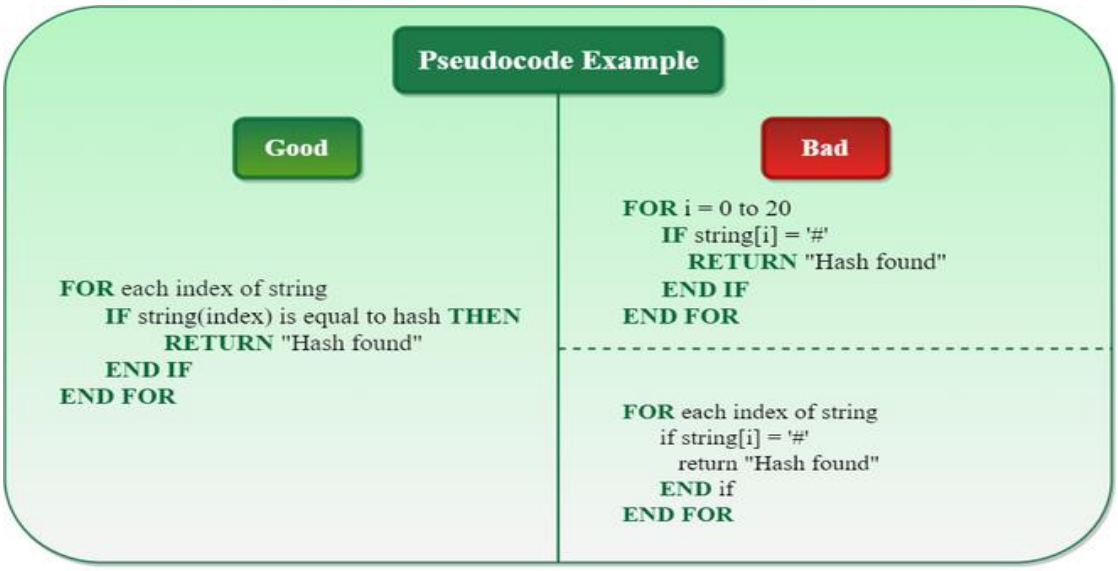
Web Development and Database Administration

Level-III

Based on November 2023, Curriculum Version II



Module Title: Designing Program Logic

Module code: EIS WDDBA3 M04 0323

Nominal duration: 50 Hours

Prepared by: Ministry of Labor and Skill

November, 2023

Addis Ababa, Ethiopia

Table of contents

Acknowledgment 1
Acronym 2
Introduction to the Module 3
Unit One:program logic design 4
1.1.Systems development life cycle 5
1.2.Design documentation 7
1.3.Design approach coding 10
Self-check-1 39
Operation sheet 1.1. Designing Program Logic 41
Lap Tests 45
Unit Two: program logic Documentation 46
2.1. Project planning standards 47
2.2. Documentation of Project scope 49
2.3. Project scope statement 50
2.4. Identification and revising References 52
2.5. Logic design templates 54
2.6. Logic design validation 55
Self-check-2 58
Reference 59
Developer's Profile 60

Acknowledgment

Ministry of Labor and Skills wish to extend thanks and appreciation to the many representatives of TVET instructors and respective industry experts who donated their time and expertise to the development of this Teaching, Training and Learning Materials (TTLM).

Acronym

CASE----------computer-aided software engineering

CRUD----------Create, Read, Update, and Delete

DFD----------Data Flow Diagram

HIPO----------Hierarchy Input Process Output Diagram

MDS---------- Master Data Services

RAD ----------Rapid Application Development model.

SDLC----------Systems development life cycle

SOW----------scope of work

UML----------Uniform Modeling Language

Introduction to the Module

This module is designed to meet the industry requirement under the Web Development and Database Administration occupational standard, particularly for the unit of competency Designing Program logic and it describe the various processes in a program to ensure that there is understanding of user and design requirements**.**

Module covers the units:

- program logic design
- program logic Documentation

Learning Objective of the Module

- Understand program logic design
- Understand program logic Documentation
- Develop Systems development life cycle
- Create a flowchart

Module Instruction

For effective use these modules trainees are expected to follow the following module instruction:

1. Read the information written in each unit
2. Accomplish the Self-checks at the end of each unit
3. Perform Operation Sheets which were provided at the end of units
4. Do the “LAP test” giver at the end of each unit
5. Read the identified reference book for Examples and exercise

Unit One:program logic design

This unit is developed to provide you the necessary information regarding the following content coverage and topics:

- Systems development life cycle
- Design documentation
- design approach coding

This unit will also assist you to attain the learning outcomes stated in the cover page.

Specifically, upon completion of this learning guide, you will be able to:

- Develop Systems development life cycle
- Understand and Design document
- Demonstrate design approach coding

1.1. Systems development life cycle

The systems development life cycle (SDLC) is a conceptual model used in project management that describes the stages involved in an information system development project, from an initial feasibility study through maintenance of the completed application. SDLC can apply to technical and non-technical systems.

The method or approach that software engineers use in solving problems in computer science is called software development method. Another name that is commonly used for the software development method is called software life cycle. The software life cycle has the following components.

- Preliminary investigation
- Analysis
- Design
- Implementation
- Deployment and Testing
- Maintenance

A. **Preliminary investigation**: The purpose of the preliminary investigation is to determine whether the problem or deficiency in the current system really exists. The project team may re-examine some of the feasibility aspects of the project. The end result is a decision to proceed further or to abandon the project.

   - ✓ **Defining the problem**: E.g. Examines documents, work papers, and procedures; Observe system operations; interview key users of the system.
   - ✓ **Suggesting a solution e.g**. often improving an existing one or building a new information system.

Determine whether the solution is feasible.

- ✓ **Technical Feasibility**: Whether implementation is possible with the available or affordable hardware, software and other technical resources.
- ✓ **Economic Feasibility**: Whether the benefits of the proposed solution outweigh the costs.

- ✓ **Operational Feasibility**: Whether the proposed solution is desirable within the existing managerial and organizational framework.

B. **Analysis (Requirement gathering):** In the analysis phase end user business requirements are analyzed and project goals converted into the defined system functions that the organization intends to develop.

- ✓ Try to understand the business in general (activities done, how it is done, etc)
- ✓ Define the specific information requirements, who needs what information, where, when and how.

C. **Design phase:** In the design phase, we describe the desired features and operations of the system. This phase includes business rules, pseudo-code, screen layouts, and other necessary documentation.

D. **Implementation:** In the implementation phase, the development team codes the product. They analyze the requirements to identify smaller coding tasks they can do daily to achieve the final result. System performance is compared to performance objectives established during the planning phase. Implementation includes user notification, user training, installation of hardware, installation of software onto production computers, and integration of the system into daily work processes. This phase continues until the system is operating in production in accordance with the defined user requirements.

E. Deployment and Testing: When teams develop software, they code and test on a different copy of the software than the one that the users have access to. The software that customers use is called production, while other copies are said to be in the build environment, or testing environment.

F. **Maintenance:** In the maintenance phase, among other tasks, the team fixes bugs, resolves customer issues, and manages software changes. In addition, the team monitors overall system performance, security, and user experience to identify new ways to improve the existing software.

Some benefits of SDLC:

- ✓ Increased visibility of the development process for all stakeholders involved
- ✓ Efficient estimation, planning, and scheduling

- ✓ Improved risk management and cost estimation
- ✓ Systematic software delivery and better customer satisfaction

1.2. Design documentation

The Design Document will be the starting point and in many cases you should, do the Design Document before doing any of the actual work of program. When you write code for a program, you usually will have clarified your requirements and planning the design before you write the first line of actual code. Having the type of documentation that we are expecting helps ensure a number of things like; you are doing what the customer wants; that when you are done, you actually did what you intended to do; that in case of personnel problems you project continuity remains; etc.

Program logic is the implementation of the program's requirements and design. If the design of the application is bad, the program logic can nevertheless be professionally implemented. For example, if the user interface is poorly conceived, the program logic can execute that second-rate interface. Programming logic involves logical operations on hard data that works according to logical principles and quantifiable results.

The important distinction here is that programming logic, and logic in general, is fundamentally set against other kinds of programming that are not built on hard logic or quantifiable states and results. For example, modal logic by its nature is set against the theoretical quantum operations that don't provide a specific set state that computers can apply logic to.

Programming logic in general rests on a foundation of computational logic that is shared by both humans and machines, which is what we explore as we continue to interact with new technologies. With that in mind, one could develop more specific definitions of a programming logic having to do with the basis of a piece of code.

Generally,

- ✓ Program logic is a picture of why and how you believe a program or a policy will work.
- ✓ Program logic demonstrates design & implementation of competence.

- ✓ Program logic provides a chain of reasoning that links investments with results.
- ✓ Program logic is a series of "if-then" relationships that, if implemented as intended, lead to the desired outcomes.

- **Logic model**

Logic models provide a kind of map for a program or initiative, helping clarify a program or policy's destination, the pathways toward the destination, and markers along the way. In this consider:

- ✓ Where are you going?
- ✓ How will you get there?
- ✓ What will tell you that you have arrived?

Logic models provide a simplified picture of the relationships between the program activities and the desired outcomes of the program. It is valuable in supporting:

- ✓ Program planning.
- ✓ Program implementation.
- ✓ Program monitoring.
- ✓ Program evaluation.

We also use a logic model to:

- ✓ Brings detail to broad goals.
- ✓ Helps identify gaps in program logic and clarify assumptions.
- ✓ Builds understanding and promotes consensus.
- ✓ Makes explicit underlying beliefs.
- ✓ Helps clarify what is appropriate to evaluate and when.
- ✓ Summarizes complex programs for effective communication.

- **Program of requirements**

A specification of requirements or Program of Requirements is a document used in a design or procurement process. An optimal construction process begins with a thorough requirements specification, which comprehensively details the demands and wishes. The aim of a Program of

Requirements is to lay down a clear framework for everyone involved in the project so that everyone understands which criteria must be met. Hence, the program of requirements is an important guideline for the designers to ensure that they deliver the desired result.

- **Understanding the Programming Process**

  There are about six programming phases:

  - ✓ Understand the problem
  - ✓ Plan the logic
  - ✓ Code the program
  - ✓ Use software to translate the program to machine language
  - ✓ Test the program
  - ✓ Deploy the program into production

A. Understanding the problem

- ✓ May be the most difficult phase
- ✓ Users may not be able to articulate their needs well
- ✓ User needs may be changing frequently
- ✓ Programmers may have to learn the user's functional job tasks
- ✓ Failure to understand the problem is the major cause of most project failures

B. **Planning the logic**

- ✓ Plan the steps that the program will take
- ✓ Use tools such as flowcharts and pseudo code to depict or illustrate the structure or steps of program.
- ✓ Flowchart: a pictorial representation of the logic steps
- ✓ Pseudo code: English-like representation of the logic
- ✓ Walk through the logic before coding by desk-checking the logic.

C. Coding the program:

- ✓ Select the programming language
- ✓ Write the instructions

D. Using software to translate the program into machine language:

- ✓ Programmers write instructions in English-like high-level languages
- ✓ Compilers or interpreters change the programs into low-level machine language that can be executed
- ✓ Syntax errors are identified by the compiler or interpreter

E. Testing the program:

- ✓ Execute it with sample data and check results
- ✓ Identify logic errors and correct them
- ✓ Choose test data carefully to exercise all branches of the logic

F. Putting the program into production

- ✓ Do this after testing is complete and all known errors have been corrected
- ✓ May require coordination with other related activities or software

1.3. Design approach coding

- **Program algorithm**

An algorithm is a set of instructions designed to perform a specific task. This can be a simple process, such as multiplying two numbers, or a complex operation, such as playing a compressed video file. Search engines use proprietary algorithms to display the most relevant results from their search index for specific queries. In computer programming, algorithms are often created as functions. These functions serve as small programs that can be referenced by a larger program. For example, an image viewing application may include a library of functions that each uses a custom algorithm to render different image file formats. An image editing program may contain algorithms designed to process image data. Examples of image processing algorithms include cropping, resizing, sharpening, blurring, red-eye reduction, and color enhancement.

In many cases, there are multiple ways to perform a specific operation within a software program. Therefore, programmers usually seek to create the most efficient algorithms possible. By using highly efficient algorithms, developers can ensure their programs run as fast as possible and use minimal system resources. Of course, not all algorithms are created perfectly the first time. Therefore, developers often improve existing algorithms and include them in future software

updates. When you see a new version of a software program that has been "optimized" or has "faster performance," it most means the new version includes more efficient algorithms.

- **Designing an algorithm/ a solution to a problem**

A program is written in order to solve a problem. A solution to a problem actually consists of two things.

- ✓ A way to organize the data.
- ✓ Sequence of steps to solve the problem.

The way data are organized in a computer memory is said to be data structure and the sequence of computational steps to solve a problem is said to be an algorithm. Therefore a program is nothing but data structure plus algorithms. An algorithm is a well-defined computational procedure that takes some values or a set of values as input and produces some values or a set of values as output. An algorithm is a procedure for solving a problem in terms of the actions to be executed and the order in which those actions are to be executed. An algorithm is,

- ✓ A step-by-step sequence of instructions.
- ✓ To solve a specific problem.
- ✓ In a finite amount of time.

- **Algorithm development**

Once the requirements of a program are defined algorithm development is the next step. An algorithm is procedure for solving a problem in terms of the action to execute (**What to do)** and the order in which these actions are executed (done). An algorithm needs to be:

- ✓ Precise and unambiguous (No ambiguity in any instruction and in the order of execution)
- ✓ Correct,
- ✓ Finite (has to have an end),
- ✓ Handles all exceptions,
- ✓ Efficient in time, memory and other resources,

Many different methods exist for constructing algorithms, (An algorithm can be expressed in many ways). Some of those methods are **Flowchart and Pseudo-code**.

- **Pseudo code**

Pseudo code is defined as a step-by-step description of an algorithm. Pseudo code does not use any programming language in its representation instead it uses the simple English language text as it is intended for human understanding rather than machine reading.

Pseudo code is the intermediate state between an idea and its implementation (code) in a high-level language.

Pseudo code is an important part of designing an algorithm; it helps the programmer in planning the solution to the problem as well as the reader in understanding the approach to the problem. Pseudo code is an intermediate state between algorithm and program that plays supports the transition of the algorithm into the program.

Pseudocode is an intermediate state between algorithm and program



- ✓ It is an informal high-level description of the operating principle of a computer program or other algorithm.
- ✓ English-like representation of a program (logic).
- ✓ It could be viewed as the outline of a program
- ✓ It can’t be compiled or executed.
- ✓ The focus is on solving the problem, not worrying about the specifics of a programming language.

It uses the structural conventions of a programming language, but is intended for human reading rather than machine reading. Pseudo code typically omits details that are not essential for human

understanding of the algorithm, such as variable declarations, system-specific code and some subroutines.

The programming language is augmented with natural language description details, where convenient, or with compact mathematical notation. The purpose of using pseudo code is that it is easier for people to understand than conventional programming language code, and that it is an efficient and environment-independent description of the key principles of an algorithm.

It is commonly used in textbooks and scientific publications that are documenting various algorithms, and also in planning of computer program development, for sketching out the structure of the program before the actual coding takes place.

The advantage of pseudo coding is that if performed fully and accurately a computer programmer can translate the Structured English directly to computer code.

Creating pseudo code is a useful step outside systems analysis: if the analyst can describe orally the steps involved, including logical dependencies, then s/he can explain the new system to any audience and will identify logical problems before they are manifested in the design.

Programmers may resist pseudo code that dictates how to solve the problem. The programmer needs to understand the logic and should not have to guess what the analyst means. On the other hand, if the analyst dictates the data structures to be used (e.g., use a hash here, an array there, or try this enumeration…) then the programmer's ability to create a viable program is impacted.

Then the programmer may determine the best way for the machine to handle the data.

Pseudo coding might be considered the textual equivalent of flowcharting and is useful for explaining logic models to people who shy from anything that looks technical.

- **Advantages of Pseudo code**

✓ Improves the readability of any approach. It's one of the best approaches to start implementation of an algorithm.

✓ Acts as a bridge between the program and the algorithm or flowchart. Also works as a rough documentation, so the program of one developer can be understood easily when a pseudo code is written out. In industries, the approach of documentation is essential. And that's where a pseudo-code proves vital.

- ✓ The main goal of a pseudo code is to explain what exactly each line of a program should do, hence making the code construction phase easier for the programmer.

Example1. A pseudo-code to calculate bonus

ACCEPT Name, Salary

Bonus=Salary x 0.1

ACCEPT Service year

IF Service year> 10 Then

Bonus + 100

ENDIF

DISPLAY Bonus

Example 2. A pseudo-code to calculate interest rate

ACCEPT Name, Principal, Rate

Interest = Principal x Rate

DISPLAY Name, Interest

When we write programs, we assume that the computer executes the program starting at the beginning and working its way to the end.

We call this sequence

SEQUENCE is a linear progression where one task is performed sequentially after another.

In pseudo code it look like this

Statement 1;

Statement 2;

Statement 3;

Statement 4;

Statement 5;

For example, for making a cup of tea:

Plug in kettle;

Put teabag in cup;

Put water into kettle;

Wait for bottle to boil;

Add water to cup;

Remove teabag with spoon/fork/;

Add milk and /or sugar;

Organize everything together;

Serve

Pseudo Code with LOOP

What if we need to tell the computer to keep doing something until some condition occurs?

Let's say we wish to indicate that you need to keep filling the kettle with water until it is full.

We call this a loop.

WHILE is a loop (repetition) with a simple conditional test at its beginning or in general

While (<condition>)
Do <Statements>;
Endwhile;

As we could state this as:

While (Kettle is not full)
Do keep filling Kettle;
EndWhile

Pseudo Code with SELECTION

What if we want to make a choice, for example, do we want add sugar or not to the tea?

We call this selection.

IF-THEN-ELSE is a decision (selection) in which a choice is made between two alternative courses of action or in general:

If (<condition>)
Then <Statement>;
Else <Statement>;
EndIf;

If (Sugar is required)
Then add Sugar;
Else do nothing;
EndIf;

Example3. A pseudo-code that calculates grade hint Grade A>=80, B>=60, C>=50, D>=40, F<40

ACCEPT Mark, Name
IF Mark>80 Then
Grade A
ELSE IF Mark >60 Then
Grade B
ELSE IF Mark >50 Then

Grade C

ELSE IF Mark >40 Then

Grade D

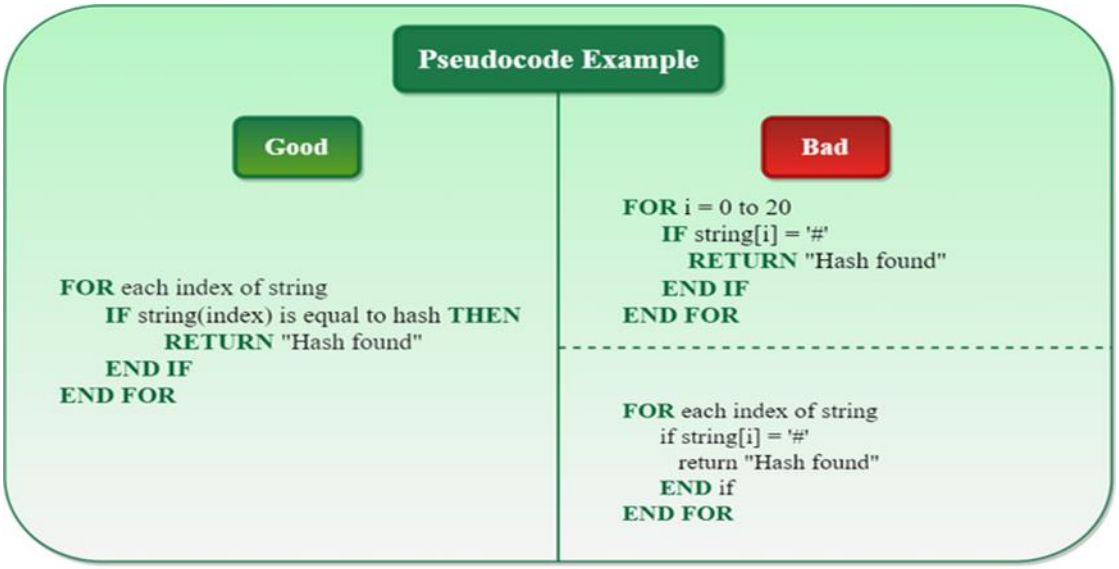
ELSE

Grade F

ENDIF

Display Grade, Name

Pseudo code good and bad writing style



- **Flowcharts**

A flow chart is a graphical or symbolic representation of a process (Pictorial representation of a program). Each step in the process is represented by a different symbol and contains a short description of the process step. The flow chart symbols are linked together with arrows showing the process flow direction. In other word Flowchart is pictorial representation of the logic & the purpose of flowcharts is to represent graphically the logical decisions and progression of steps in the physical completion of a task. As such, flowcharts are the lowest level of decomposition.

It specifies the order in which tasks are performed. Therefore, anyone who knows about the function can recognize immediately changes to the actual performance of work. Since the logic of

the work is tested, it is here the analyst identifies and documents changes to the physical work flow of people or the design of a computer program.

In a large system the flow charting steps may be complex and the analyst may choose to represent the system with algebra, structured English or pseudo code, or some type of decision tree or decision table.

When describing a complicated system to others it may be useful to use hierarchy charts or structure charts. But for the information systems analysis, the best options are flow charts and pseudo coding/ Structured English.

Before beginning the flow charts, the analyst must have a complete understanding of the entire process and participants (context level), the subroutines and their interdependencies (level-n diagrams), the data (data dictionary, inputs/outputs) and transformation of data (DFDs).

The flow chart documents, the flow of logic, control and the actual data through a routine or procedure activity completes the logical part of analysis. From here we proceed to integrate the whole decomposed process into something new the revised, more efficient and effective system. After flowcharting, we examine how the pieces fit back together and express this in a system flowchart.

The system flowchart represents the physical system; while the flowcharting at the end of process decomposition represents the logical. Can be created with pencil and paper, Microsoft Word, PowerPoint, and Visio etc.

- ✓ **Symbol set**

There are seven commonly used symbols in flowcharting. However, be aware that people can create their own symbol set. The most common symbols that can be used to represent the main concept include:

Terminator: marks the beginning and end of a flowchart.

Process: indicates that an operation changed or manipulated the data in some way.

Data or input/output: an operation that inputs or outputs data.

Decision: a logical decision point (e.g., if, then; true/false).

On-page connector: the logic continues at another place on the same page.

Off-page connector: the logic continues on a different page.

Predefined process: a predefined process or subroutine (similar to a predefined function or process in database programming).

As a general rule, a flowchart has a single entry and exit point. In procedural programming it is fairly easy to create such charts but Object oriented programming, on the other hand, can become completed very quickly and using single-entry, single-exit logic may be difficult, at times impossible. As a consequence, another kind of language has developed, Uniform Modeling Language (UML).

Six basic flowchart shapes



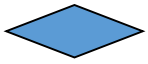
Terminal symbol -Indicates the beginning or end of a program



Input/output symbol- Represents information coming into a program or going out of a program



Processing symbol- Shows calculations that a program must do



Diamond symbol represents decision- that a program must make



The circle is for connecting parts of the flowchart to other parts. This is especially Useful if the flowchart covers more than one page

Flow lines connect the shapes within a flowchart

Program logic has three patterns: sequence, decision, and repetition.

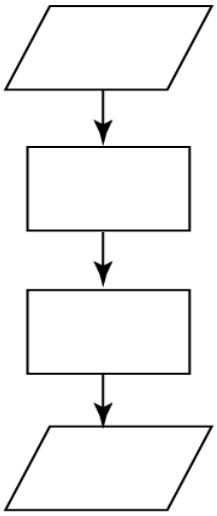
Sequence:

A decision block implies IF-THEN-ELSE. A condition ("it's the case that something is true …", or "while x is true …") is tested: if the condition is true, the logic associated with the "THEN" branch is performed while the ELSE block is skipped. If the condition is false, then the ELSE logic is executed and the THEN logic is skipped.

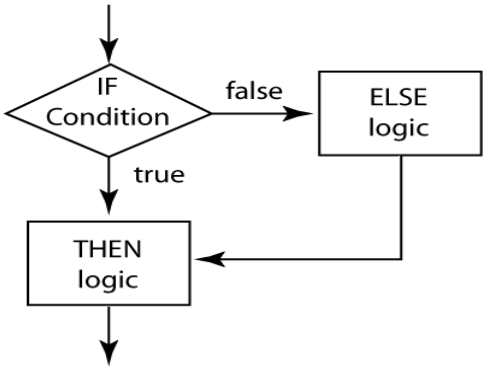
For example, "IF there's enough gas in the car (condition), THEN (true) I'll drive to my friends' house; ELSE (false, not enough gas) I'll go to the gas station.

Sometimes the if/then are nested in other if/then conditions: If there's enough gas in the car, I'll go to my friend's house; if I don't reach there by noon, I'll stop at a restaurant along the way.

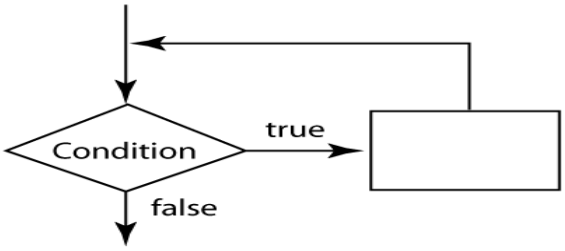
1. Sequence logic



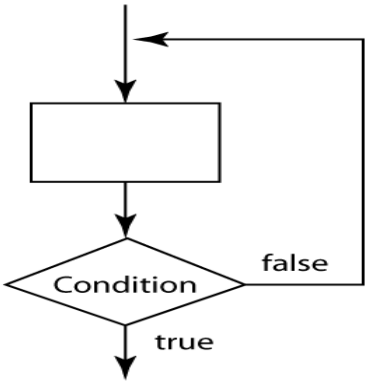
2. Decision logic



3. Repetitive logic



do while



do until

Example 4 Add three numbers

A program is required to read three numbers, add them together and print their total.

Pseudo code

Add_three_numbers

Read number1, number2, number3

Total = number1 + number2 + number3

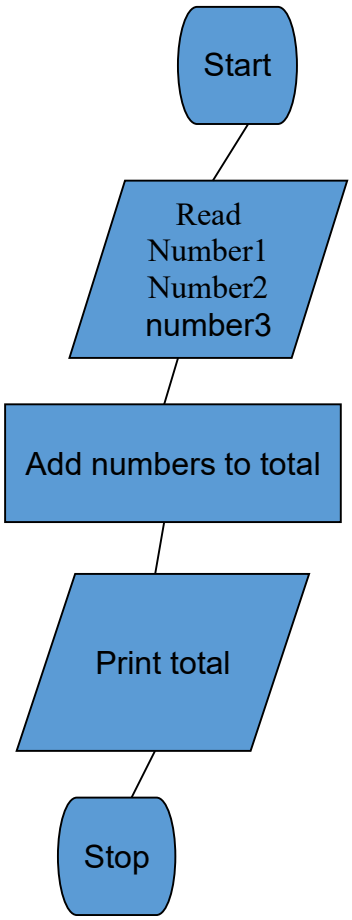
Print total

END

Defining diagram

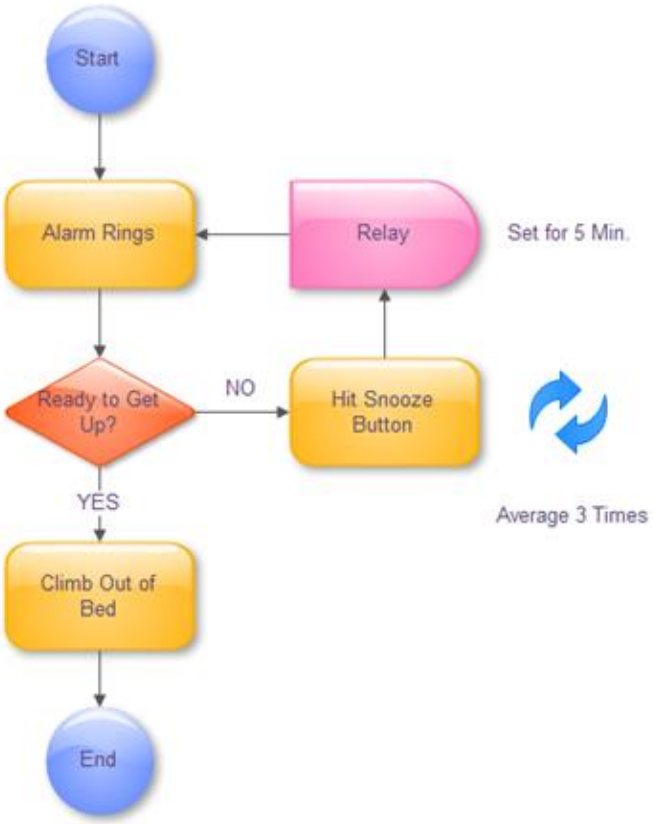
Input	**Processing**	**Output**
Number1 Number2 Number3	Read three numbers Add number together Print total number	total

Flow Chart



Example 5 Getting out of bed in the morning

The following flow chart example presents a very simple flow chart for the process of getting out of bed in the morning.



Example6 :- Find average temperature

A program is required to prompt the terminal operator for the maximum and minimum temperature readings on a particular day, accept those readings as integers, and calculate and display to the screen the average temperature, calculated by (maximum temperature + minimum temperature)/2.

Defining diagram

Input	**Processing**	**Output**
Max_temp Min_temp	Prompt for temperatures Get temperatures Calculate average temperature Display average temperature	Avg_temp

Pseudo code

Find average_temperature

Prompt operator for max_temp, min_temp

Get max_temp, min_temp

Avg_temp= (max_Temp + min_temp)/2

Output avg_temp to the screen

END

Flow Chart

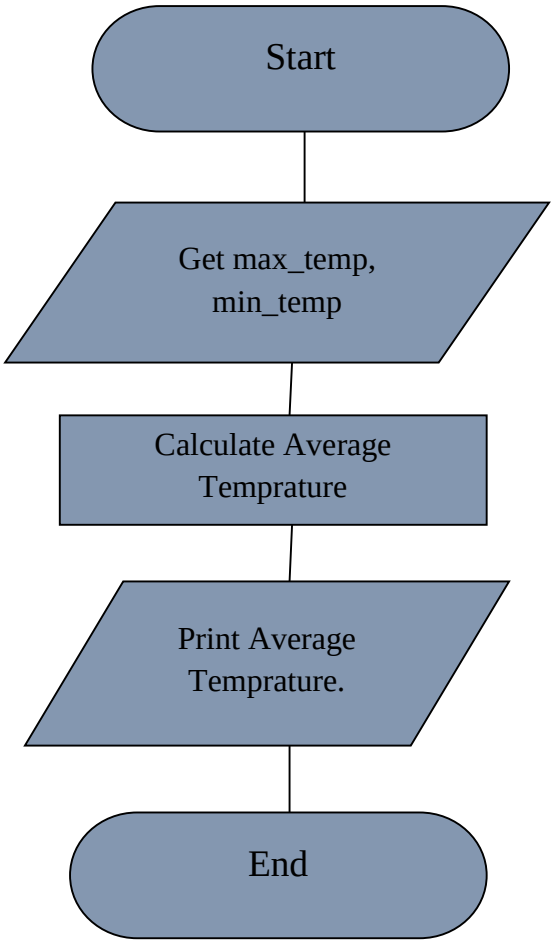


Table 1.1 Difference between Flowchart and Pseudo code

Flowchart	Pseudo code
➢ Flowchart is pictorial representation of flow of an algorithm.	➢ Pseudo code is a step-by-step description of an algorithm in code like structure using plain English text.
➢ Flowchart uses standard symbols for input, output decisions and start stop statements. Only uses different shapes like box, circle and arrow.	➢ Pseudo code uses reserved keywords like if-else, for, while, etc.
➢ This is a way of visually representing data, these are nothing but the graphical representation of the algorithm for a better understanding of the code	➢ These are fake codes as the word pseudo means fake, using code like structure but plain English text instead of programming language
➢ Flowcharts are good for documentation	➢ Pseudo code is better suited for the purpose of understanding

- **DFD's**

✓ Data Flow Diagram (DFD)

The Data Flow Diagram (DFD) is a graphical representation of the flow of data through an information system.

A structured analysis technique that employs a set of visual representations of the data that moves through the organization, the paths through which the data moves, and the processes that produce, use, and transform data. It enables you to represent the processes in your information system from the viewpoint of data. The DFD lets you visualize how the system operates, what the system accomplishes and how it will be implemented, when it is refined with further specification.

Data flow diagrams are used by systems analysts to design information-processing systems but also as a way to model whole organizations.

Why Data Flow Diagrams?

- Can diagram the **organization** or the **system**
- Can diagram the **current** or **proposed** situation
- Can facilitate **analysis** or **design**
- Provides a good bridge from analysis to design
- Facilitates communication with the user at all stages

Types of DFDs

- **Current** - how data flows now
- **Proposed** - how we'd like it to flow
- **Logical** - the "essence" of a process (i.e. implementation-independent and describe the system, rather than how activities are accomplished.)
- **Physical** - the implementation of a process (i.e. are implementation-dependent and describe the actual entities (devices, department, people, etc.) involved in the current system.)
- **Partitioned physical** - system architecture or high-level design

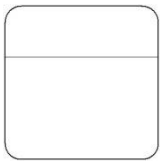
Levels of Details

Context Diagram- Shows just the inputs and outputs of the system

Level 0 Diagram- Decomposes the process into the major sub processes and identifies what data flows between them

Child diagrams- Increasing levels of detail (i.e. Level 1, Level 2 etc)

Four Basic Symbols



Process

Data Flow



Data Store



Source/Sink (External Entity)

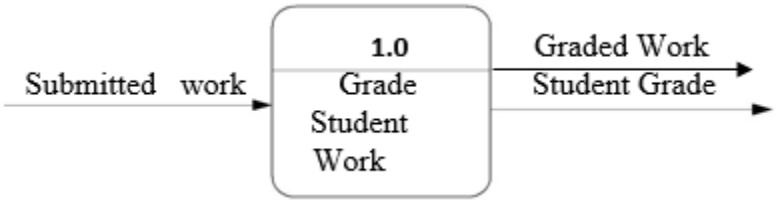
Process

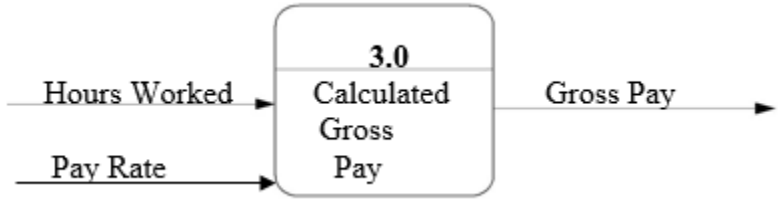


- Work or actions performed on data (inside the system)
- Labels should be **verb phrases**
- Receives input data and produces output

Rule 1: Process

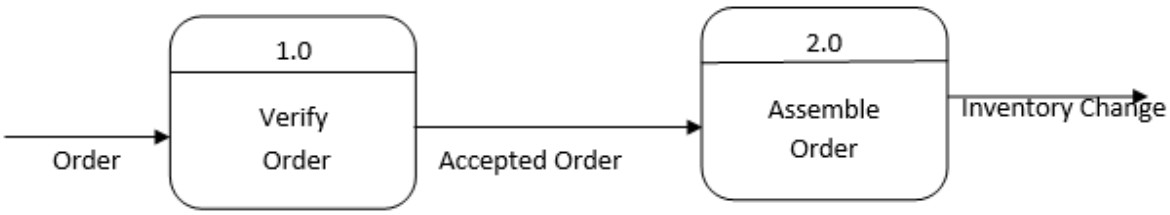
- Can have more than one outgoing data flow or more than one incoming data flow





Rule 2: Process

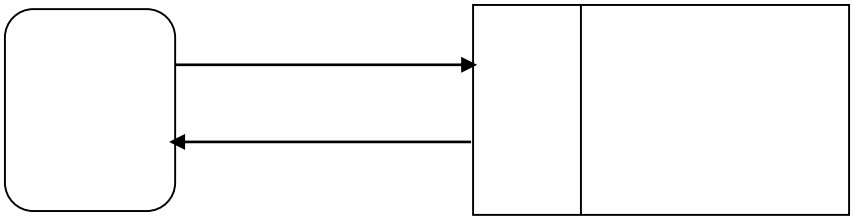
- Can connect to any other symbol (including another process symbol)



Data Flow Deposit



- Is a path for data to move from one part to another
- Arrows depicting movement of data
- Can represent flow between process and data store by two separate arrows



Data Store

D1	**Students**

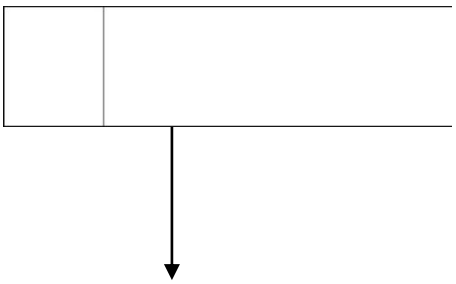
- Is used in a DFD to represent data that the system stores
- Labels should be noun phrases

Rule: Data Store

- Must have at least one incoming and one outgoing data flow

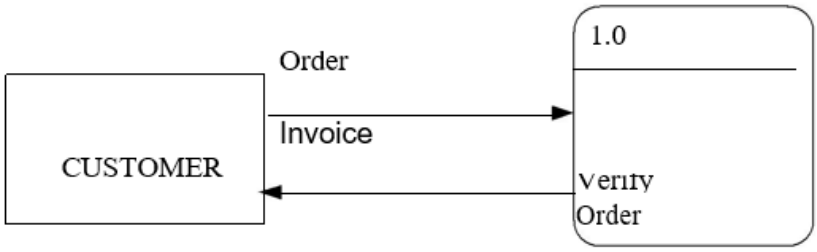


Customer Payment



Daily Payment

Source/Sink (External Entity)



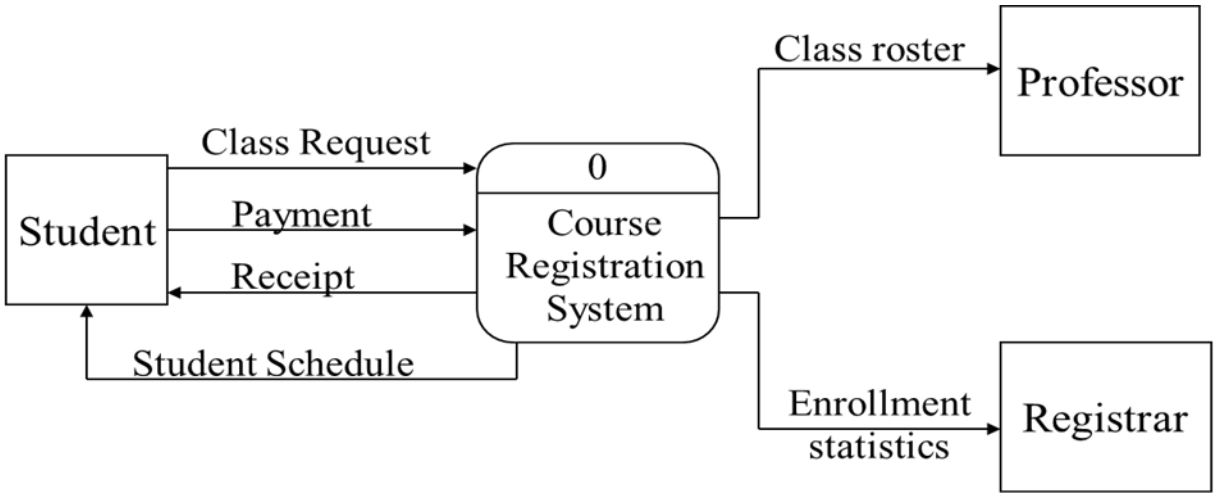
- External entity that is origin or destination of data (outside the system)
- Labels should be **noun phrases**
- Source –Entity that supplies data to the system
- Sink –Entity that receives data from the system

Context Level Diagram

A context diagram is a data flow diagram that only shows the top level, otherwise known as Level 0. At this level, there is only one visible process node that represents the functions of a complete system in regards to how it interacts with external entities. Some of the benefits of a Context Diagram are:

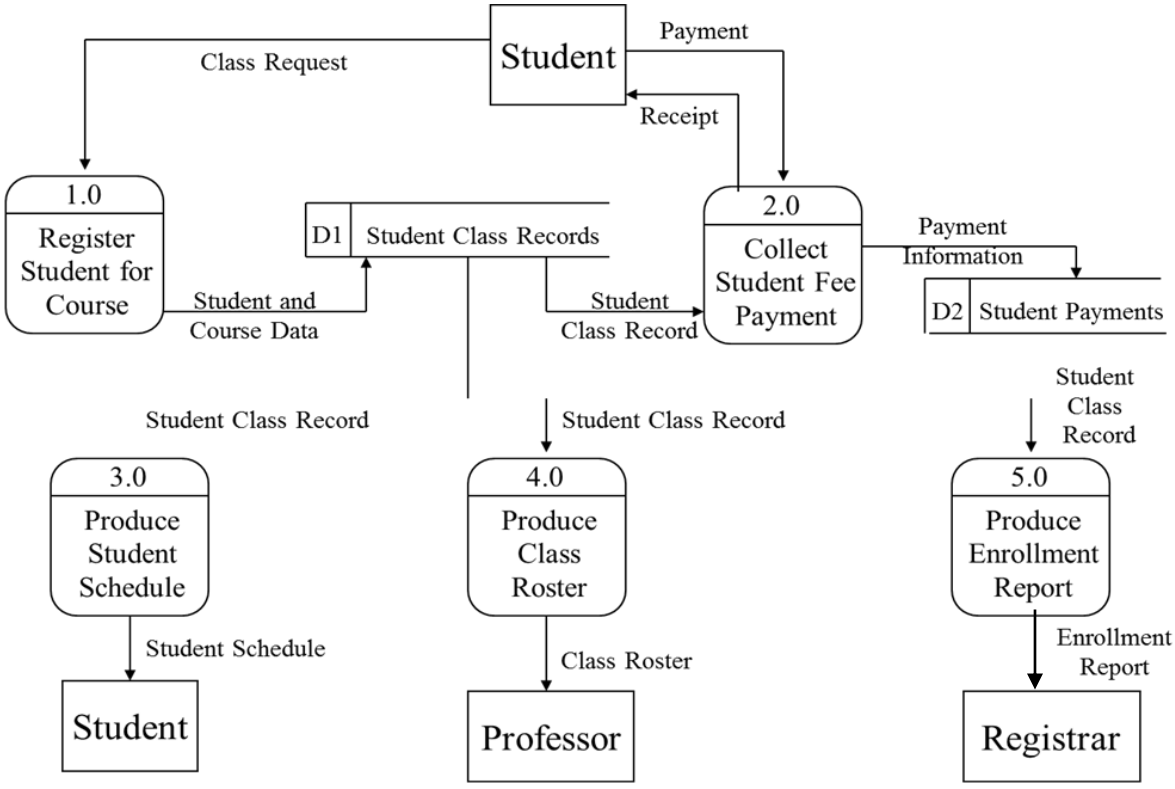
- Shows the overview of the boundaries of a system
- No technical knowledge is required to understand with the simple notation
- Simple to draw, amend and elaborate as its limited notation

Example: -Course Registration: Context level Diagram



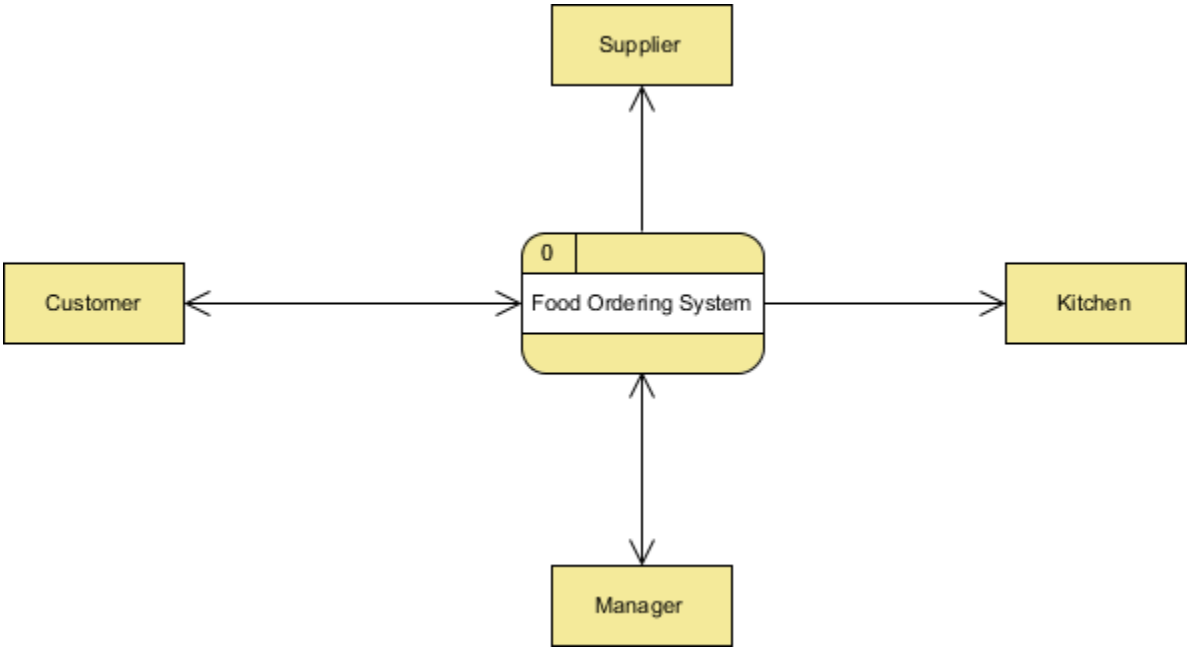
Level 0 Diagram

- Process is “exploded”
- Sources, sinks, and data flows repeated from context diagram
- Process broken down into sub processes, numbered sequentially
- Lower-level data flows and data stores added



Example: -Food Ordering System: Context level Diagram

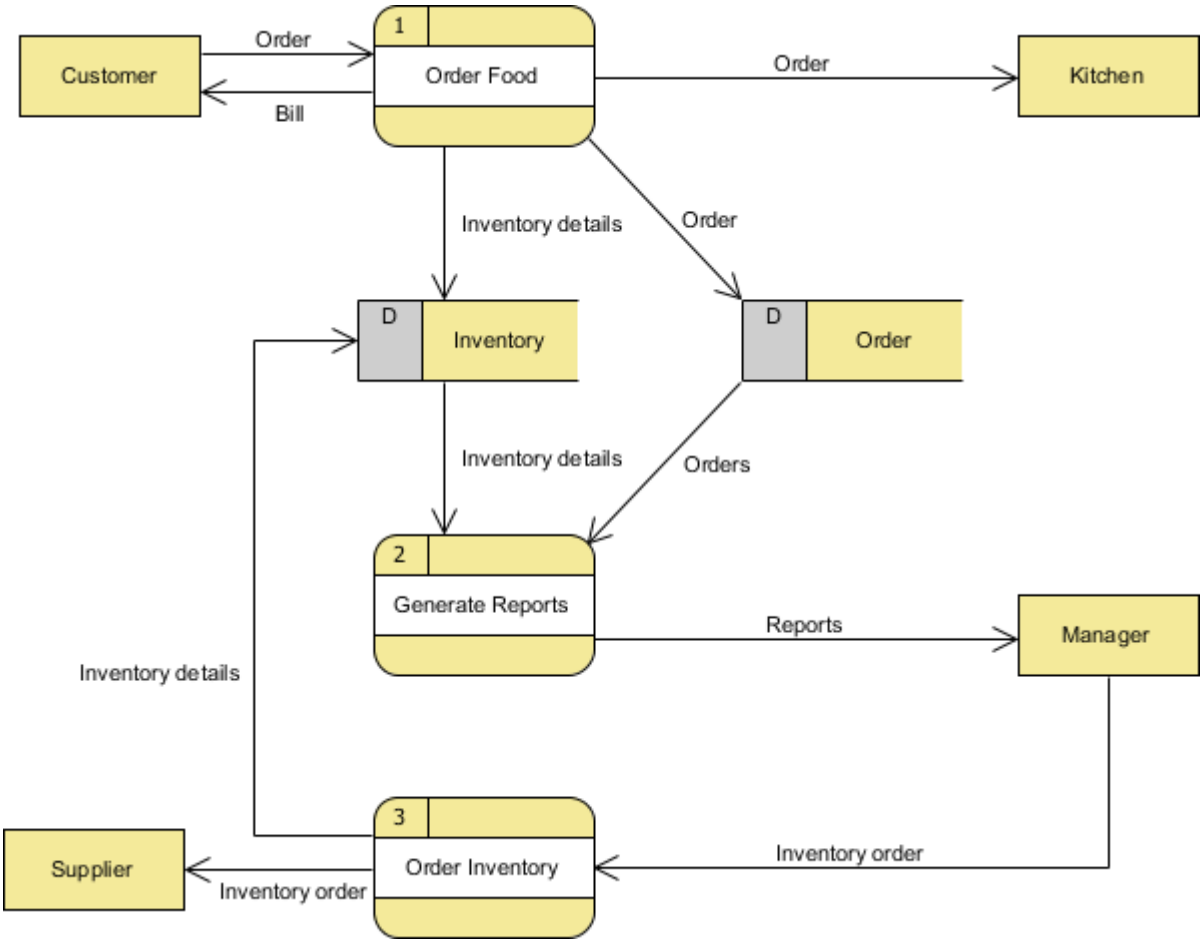
The figure below shows a context Data Flow Diagram that is drawn for a Food Ordering System. It contains a process (shape) that represents the system to model, in this case, the "Food Ordering System". It also shows the participants who will interact with the system, called the external entities. In this example, Supplier, Kitchen, Manager and Customer are the entities who will interact with the system. In between the process and the external entities, there are data flow (connectors) that indicate the existence of information exchange between the entities and the system.



Context DFD is the entrance of a data flow model. It contains one and only one process and does not show any data store.

Level 1 DFD

The figure below shows the level 1 DFD, which is the decomposition (i.e. break down) of the Food Ordering System process shown in the context DFD. Read through the diagram and then we will introduce some of the key concepts based on this diagram.



The Food Order System Data Flow Diagram example contains three processes, four external entities and two data stores.

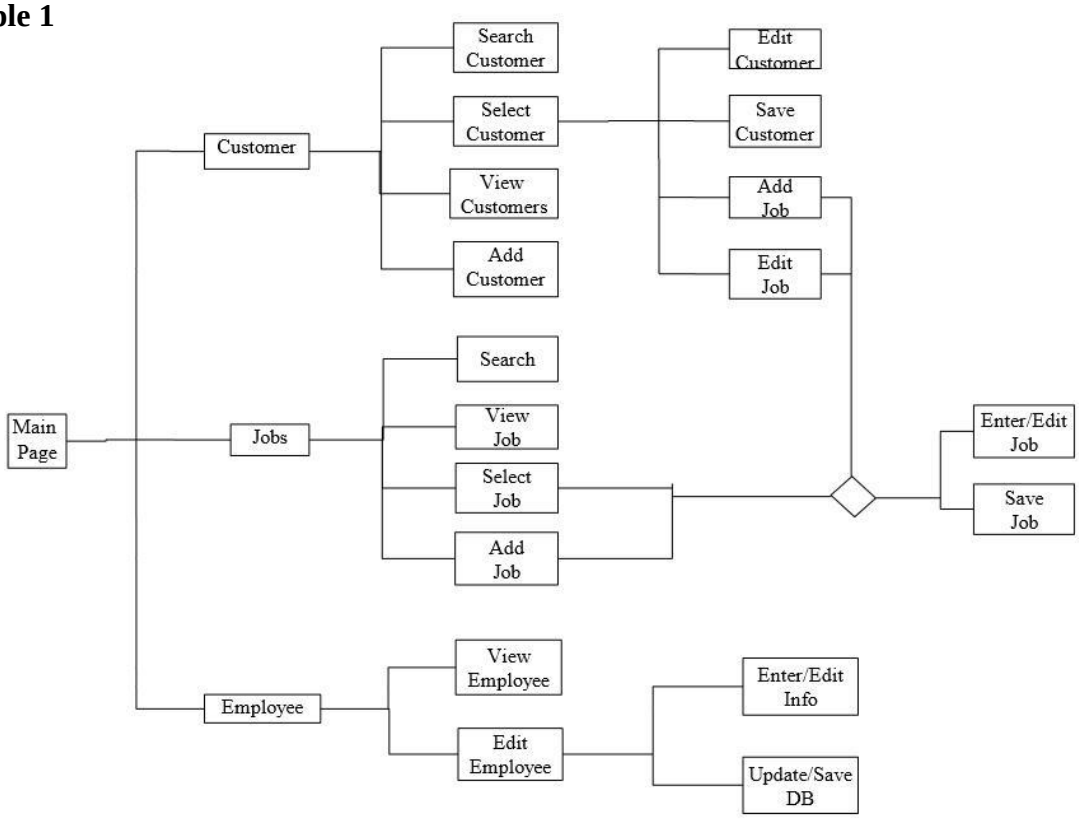
- ✓ Based on the diagram, we know that a Customer can place an Order. The Order Food process receives the Order, forwards it to the Kitchen, store it in the Order data store, and store the updated Inventory details in the Inventory data store. The process also deliver a Bill to the Customer.
- ✓ Manager can receive Reports through the Generate Reports process, which takes Inventory details and Orders as input from the Inventory and Order data store respectively.
- ✓ Manager can also initiate the Order Inventory process by providing Inventory order. The process forwards the Inventory order to the Supplier and stores the updated Inventory details in the Inventory data store.

- **Hierarchy Input Process Output Diagram (HIPO)**

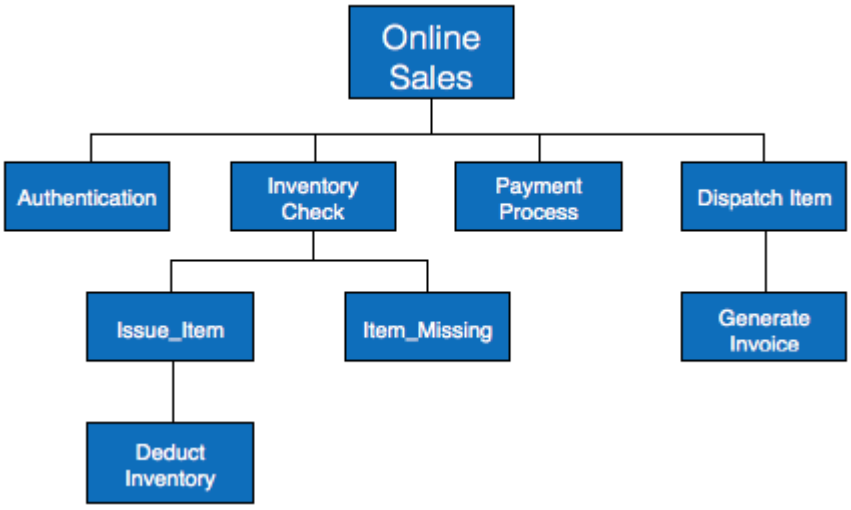
It is a systems analysis design aid and documentation technique from the 1970s, used for representing the modules of a system as a hierarchy and for documenting each module.

It is a hierarchy chart that graphically represents the program's control the functions (or processes) performed by each module on the system.

Example 1



Example 2



- **RAD**

What is RAD model- advantages, disadvantages and when to use it?

RAD model is Rapid Application Development model. It is a type of incremental model. In RAD model the components or functions are developed in parallel as if they were mini projects. The developments are time boxed, delivered and then assembled into a working prototype. This can quickly give the customer something to see and use and to provide feedback regarding the delivery and their requirements.

Diagram of RAD-Model:

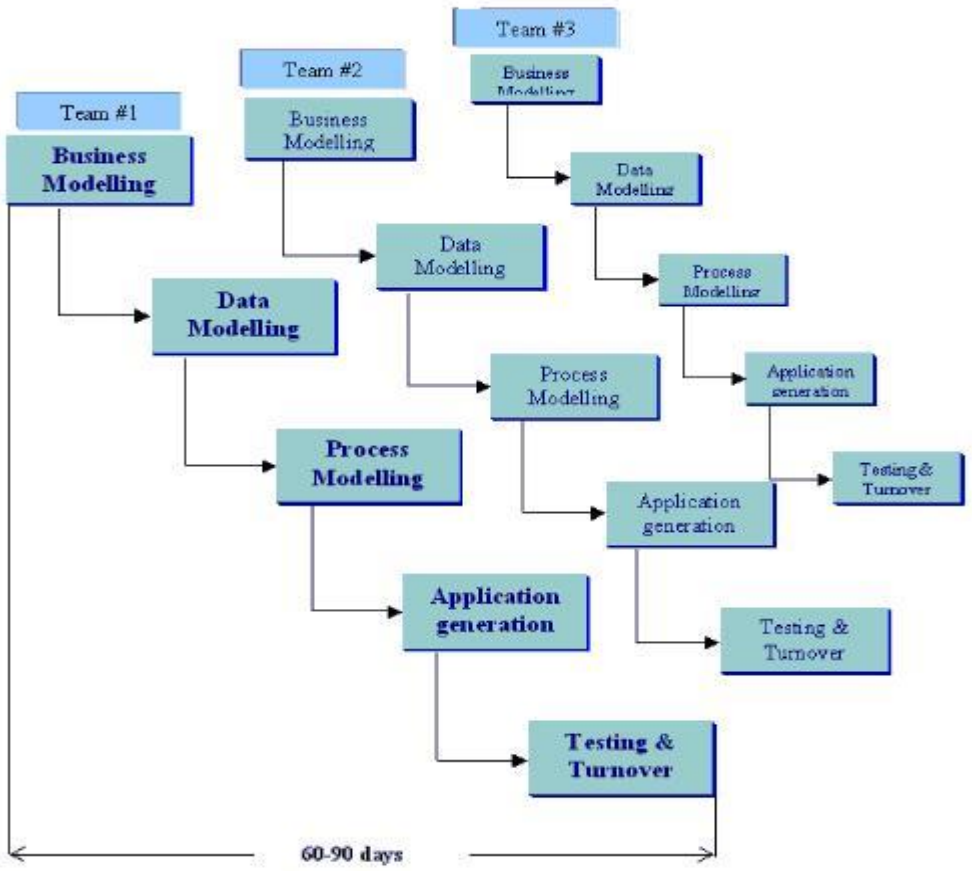


Figure 1.5 – RAD Model

The phases in the rapid application development (RAD) model are:

Business modeling: The information flow is identified between various business functions.

Data modeling: Information gathered from business modeling is used to define data objects that are needed for the business.

Process modeling: Data objects defined in data modeling are converted to achieve the business information flow to achieve some specific business objective. Description are identified and created for CRUD of data objects.

Application generation: Automated tools are used to convert process models into code and the actual system.

Testing and turnover: Test new components and all the interfaces.

Advantages of the RAD model:

- ✓ Reduced development time.
- ✓ Increases reusability of components
- ✓ Greater customer satisfaction
- ✓ Encourages customer feedback
- ✓ Faster delivery time
- ✓ simple and better quality

Disadvantages of RAD model:

- ✓ Depends on strong team and individual performances for identifying business requirements.
- ✓ Only system that can be modularized can be built using RAD
- ✓ Requires highly skilled developers/designers.
- ✓ It is not appropriate when technical risk is high
- ✓ High dependency on modeling skills
- ✓ Inapplicable to cheaper projects as cost of modeling and automated code generation is very high.

When to use RAD model:

- ✓ RAD should be used when there is a need to create a system that can be modularized in 2-3 months of time.
- ✓ It should be used if there's high availability of designers for modeling and the budget is high enough to afford their cost along with the cost of automated code generating tools.
- ✓ RAD model should be chosen only if resources with high business knowledge are available and there is a need to produce the system in a short span of time (2-3 months).

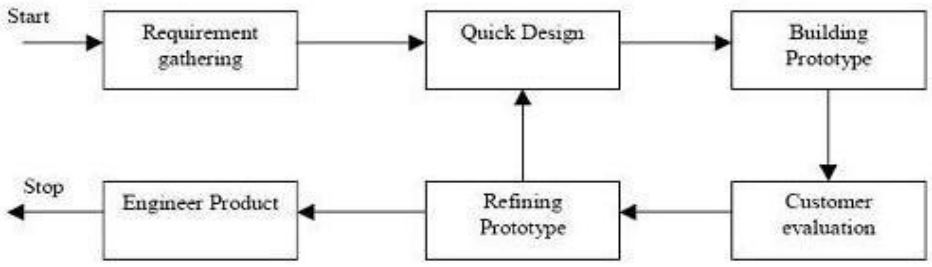
- **Prototyping**

What is Prototype model- advantages, disadvantages and when to use it?

This prototype is developed based on the currently known requirements. By using this prototype, the client can get an "actual feel" of the system, since the interactions with prototype can enable

the client to better understand the requirements of the desired system. Prototyping is an attractive idea for complicated and large systems for which there is no manual process or existing system to help determining the requirements. The prototypes are usually not complete systems and many of the details are not built in the prototype. The goal is to provide a system with overall functionality.

Diagram of Prototype model:



Prototyping Model

Advantages of Prototype model:

- ✓ Users are actively involved in the development
- ✓ Errors can be detected much earlier.
- ✓ Quicker user feedback is available leading to better solutions.
- ✓ Missing functionality can be identified easily
- ✓ Confusing or difficult functions can be identified

Disadvantages of Prototype model:

- ✓ It's a slow in process
- ✓ It's not complete model
- ✓ It is time consuming model
- ✓ It should be developed or built on company's cost

When to use Prototype model:

- ✓ Prototype model should be used when the desired system needs to have a lot of interaction with the end users.

- ✓ Typically, online systems, web interfaces have a very high amount of interaction with end users, are best suited for Prototype model. It might take a while for a system to be built that allows ease of use and needs minimal training for the end user.
- ✓ Prototyping ensures that the end users constantly work with the system and provide a feedback which is incorporated in the prototype to result in a useable system. They are excellent for designing good human computer interface systems.

- **Case Tools**

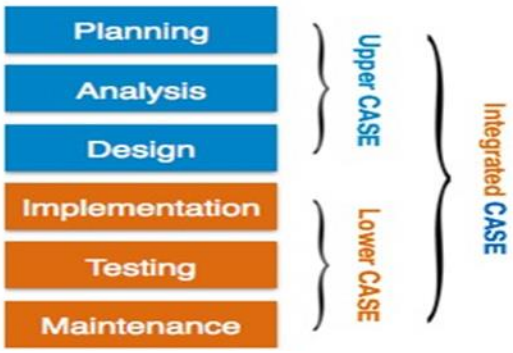
CASE (computer-aided software engineering) tools have been widely in use since the 1990's. They are specialized, computerized tools that aid in the design, development and maintenance of software. The term CASE was originally coined by software company, Nastec Corporation of Southfield, Michigan in 1982 with their original integrated graphics and text editor GraphiText.

Many different definitions of CASE tools exist, below are just a few of those:
Galin - CASE tools are computerized software development tools that support the developer when performing one or more phases of the software life cycle and/or support software maintenance.

- ✓ **Components of CASE Tools**

CASE tools can be broadly divided into the following parts based on their use at a particular SDLC stage:

- ✓ Central Repository - Central repository is a central place of storage where a product specification, requirement documents, related reports and diagrams, other useful information regarding management is stored. Central repository also serves as data dictionary.



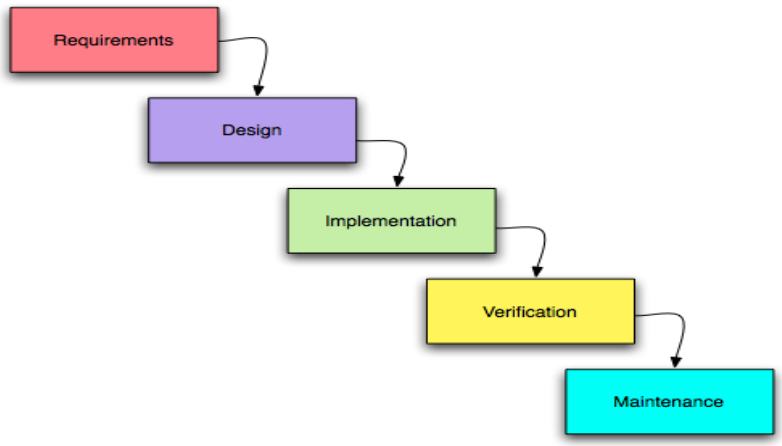
Types of CASE tools

Classic CASE tools - established software development support tools (e.g. interactive debuggers, compilers, etc.)

Real CASE tools - can be separated into three different categories, depending on where in the development process they are most involved in:

- Upper - support analysis and design phases
- Lower - support coding phase
- Integrated - also known as I-CASE support analysis, design and coding phases

Upper and lower CASE tools are named as such because of where the phases they support are in the Waterfall Model (see below)



Why we use CASE tools?

CASE tools offer:

- ✓ Savings in resources required for software development - with less
- ✓ Quicker development phase (i.e. shorter time to market)
- ✓ Reduction of generation of defects
- ✓ Easier identification of defects during development
- ✓ Savings in maintenance resources required
- ✓ Greater standardization of software systems and therefore increased reuse possibilities

Self-check-1

Part-I: Choose the best answer

1._____provide a simplified picture of the relationships between the program activities and the desired outcomes of the program

A. Logic models
B. Program planning.
C. Program implementation.
D. None

2._____Logic model is valuable in supporting all the following except one.

A. Program planning.
B. Program implementation.
C. Program monitoring.
D. Program execution.

3._____It defines a picture of why and how you believe a program or a policy will work.

A. Program planning.
B. Program implementation.
C. Program logic
D. Program monitoring.

4. We use a logic model for all the following activities except one.

A. Brings detail to broad goals.
B. Helps identify gaps in program logic and clarify assumptions.
C. Builds understanding and promotes consensus.
D. Makes implicit underlying beliefs.

5. One of the following will be the starting point in many cases you should do before doing any of the actual work of program.

A. Design Document
B. Coding program
C. Testing program

Part-II: Answer the following questions briefly

1. What is systems development life cycle?
2. List three benefits of SDLC?
3. What is programing algorithm?
4. What is the advantage of pseudo cod?
5. What is Data Flow Diagram (DFD)?
6. What are the Advantages of the RAD model?
7. List three the disadvantages of Prototype model?

Operation sheet 1.1. Designing Program Logic

Operation Title: create flow chart **proposal development process use Microsoft Visio 2007 and 2010.**

Conditions or situations for the operations:

- ✓ Safe working area
- ✓ Properly operated tools and equipment

Equipment Tools and Materials:

- ✓ computer
- ✓ Microsoft Visio 2007 and 2010.
- ✓ Edraw Max
- ✓ Visual Paradigm Online

Steps in doing the task

1. Start **Visio.**
2. Click the **Flowchart category**.
3. Double-click **Basic Flowchart.**
4. For each step in the process that you are documenting, drag a flowchart shape onto your drawing.
5. Connect the flowchart shapes by holding the mouse pointer over the **first shape**, and then clicking the **small arrow** that appears that points to the shape you want to connect to. If the second shape is not directly across from the first shape, click and hold the small arrow, **drag it to the second shape**, and **drop the connector** in the middle of the second shape.
6. To add text to a shape or connector, select it, and then type. When you are finished typing, click on a blank area of the page.
7. To change the direction of a connector's arrow, select the connector, and then, on the **Shape tab**, in the **Shape Styles** group, **click Line**, point to **Arrows,** and select the arrow direction and style that you want.

Flow chart proposal development process steps:

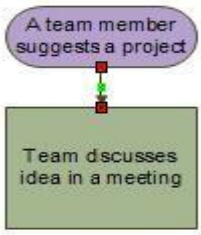
1. On the File menu, point to new, point to **Flowchart** and then click **Basic Flowchart,** then **click Ok.**
2. For each step in the process that you are **documenting, drag a flowchart shape** onto your drawing. First see the section what the flowchart shapes represent for information.
3. Connect the flowchart shapes in either of the following ways. For information on other ways to connect shapes, **see Add and glue connectors** with the **Connector tool.** Proposal Connect two shapes together.

   Connect two shapes together.
4. Click the Connector tool **on the Standard toolbar**

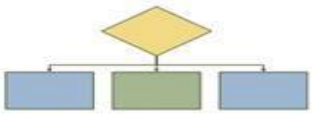




5. Drag from a connection point on the first shape to a connection point on the second shape. The connector endpoints turn red when the shapes are connected



6. Connect one shape to many from a single connection point. By default connectors are set to **Right-Angle** so that if you connect a single point on one shape to three other shapes it will look like the figure below.



To have each connector radiate straight from the central point on the first shape to points on each of the other shapes you need to set the connectors to **Straight Connector** as shown in the following figure.



7. **Click the Connector tool**  **on the Standard toolbar**
8. For each shape you want to connect to, drag from the same connection point on the first shape to a connection point on each of the other shapes
9. **Right-click each connector and click Straight Connector**
10. **Click the Pointer tool**  on the Standard toolbar to return to normal editing
11. To add text to a shape or connector, select it, and then type. When you are finished typing, click on a blank area of the page.
12. To change the direction of a connector's arrow, **select the connection**, and then on the **Shape menu**, point to Operations, and click **Reverse Ends.**

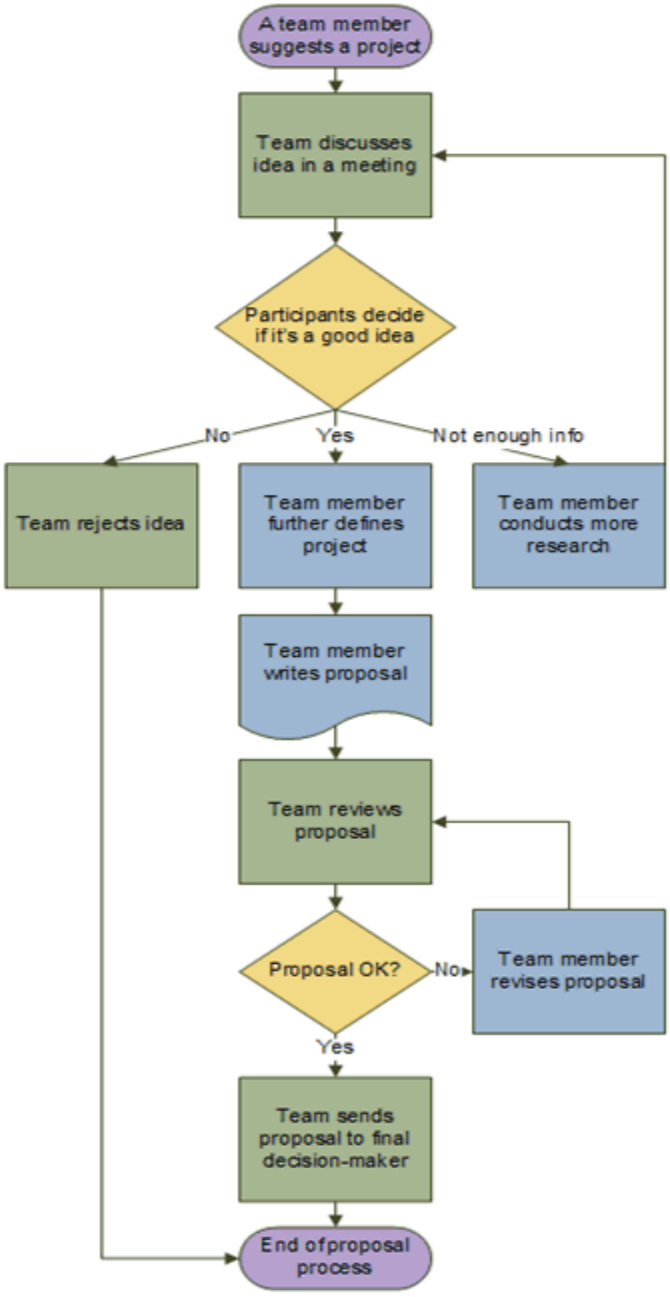


Figure 1: Flowchart of proposal development process

Quality Criteria: To ensure that the final product meets the required quality standards of proposal development process.

Lap Tests

Instructions: Given necessary templates, tools and materials you are required to perform the following tasks accordingly.

Task 1: Recognize Symbols Used in a Flowchart and List Logical Process to Solve a Problem

Task 2: Draw the Flowchart to Illustrate the Problem Solving Process

Unit Two: program logic Documentation

This unit is developed to provide you the necessary information regarding the following content coverage and topics:

- Project planning standards
- documentation of Project scope
- project scope statement
- Identification and revising References.
- Logic design templates
- Logic design validation

This unit will also assist you to attain the learning outcomes stated in the cover page.

Specifically, upon completion of this learning guide, you will be able to:

- Understand Project standards
- Understand and document Project scope
- Understand project scope statement
- Identify and revising References
- Understand logic design templates
- Check Logic design validation

2.1. Project planning standards

Project planning standards are set to attain the project goals. Project planning standards may obviously vary from project to project, but the goals are usually the same to complete the project within the timeframe and without exceeding the allotted resources.

Any effective project planning standards must lay down project management methodologies, provide a step-by-step guide and inputs for setting standards for project management, facilitate project reporting and offer required documentation throughout the tenure of the project.

- **Setting Standards for Project Management and the Creating Project Plan**

Project planning standards should be able to clearly define general project activities and address the specific requirements of individual projects. The project standards should provide adequate details to ensure all team members can identify project objectives and relate to its fulfillment. Clearly laid-down objectives are a basic prerequisite for timely and successful completion of projects. The project standards should, as far as possible, involve all persons who can meaningfully contribute to the functional requirements of a project.

Project planning standards must take into account the overall project management system, its merits and limitations, define goals of the project, organize the information system with easy identification of project objectives and also plan for bridging a system network to monitor and control the projects efficiently. The more elaborate the project planning standards are, the more effective they will be to enhance the productivity of team members, allowing all project team members to better understand the project objectives.

Organizations that have to manage multiple projects through effective collaboration and coordination should establish planning standards for coordinating and managing the various projects. The project planning standards for coordinating multiple projects should outline procedures for project prioritization, resource utilization and the project status updating and reporting.

- **Project Planning Standards - Parameters and Applications**

Project planning standards, for them to be true to definition, should include features, such as:

- ✓ Quality Assurance Necessities
- ✓ Risk Management Plan

- ✓ Security Measures
- ✓ Testing Techniques
- ✓ Documentation and Portfolio Requirements

No project plan standards can be deemed complete without cost details, staffing needs, resource deployment (identifying project participants and project management tools) and training facilities. Project planning standards should also include configuration management standards aimed at minimizing disruptions affecting a project management system and ensuring smooth flow of project progress. Configuration standards should be evolved and put in place to ensure all steps are scrutinized properly, approved after thorough evaluation and drafted and documented as per requirement, and information shared.

Quality assurance also forms an indispensable part of project planning standards and quality assurance procedures should, amongst other things, ensure commitment from all team members. Audit and compliance authorities should assist quality assurance personnel to verify, at every stage, conformity of project requirements (internal or external). Project scalability varies from project to project. Quality assurance standards must obviously live up to the project size, project traits and the risks involved.

Needless to say, risk management standards are critically important while implementing project planning standards. Risk management standards must recognize internal and external project risks and include procedures for identifying and managing such risks.

No less important in project planning is establishing appropriate testing standards based on test plans which are comprehensive (subjective) in nature. These testing standards are the outcome of active participation of end-users and proper documentation of project progress at every stage. Here it is important to note that if copies of customer/client (user-oriented) data are used during test management procedures, then it becomes extremely important to set standards for the protection of data confidentiality.

Finally, care must be exercised to ensure that documentation standards comprehensively cover all details, system methodologies and technological resources. Documentation is critical for helping the team members to periodically review and modify the technology applications, system operability and standard project management procedures whenever necessary.

2.2. Documentation of Project scope

The larger the project, the more important it becomes to use CASE tools in the software's development. The standardization provided by using CASE tools allows many developers to work on one project without spending time working out what certain sections of code does, it also enables easier debugging which becomes more important with a project that involves a large amount or complex coding as software such as this is more likely to have errors.

The **purpose** of a **scope and limitations** statement is to **provide an outline of what your project will address and what it won't address**. What you turn in will include the following:

- ✓ a description of the project (what technology you are addressing and the rationale for choosing that technology),
- ✓ an explanation of what kind of data you will collect, the methods you will use (interviews, library research, observation, etc.), and how much of each kind of data you will collect,
- ✓ an explanation of how you are approaching the challenge of making something 'better' (making it less expensive, easier to use, available to a more diverse population, more environmentally friendly, etc.),
- ✓ A proposal about how you will test your initial design suggestions. Remember the design guidelines provided during Unit 1, and address these as you outline your approach, and
- ✓ A timeline with major tasks to be accomplished for the project and which team member will take a leadership role and responsibility for the task.

While the actual write-up you hand in may be as short as one page, I expect a significant level of detail in your descriptions of your approach, and I also expect that the paper will reflect a considerable amount of thought and the contributions of each group member. Keep in mind that my expectation is that three or four people will be contributing content to this statement, and the work should reflect that level of collective effort.

A project scope document sometimes called a scope of work (SOW) is a critical piece of project paperwork that gets teams and stakeholders aligned on the boundaries of a project before it even begins. A well-defined scope document can save you from major headaches by defining the following project elements:

- ✓ Project goals
- ✓ Requirements
- ✓ Major deliverables
- ✓ Key milestones
- ✓ Assumptions
- ✓ Constraints

These critical scope aspects enable you to say no more easily when new requests arise as you are trying to deliver a project on time and under budget. In the end, a well-documented scope statement gets everyone team and stakeholders alike aligned around these important details that can make or break a project.

2.3. Project scope statement

A scope of work (SOW) document is an agreement on the work you're going to perform on the project. The document includes it's a section of the document that delineates the major phases across the schedule of the project's duration. It should also mark the points in the project when your deliverables are ready. There's no doubt that a lot of thought, discussion, and sometimes even debate goes into finalizing a solid scope. But all that work is worth it because having a well-considered scope document can increase your chances of leading a project to successful completion. There are lots of different ways to write a scope statement.

- **The list of possible elements you should consider adding to your project scope statement.**

- ✓ Business case and goals: Every project has goals, and this is where you'll define them. This typically includes the reasons the project is being supported (or funded), along with a set of business goals or intended project outcomes for your team to keep in mind while executing the project. These details are critical to document because there will be times when stakeholder (and sometimes even team) requests creep in and put your timeline and budget at risk. But you can push those risks away if change requests don't meet the documented business case.
- ✓ Project description and deliverables: This is a plain language overview of the project's deliverables. Avoid confusion by clearly outlining what will be delivered for approval

through the course of the project, as well as the final deliverable. For instance, if you're creating a database stores student data for a client, you might say something like: The student database stores all the information of students and subjects lists for easily manageable student profiles.

- ✓ Acceptance criteria: Your scope should help you come to an agreement on what will be delivered and leave no question when the project is complete. Acceptance criteria can be measured, achieved, and used to prove that work is complete. Examples of some of the conditions or criteria of acceptance can be found in project requirements, user acceptance testing, or even just a final stakeholder review and approval.
- ✓ Limitations: Every project has its limits, and you need to be sure you're not exceeding those limits to complete a project on time and under budget. Limitations can come in many forms, but one example would be technology. For instance, if you're building an application that depends on a specific technology, be sure to mention that. There may be several ways to code that website, but if you're boxed into a complicated technology, you can cover yourself by specifying those limitations in your scope. Doing so will help you when you run into a limitation and don't have the time or budget to explore alternatives. Think of it as an insurance policy for your project.
- ✓ Assumptions: You know what they say about assumptions, and you probably know it's true. If you don't outline them, you'll end up with confusion, missed expectations, and project problems. So take time to list out all the assumptions you've thought about that will affect the work you'll do or the outcomes of that work.
- ✓ Exclusions: You've already listed out the deliverables you will provide, but sometimes it's just as important to itemize what you will NOT deliver. This helps you avoid awkward "But weren't you going to" questions or requests. Really, it's about setting expectations and avoiding any miscommunication around the work you have planned.
- ✓ Costs: This is an optional portion of your project SOW, depending on the type of organization you work in. If you're part of a consulting agency that charges external clients for your work, you'll want to outline project costs, possibly even on the phase or milestone level. You have to do what feels right for your project and organization. But the clearer

you can be about costs and the work associated with it, the easier it will be for you to manage it and make a case for more funds when additional scope creeps in.

- ✓ Agreement: Scope documents create agreement by nature, but sometimes you need proof. So include a signature field in your scope document and have your lead stakeholder or project funder sign the document. On that note, it's important to remember that if you're collecting money for the work or if there are high stakes you'll likely want to have your scope document reviewed by a lawyer before it's signed. After all, the scope document is a contract.

2.4. **Identification and revising References**

MDS allows the definition of business rules to enforce data quality and consistency. This ensures that the entered data adheres to predefined standards and validation criteria.

Master Data Services (MDS): Master data service implementations may vary; the services can be characterized within three layers:

- ✓ **Core services**, which focus on data object life cycle actions, along with the capabilities necessary for supporting general functions applied to master data
- ✓ **Object services**, which focus on actions related to the master data object type or classification (such as customer or product)
- ✓ **Business application services**, which focus on the functionality necessary at the business level

- **Reference Tables**

- ✓ **Calendar Tables:** Tables that display days, weeks, and months in a calendar format, often used for scheduling and planning. Reference data is not purely descriptive. It lives in the context of other information. This is where reference tables come into play.

Reference tables are used to store information that is commonly used to set up context and describe other business keys. In many cases, these are standard codes and descriptions or classifications of information. The most basic reference table is just a typical table in third or second normal form. This basic table is used when there is no need to store history for the reference data. That is often the case for reference data that is not going to change or that will change very seldom. Typical example, calendar dates.

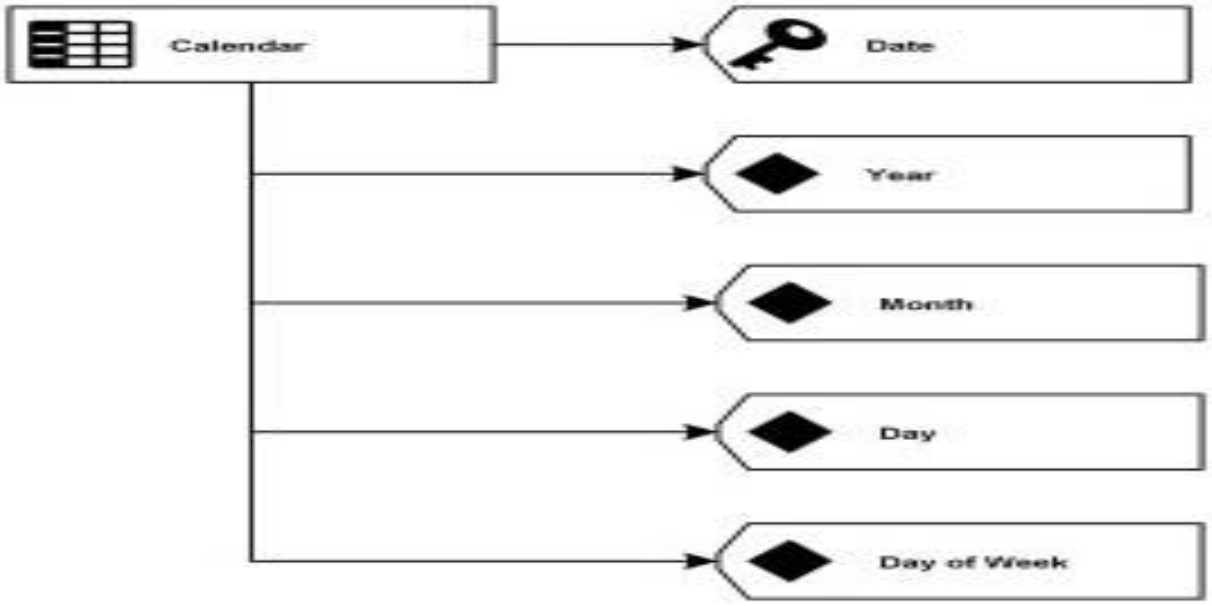


Figure 2.1: Figure for a non-histories reference table for calendar (logical design)

2.5. Logic design templates

Programming logic model template is a template that is simply-formatted to use when developing a program's goals and objectives.

Table 2.6 sample Logic design templates

Project: **Goal:**					
INPUTS	**ACTIVITIES**		**OUTCOMES**		
What we invest	**What we do**	**Who we reach**	**Why this project: shortterm results**	**Why this project: intermediate results**	**Why this project: long-term results**

Assumptions	***External Factors***

2.6. Logic design validation

- **Verification and Validation**

Software verification provides objective evidence that the design outputs of a particular phase of the software development life cycle meet all of the specified requirements for that phase. Software verification looks for consistency, completeness, and correctness of the software and its supporting documentation, as it is being developed, and provides support for a subsequent conclusion that software is validated. Software testing is one of many verification activities intended to confirm that software development output meets its input requirements. Other verification activities include various static and dynamic analyses, code and document inspections, walkthroughs, and other techniques.

Software validation is a part of the design validation for a finished device, but is not separately defined in the Quality System regulation. For purposes of this guidance, FDA considers software validation to be "confirmation by examination and provision of objective evidence that software specifications conform to user needs and intended uses, and that the particular requirements implemented through software can be consistently fulfilled." In practice, software validation activities may occur both during, as well as at the end of the software development life cycle to ensure that all requirements have been fulfilled. Since software is usually part of a larger hardware system, the validation of software typically include evidences that all software requirements have been implemented correctly and completely and are traceable to system requirements. A conclusion that software is validated is highly dependent upon comprehensive software testing, inspections, analyses, and other verification tasks performed at each stage of the software development life cycle. Testing of device software functionality in a simulated use environment, and user site testing are typically included as components of an overall design validation program for a software automated device.

Software verification and validation are difficult because a developer cannot test forever, and it is hard to know how much evidence is enough. In large measure, software validation is a matter of developing a "level of confidence" that the device meets all requirements and user expectations for the software automated functions and features of the device. Measures such as defects found

in specifications documents, estimates of defects remaining, testing coverage, and other techniques are all used to develop an acceptable level of confidence before shipping the product. The level of confidence, and therefore the level of software validation, verification, and testing effort needed, will vary depending upon the safety risk (hazard) posed by the automated functions of the device.

- **Gaining Feedback or Input**

The great leadership guru Ken Blanchard likes to say that "Feedback is the breakfast of champions." He is right; getting high-quality feedback from customers can make a phenomenal difference to a company's bottom line. Yet so, few business owners actually seek feedback on their performance. Maybe it is because they are too busy serving their customers, or perhaps it is because they are scared of what their clients might say when asked. Regardless, not asking for feedback is a crazy mistake.

Good customer feedback can help to do a number of things, including:

- ✓ Improve products.
- ✓ Alter pricing structure.
- ✓ Retain customers for longer.
- ✓ Identify high- and low-achieving staff.

A. Ask new customers why they chose your service brand. This can be hard to do as many customers will feel uncomfortable speaking with a company representative directly. For this reason, it is important to establish a comfortable rapport so that customers feel they may speak openly and honestly. An alternative is to hire a consultant to perform this survey as a third party may uncover sensitive information more readily**.**

B. Ask current and past customers what differentiates your service brand from others. Another approach is to ask a customer what thought or image first comes to mind when they consider your company. Then the first thought or image when considering a competitor's brand. There is no right or wrong answers. Take each answer at face value and recognize that the stated differences will be very instructive in terms of how your brand is being perceived by your target audience. Any misalignments uncovered with regard to your branding message and audience perception can then be readily addressed**.**

C. **Who are your repeat customers?** Look for areas of commonality among your repeat customers. This is an important clue to how your brand is perceived in the marketplace. For example, do your customers seem to fall within a certain age or demographic group? Contact these repeat patrons to understand what drives their customer loyalty.

D. **Is your business obtaining referrals?** If your brand is perceived accurately by customers, then word-of-mouth is a great source of new business. What customers say about your company's brand speaks louder than any promotional message.

E. **Contact customers who chose a competitor.** Assume a friendly tone and stress that your call is not sales-oriented. You simply wish to follow up on why they didn't choose your firm. Again, a consultant performing this survey may obtain more detailed information during a conversation with a company employee.

F. **Is your brand an integral part of your core values?** Both Starbucks and Union Square align their service branding strategy with company values. Company core values are just that: they are the rock upon which your firm is built and, as such, unchanging. Building your service brand on core values ensures message consistency and facilitates the development strong and attractive of a brand image in the minds of your audience.

Spending time understanding the perception of your service brand is a highly valuable exercise. It ensures that your firm is reaching its intended audience and aligning customer expectations with service delivery.

Continually seeking feedback through a variety of sources can help you identify trends, gain competitive advantages, and cut costs. Failure to seek feedback can cost you customers. And remember, you can't make a profit if you're not satisfying your customers.

Self-check-2

Part-I: Say TRUE or FALSE

1. _____Project planning standards are set to attain the project goals
2. _____object services focus on data object life cycle actions, along with the capabilities necessary for supporting general functions applied to master data?
3. _____Project scope document also called a scope of work (SOW)?
4. _____Quality assurance standards must obviously live up to the project size, project traits and the risks involved.

Part-II: Answer the following questions briefly

1. What is Reference Tables?
2. List at least three good customer feedbacks?
3. What is Software validation?
4. What is Software verification?
5. List at least three project elements?

Reference

Books

- Programming Logic and Design, Introductory, 9th Edition Joyce Farrell - ©2018 (Author)
- Programming Logic and Design, Introductory, 9th Edition Joyce Farrell - ©2018 (Author)
- Programming Logic and Design, Comprehensive 8th Edition by Joyce Farrell (Author)
- Fundamentals of Database System 4th Edition Ramez Elmasri
- Beginning SQL Server 2005 For Developers (2006) Robin Dewson,
- Database Design for Mere Mortals™, Second Edition
- Introduction to SQL: Mastering the Relational Database Language, Fourth Edition/20th Anniversary Edition By Rick F. van der Lans

URL

https://www.techtarget.com/searchsoftwarequality/definition/systems-development-life-cycle

https://www.geeksforgeeks.org/how-to-write-a-pseudo-code/

https://www.geeksforgeeks.org/what-is-a-flowchart-and-its-types/

https://www.breezetree.com/articles/what-is-a-flow-chart

Developer's Profile

No	Name	Qualification	Field of Study	Organization/ Institution	Mobile number	E-mail
1	Frew Atkilt	M-Tech	Network & Information Security	Bishoftu Polytechnic College	0911787374	frew.frikii@gmail.com
2	Gari Lencha	MSc	ICT Managment	Gimbi Polytechnic	0917819599	Garilencha12@gmail.com
3	Kalkidan Daniel	BSc	Computer Science	Entoto Polytechnic	0978336988	kalkidaniel08@gmail.com
4	Solomon Melese	M-Tech	Computer Engineering	M/G /M /Polytechnic College	0918578631	solomonmelese6@gmail.com
5	Tewodros Girma	MSc	Information system	Sheno Polytechnic College	0912068479	girmatewodiros@gmail.com
6	Yohannes Gebeyehu	BSc	Computer Science	Entoto Polytechnic College	0923221273	yohannesgebeyehu73@gmail.com