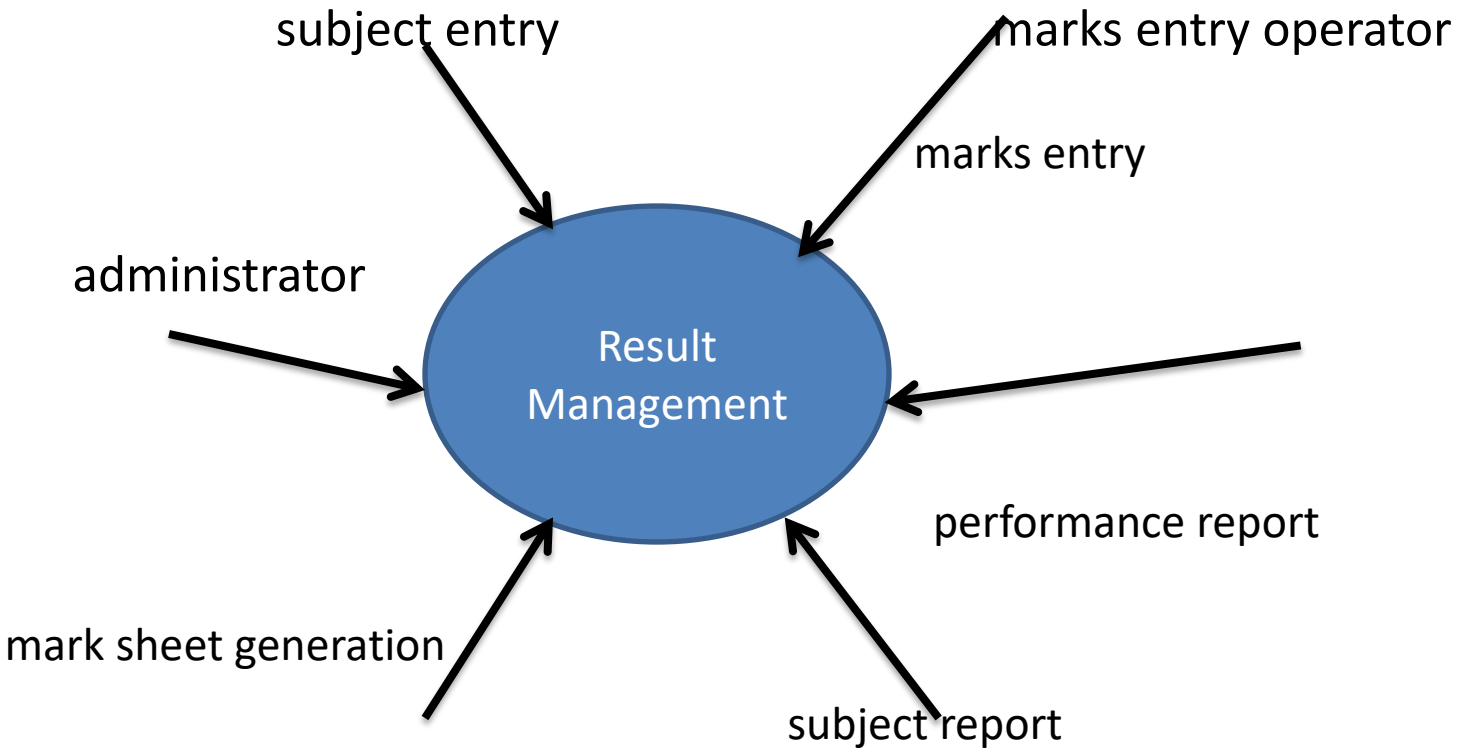
Draw the context diagram

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.

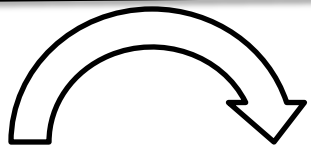
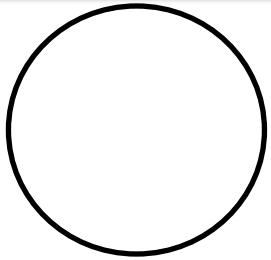
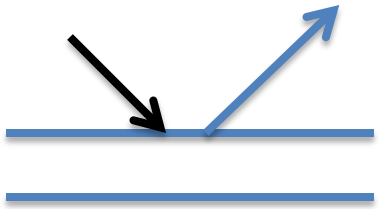
Elicitation and analyze activities have provided better insight into the system.

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:

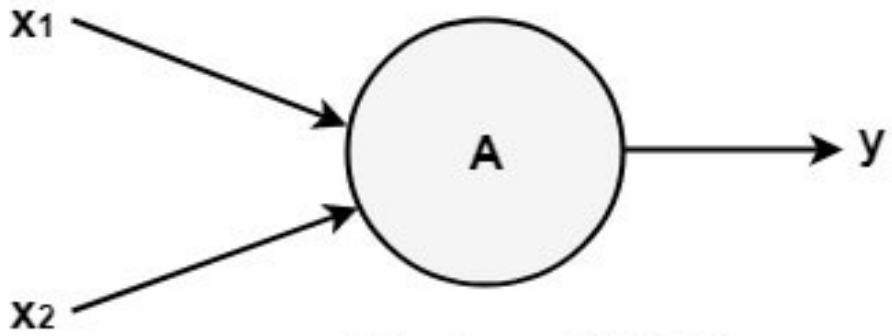


Fig: Level-0 DFD.

The Level-0 DFD, also called context diagram.

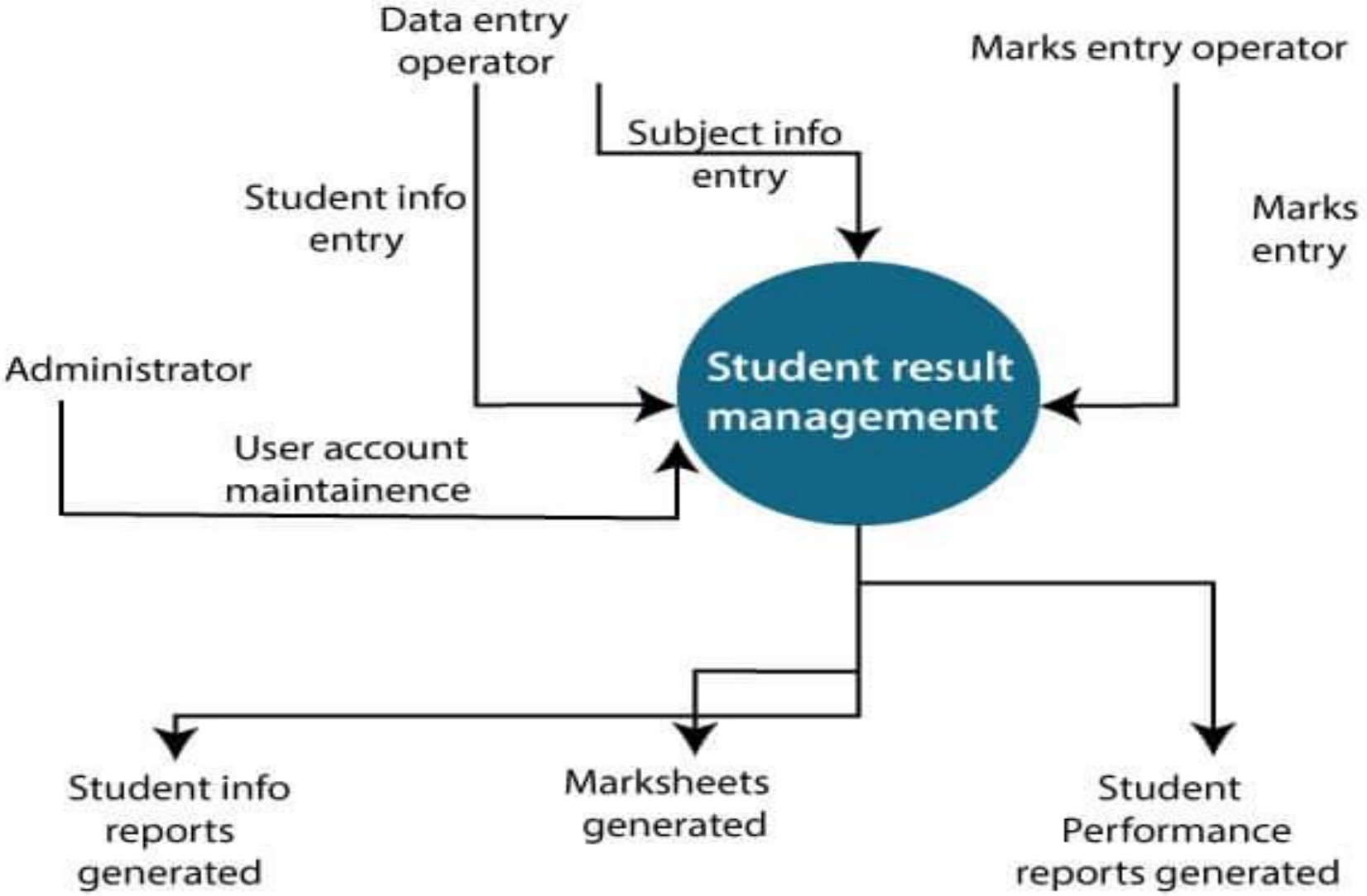
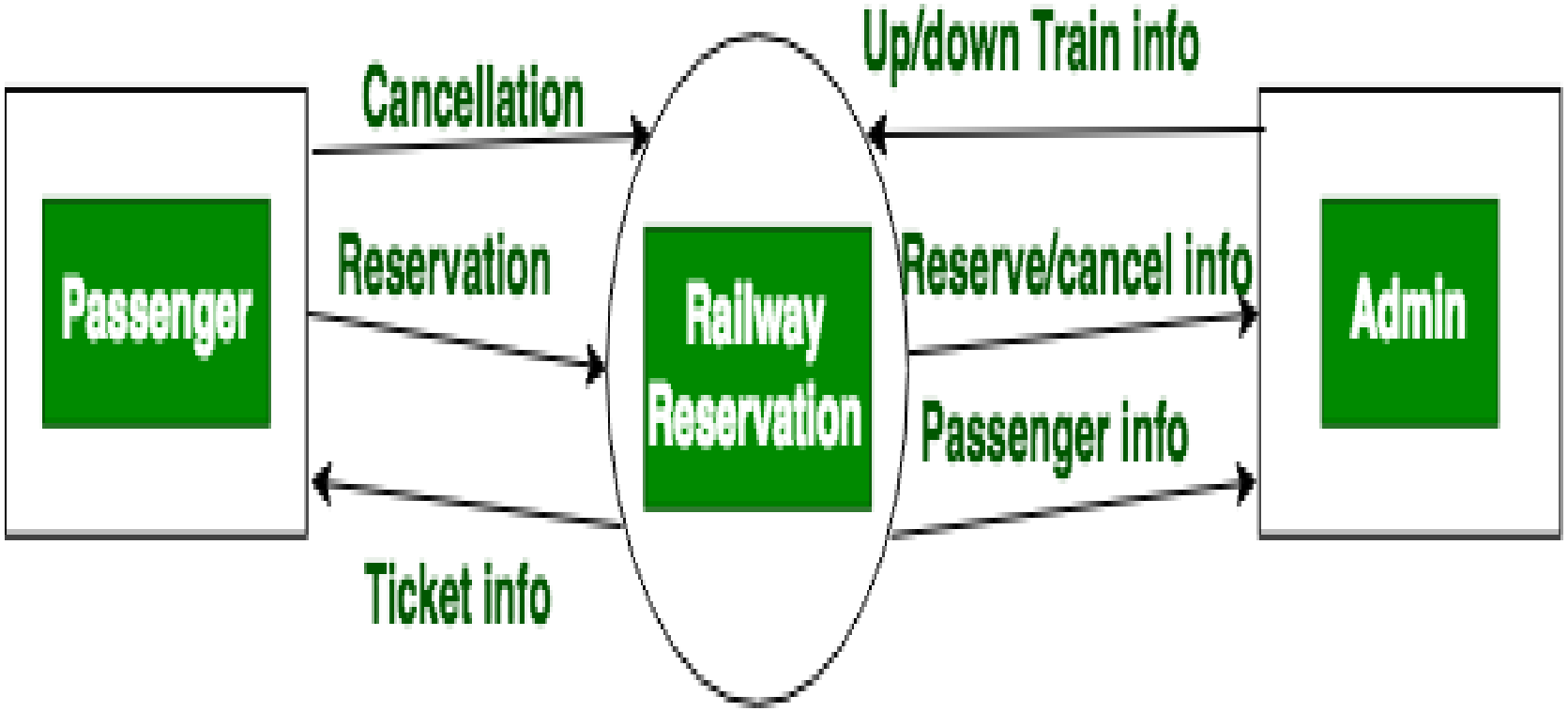
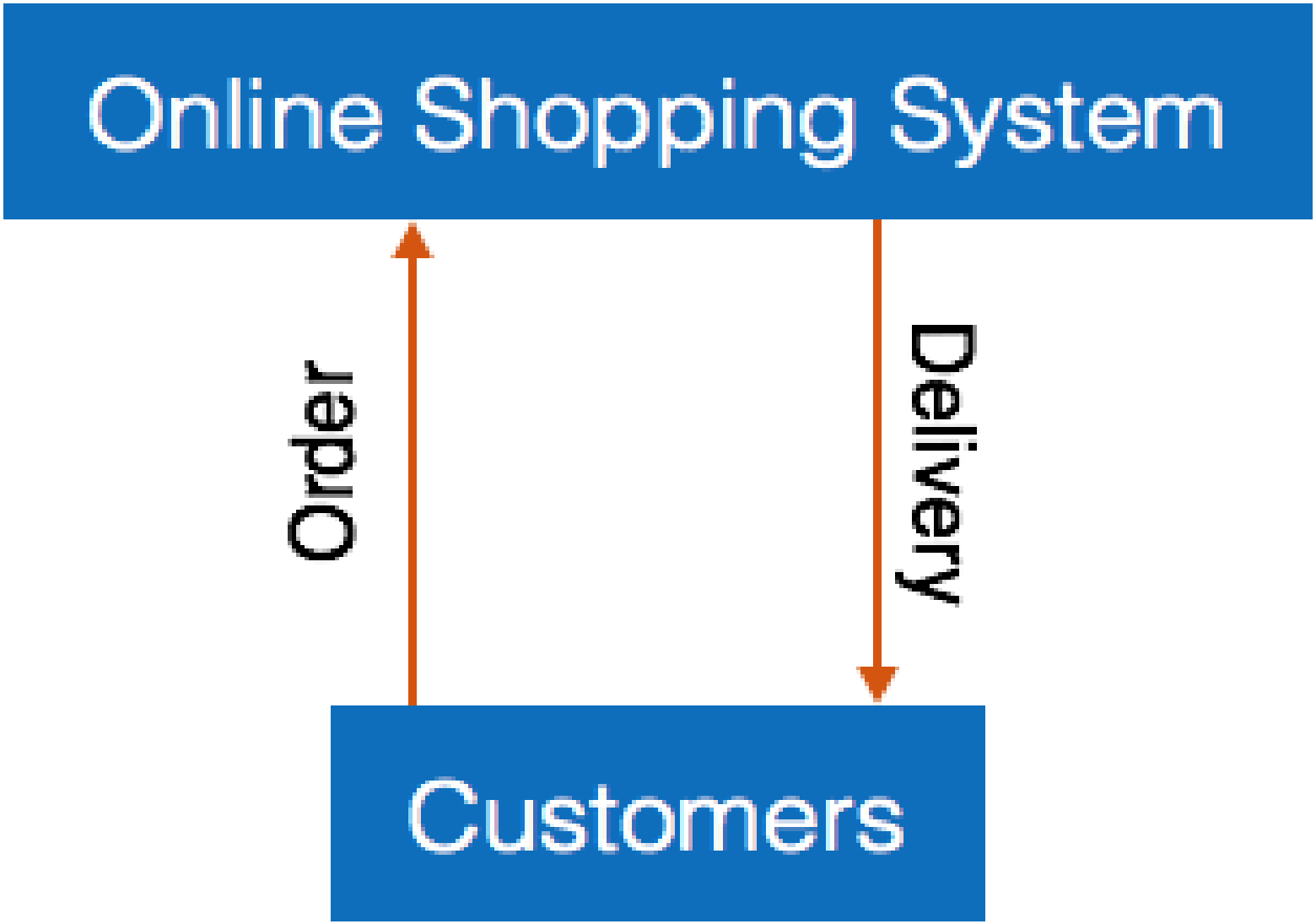


Fig: Level-0 DFD of result management system


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

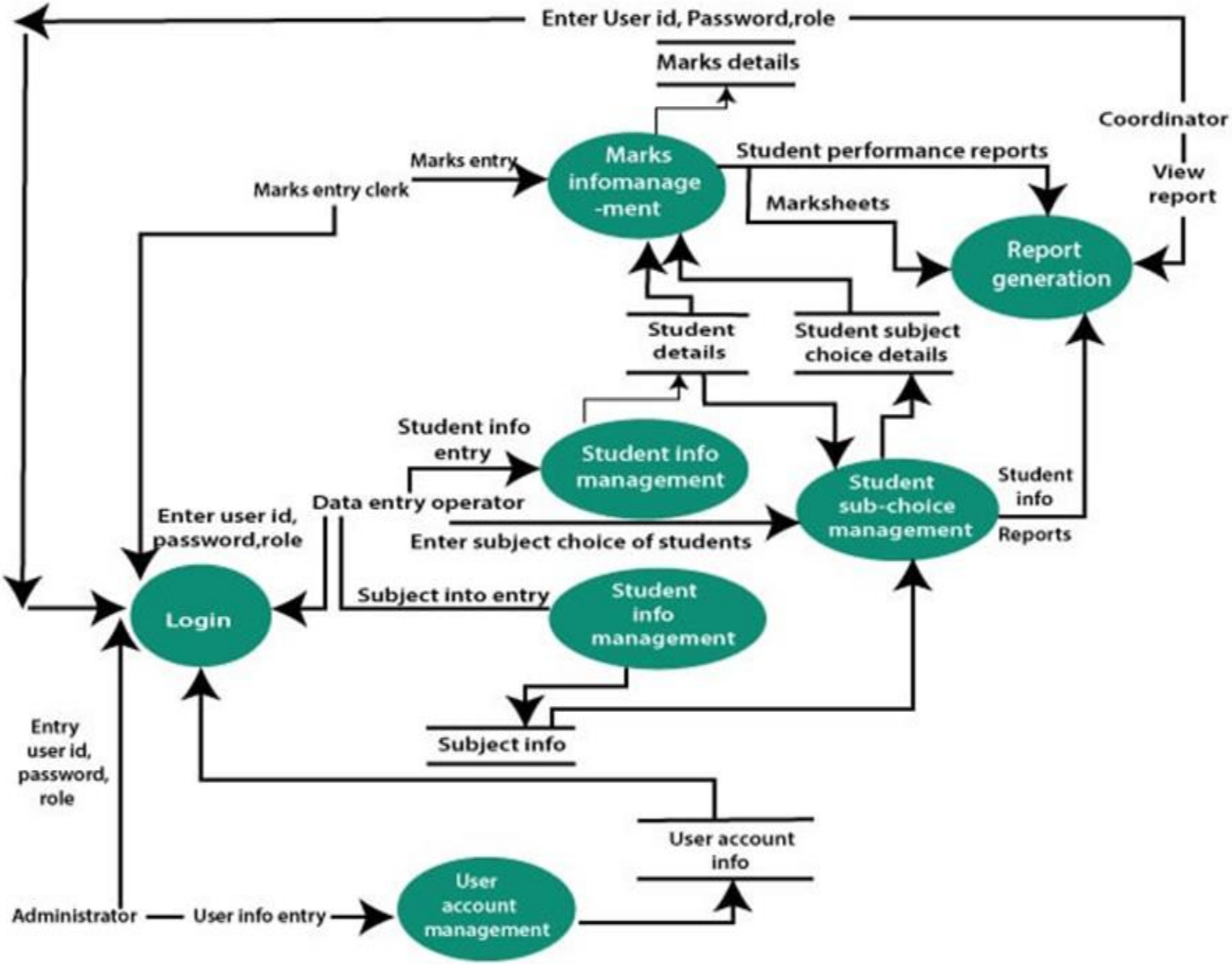
0-LEVEL DFD

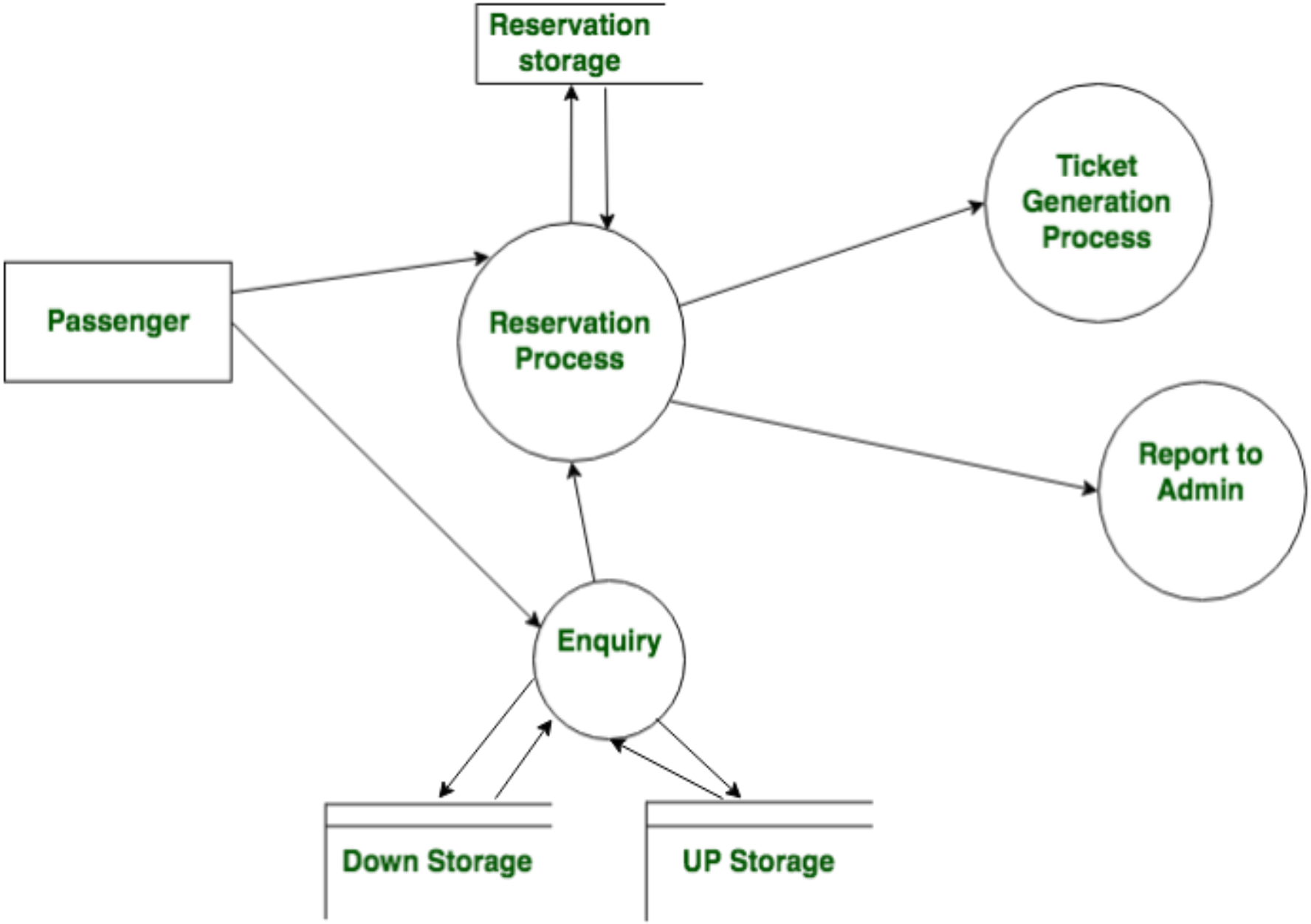

Online Shopping System
Order
Delivery
Customers

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.

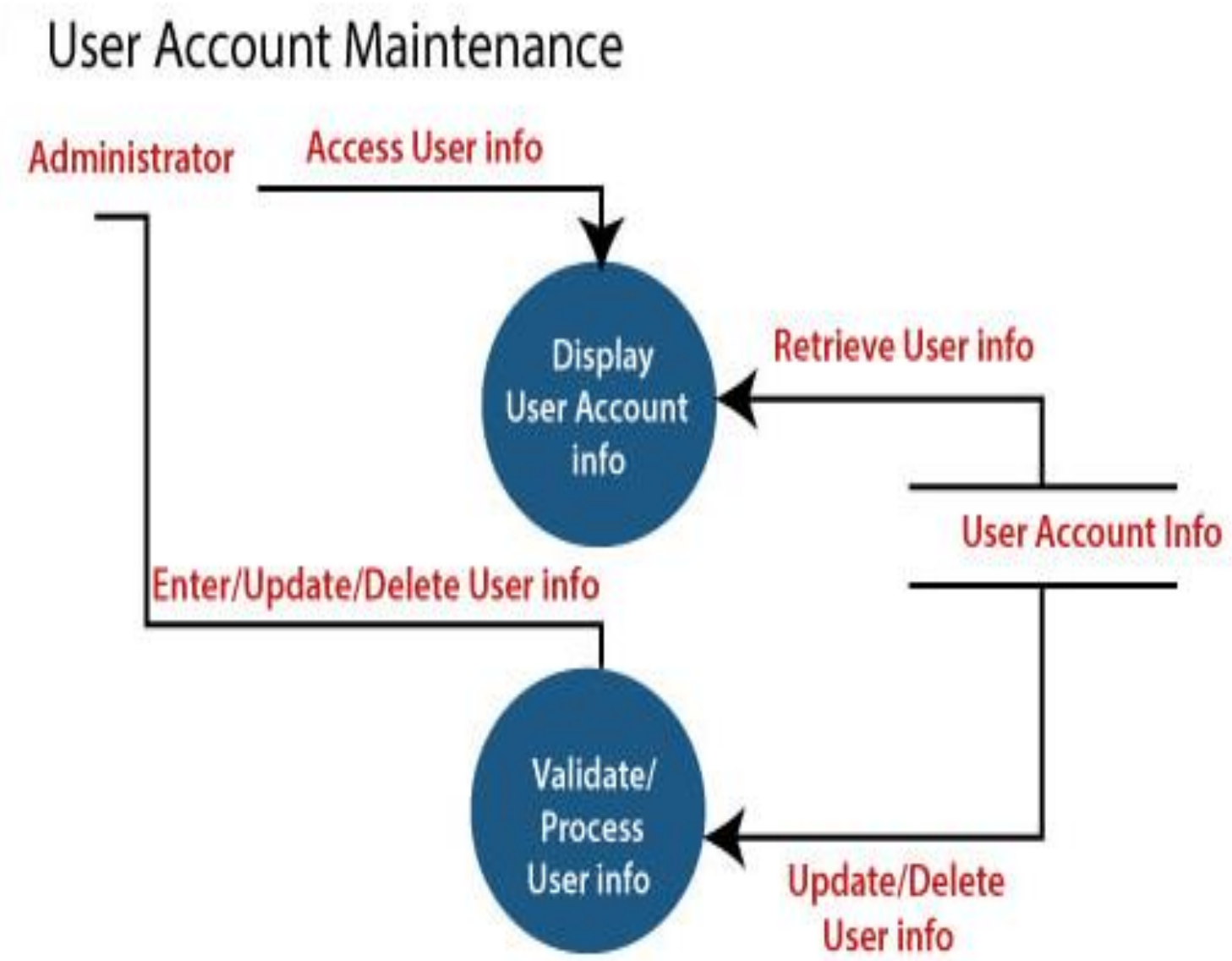

Enter User id, Password,role
Marks details
Coordinator
Marks entry
Marks entry clerk
Marks infomanage -ment
Student performance reports
View report
Marksheets
Report generation
Student details
Student subject choice details
Student info entry
Student info management
Data entry operator
Student sub-choice 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

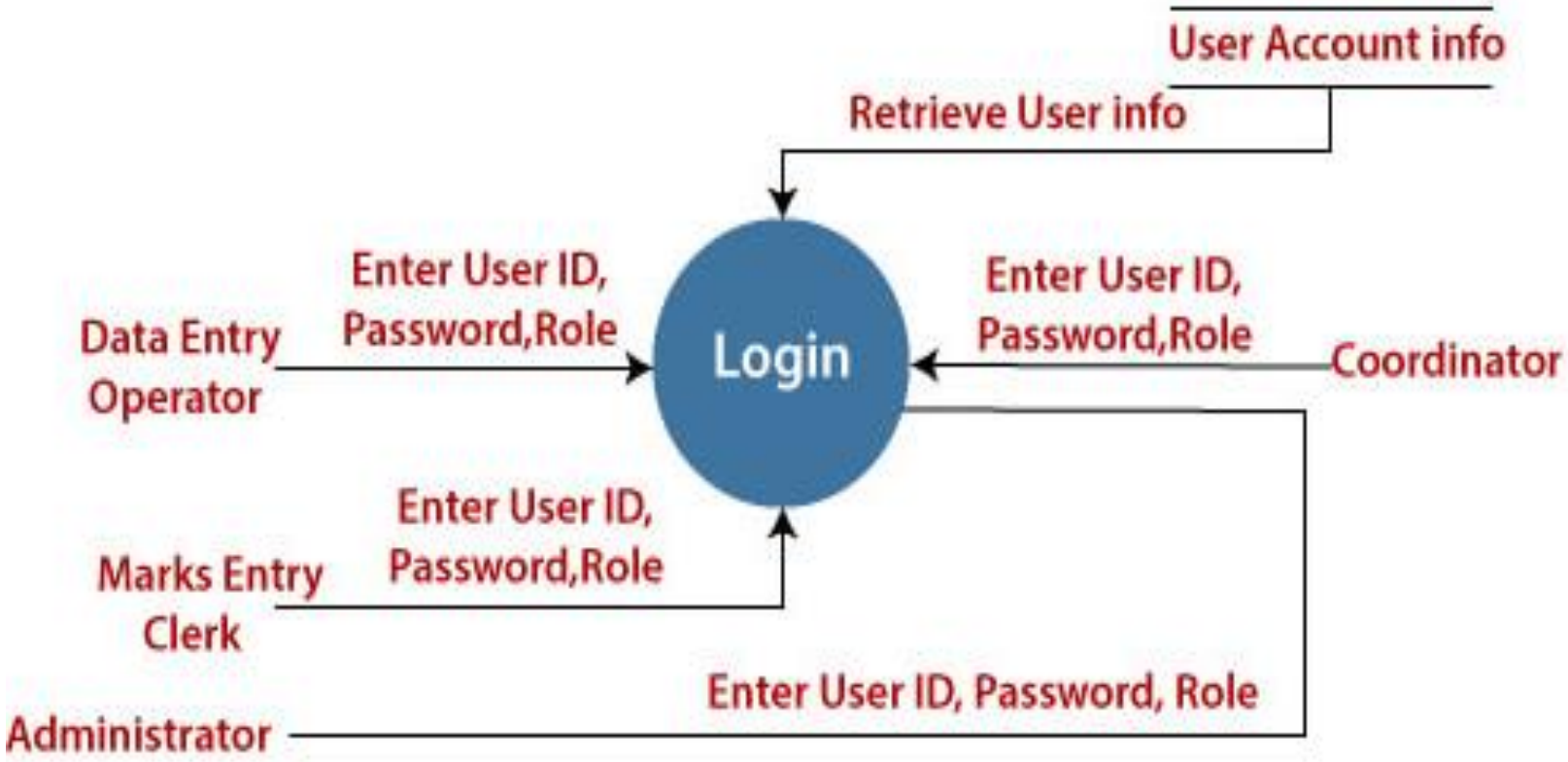
2-Level DFD

- 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:



GLA UNIVERSITY
MATHURA
Recognised by UGC Under Section 2(f)

Subject Name Software Engineering

Requirement Documentation

Requirement Documentation

- Important activity after requirement gathering.
- It is also known as Software Requirement Specification (SRS).
- It serves number of purposes.
- Increase customer satisfaction level.

Nature of SRS

- Functionality
- External Interfaces
- Performance
- Attributes
- Design constraints imposed on an implementation

Characteristics of a good SRS

The SRS should be:

- Correct
- Unambiguous
- Complete
- Consistent

Characteristics of a good SRS

- Ranked for importance and/or stability
- Verifiable
- Modifiable
- Traceable

Organization of SRS

IEEE has published guidelines & standards to organize an SRS document.

Different projects may require their requirements to be organized differently.

Cont……

In order to conduct a successful software project, we must understand:

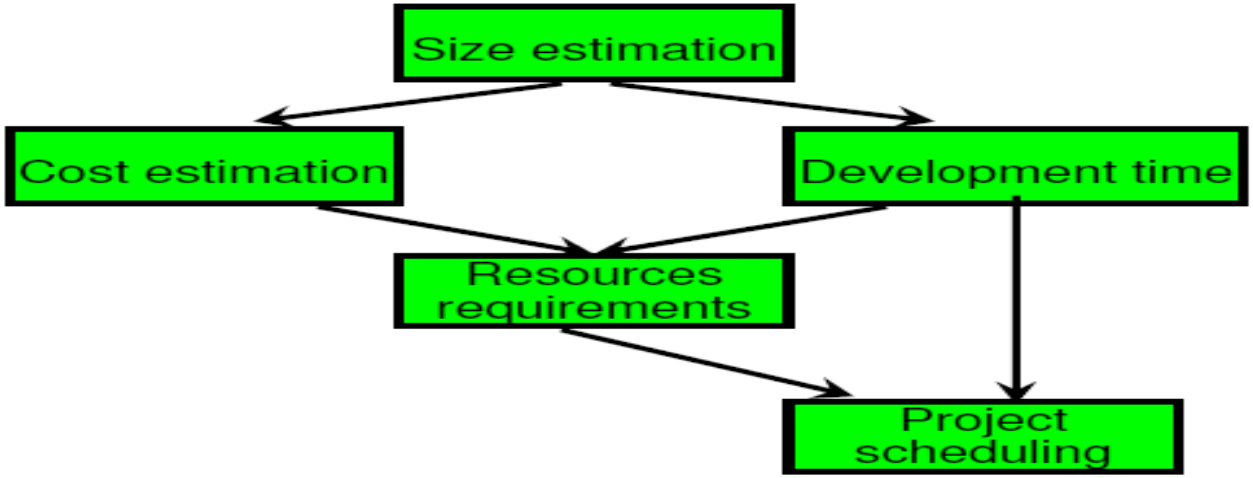
- Scope of work to be done
- The risk to be incurred
- The resources required
- The task to be accomplished
- The cost to be expended
- The schedule to be followed

Software Project Planning

After the finalization of SRS, we would like to estimate size, cost and development time of the project. Also, in many cases, customer may like to know the cost and development time even prior to finalization of the SRS.

Cont….

Software planning begins before technical work starts, continues as the software evolves from concept to reality, and culminates only when the software is retired.



Activities during Software Project Planning

Project size estimation techniques

- Estimation of the size of software is an essential part of Software Project Management. It helps the project manager to further predict the effort and time which will be needed to build the project. Various measures are used in project size estimation. Some of these are:
- Lines of Code
- Number of entities in ER diagram
- Total number of processes in detailed data flow diagram
- Function points

Lines of Code (LOC)

If LOC is simply a count of the number of lines then figure shown below contains 18 LOC .

"A line of code is any line of program text that is not a comment or blank line, regardless of the number of statements or fragments of statements on the line. This specifically includes all lines containing program header, declaration, and executable and non-executable statements".

This is the predominant definition for lines of code used by researchers. By this definition, figure shown above has 17 LOC.

```
1.  int. sort (int x[ ], int n)
2.  {
3.  int i, j, save, im1;
4.  /*This function sorts array x in ascending order */
5.  If (n<2) return 1;
6.  for (i=2; i<=n; i++)
7.  {
8.  im1=i-1;
9.  for (j=1; j<=im; j++)
10. if (x[i] < x[j])
11. {
12. Save = x[i];
13. x[i] = x[j];
14. x[j] = save;
15. }
16. }
17. return 0;
18. }
```

Function Count

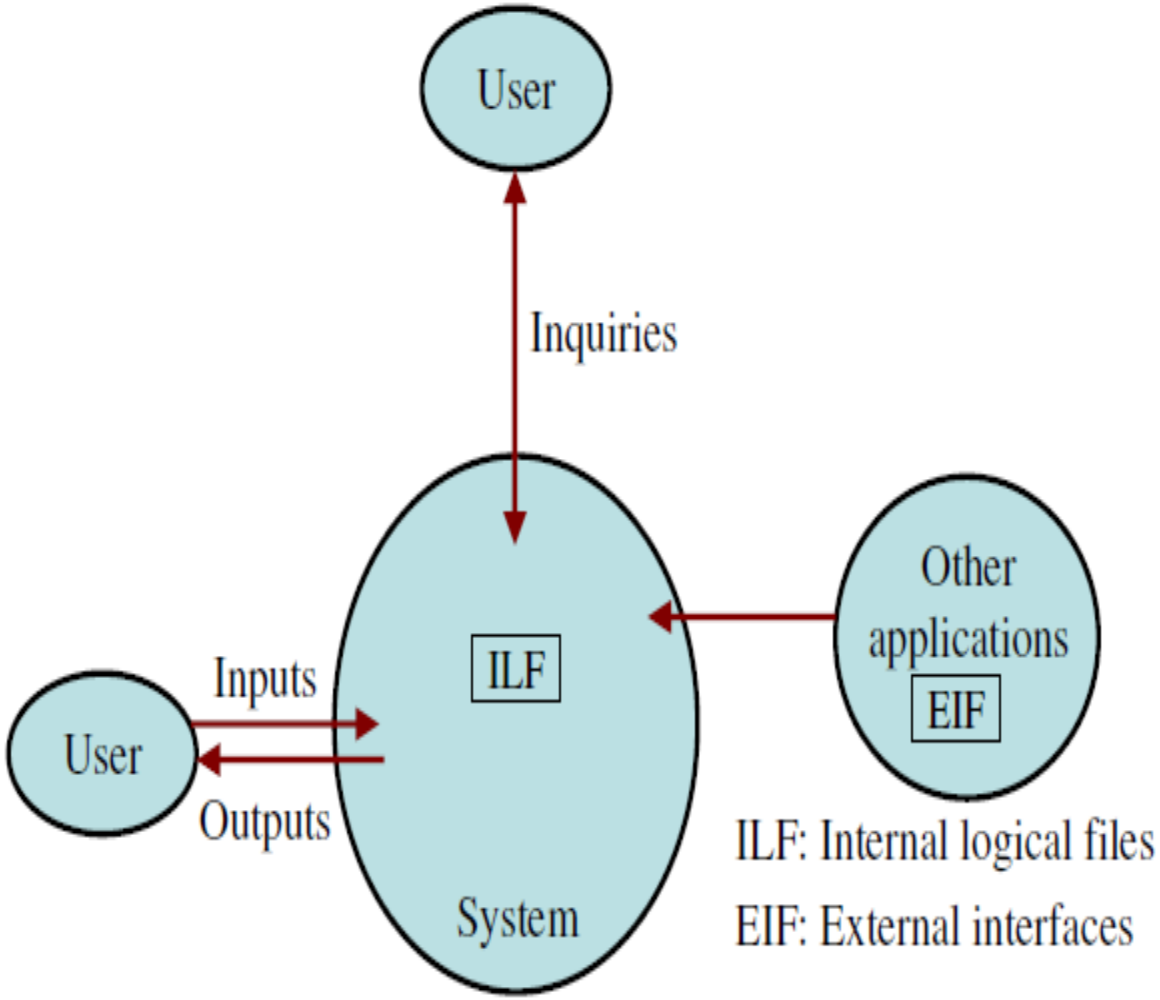
Alan Albrecht while working for IBM, recognized the problem in size measurement in the 1970s, and developed a technique (which he called Function Point Analysis), which appeared to be a solution to the size measurement problem.

Software Project Planning

The principle of Albrecht's function point analysis (FPA) is that a system is decomposed into functional units.

▪ Inputs	:	information entering the system
▪ Outputs	:	information leaving the system
▪ Enquiries	:	requests for instant access to information
▪ Internal logical files	:	information held within the system
▪ External interface files	:	information held by other system that is used by the system being analyzed.

The FPA functional units are shown in figure given below:



FPAs functional units System

The five functional units are divided in two categories:

(i) Data function types

- Internal Logical Files (ILF): A user identifiable group of logical related data or control information maintained within the system.
- External Interface files (EIF): A user identifiable group of logically related data or control information referenced by the system, but maintained within another system. This means that EIF counted for one system, may be an ILF in another system.

(ii) Transactional function types

- External Input (EI): An EI processes data or control information that comes from outside the system. The EI is an elementary process, which is the smallest unit of activity that is meaningful to the end user in the business.
- External Output (EO): An EO is an elementary process that generate data or control information to be sent outside the system.
- External Inquiry (EQ): An EQ is an elementary process that is made up to an input-output combination that results in data retrieval.

Special features

- Function point approach is independent of the language, tools, or methodologies used for implementation; i.e. they do not take into consideration programming languages, data base management systems, processing hardware or any other data base technology.
- Function points can be estimated from requirement specification or design specification, thus making it possible to estimate development efforts in early phases of development.

- Function points are directly linked to the statement of requirements; any change of requirements can easily be followed by a re-estimate.
- Function points are based on the system user's external view of the system, non-technical users of the software system have a better understanding of what function points are measuring.

Calculation of Function Point

Calculate Function Point.FP = UFP * CAF

Where,UFP=unadjusted function point

Complexity Adjustment Factor (CAF).

CAF = 0.65 + (0.01 * F)

Where F=14*Scale, and scale can be calculated

0 - No Influence

1 - Incidental

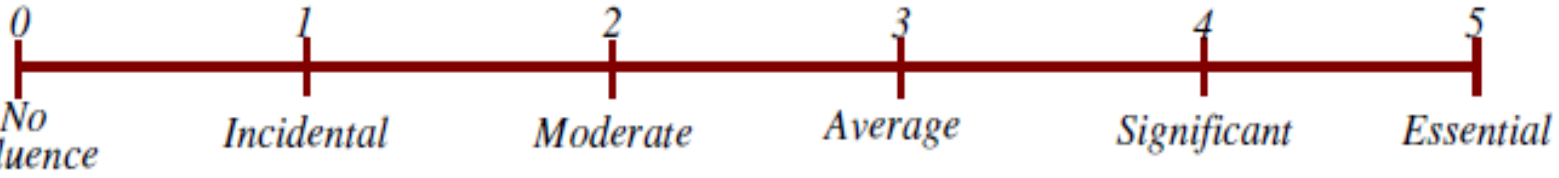
2 - Moderate

3 - Average

4 - Significant

5 - Essential

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 ?

Calculate Unadjusted Function Point (UFP).

FUNCTION UNITS	LOW	AVG	HIGH
EI	3	4	6
EO	4	5	7
EQ	3	4	6
ILF	7	10	15
EIF	5	7	10

Multiply each individual function point to corresponding values in TABLE.

Ex-1(Calculation of Functional Point)

- Given the following values, compute function point when all complexity adjustment factor (CAF) and weighting factors are average.

- User Input = 50 User Output = 40 User Inquiries = 35 User Files = 6 External Interface = 4

- **Step-1:** As complexity adjustment factor is average (given in question), hence, scale = 3.
- F = 14 * 3 = 42
- **Step-2:**CAF = 0.65 + (0.01 * 42) = 1.07
- **Step-3:** As weighting factors are also average (given in question) hence we will multiply each individual function point to corresponding values in TABLE.
- UFP = (50*4) + (40*5) + (35*4) + (6*10) + (4*7) = 628
- **Step-4:**Function Point = 628 * 1.07 = 671.96

Software Project Planning

An application has the following:

10 low external inputs, 12 high external outputs, 20 low internal logical files, 15 high external interface files, 12 average external inquiries, and a value of complexity adjustment factor of 1.10.

What are the unadjusted and adjusted function point counts ?

Solution

Unadjusted function point counts may be calculated using as:

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

= 10 x 3 + 12 x 7 + 20 x 7 + 15 + 10 + 12 x 4

= 30 + 84 +140 + 150 + 48

= 452

FP = UFP x CAF

= 452 x 1.10 = 497.2.

Software Project Planning

Consider a project with the following parameters.

(*i*) External Inputs:
- (a) 10 with low complexity
- (b) 15 with average complexity
- (c) 17 with high complexity

(*ii*) External Outputs:
- (a) 6 with low complexity
- (b) 13 with high complexity

(*iii*) External Inquiries:
- (a) 3 with low complexity
- (b) 4 with average complexity
- (c) 2 high complexity

Software Project Planning

(*iv*) Internal logical files:

(a) 2 with average complexity

(b) 1 with high complexity

(v) External Interface files:

(a) 9 with low complexity

In addition to above, system requires

i. Significant data communication

ii. Performance is very critical

iii. Designed code may be moderately reusable

iv. System is not designed for multiple installation in different organizations.

Other complexity adjustment factors are treated as average. Compute the function points for the project.

Software Project Planning

Solution: Unadjusted function points may be counted using table 2

Functional Units	Count	Complexity		Complexity Totals	Functional Unit Totals
External Inputs (EIs)	10	Low x 3	=	30	
	15	Average x 4	=	60	
	17	High x 6	=	102	192
External Outputs (EOs)	6	Low x 4	=	24	
	0	Average x 5	=	0	
	13	High x 7	=	91	115
External Inquiries (EQs)	3	Low x 3	=	9	
	4	Average x 4	=	16	
	2	High x 6	=	12	37
External logical Files (ILFs)	0	Low x 7	=	0	
	2	Average x 10	=	20	
	1	High x 15	=	15	35
External Interface Files (EIFs)	9	Low x 5	=	45	
	0	Average x 7	=	0	
	0	High x 10	=	0	45
Total Unadjusted Function Point Count					424

Software Project Planning

$$\sum_{i=1}^{14} F_i = 3+4+3+5+3+3+3+3+3+3+2+3+0+3=41$$

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

= $(0.65 + 0.01 \times 41)$

= 1.06

FP = UFP x CAF

= 424 x 1.06

= 449.44

Hence **FP = 449**

Software Project Planning

Language	SLOC/UFP
Ada	71
AI Shell	49
APL	32
Assembly	320
Assembly (Macro)	213
ANSI/Quick/Turbo Basic	64
Basic-Compiled	91
Basic-Interpreted	128
C	128
C++	29

Converting function points to lines of code

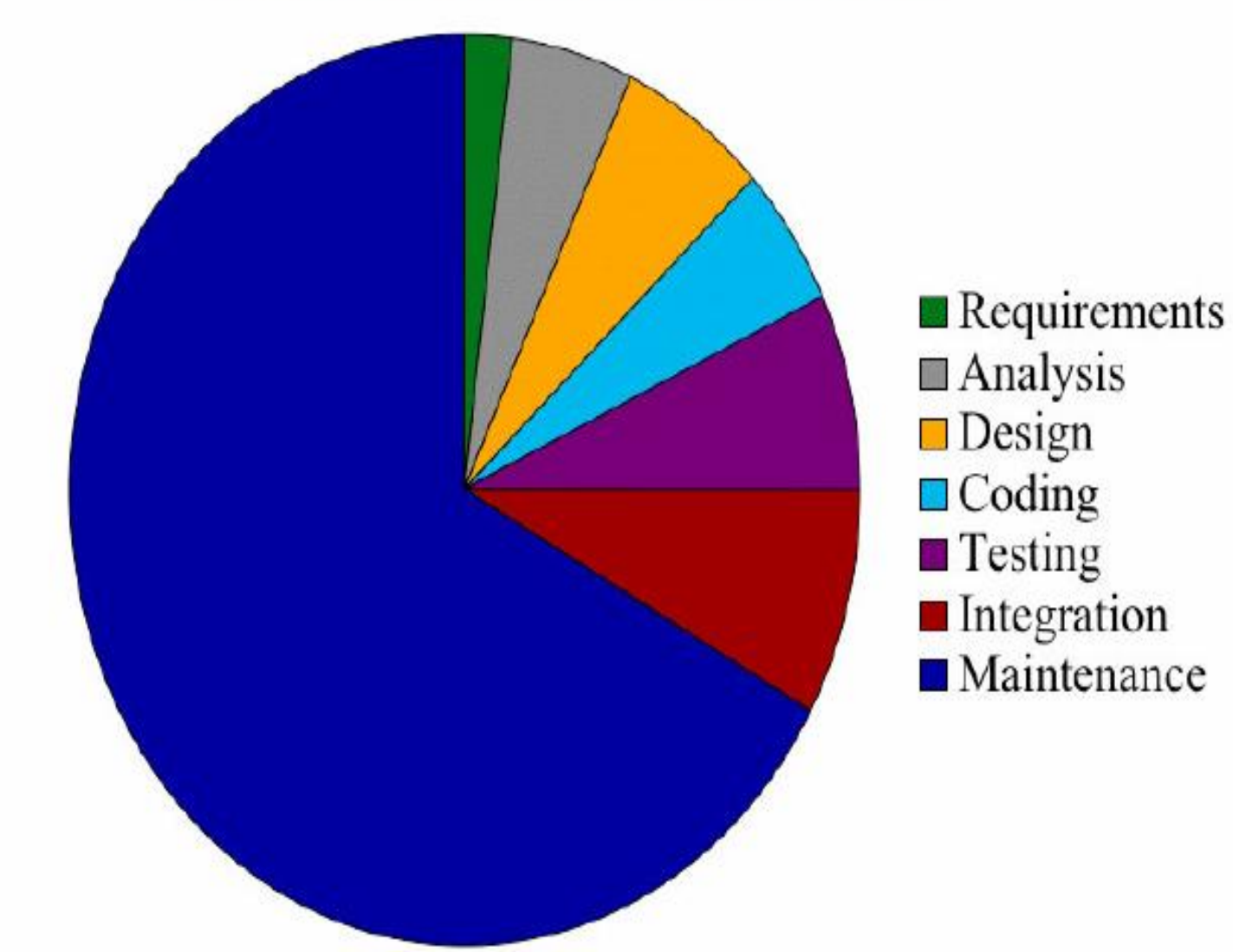
Software Project Planning

Language	SLOC/UFP
ANSI Cobol 85	91
Fortan 77	105
Forth	64
Jovial	105
Lisp	64
Modula 2	80
Pascal	91
Prolog	64
Report Generator	80
Spreadsheet	6

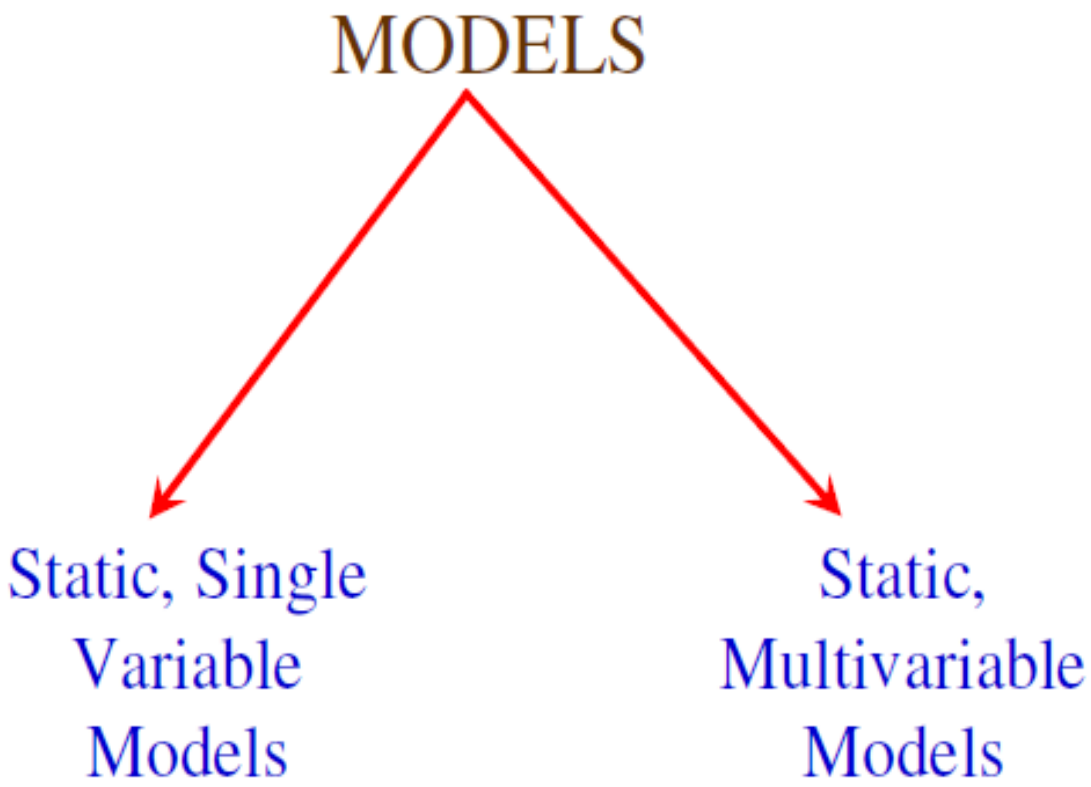
Converting function points to lines of code

Software Project Planning

Relative Cost of Software Phases



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.

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

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

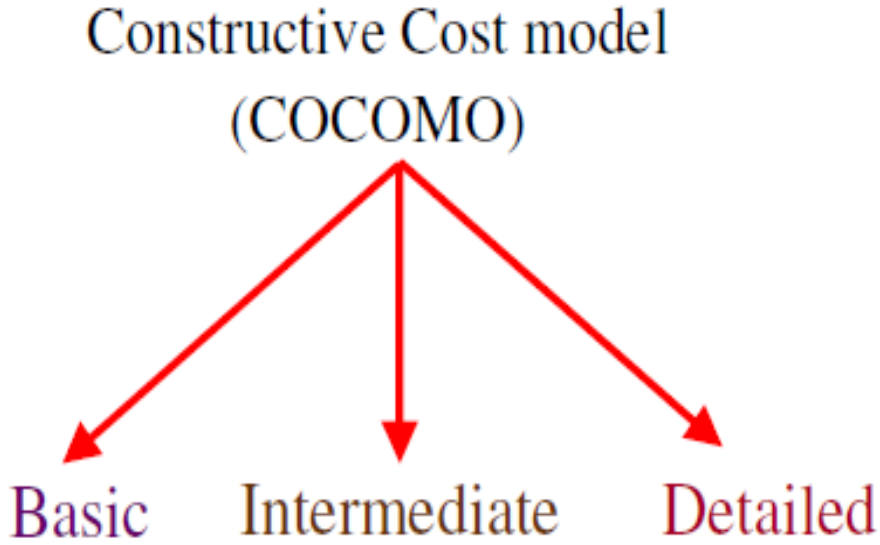
The productivity index uses 29 variables which are found to be highly correlated to productivity as follows:

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

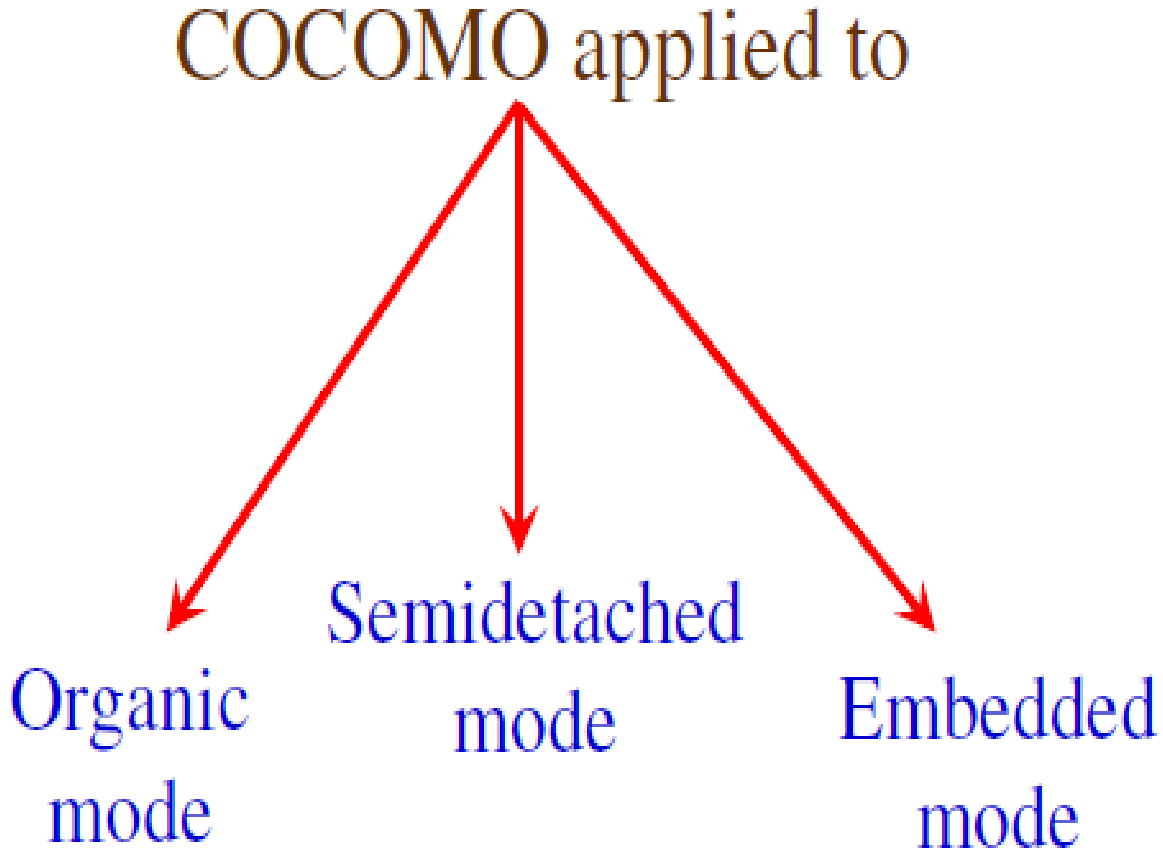
Where $\mathbf{W_i}$ is the weight factor for the $\mathbf{i^{th}}$variable and $\mathbf{X_i}$**={-1,0,+1}** the estimator gives $\mathbf{X_i}$one of the values **-1, 0 or +1** depending on the variable decreases, has no effect or increases the productivity.

WALSTON and FELIX develop the models at IBM.

The Constructive Cost Model (COCOMO)



Model proposed by
B. W. Boehm's
through his book
Software Engineering Economics in 1981


COCOMO applied to
Organic mode
Semidetached mode
Embedded mode

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

Types of Models:
COCOMO consists of a hierarchy of three increasingly detailed and accurate forms.

Any of the three forms can be adopted according to our requirements.

These are types of COCOMO model:

- Basic COCOMO Model
- Intermediate COCOMO Model
- Detailed COCOMO Model

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. a_b, b_b, c_b and d_b are The coefficients

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

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:

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

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.

Solution

The basic COCOMO equation take the form:

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

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

Estimated size of the project = 400 KLOC

(i) Organic mode

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

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

(ii) Semidetached mode

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

$$D = 2.5(2462.79)^{0.35} = 38.45 \text{ PM}$$

(iii) Embedded mode

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

$$D = 2.5(4772.8)^{0.32} = 38 \text{ PM}$$

Example

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.

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

$D = 2.5(1133.12)^{0.35} = 29.3$ PM

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

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

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

$$P = 176\, LOC / PM$$

Intermediate Model

The basic COCOMO model considers that the effort is only a function of the number of lines of code and some constants calculated according to the various software systems. The intermediate COCOMO model recognizes these facts and refines the initial estimates obtained through the basic COCOMO model by using a set of 15 cost drivers based on various attributes of software engineering.

Classification of Cost Drivers and their attributes

- **(i) Product attributes -**
- Required software reliability extent
- Size of the application database
- The complexity of the product
- **Hardware attributes -**
- Run-time performance constraints
- Memory constraints
- The volatility of the virtual machine environment
- Required turnabout time

- **Personnel attributes -**
- Analyst capability
- Software engineering capability
- Applications experience
- Virtual machine experience
- Programming language experience
- **Project attributes -**
- Use of software tools
- Application of software engineering methods
- Required development schedule

COST DRIVERS	VERY LOW	LOW	NOMINAL	HIGH	VERY HIGH
Product Attributes					
Required Software Reliability	0.75	0.88	1.00	1.15	1.40
Size of Application Database		0.94	1.00	1.08	1.16
Complexity of The Product	0.70	0.85	1.00	1.15	1.30

COST DRIVERS	VERY LOW	LOW	NOMINAL	HIGH	VERY HIGH
Hardware Attributes					
Runtime Performance Constraints			1.00	1.11	1.30
Memory Constraints			1.00	1.06	1.21
Volatility of the virtual machine environment		0.87	1.00	1.15	1.30
Required turnabout time		0.94	1.00	1.07	1.15

COST DRIVERS	VERY LOW	LOW	NOMINAL	HIGH	VERY HIGH
Personnel attributes					
Analyst capability	1.46	1.19	1.00	0.86	0.71
Applications experience	1.29	1.13	1.00	0.91	0.82
Software engineer capability	1.42	1.17	1.00	0.86	0.70
Virtual machine experience	1.21	1.10	1.00	0.90	
Programming language experience	1.14	1.07	1.00	0.95	

COST DRIVERS	VERY LOW	LOW	NOMINAL	HIGH	VERY HIGH
Project Attributes					
Application of software engineering methods	1.24	1.10	1.00	0.91	0.82
Use of software tools	1.24	1.10	1.00	0.91	0.83
Required developme nt schedule	1.23	1.08	1.00	1.04	1.10

Intermediate COCOMO equations

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

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

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

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	..

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	..

Detailed COCOMO Model

Detailed COCOMO incorporates all characteristics of the intermediate version with an assessment of the cost driver's impact on each step of the software engineering process. The detailed model uses different effort multipliers for each cost driver attribute. In detailed cocomo, the whole software is divided into different modules and then we apply COCOMO in different modules to estimate effort and then sum the effort.

Development Phase

Plan / Requirements

EFFORT : 6% to 8%

DEVELOPMENT TIME : 10% to 40%

% depend on mode & size

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%

Principle of the effort estimate

Size equivalent

As the software might be partly developed from software already existing (that is, re-usable code), a full development is not always required. In such cases, the parts of design document (DD%), code (C%) and integration (I%) to be modified are estimated. Then, an adjustment factor, A, is calculated by means of the following equation.

$$A = 0.4\,DD + 0.3\,C + 0.3\,I$$

The size equivalent is obtained by

$$S\,(\text{equivalent}) = (S \times A) / 100$$

$$E_p = \mu_p E$$

$$D_p = \tau_p D$$

Lifecycle Phase Values of μ_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

Lifecycle Phase Values of τ_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

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 ?

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

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

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

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

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

$$EAF = 1.29 \times 0.95 = 1.22$$

$$E = 3712 \times 1.22 = 4528 \text{ PM}$$

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

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.

Problem

- 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.

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**

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

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

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

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

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

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

(b) Using the following equations and referring Table 7, phase wise cost and schedule estimates can be calculated.

$$E_p = \mu_p E$$

$$D_p = \tau_p D$$

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

GLA UNIVERSITY
MATHURA
Recognised by UGC Under Section 2(f)

Topic: Function Oriented Design

Content

- Software Design
- Problem Partitioning
- Abstraction
- Strategy of Design
  - ❖ Bottom Up Design
  - ❖ Top Down Design
  - ❖ Hybrid Approach

Software Design

- Software Design is more creative process than analysis as it deals with the development of the actual mechanics for a new workable system.
- Design is a problem solving activity.
- Designers with users has to search for a solution until a satisfactory solution has been found.
- Design is a phase where designers plans “How” a software system should be produced.

Problem Partitioning

- For solving larger problems, the basic principle is “Divide & Conquer”.
- The idea behind this principle is to divide the problems into smaller & manageably pieces, so that each piece can be conquered separately.
- These different pieces can not be entirely independent of each other, as they together form the system.
- These different pieces have to cooperate & communicate to solve the larger problem.
- The designer has to make the judgement about when to stop partitioning.

Abstraction

- An abstraction of a component describes the external behavior of that component.
- Abstraction is an indispensable part of the design process & is essential for problem partitioning.
- Abstraction is used for existing components as well as the components that are being designed.
- During the design process, abstractions are used in the reverse manner than in the process of understanding a system.
- There are two common abstraction mechanism for software system:
  - ❖ Functional abstraction
  - ❖ Data abstraction

Strategy of Design

- A good system design strategy is to organize the program modules in such a way that are easy to develop and latter to, change.

- Structured design techniques help developers to deal with the size and complexity of programs.

- Analysts create instructions for the developers about how code should be written and how pieces of code should fit together to form a program.

Strategy of Design

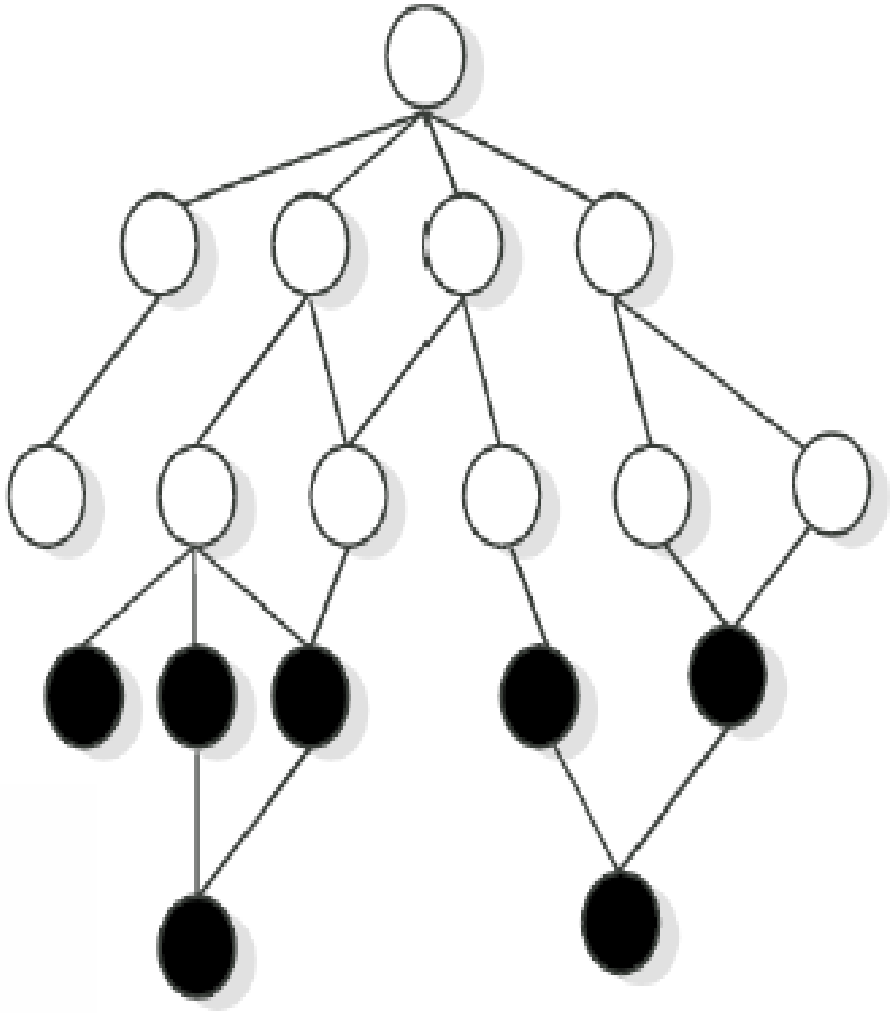
There are many strategies or techniques for performing system design. They include:

- Bottom Up Design
- Top Down Design &
- Hybrid Approach

Bottom Up Design

- The bottom up design model starts with most specific and basic components.
- It proceeds with composing higher level of components by using basic or lower level components.
- It keeps creating higher level components until the desired system is not evolved as one single component.
- With each higher level, the amount of abstraction is increased.
- Bottom-up strategy is more suitable when a system needs to be created from some existing system, where the basic primitives can be used in the newer system.

Bottom Up Design



Top Down Design

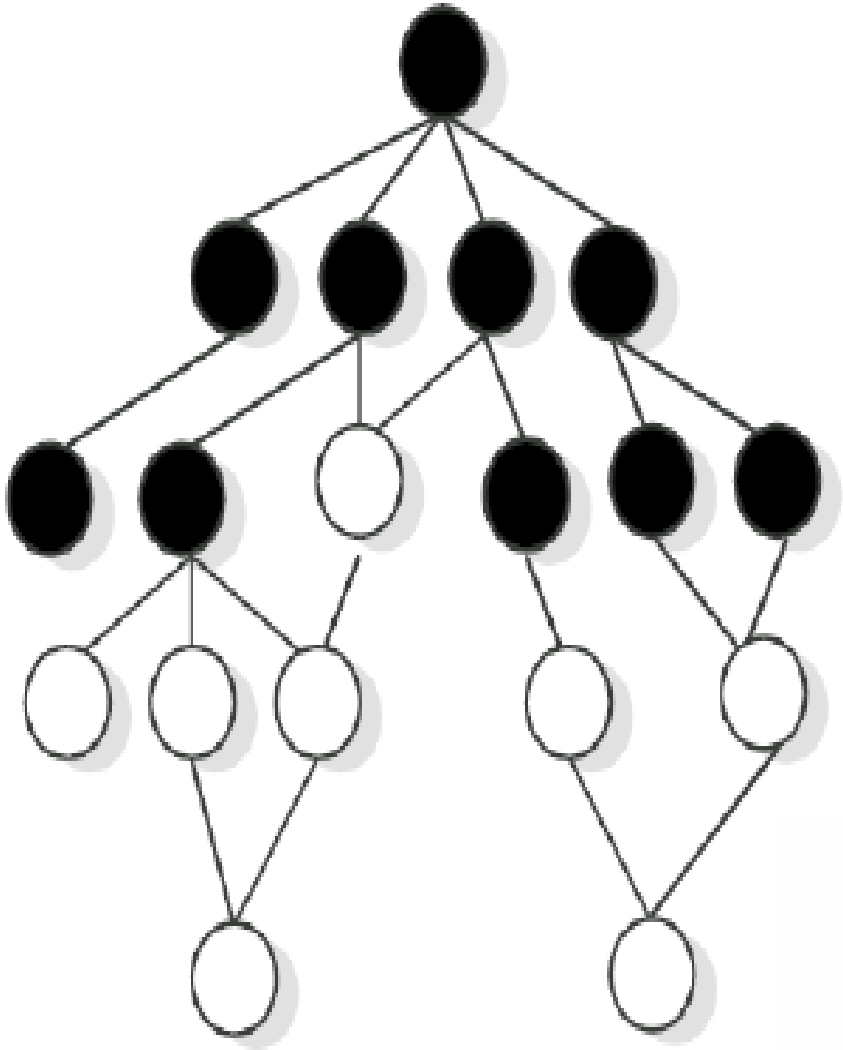
- Top-down design takes the whole software system as one entity and then decomposes it to achieve more than one sub-system or component based on some characteristics.

- Each sub-system or component is then treated as a system and decomposed further.

- This process keeps on running until the lowest level of system in the top-down hierarchy is achieved.

- Top-down design is more suitable when the software solution needs to be designed from scratch and specific details are unknown.

Top Down Design

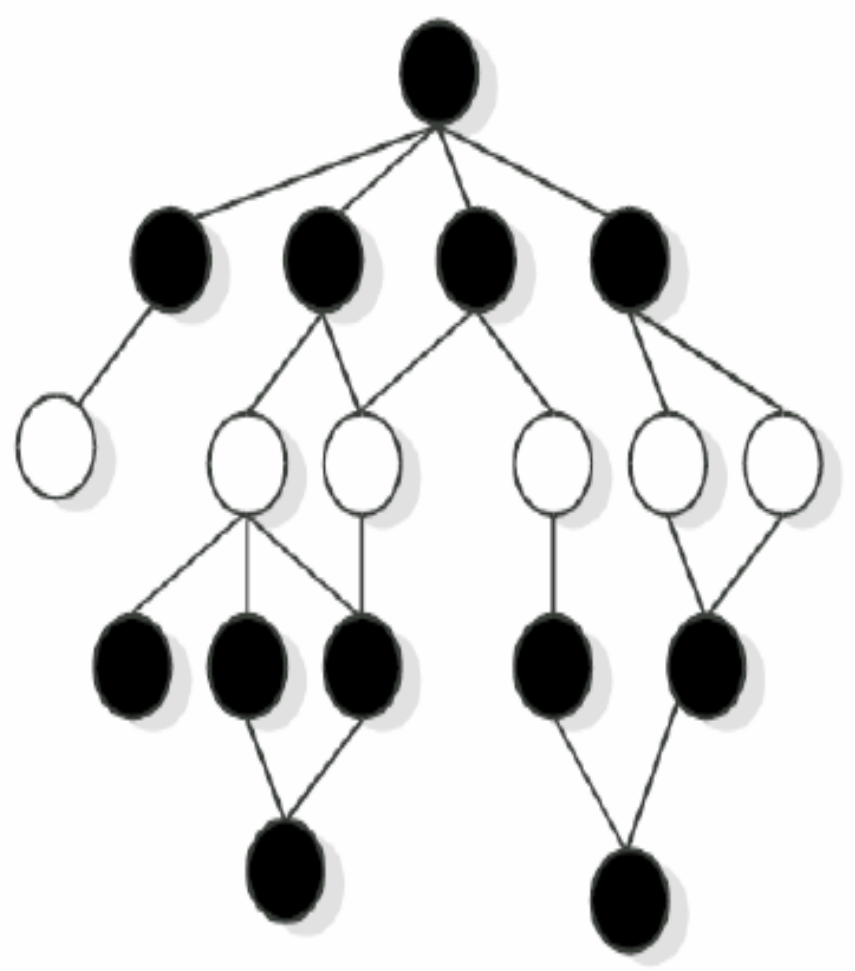


GLA UNIVERSITY
MATHURA

Hybrid Approach

- Pure top-down or pure bottom-up approaches are often not practical.
- For a bottom-up approach to be successful, we must have a good notion of the top to which the design should be heading.
- Hybrid approach is a combination of both the top – down and bottom – up design strategies. In this we can reuse the modules.

Hybrid Approach



References

- K.K. Aggarwal and Yogesh Singh, “Software Engineering”, Third Ed., New Age International (P) Limited Publishers. (2009).
- Pankaj Jalote, “An Integrated Approach to Software Engineering”, Third Ed., Narose Publishing House, (2008).
- Rajib Mall, “Fundamentals of Software Engineering”, Fourth Ed., PHI Learning Private Limited, (2016).
- Roger S. Pressman, “Software Engineering: A Practitioner’s Approach”, Fifth Ed., McGraw–Hill Higher Education, (2001).

Thank You !!!

GLA UNIVERSITY
MATHURA
Recognised by UGC Under Section 2(f)

Topic: Module Level Concepts: Co on

Topics to be covered

- Module
- Modularity
- Module Coupling
- Module Cohesion &
- Relationship between Coupling & Cohesion

Module

- A module is a logically separable part of a program.
- It is a program unit that is discrete & identifiable with respect to compiling & loading.
- In programming terminology, a module can be a macro, a function, a procedure (or subroutine), a process, or a package.
- In systems using functional abstraction, a module is usually a procedure of function.

Modularization

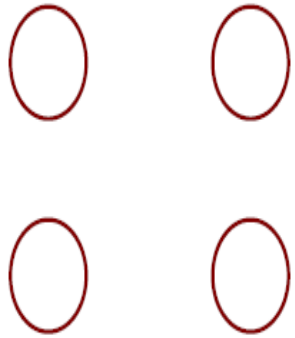
- Modularization is the process of dividing a software system into multiple independent modules where each module works independently.

- There are many advantages of Modularization in software engineering. Some of these are given below:

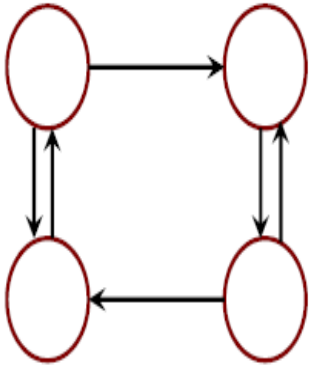
  - ❖ Easy to understand the system.
  - ❖ System maintenance is easy.
  - ❖ A module can be used many times as their requirements.

Module Coupling

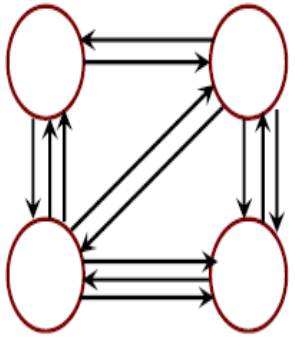
- Coupling is the measure of the degree of interdependence between the modules.
- A good software will have low coupling.



(Uncoupled : no dependencies)



Loosely coupled:
some dependencies

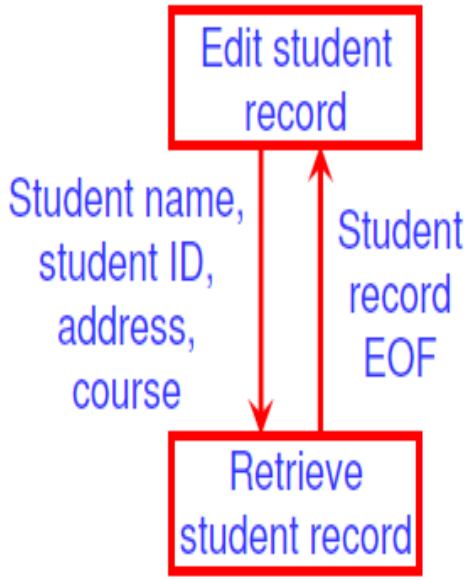


Highly coupled:
many dependencies

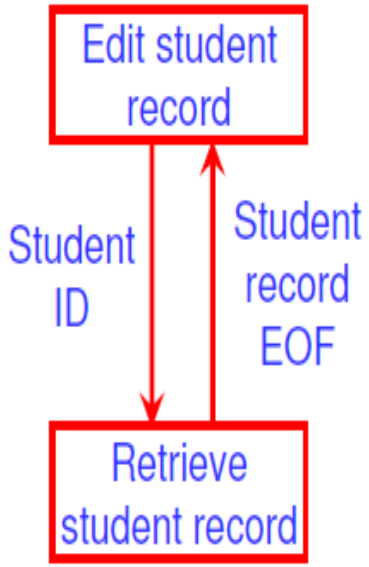
Module Coupling

For Example,

Consider a case of editing a student record in a "Student Information System".



Poor design: Tight Coupling



Good design: Loose Coupling

Module Coupling: Types

Module Coupling can be classified into 6 different types as follows:

Data coupling	Best
Stamp coupling	
Control coupling	
External coupling	
Common coupling	
Content coupling	Worst

Given two procedures A & B, we can identify number of ways in which they can be coupled.

Module Coupling

- Data Coupling:

The dependency between module A and B is said to be data coupled if their dependency is based on the fact they communicate by only passing of data.

- Stamp Coupling:

Stamp coupling occurs between module A and B when complete data structure is passed from one module to another.

Module Coupling

- Control Coupling:

Module A & B are said to be control coupled if they communicate by passing of control information. This is usually accomplished by means of flags that are set by one module and reacted upon by the dependent module.

- External Coupling:

In external coupling, the modules depend on other modules, external to the software being developed or to a particular type of hardware. Ex- protocol, external file, device format, etc.

Module Coupling

- Common Coupling:

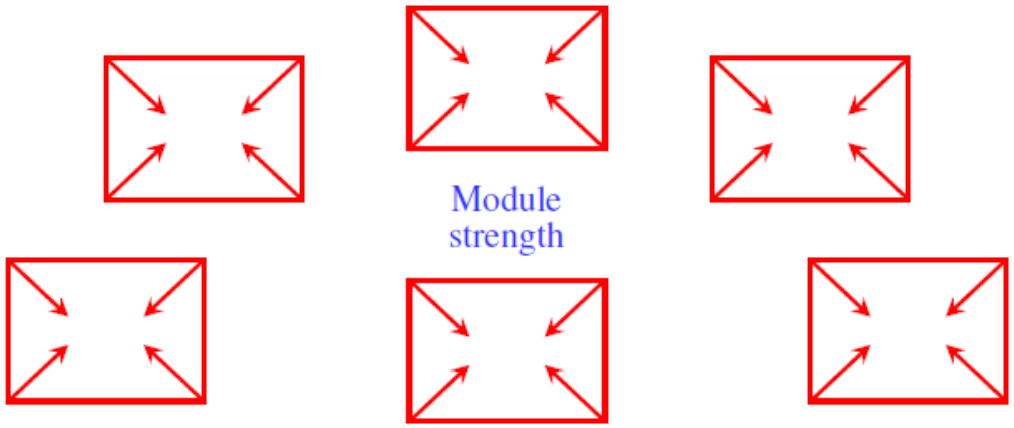
With common coupling, module A & B have shared data. Global data areas are commonly found in programming languages. Making a change to the common data means tracing back to all the modules which access that data to evaluate the effect of changes.

- Content Coupling:

In a content coupling, one module can modify the data of another module or control flow is passed from one module to the other module. This is the worst form of coupling and should be avoided.

Module Cohesion

- Cohesion is a measure of the degree to which the elements of the module are functionally related.
- A good software design will have high cohesion.



Module Cohesion: Types

Module Cohesion can be classified into 7 different types as follows:

Functional Cohesion	Best (high)
Sequential Cohesion	
Communicational Cohesion	
Procedural Cohesion	
Temporal Cohesion	
Logical Cohesion	
Coincidental Cohesion	Worst (low)

Module Cohesion

- Functional Cohesion:

Every essential element for a single computation is contained in the component. A functional cohesion performs the task and functions. It is an ideal situation.

- Sequential Cohesion:

An element outputs some data that becomes the input for other element, i.e., data flow between the parts. It occurs naturally in functional programming languages.

Module Cohesion

- Communicational Cohesion:

Two elements operate on the same input data or contribute towards the same output data. Example- update record int the database and send it to the printer.

- Procedural Cohesion:

Elements of procedural cohesion ensure the order of execution. Actions are still weakly connected and unlikely to be reusable. Ex- calculate student GPA, print student record, calculate cumulative GPA, print cumulative GPA.

Module Cohesion

- Temporal Cohesion:

The elements are related by their timing involved. A module connected with temporal cohesion all the tasks must be executed in the same time-span. This cohesion contains the code for initializing all the parts of the system.

- Logical Cohesion:

The elements are logically related and not functionally. Ex- A component reads inputs from tape, disk, and network. All the code for these functions is in the same component. Operations are related, but the functions are significantly different.

Module Cohesion

- Coincidental Cohesion:

The elements are not related(unrelated).The elements have no conceptual relationship other than location in source code. It is accidental and the worst form of cohesion.

Relationship between Cohesion & Coupling

If the software is not properly modularized, a host of seemingly trivial(normal) enhancement or changes will result into death of the project. Therefore, a software engineer must design the modules with goal of high cohesion and low coupling.

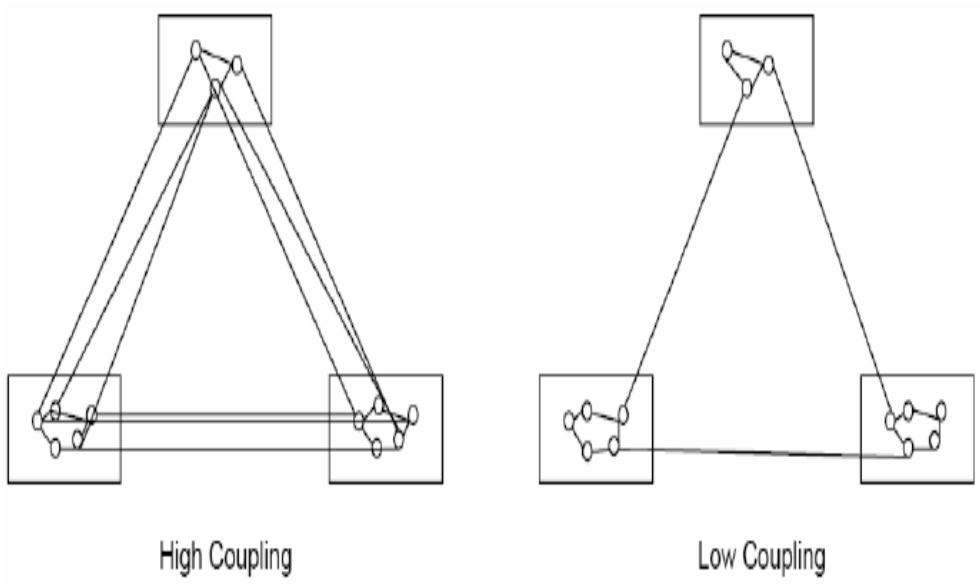


Fig. View of Cohesion & Coupling

References

- K.K. Aggarwal and Yogesh Singh, “Software Engineering”, Third Ed., New Age International (P) Limited Publishers. (2009).
- Pankaj Jalote, “An Integrated Approach to Software Engineering”, Third Ed., Narose Publishing House, (2008).
- Rajib Mall, “Fundamentals of Software Engineering”, Fourth Ed., PHI Learning Private Limited, (2016).
- Roger S. Pressman, “Software Engineering: A Practitioner’s Approach”, Fifth Ed., McGraw–Hill Higher Education, (2001).
- https://www.geeksforgeeks.org/software-engineering-coupling-and-cohesion/last accessed on 28/8/2020.

Topic: Design Notation & Specification – Structure Charts

Design Notations and Specifications- Structure Charts

Design Notations:

Design notations are largely meant to be used during the process of design and are used to represent design or design decisions. For a function oriented design, the design can be represented graphically or mathematically by the following:

- Data flow diagrams
- Data Dictionaries
- Structure Charts
- Pseudocode

Design Notations and Specifications- Structure Charts

- Structure chart breaks down the entire system into lowest functional modules, describes functions and sub-functions of each module of the system to a greater detail than DFD.

- Structure chart represents hierarchical structure of modules. At each layer a specific task is performed.

Design Notations and Specifications- Structure Charts

- Every program has a structure
- Structure Chart - graphic representation of structure
- Structure Chart represents modules
- Each module is represented by a box
- If A invokes(call) B, an arrow is drawn from A to B
- Arrows are labeled by data items
- Different types of modules in a Stru. Chart
- Input, output, transform and coordinate module
- Stru. Chart shows the static structure, not the logic
- Major decisions and loops can be shown
- Structure is decided during design
- Implementation does not change structure
- Structure effects maintainability

Design Notations and Specifications- Structure Charts

Structure Chart

It partition a system into block boxes. A black box means that functionality is known to the user without the knowledge of internal design.

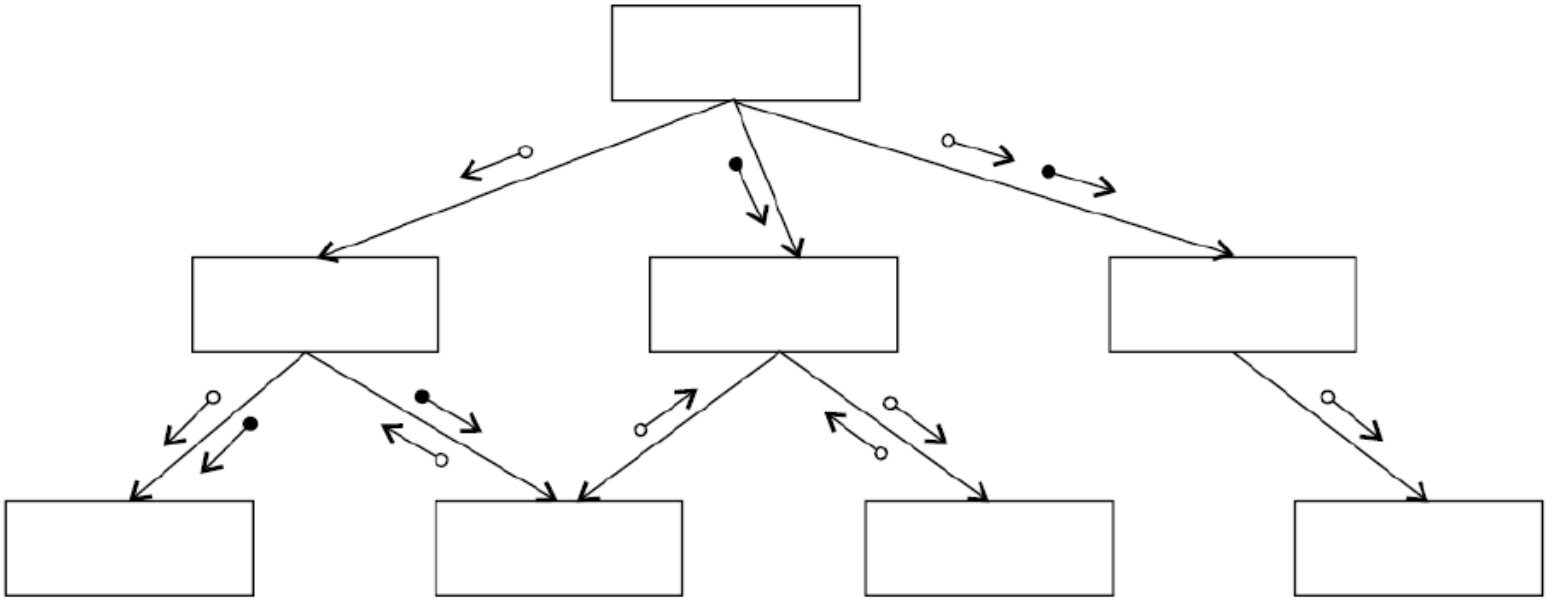


Fig. 16 : Hierarchical format of a structure chart

Design Notations and Specifications- Structure Charts

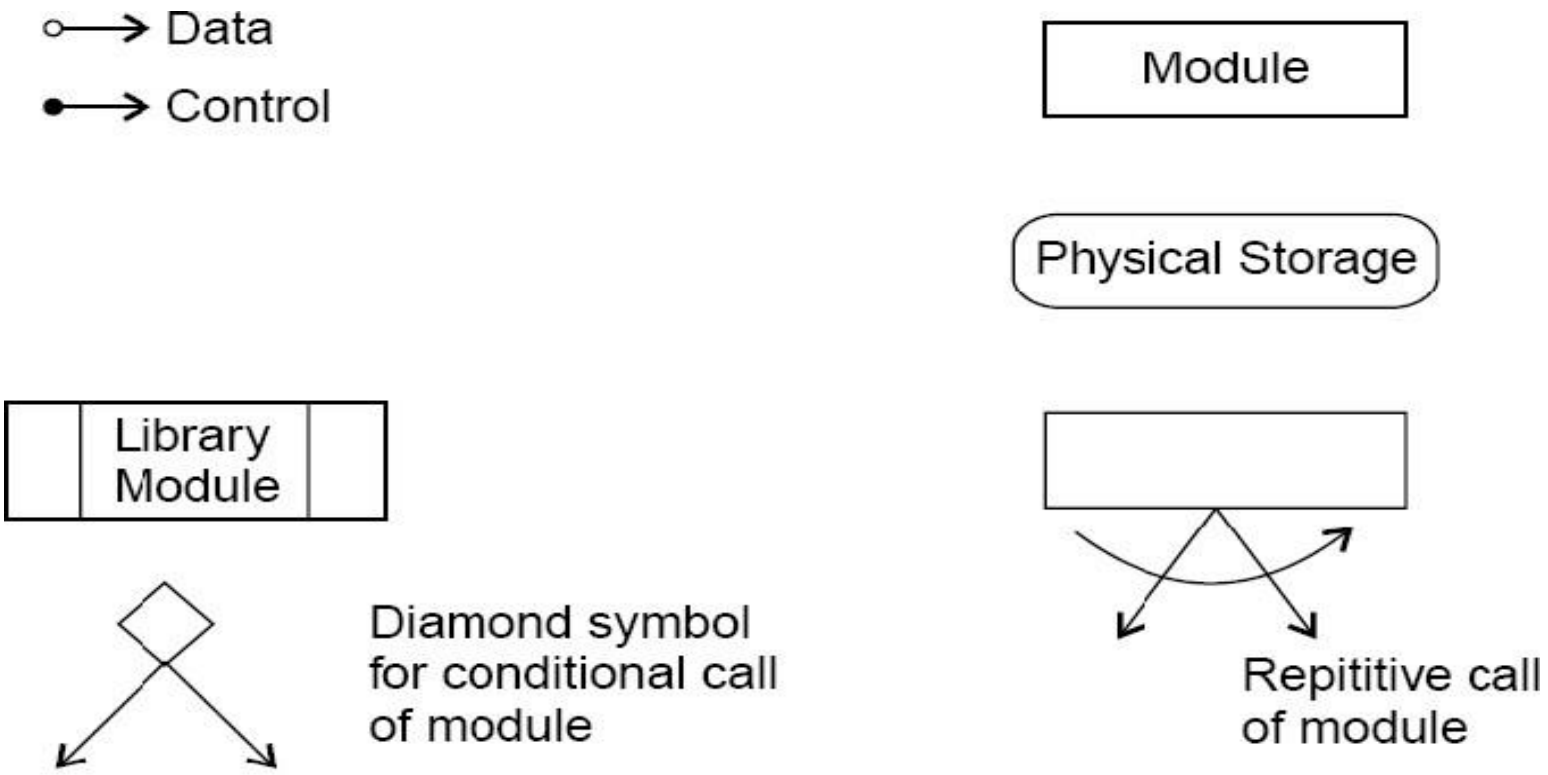
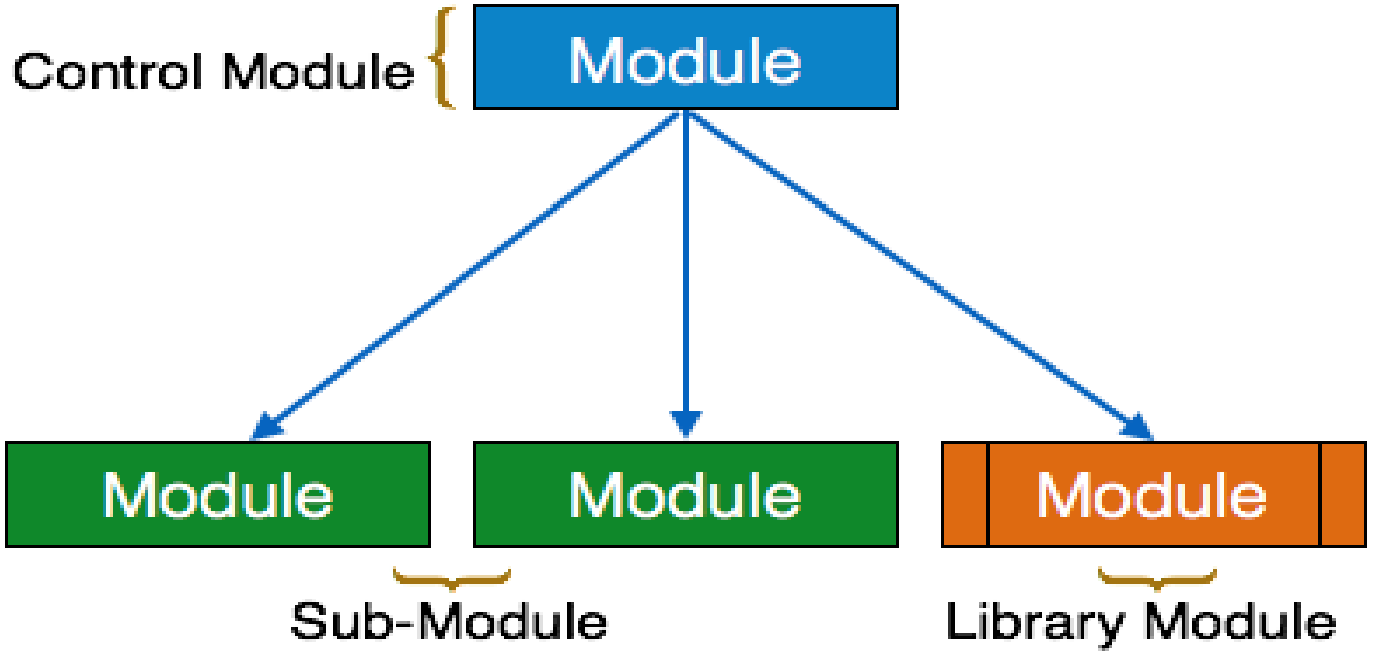


Fig. 17 : Structure chart notations

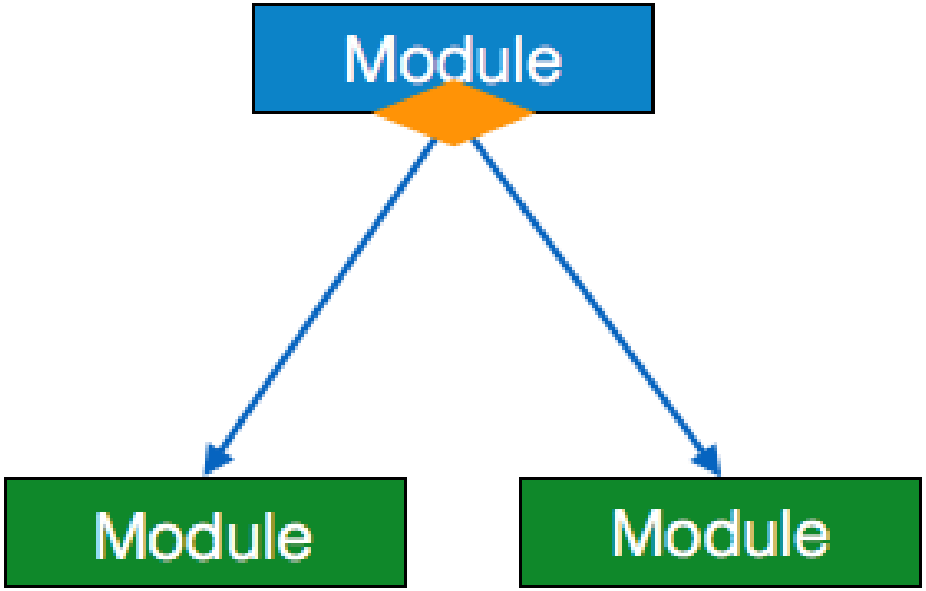
Design Notations and Specifications- Structure Charts

Module - It represents process or subroutine or task. A control module branches to more than one sub-module. Library Modules are re-usable and invokable from any module.



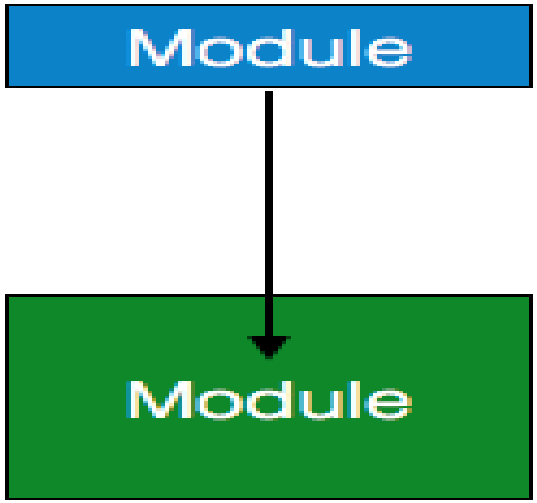
Design Notations and Specifications- Structure Charts

Condition - It is represented by small diamond at the base of module. It depicts(represent) that control module can select any of sub-routine based on some condition.



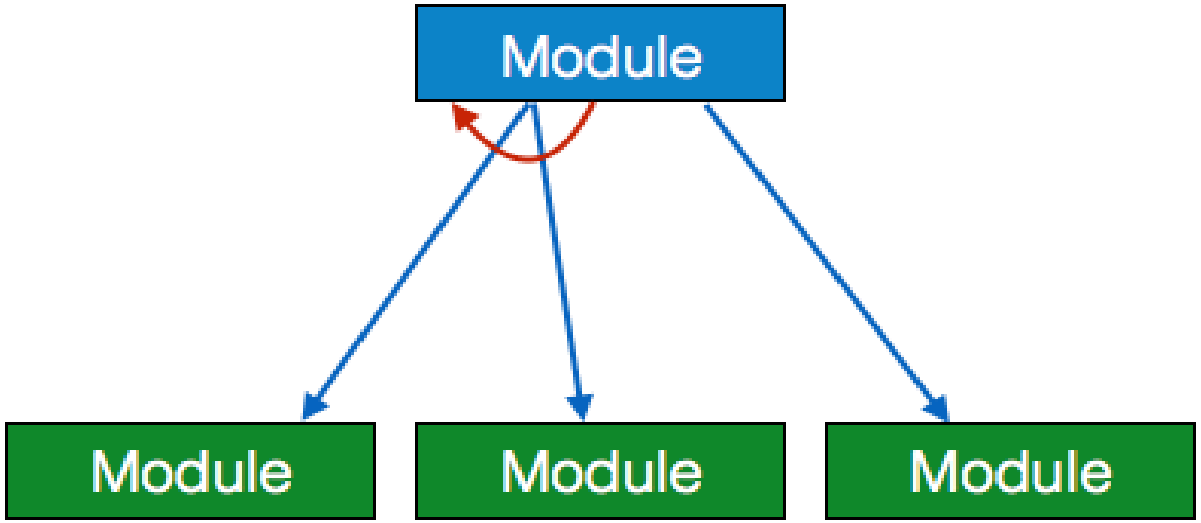
Design Notations and Specifications- Structure Charts

Jump - An arrow is shown pointing inside the module to depict that the control will jump in the middle of the sub-module.



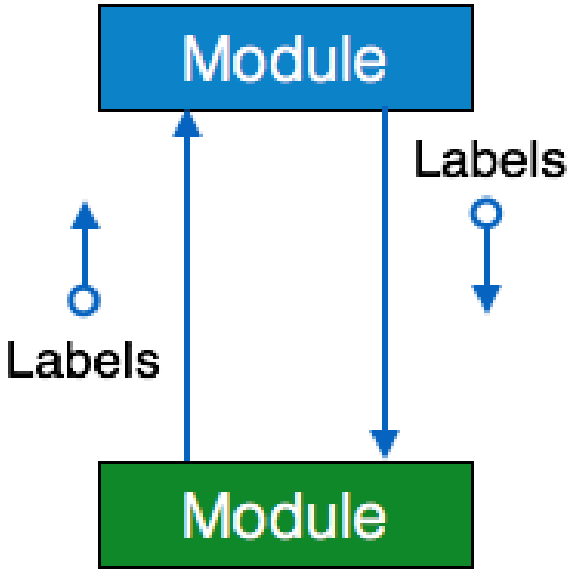
Design Notations and Specifications- Structure Charts

Loop - A curved arrow represents loop in the module. All sub-modules covered by loop repeat execution of module.



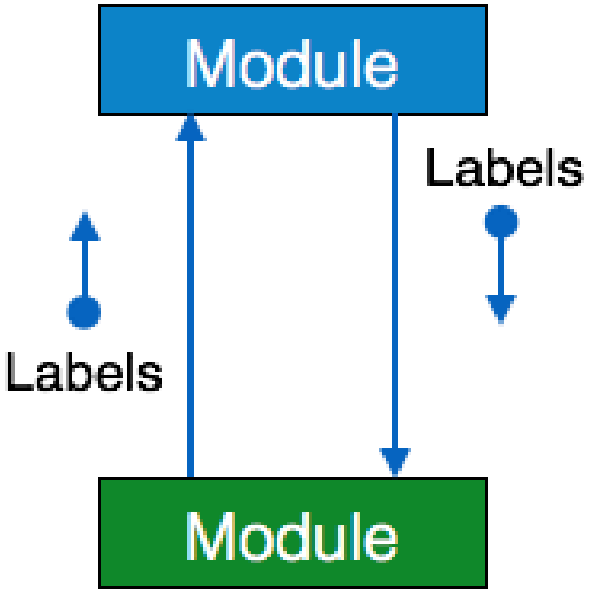
Design Notations and Specifications- Structure Charts

Data flow - A directed arrow with empty circle at the end represents data flow.



Design Notations and Specifications- Structure Charts

Control flow - A directed arrow with filled circle at the end represents control flow.



Design Notations and Specifications- Structure Charts

A structure chart for "update file" is given in fig. 18.

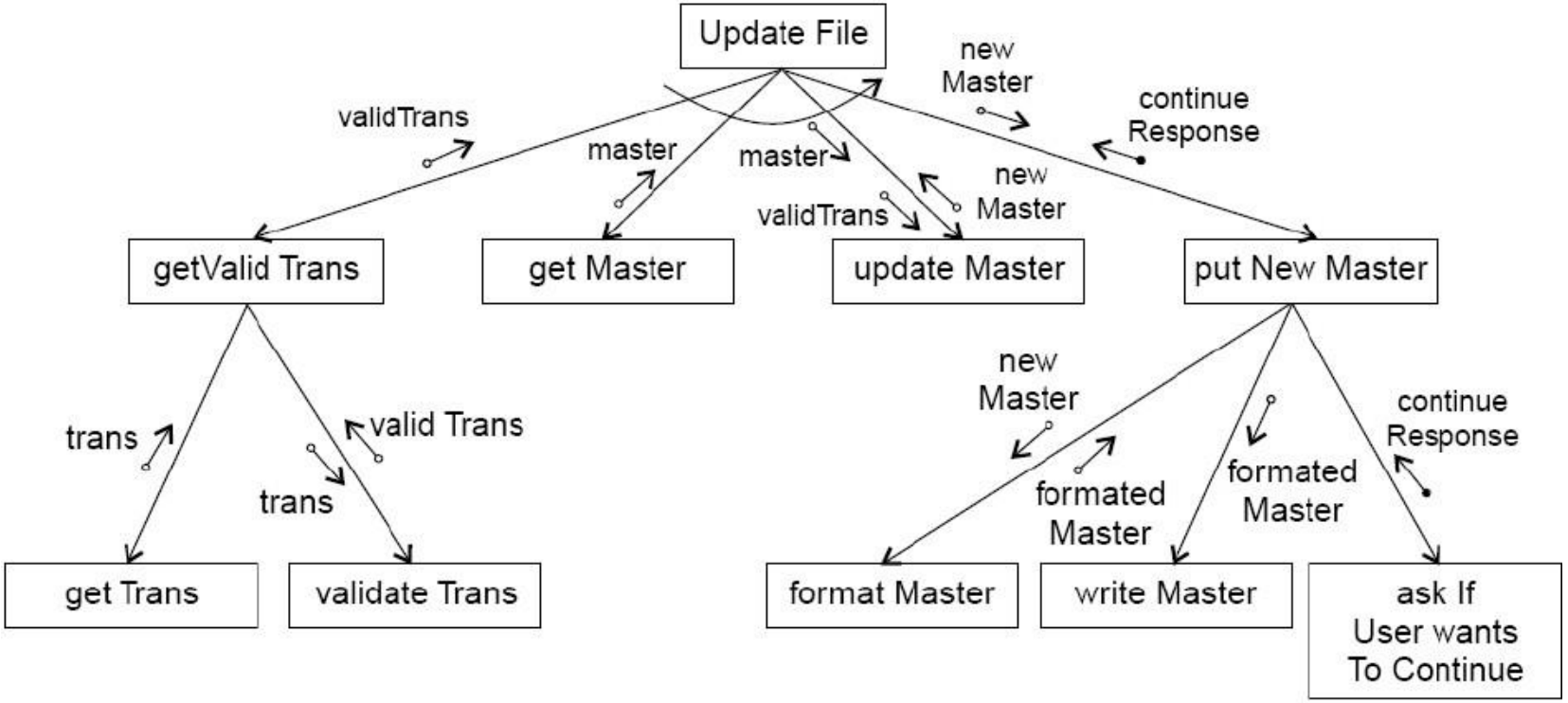


Fig. 18 : Update file

Design Notations and Specifications- Structure Charts

A transaction centered structure describes a system that processes a number of different types of transactions. It is illustrated in Fig.19.

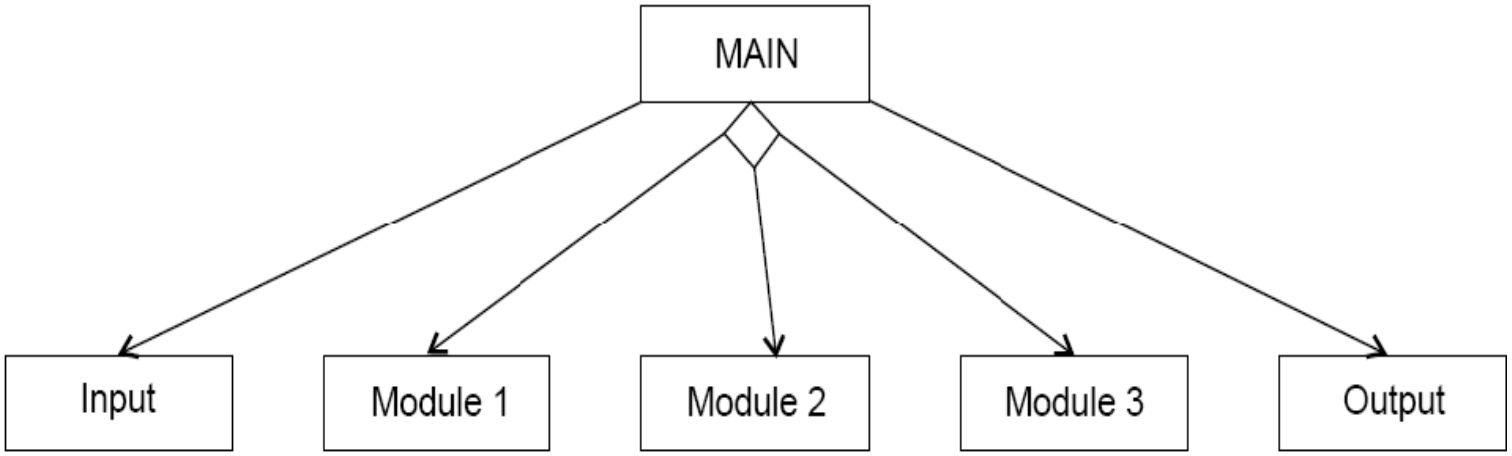
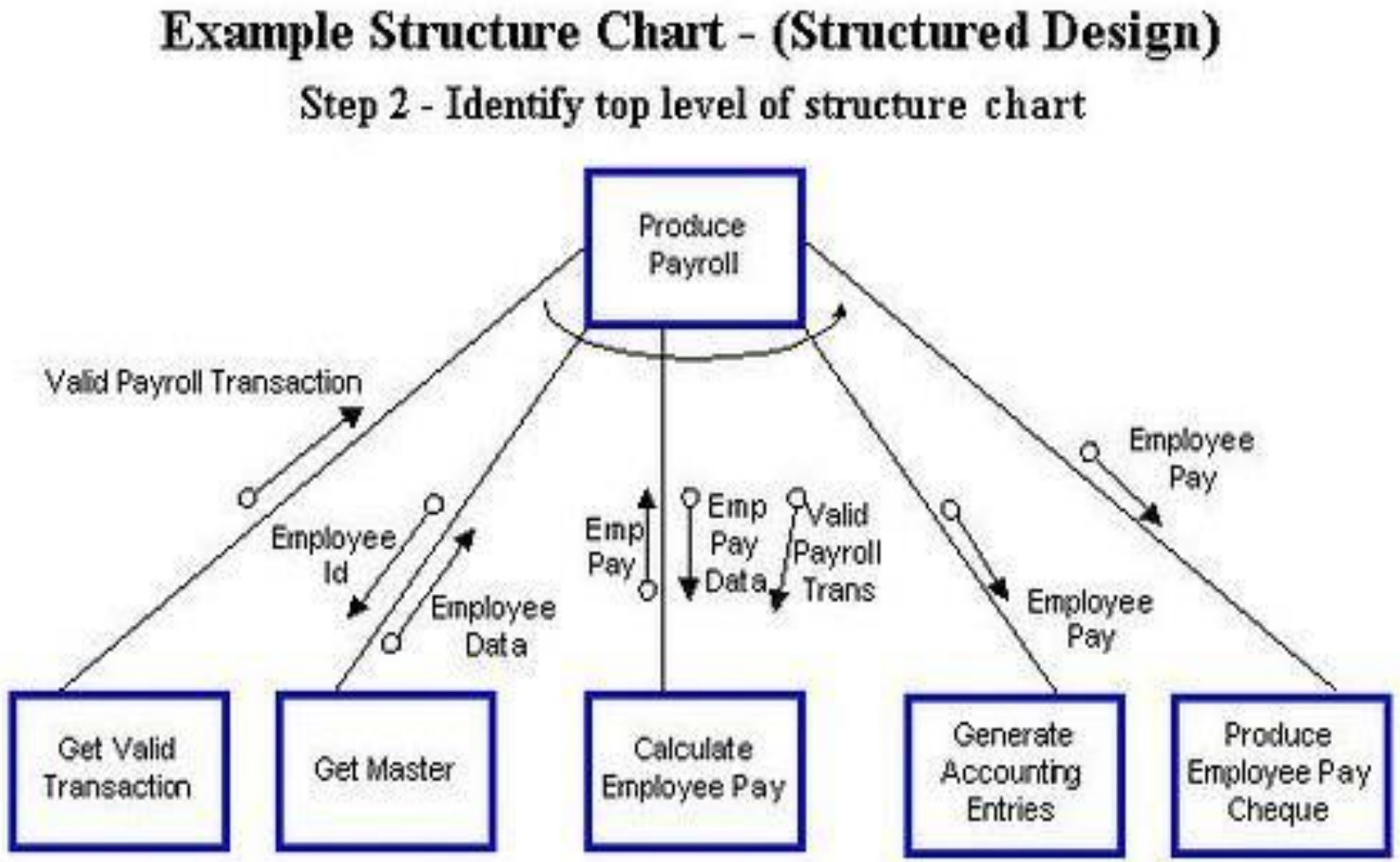
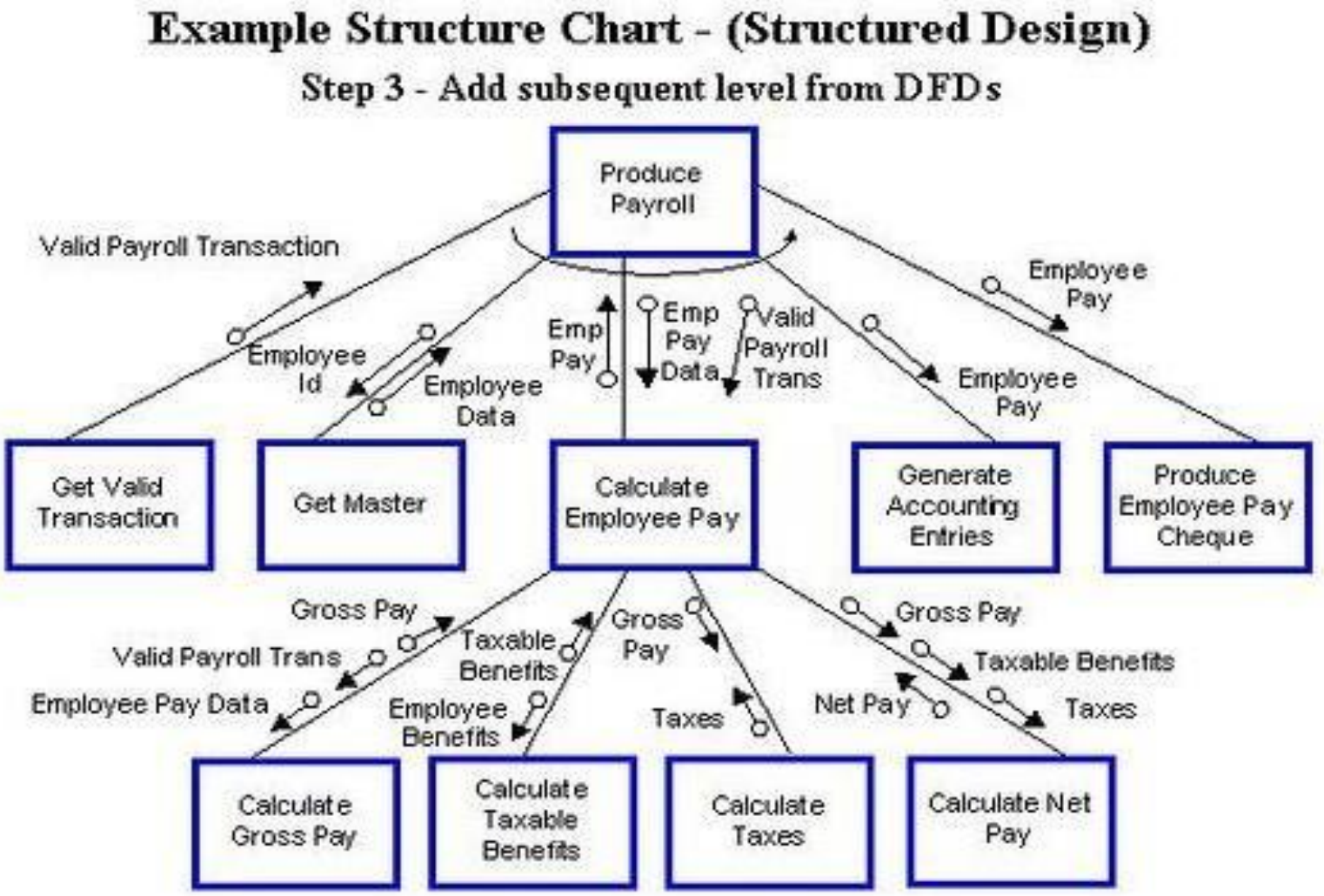


Fig. 19 : Transaction-centered structure

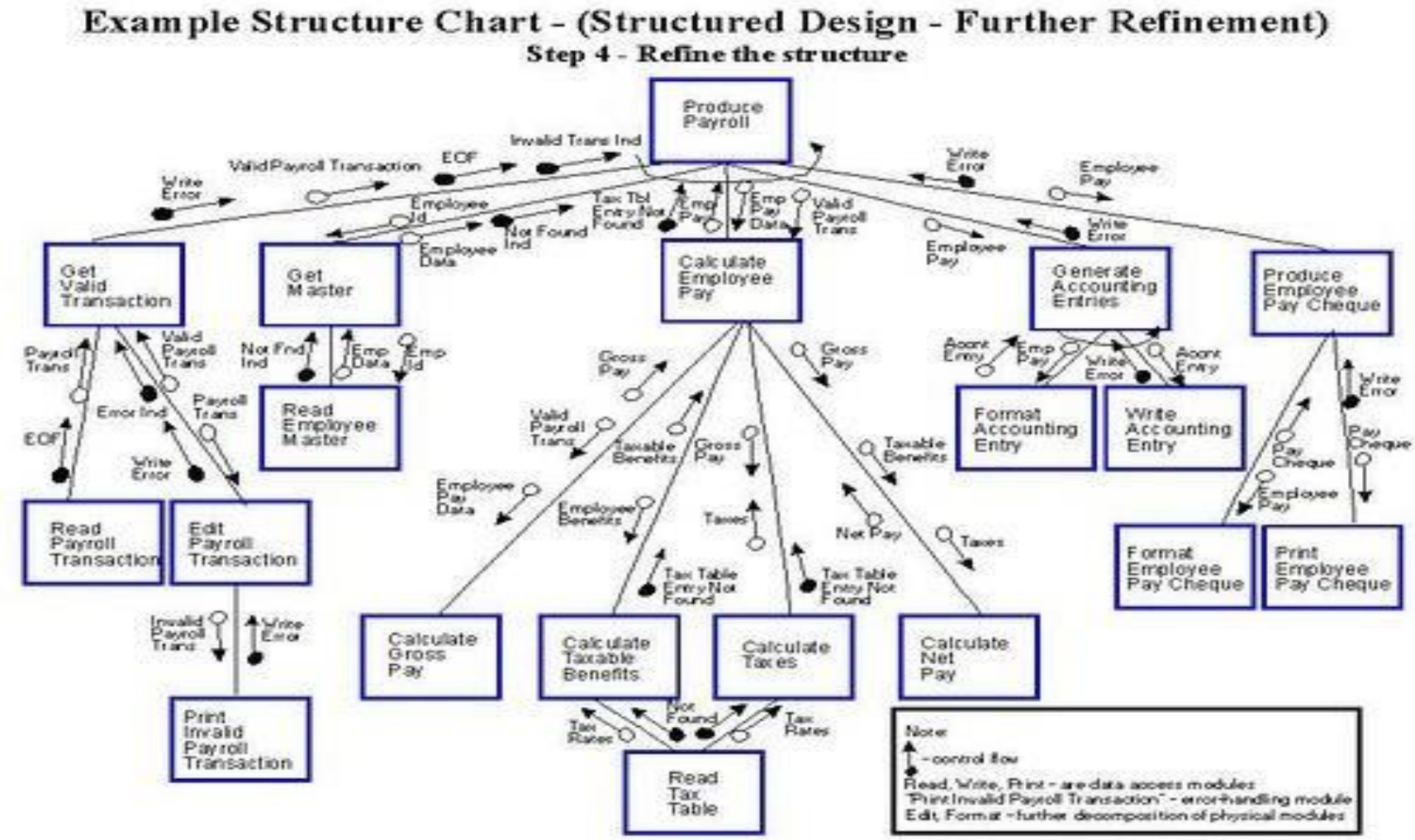
Design Notations and Specifications- Structure Charts



Design Notations and Specifications- Structure Charts



Design Notations and Specifications- Structure Charts



Design Notations and Specifications- Structure Charts

As an example consider the structure of the following program, whose structure is shown in Figure

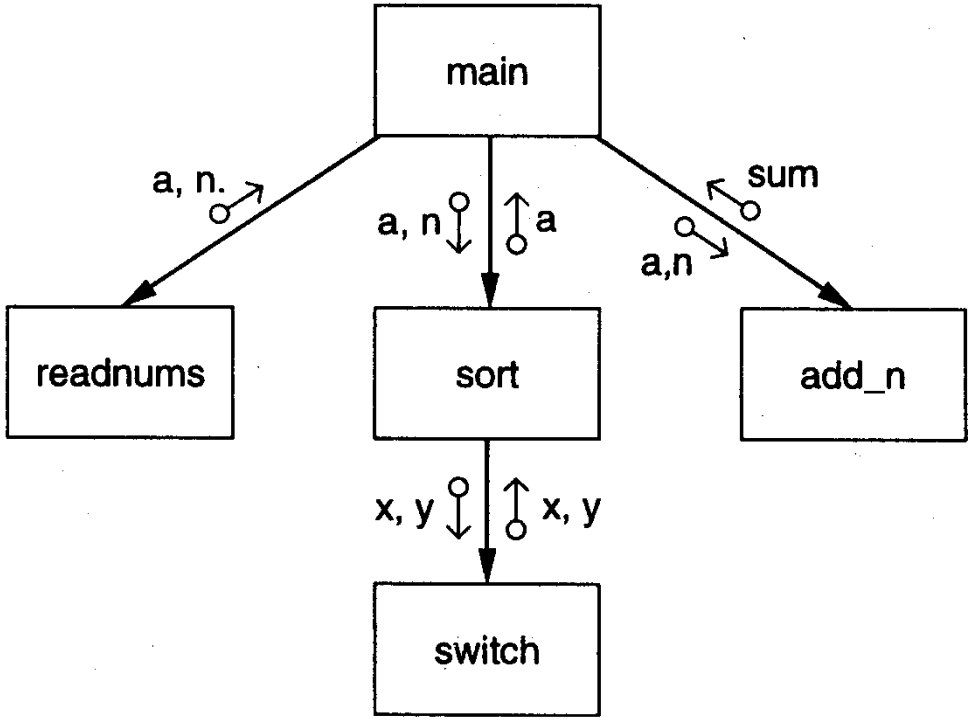
```
main()
{
    int sum, n, N, a[MAX];
    readnums(a, &N); sort(a, N); scanf(&n);
    sum = add_n(a, n); printf(sum);
}

readnums(a, N)
int a[], *N;
{
    :
}
```

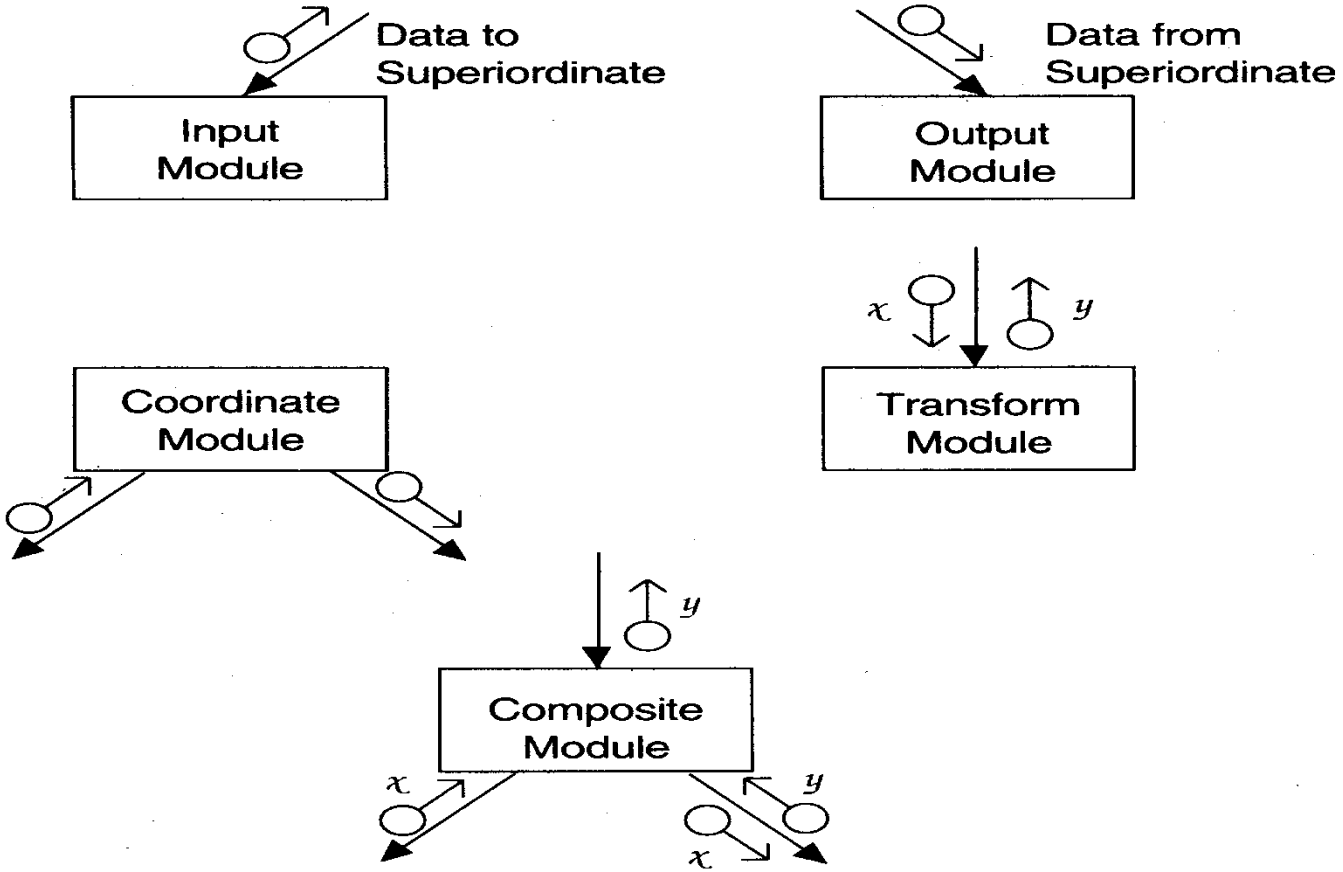
```
sort(a, N)
int a[], N;
{
    :
    if (a[i] > a[t]) switch(a[i], a[t]);
    :
}

/* Add the first n numbers of a */
add_n(a, n)
int a[], n;
{
    :
}
```

Design Notations and Specifications- Structure Charts

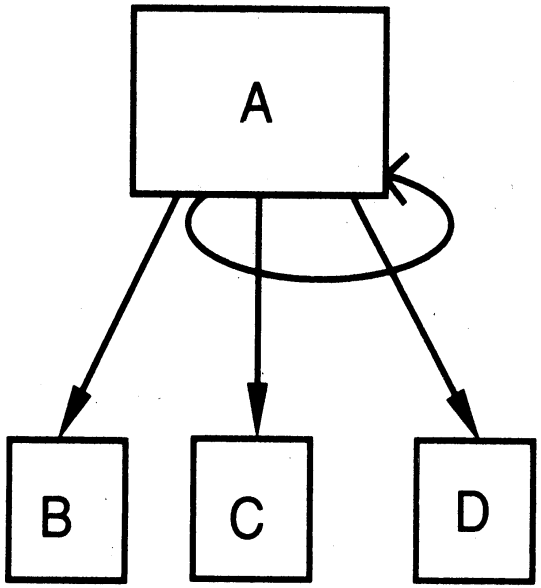


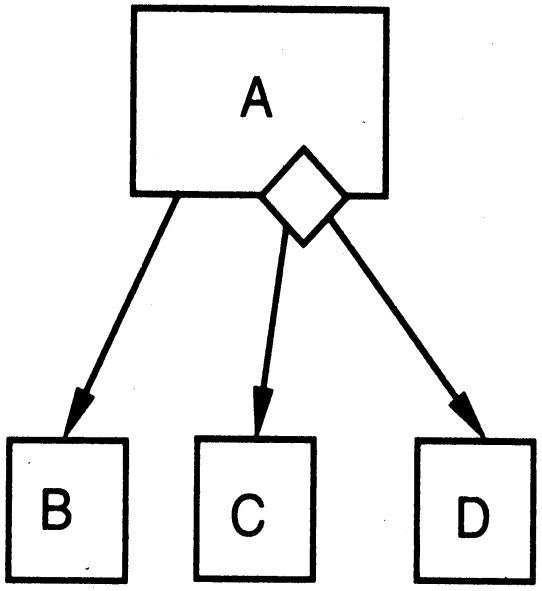
Design Notations and Specifications- Structure Charts



Design Notations and Specifications- Structure Charts

Iteration and decision





References

- K.K. Aggarwal and Yogesh Singh, “Software Engineering”, Third Ed., New Age International (P) Limited Publishers. (2009).
- Pankaj Jalote, “An Integrated Approach to Software Engineering”, Third Ed., Narose Publishing House, (2008).
- Rajib Mall, “Fundamentals of Software Engineering”, Fourth Ed., PHI Learning Private Limited, (2016).
- Roger S. Pressman, “Software Engineering: A Practitioner’s Approach”, Fifth Ed., McGraw–Hill Higher Education, (2001).

Structured Design Methodology

- Creating the software system design is the major concern of the design phase.
- The aim of design methodologies is not to reduce the process of design to a sequence of mechanical steps but to provide guidelines to aid the designer during the design process.
- Structured design methodology (SDM) views every software system as having some inputs that are converted into the desired outputs by the software system.

Structured Design Methodology

- SDM views software as a transformation function that converts given inputs to desired outputs.
- The focus of SD is the transformation function.
- Uses functional abstraction and functional decomposition(A method of analysis to dissects a complex process to examine its individual elements).
- Goal of SDM: Specify functional modules and connections.
- Low coupling and high cohesion is the objective.

Structured Design Methodology

- There are four major steps in this strategy:
  - Draw a DFD of the system.
  - Identify most abstract inputs and most abstract outputs.
  - First level factoring.
  - Factoring of input, output, and transform

Data Flow Diagram

- Struc. Design starts with a DFD to capture flow of data in the proposed system.
- DFD provides a high level view of the system.
- Emphasizes the flow of data through the system.
- Ignores procedural aspects.
- (Purpose here is different from DFDs used in requirements analysis, thought notation is the same)

Data Flow Diagram

- However, there is a fundamental difference between the DFDs drawn during requirements analysis and those drawn during structured design.
- In the requirements analysis, a DFD is drawn to model the problem domain. The analyst has little control over the problem, and hence his task is to extract from the problem all the information and then represent it as a DFD.
- The DFD during design represents how the data will flow in the system when it is built. In this modeling, the major transforms or functions in the software are decided, and the DFD shows the major transforms that the software will have and how the data will flow through different transforms.

1:Draw Data Flow Diagram

- Start with identifying the inputs and outputs
- Work your way from inputs to outputs, or vice versa
  - If stuck, reverse direction
  - Ask: "What transformations will convert the inputs to outputs"
- Never try to show control logic.
- If thinking about loops, if-then-else, start again
- Label each arrow carefully
- Ignore minor functions in the start
- For complex systems, make DFD hierarchical

Consider the example of the simple automated teller machine that allows customers to withdraw money. A DFD for this ATM is shown in Figure.

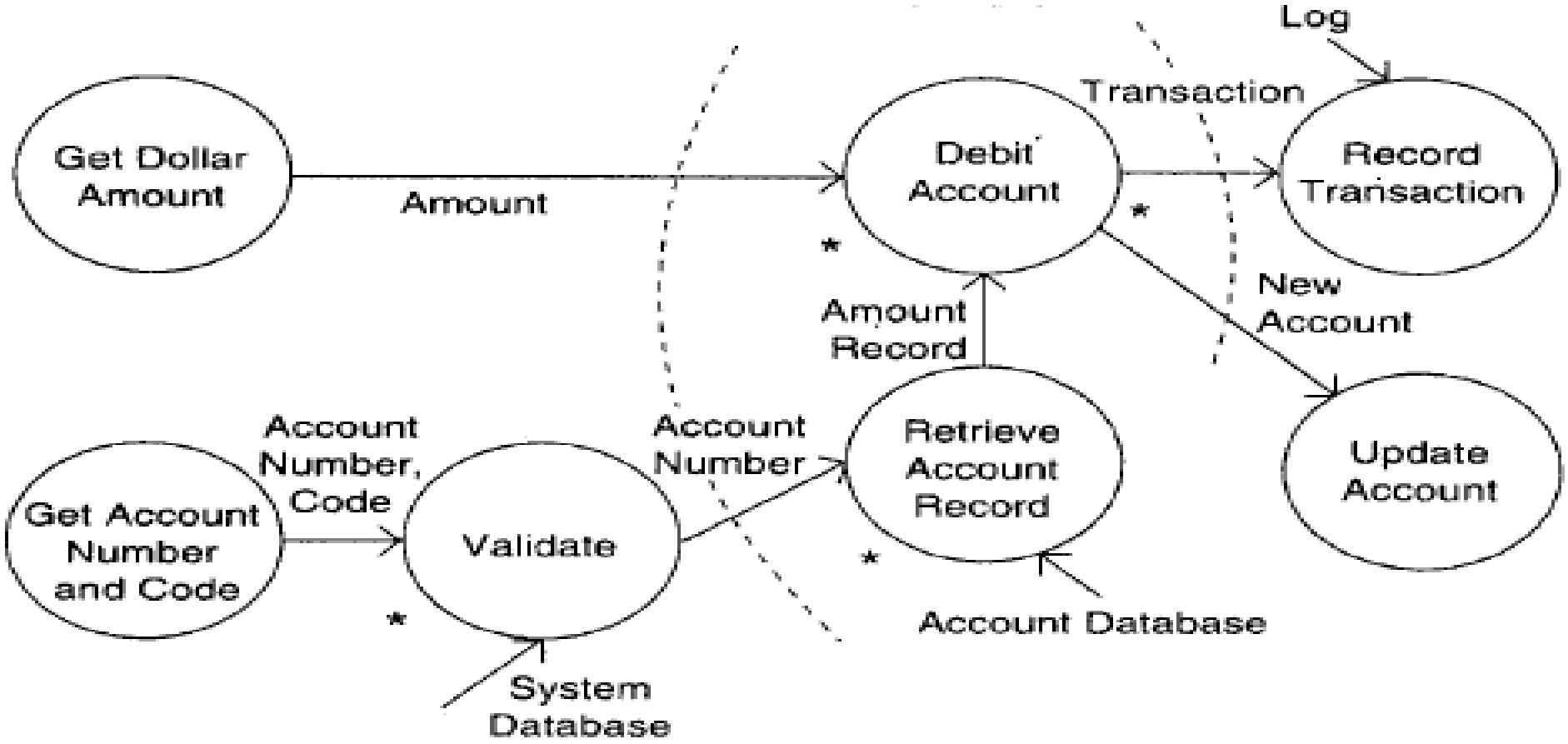
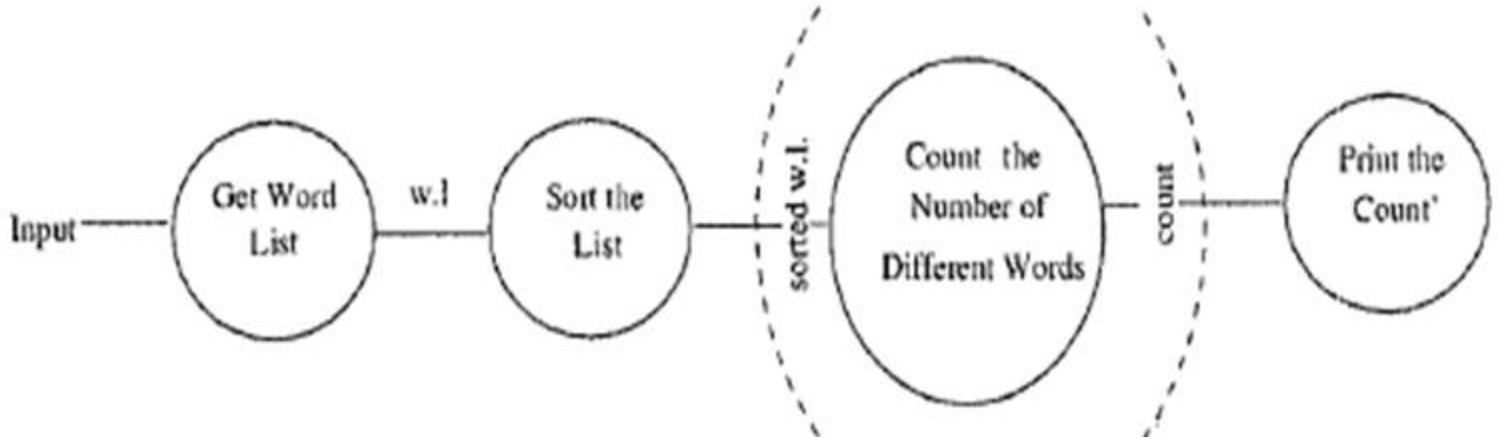


Figure 6.4: Data flow diagram of an ATM.

There are two major streams of input data in this diagram. The first is the account number and the code, and the second is the amount to be debited. The DFD is self-explanatory. Notice the use of * at different places in the DFD. For example, the transform "validate," which verifies if the account number and code are valid, needs not only the account number and code, but also information from the system database to do the validation. And the transform debit account has two outputs, one used for recording the transaction and the other to update the account.



This problem has only one input data stream, the input file, while the desired output is the count of different words in the file. To transform the input to the desired output, the first thing we do is form a list of all the words in the file. It is best to then sort the list, as this will make identifying different words easier. This sorted fist is then used to count the number of different words, and the output of this transform is the desired count, which is then printed. This sequence of data transformation is what we have in the data flow diagram.