DATABASE ADMINSTRATION Level III

Based on August, 2011, Version 3 Occupational Standards (OS) and curriculum

Module Title : Designing Program Logic

LG Code: EIS DBA3 M06 1220 Lo (1-3) LG (21-23)
TTLM Code: EIS DBA3 TTLM06 1220v1

December, 2020
Bishoftu, Ethiopia

Table of Contents

L #21	LO #1- Select the Program Design Approach

Instruction sheet

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

- Obtaining Design documentation
- Reviewing and clarifying requirements for the program
- Determining Design approach

This guide 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:

- Obtain Design documentation
- Review and clarifying requirements for the program
- Determine Design approach

Learning Instructions:

Read the specific objectives of this Learning Guide.

1. Follow the instructions described below.
2. Read the information written in the “Information Sheets”. Try to understand what are being discussed. Ask your trainer for assistance if you have hard time understanding them.
3. Accomplish the “Self-checks” which are placed following all information sheets.
4. Ask from your trainer the key to correction (key answers) or you can request your trainer to correct your work. (You are to get the key answer only after you finished answering the Self-checks).
5. If you earned a satisfactory evaluation proceed to “Operation sheets”
6. Perform “the Learning activity performance test” which is placed following “Operation sheets” ,
7. If your performance is satisfactory proceed to the next learning guide,
8. If your performance is unsatisfactory, see your trainer for further instructions or go back to “Operation sheets”.

Information Sheet 1.1: Obtaining Design documentation

Introduction

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.

1.1 Obtaining 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.

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.

Self-Check 1.1	**Written Test**

Directions: Answer all the questions listed below. Use the Answer sheet provided in the next page:

I. Choose the best answer (each 1 point)

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

Note: **Satisfactory rating - 3 points** **Unsatisfactory - below 3 points**

You can ask you teacher for the copy of the correct answers.

Score = ___________

Rating: ____________

Information Sheet 1.2: Reviewing and clarifying requirements for the program

1.2 Reviewing and clarifying requirements for the program

A 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 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

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

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.

Coding the program:

- Select the programming language
- Write the instructions

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

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

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

Self-Check 1.2	**Written Test**

Directions: Answer all the questions listed below. Use the Answer sheet provided in the next page:

I. Choose the best answer (each 1 point)

1. The aim of a program of requirements is _____
   A. to lay down a clear framework for everyone involved in the project.
   B. to set the criteria for everyone involved in the project.
   C. to inform the criteria for everyone involved in the project.
   D. All

2. Which one is **not** true about the programming phases?
   A. Understand the problem
   B. Plan the logic
   C. Selection of diagram
   D. Code the program

3. Which one is **not** the element of understanding the problem?
   A. May be the most difficult phase
   B. Users may not be able to articulate their needs well
   C. User needs may be changing frequently
   D. None

4. Testing the program is _______________
   A. Execute it with sample data and check results
   B. Identify logic errors and correct them
   C. Choose test data carefully to exercise all branches of the logic
   D. All

5. Which one is a document used in a design or procurement process?
   A. Program planning
   B. Setting framework
   C. Program of requirements
   D. Monitoring planning

Note: **Satisfactory rating - 3 points** **Unsatisfactory - below 3 points**

You can ask you teacher for the copy of the correct answers.

Score = ___________

Rating: ____________

Information Sheet 1.3: Determining Design approach

1.3 Determining Design approach

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.

1. Preliminary investigation
2. Analysis
3. Design
4. Implementation
5. Testing
6. Maintenance

- **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.

- **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.

  ✓ Present a detail description of the functions the new system must perform.

In this phase we identify:

- ✓ Inputs to the problem and their form
- ✓ Outputs expected from the solution and their form
- ✓ Constraints (What are the limits on the data? E.g. Income cannot be a negative number)
- ✓ Assumptions (Problem deals)
- ✓ Formulas (e.g. For determining the area. It is length x Width)

The three primary activities involved in the analysis phase are as follows:

1. Gathering business requirement
2. Creating process diagrams
3. Performing a detailed analysis

Business requirement gathering is the most crucial part in program logic. Business requirements are a brief set of business functionalities that the system needs to meet in order to be successful. Technical details such as the types of technology used in the implementation of the system need not be defined in this phase. A sample business requirement might look like “The system must track all the employees by their respective department, region, and the designation”. This requirement is showing no such detail as to how the system is going to implement this requirement, but rather what the system must do with respect to the business.

- **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. The two primary activities involved in the design phase are as follows:

  1. Designing of IT infrastructure
  2. Designing of system model

To avoid any crash, malfunction or lack of performance, the IT infrastructure should have solid foundations. In this phase, the specialist recommends the kinds of clients and servers needed on a cost and time basis, and technical feasibility of the system. Also, in this phase, the organization creates interfaces for user interaction. Other than that, data models and entity relationship diagrams (ERDs) are also created in the same phase.

Based on the requirements specified in the analysis phase algorithms are developed at design stage. Develop a series of steps with a logical order which, when applied to the input would produce the specified output. Consider alternative technology configurations to develop the system (hardware, software, security capability of the system, network alternatives, etc.)

Management and control of the technical realization of the system: Detailed program specification, detail system specifications for the functions identified in the analysis: managerial, organizational and technological components of the system solution (input, processes, output, user interface, database design, processing, manual procedures, controls, and procedural controls; security, documentation, training, organizational change).

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),
- Simple,
- General (1 inch is equal to 2.54cm is not an algorithm, it has to convert a supplied number of inches)
- 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 N**arrative, Flowchart** and **Pseudo-code.**

Narrative: Often used to narrate the algorithm, can be understood by any user who may not have any knowledge of computer programming. Too wordy, Too ambiguous and can be interpreted in different ways. **Example:** Accept salary of the employee. Calculate bonus as 10% of salary and add it to salary. Accept service year of employee. If the service year is greater than 10, give additional 100 birr as bonus. Display the bonus of the employee.

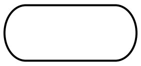
1.3.1 Flowcharts diagrams

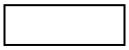
Flowchart: A flowchart is a graphical representation for an algorithm. It is a diagram consisting of labeled symbols, together with arrows connecting one symbol to another. A flowchart is a type of diagram that represents an algorithm or process, showing the steps as boxes of various kinds, and their order by connecting those with arrows. This diagrammatic representation can give a step-by-step solution to a given problem. Flowcharts are used in analyzing, designing, documenting or managing a process or program in various fields.

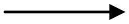
A flow chart can be used to:

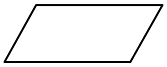
- Define and analyze processes.
- Build a step-by-step picture of the process for analysis, discussion, or communication.
- Define, standardize or find areas for improvement in a process,

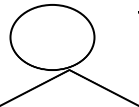
A picture can say a thousand words! That is why programmers use flowcharts to display their ideas. Flowcharts easily display the way a program naturally flows from one statement to the next. The following are the symbols used when drawing flowcharts.

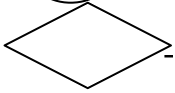

----- **Terminal point**: Makes the beginning or end of a program.

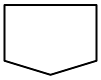

----- **Process**: Indicates any arithmetic computations or data movement.

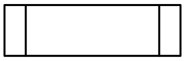

------**Flow line**: It is used to show the direction of logical flow.

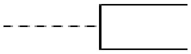

-----**Input or output data**: It is used to show input or output data


-------**Connector:** connects sections of the flowchart, so that the diagram can maintain a smooth, linear flow


------**Decision**: Indicates a yes/no decision to be made by the program.


--------**Inter-page connector**: Used as exit or entry from a flowchart on one page to a flowchart in another page.


----**Predefined process**: Indicates a process defined elsewhere.


---**Annotation flag**: Used to add clarifying comments or descriptions.

You can use different Flowchart Software's to draw a flowchart

Here is a shortlist of the best software for flowcharts:

1. Creately
2. Miro
3. Microsoft Visio
4. Gliffy
5. Edraw Max

6. ConceptDraw Diagram
7. Cacoo
8. Terrastruct
9. VisualParadigm Online
10. Draw.io

For example to create a Flowchart use a Microsoft Visio 2007

Flowcharts are diagrams that show the steps in a process. Basic flowcharts are easy to create and, because the shapes are simple and visual, they are easy to understand. The Basic Flowchart template in Microsoft Office Visio 2007 comes with shapes that you can use to show many kinds of processes, and it is especially useful for showing basic business processes like the proposal development process shown in the following figure.

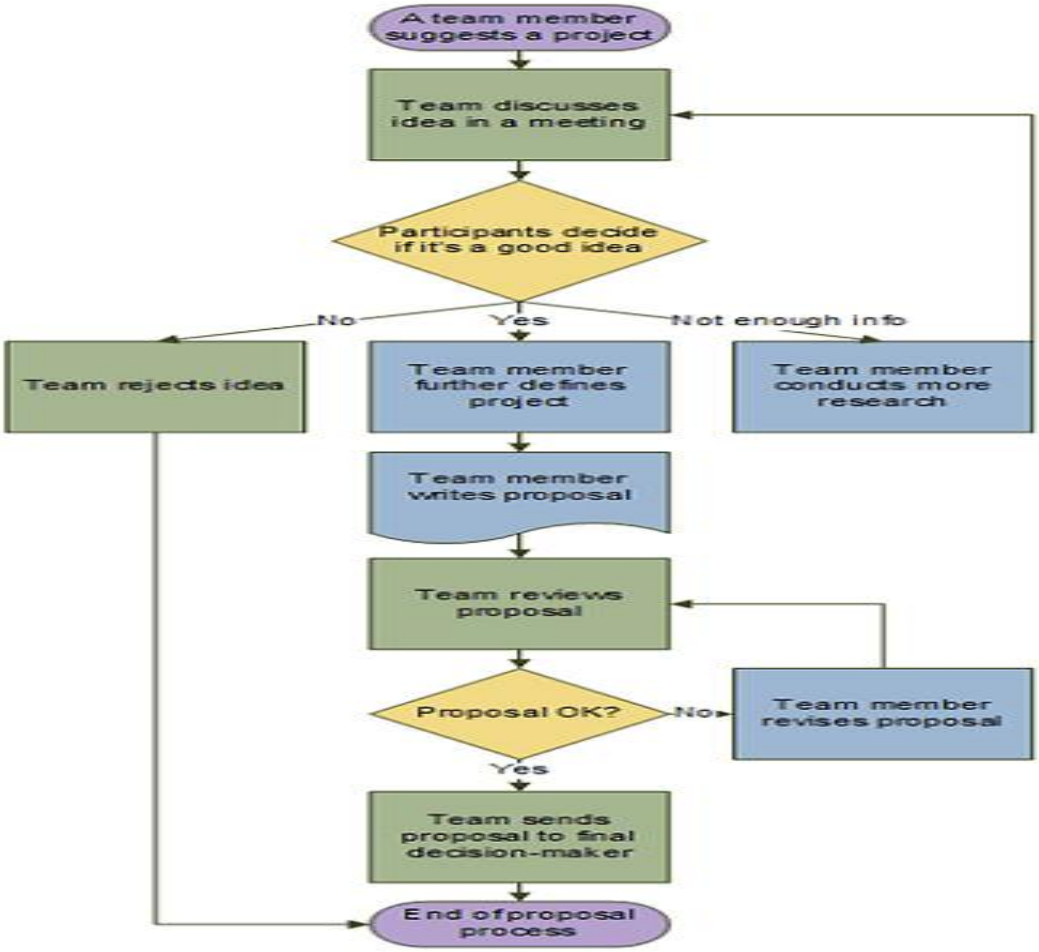

A team member suggests a project
Team discusses idea in a meeting
Participants decide if it's a good idea
No
Yes
Not enough info
Team rejects idea
Team member further defines project
Team member conducts more research
Team member writes proposal
Team reviews proposal
Proposal OK?
No
Team member revises proposal
Yes
Team sends proposal to final decision-maker
End of proposal process

Figure 1: Flowchart shows proposal development process

In addition to the Basic Flowchart template, Visio provides a variety of templates for more specific kinds of diagrams, such as data flow diagrams, timelines, and software modeling.

What the flowchart shapes represent

When you open the Basic Flowchart template, the Basic Flowchart Shapes stencil opens too. Each shape on the stencil represents a different step in a process.

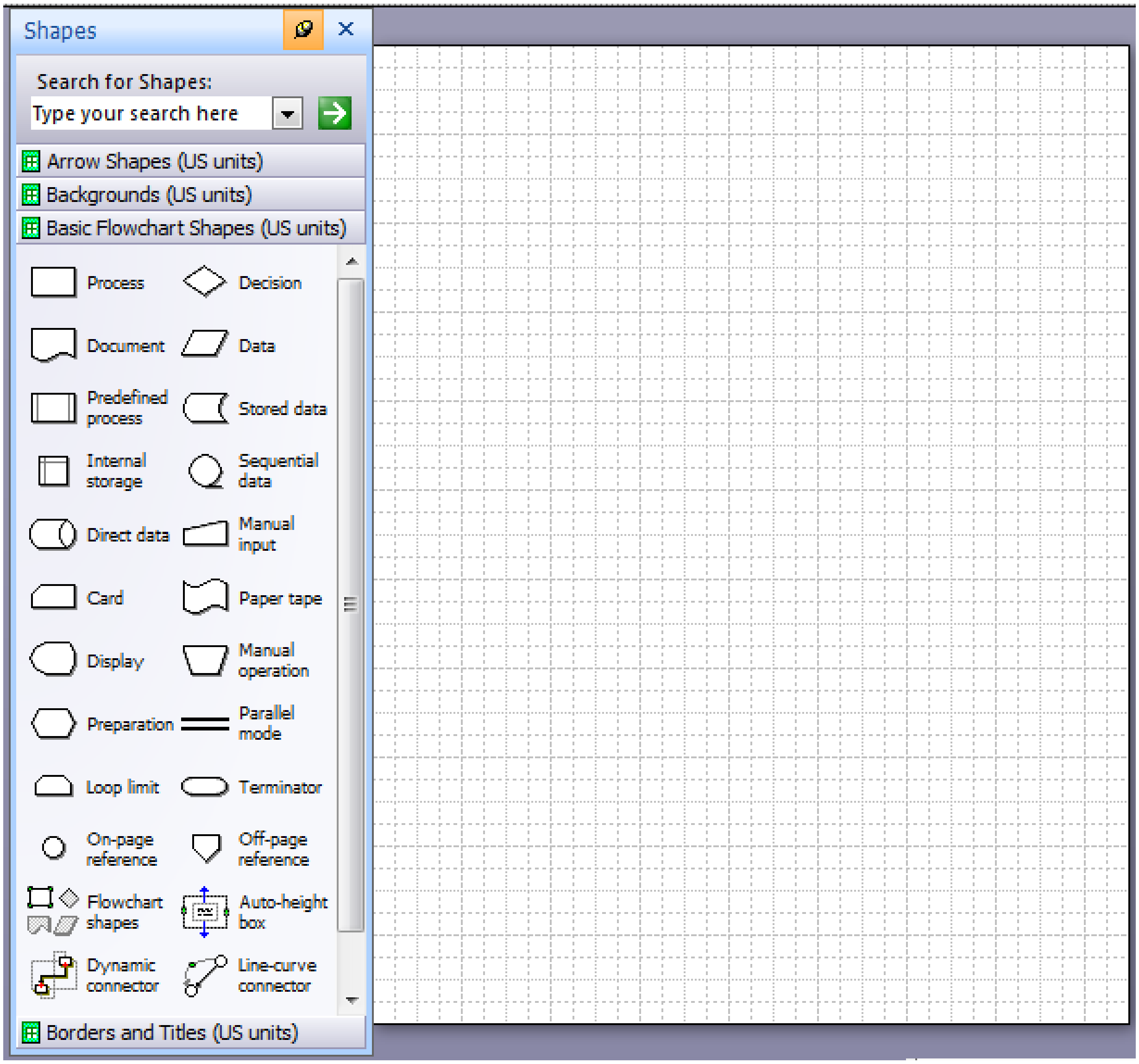


Figure 2: Flowchart Shapes stencil

Of all of the shapes on the Basic Flowchart Shapes stencil, only a few are commonly used.

Example 1: A certain company plan to give 10% bonus to each of its employees at the end of every year. If an employee has been working for 10 years or more at the company, she/he is to get an additional birr 100. Draw a flowchart of an algorithm to calculate and print the bones for a given employee.

Solution: Analysis problem is to compute bonus for employee. To do so, the salary and number of service years of the employee must be known. Let the salary be denoted by SAL, service denoted by YEAR and bonus by BONUS. To compute the bonus, we multiply the salary by 10% and assign the result to BONUS. Thus, BONUS = SAL*0.10. If the employee has served at least 10 years, birr 100 is added to the computed bonus of the employee and the result is displayed. Otherwise the originally computed bonus is displayed.

Algorithm: Uses English to write operations in a group.

1. Read employee's salary and year of service.
2. Calculate the employee's bonus.
3. If years of service is greater than or equal to 10 then increase the bonus. Otherwise only print the bonus.

Example 1: A flowchart for calculating bonus

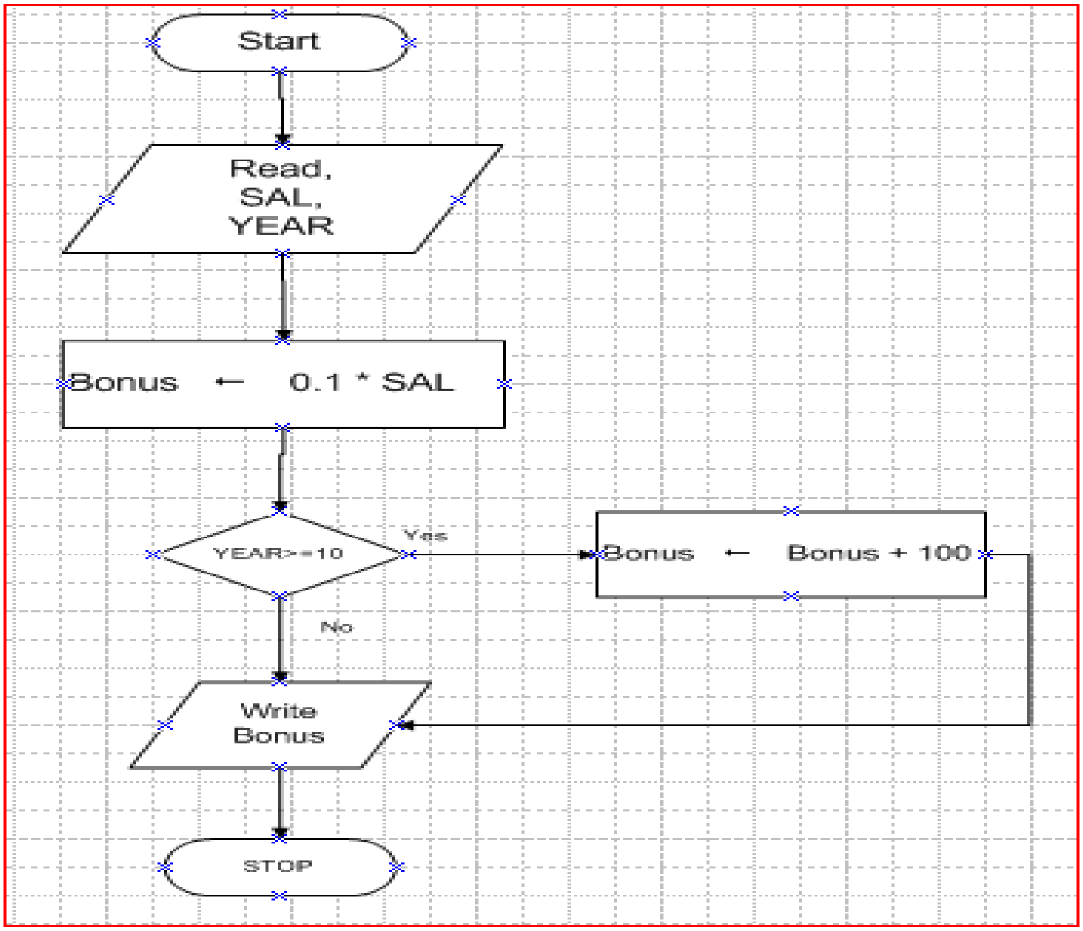


Figure 3: A flowchart for calculating bonus

Example 2. A flow chart for calculating interest amount

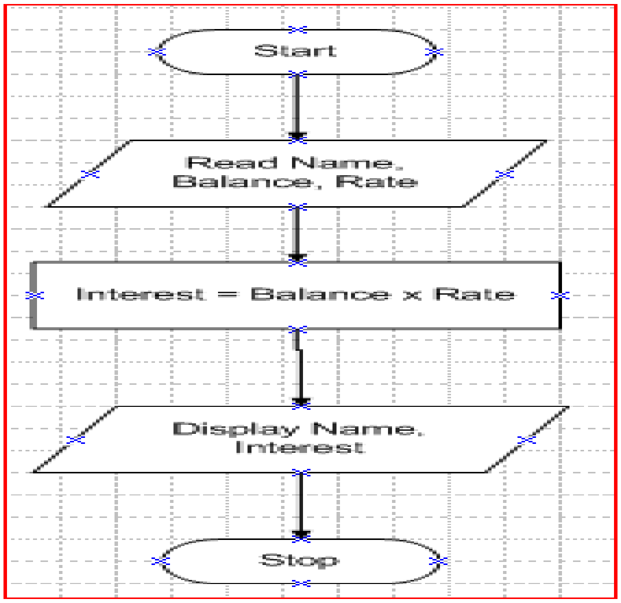


Figure 4: A flow chart for calculating interest amount

Example 3. A program that identifies a larger and smaller number from two numbers

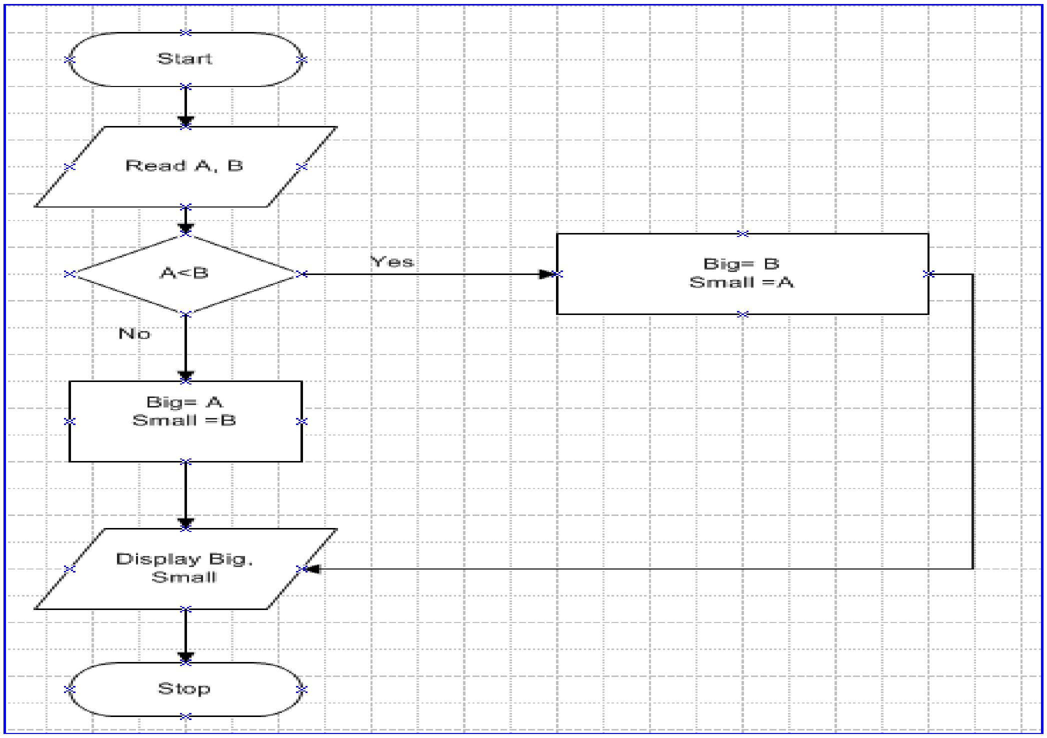


Figure 5: identifies a larger and smaller number

Example 4. Calculate grade for ten students based on the scale: >80-A, >60-B, > 50-C, >40-D, <40-F.

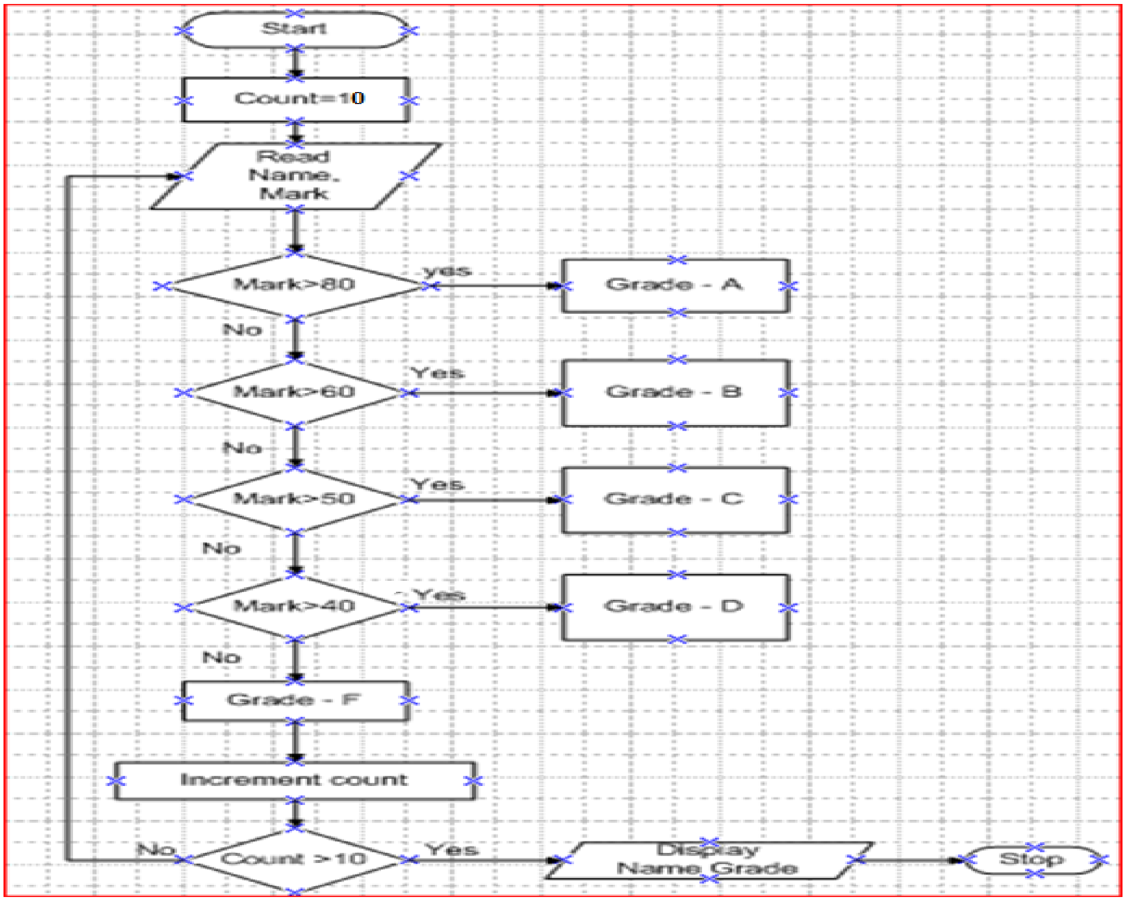


Figure 6: Calculate grade for ten students

1.3.2 Pseudo codes

Pseudo-code is an informal high-level description of the operating principle of a computer program or other algorithm. **Pseudo-code** is used in planning an algorithm with sketching out the structure of the program before the actual coding takes place. It is not an actual programming language. So it can't be compiled into an executable program. **Pseudo-code** is writing an algorithm as close as possible to computing languages. Pseudo-code is useful for describing algorithms in a structured way. The purpose of using pseudo-code is an efficient key principle of an algorithm. An algorithm is merely the sequence of steps taken to solve a problem. The steps are normally "**sequence**," "**selection**," "**iteration**," and a **case-type** statement. The "**selection**" is the "**if then else**" statement, and the iteration is satisfied by a number of statements, such as the "**while**," " **do**," and the "**for**," while the case-type statement is satisfied by the "**switch**" statement.

Pseudo Code with SEQUENCE

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 looks like this:

```
Statement 1;
Statement 2;
Statement 3;
Statement 4;
Statement 5;
```

For example, for making a cup of tea:

```
Organize everything together;
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;
Serve;
```

Pseudo Code with LOOP

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;
```

Example,

```
Organize everything together;
Plug in kettle;
Put teabag in cup;
While (Kettle is not full)
Do keep filling Kettle;
EndWhile;
Wait for bottle to boil;
Add water to cup;
Remove teabag with spoon/fork/;
Add milk and /or sugar;
Serve;
```

Pseudo Code with 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;
```

So, we could state this as:

```
If (Sugar is required)
Then add Sugar;
Else don’t add sugar;
EndIf
```

Example,

```
Organize everything together;
Plug in kettle;
Put teabag in cup;
While (Kettle is not full)
Do keep filling Kettle;
EndWhile;
Wait for bottle to boil;
Add water to cup;
Remove teabag with spoon/fork/;
Add milk;
If (Sugar is required)
Then add Sugar;
Else do nothing;
EndIf;
```

In computer science and numerical computation, pseudo-code is an informal high-level description of the operating principle of a computer program or other algorithm. An outline of a program written in a form of pseudo-code can be converted easily into real programming statements. It cannot be compiled nor executed, and there is no real formatting or syntax rules. It is simply an important step in producing the final code. Pseudo-code makes use of simple English-like statements. It is much similar to real code. We use verbs to write pseudo-code. Capitalize important words that show actions.

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
```

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
```

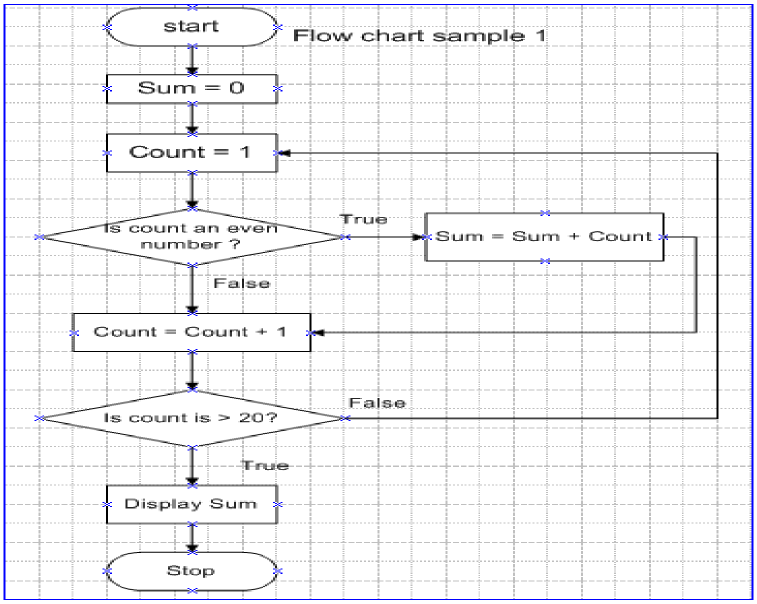
Algorithm vs. **Pseudo-code:** Both Algorithm and Pseudo code more or less describe the logical sequence of steps that follow in solving a problem.

- **Pseudo-code** consists of short readable and formally-styled natural language that used to explain specific tasks within a program's algorithm while an **Algorithm** is a group of instructions or a set of steps applied to solve a particular problem.
- **Pseudo code** is a method used to define an algorithm. An algorithm is written in a natural language while pseudo code can be written in high level programming languages.
- **Pseudo-code** cannot be executed on a real computer, but it models and resembles real programming code, and is written at roughly the same level of detail.

Example: Design the **Pseudo-code** that sums all the even numbers between 1 and 20 inclusive and then displays the sum and draw flowchart for it. It uses a repeat loop and contains a null else within the repeat loop.

Solution:

```
Start
Sum = 0
count = 1
REPEAT
IF count is even THEN sum = sum + count
count = count + 1
UNTIL count > 20
DISPLAY sum
Stop
```



1.3.3 Entity-relationship diagrams (ERDs)

ERD is a data modeling technique that creates a graphical representation of the entities, and the relationships between entities, within an information system. **Data modeling** is the formalization and documentation of existing processes and events that occur during application software design and development. Entity-Relationship model is a type of database model based on the notion of real world entities and relationship among them. We can map real world scenario onto ER database model. ER Model creates a set of entities with their attributes, a set of constraints and relation among them. ER Model is best used for the conceptual design of database.

The three main components of an ERD are:

- **Entity**: the entity can be a person, object, place or event for which data is collected. **Example**: if you consider the information system for a business, entities would include not only customers, but also the customer's address and orders as well. The entity is represented by a **rectangle** and labeled with a singular noun.
- **Relationship**: the Relationship is the interaction between the entities. A relationship may be represented by a **diamond** shape that can be connected by the line to the entities. Verbs are used to label the relationships.

- **Cardinality**: the cardinality defines the relationship between the entities in terms of numbers. The three main cardinal relationships are: one-to-one, expressed as 1:1; one-to-many, expressed as 1:M; and many-to-many, expressed as M:N.

The steps involved in creating an ERD are:

1. Identify Entities
2. Find Relationships
3. Draw Rough ERD
4. Fill in Cardinality
5. Define Primary Keys
6. Draw Key-Based ERD
7. Identify Attributes
8. Map Attributes
9. Draw fully attributed ERD
10. Check Results

Step 1. Identify Entities

A data entity is anything real or abstract about which we want to store data. Entity types fall into five classes: roles, events, locations, tangible things, or concepts. The best way to identify entities is to ask the system owners and users to identify things about which they would like to capture, store and produce information. Another source for identifying entities is to study the forms, files, and reports generated by the current system. E.g. a student registration form would refer to Student (a role), but also Course (an event), Instructor (a role), Advisor (a role), Room (a location), etc.

Step 2. Find Relationships

There are natural associations between pairs of entities. Listing the entities down the left column and across the top of a table, we can form a relationship matrix by filling in an active verb at the intersection of two entities which are related. Each row and column should have at least one relationship listed or else the entity associated with that row or column does not interact with the rest of the system. In this case, you should question whether it makes sense to include that entity in the system. A student is enrolled in one or more courses

Step 3. Draw Rough ERD

Using rectangles for entities and lines for relationships, we can draw an Entity Relationship Diagram.

Step 4. Fill in Cardinality

At each end of each connector joining rectangles, we need to place a symbol indicating the minimum and maximum number of instances of the adjacent rectangle there are for one instance of the rectangle at the other end of the relationship line. The placement of these numbers is often confusing. The first symbol is either 0 to indicate that it is possible for no instances of the entity joining the connector to be related to a given instance of the entity on the other side of the relationship, 1 if at least one instance is necessary or it is omitted if more than one instance is required. For example, more than one student must be enrolled in a course for it to run, but it is possible for no students to have a particular instructor (if they are on leave). The second symbol gives the maximum number of instances of the entity joining the connector for each instance of the entity on the other side of the relationship. If there is only one such instance, this symbol is 1. If more than 1, the symbol is a crows foot opening towards the rectangle. i.e. A student is enrolled in one or more courses.

Step 5. Define Primary Keys

For each entity we must find a unique primary key so that instances of that entity can be distinguished from one another. Often a single field or property is a primary key (e.g. a Student ID). Other times the identifier is a set of fields or attributes (e.g. a course needs a department identifier, a course number, and often a section number; a Room needs a Building Name and a Room Number). When the entity is written with all its attributes, the primary key is underlined.

Step 6. Draw Key-Based ERD

Looking at the Rough Draft ERD, we may see some relationships which are non-specific or many-to-many. I.e., there are crows feet on both ends of the relationship line. Such relationships spell trouble later when we try to implement the related entities as data stores or data files, since each record will need an indefinite number of fields to maintain the many-to-many relationship. The key-based ERD has no many-to-many relationships and each entity has its primary and foreign keys listed below the entity name in its rectangle.

Step 7. Identify Attributes

A data attribute is a characteristic common to all or most instances of a particular entity. In this step we try to identify and name all the attributes essential to the system we are studying without trying to match them to particular entities. The best way to do this is to study the forms, files and reports currently kept by the users of the system and circle each data item on the paper copy.

Step 8. Map Attributes

For each attribute we need to match it with exactly one entity. Often it seems like an attribute should go with more than one entity (e.g. Name). In this case you need to add a modifier to the attribute name to make it unique (e.g. Customer Name, Employee Name, etc.) or determine which entity an attribute "best' describes. If you have attributes left over without corresponding entities, you may have missed an entity and its corresponding relationships. Identify these missed entities and add them to the relationship matrix now.

Step 9. Draw Fully-Attributed ERD

If you introduced new entities and attributes in step 8, you need to redraw the entity relationship diagram. When you do so, try to rearrange it so no lines cross by putting the entities with the most relationships in the middle. If you use a tool like Systems Architect, redrawing the diagram is relatively easy.

Step 10. Check Results

Look at your diagram from the point of view of a system owner or user. Is everything clear? Check through the Cardinality pairs. Also, look over the list of attributes associated with each entity to see if anything has been omitted.

The basic building blocks of ER-diagram are:

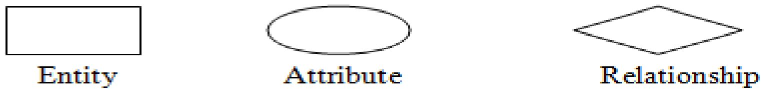


Figure 7: ER-Diagram basic Symbol

Example: A company has several departments. Each department has a supervisor and at least one employee. Employees must be assigned to at least one, but possibly more departments. At least one employee is assigned to a project, but an employee may be on vacation and not assigned to any projects. The important data fields are the names of the departments, projects, supervisors and employees, as well as the supervisor and employee number and a unique project number.

1. **Identify Entities:** The entities in this system are Department, Employee, Supervisor and Project. One is tempted to make Company an entity, but it is a false entity because it has only one instance in this problem. True entities must have more than one instance.
2. **Find Relationships:** We construct the following Entity Relationship Matrix:

	Department	**Employee**	**Supervisor**	**Project**
Department		is assigned	run by	
Employee	belongs to			works on
Supervisor	Runs			
Project		uses		

3. **Draw Rough ERD:** We connect the entities whenever a relationship is shown in the entity Relationship Matrix.

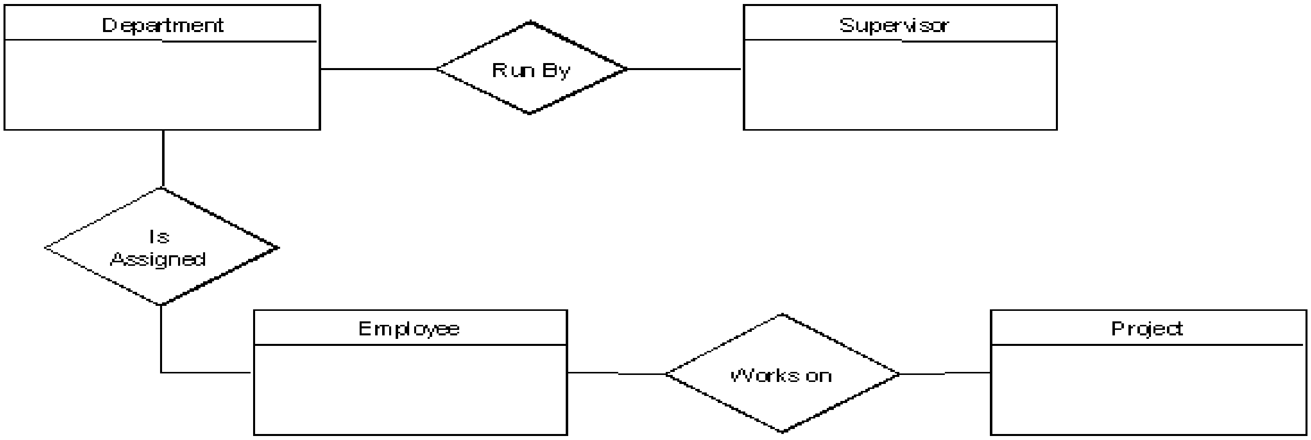


Figure 8: ERD shows relationship

4. **Fill in Cardinality**: From the description of the problem we see that:
    - Each department has exactly one supervisor.
    - A supervisor is in charge of one and only one department.
    - Each department is assigned at least one employee.
    - Each employee works for at least one department.
    - Each project has at least one employee working on it.
    - An employee is assigned to 0 or more projects.

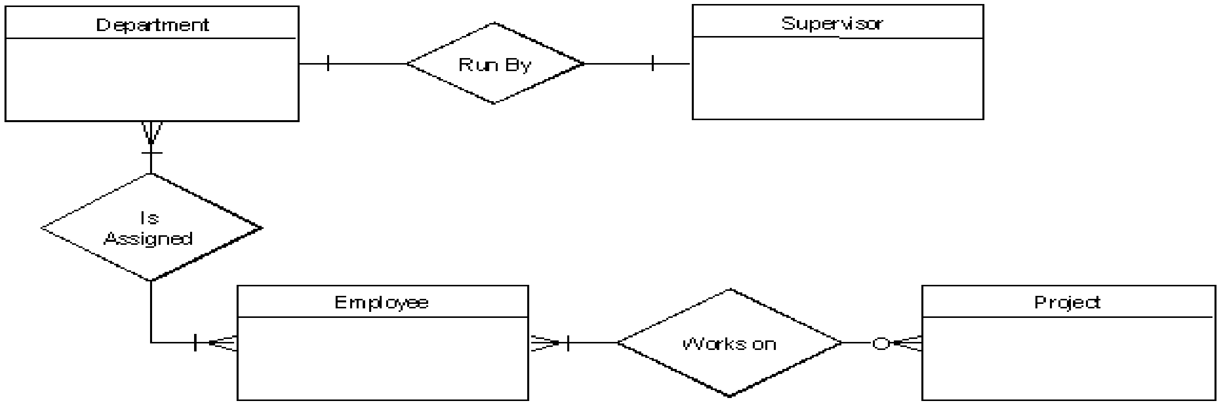


Figure 9: ERD shows cardinality ratio

5. **Define Primary Keys:** The primary keys are Department Name, Supervisor Number, Employee Number, Project Number.
6. **Draw Key-Based ERD:** There are two many-to-many relationships in the rough ERD above, between Department and Employee and between Employee and Project. Thus we need the associative entities Department-Employee and Employee-Project. The primary key for Department-Employee is the concatenated key Department Name and Employee Number. The primary key for Employee-Project is the concatenated key Employee Number and Project Number.

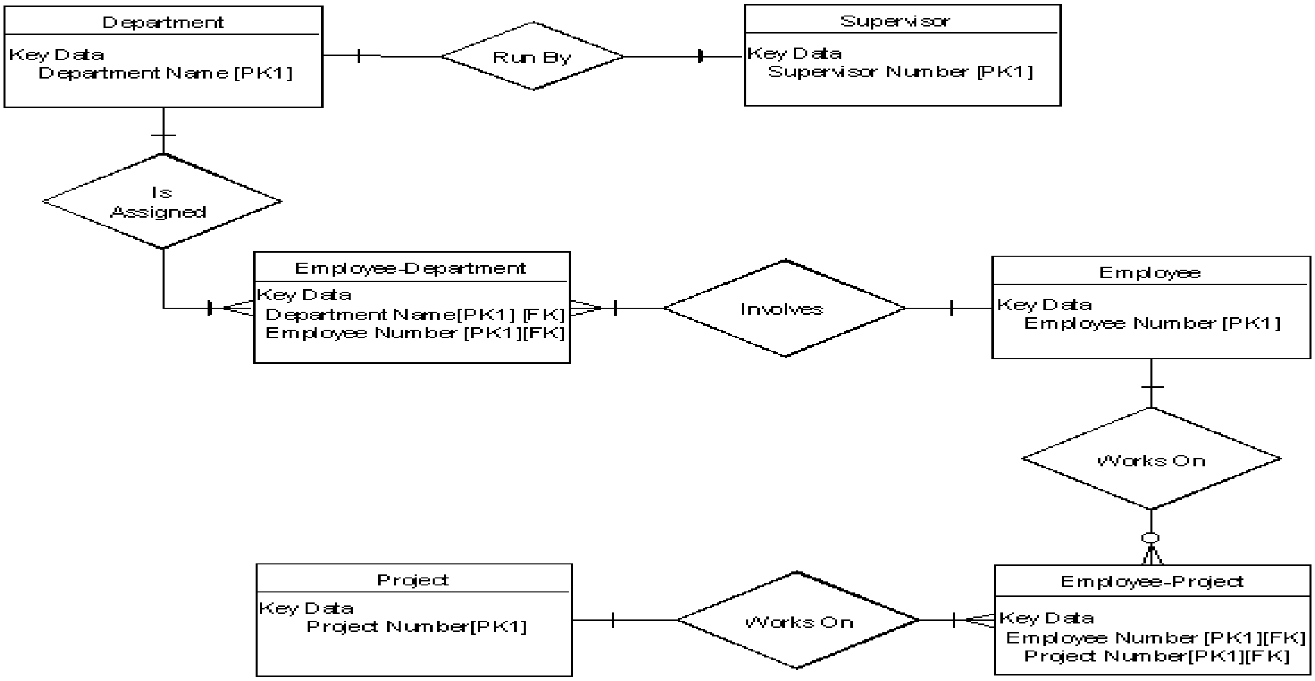


Figure 10: ERD shows that Key based

7. **Identify Attributes**: The only attributes indicated are the names of the departments, projects, supervisors and employees, as well as the supervisor and employee NUMBER and a unique project number.
8. **Map Attributes:** For each attribute, match it with exactly one entity that it describes.

Attribute	Entity	Attribute	Entity
Department Name	Department	Supervisor Number	Supervisor
Employee umber	Employee	Supervisor Name	Supervisor
Employee Name	Employee	Project Name	Project
		Project Number	Project

9. Draw Fully Attributed ERD:

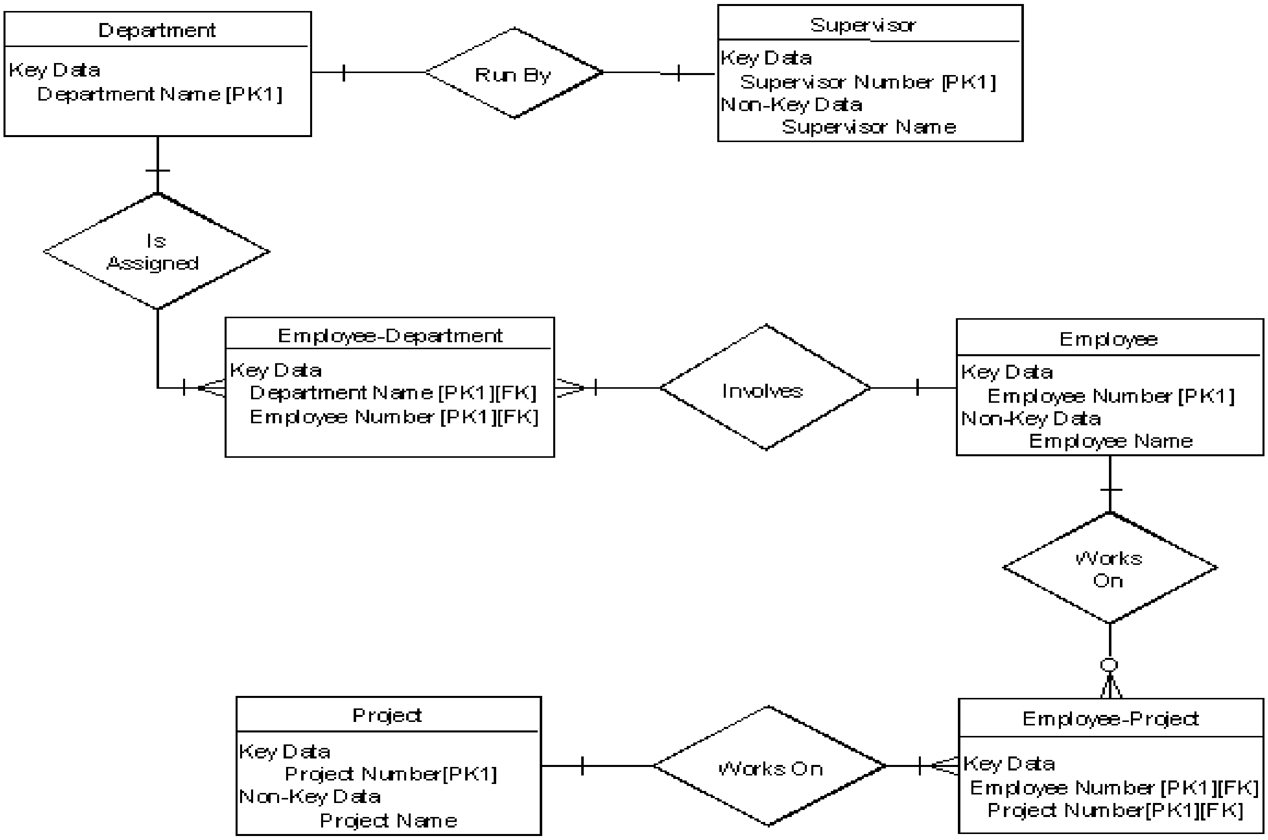


Figure 11: ERD with full Attributes

10. Check Results*:* The final ERD appears to model the data in this system well.

1.3.4 HIPO Charts (HIPO Diagram)

HIPO stands for **Hierarchical Input Process Output.** HIPO diagram is a combination of two organized method to analyze the system and provide the means of documentation. HIPO model was developed by IBM in year 1970. The HIPO acts as a hierarchical chart for the function performed by the system. The **HIPO chart** is a tool used to analyze a problem and visualize a solution using the **top down design** approach. Starting at the global (macro) level, the chart is decomposed repeatedly at ever-greater levels of detail until the **logical building blocks (functions)** are identified.

A HIPO model consists of a hierarchy chart that graphically represents the program's control structure and a set of IPO (Input-Process-Output) charts that describe the inputs to, the outputs from, and the functions performed by each module on the hierarchy chart. HIPO diagram represents the hierarchy of modules in the software system. Analyst uses HIPO diagram in order to obtain high-level view of system functions. It decomposes functions into sub-functions in a hierarchical manner. It depicts the functions performed by system. HIPO diagrams are good for documentation purpose. Their graphical representation makes it easier for designers and managers to get the pictorial idea of the system structure.

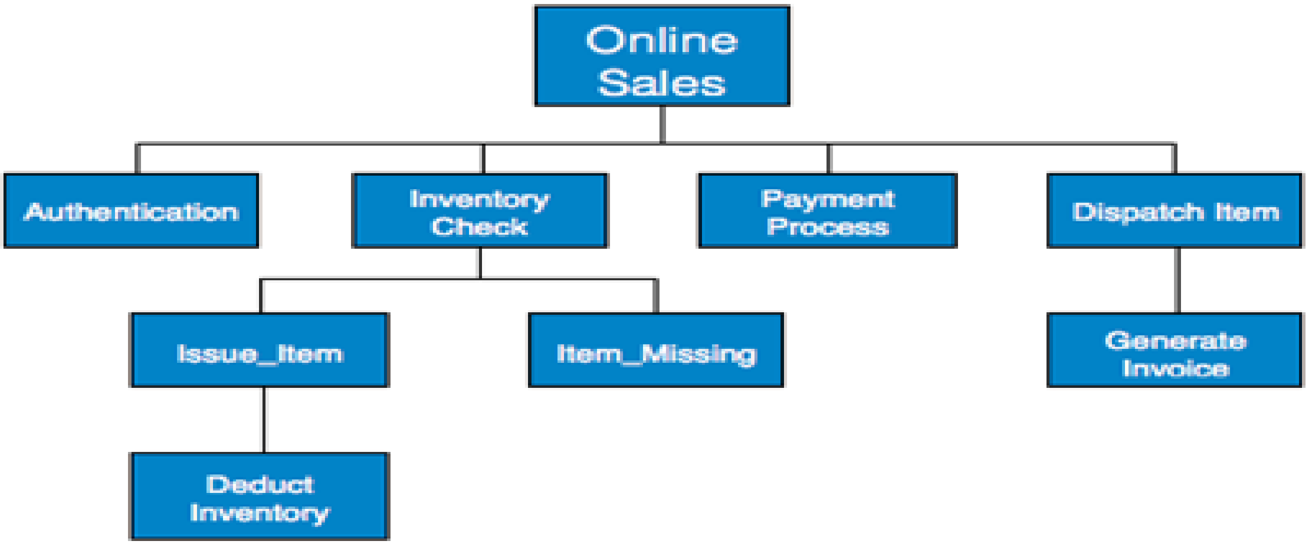


Figure 12: HIPO chart shows system structure

In contrast to IPO (Input Process Output) diagram, which depicts the flow of control and data in a module, HIPO does not provide any information about data flow or control flow.

The following shows general model:

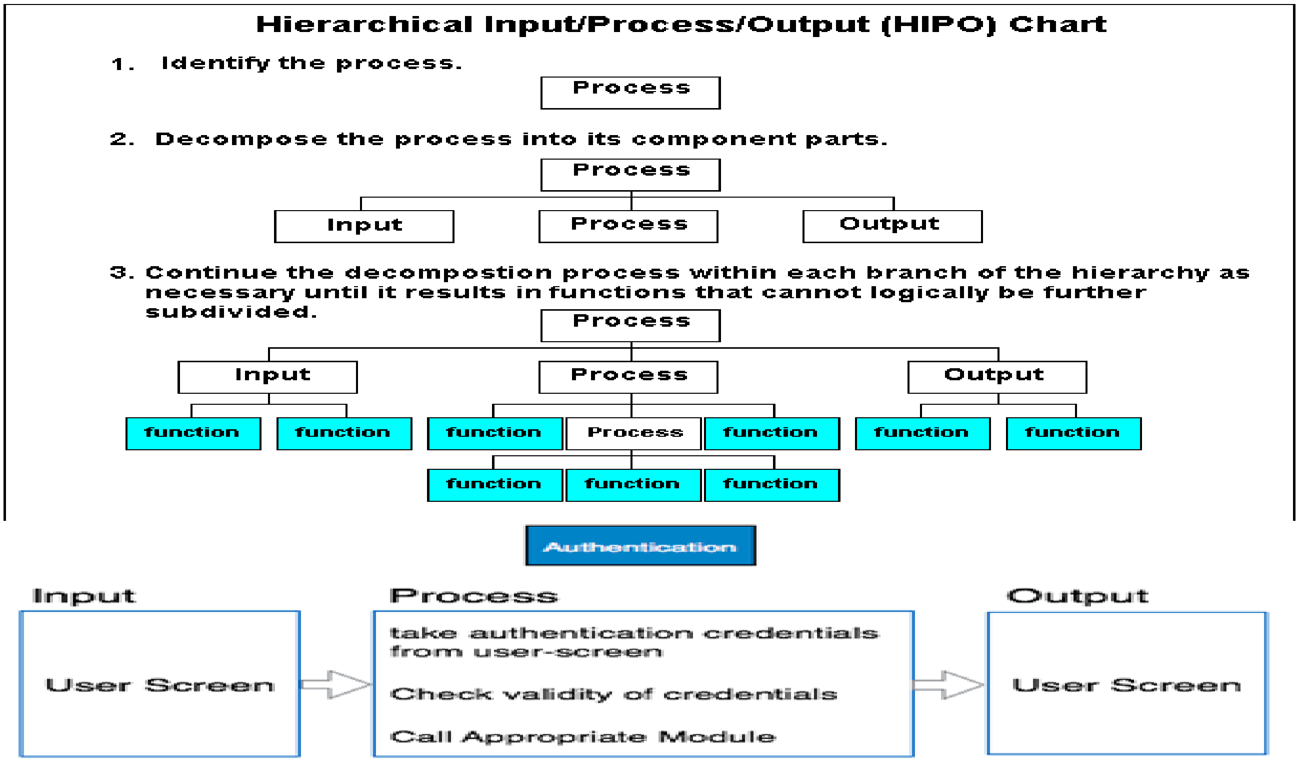

Hierarchical Input/Process/Output (HIPO) Chart
1. Identify the process.
Process
2. Decompose the process into its component parts.
Process
Input
Process
Output
3. Continue the decompostion process within each branch of the hierarchy as necessary until it results in functions that cannot logically be further subdivided.
Process
Input
Process
Output
function
function
function
Process
function
function
function
function
function
function
Authentication
Input
User Screen
Process
take authentication credentials from user-screen
Check validity of credentials
Call Appropriate Module
Output
User Screen

Figure 13: HIPO shows general model

Example: Both parts of HIPO diagram, Hierarchical presentation and IPO Chart are used for structure design of software program as well as documentation of the same.

1.3.5 Data flow diagrams (DFDs)

Software Analysis & Design Tools

Software analysis and design is the intermediate stage, which helps human-readable requirements to be transformed into actual code.

Analysis and design tools used by software designers includes

- ☞ **Data Flow Diagram (DFD):** It is graphical representation of flow of data in an information system. It is capable of depicting incoming data flow, outgoing data flow and stored data. The DFD does not mention anything about how data flows through the system. There is a prominent difference between DFD and Flowchart. The flowchart depicts flow of control in program modules. DFDs depict flow of data in the system at various levels. DFD does not contain any control or branch elements.

Types of DFD

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

DFD Components

DFD can represent Source, destination, storage and flow of data using the following set of components

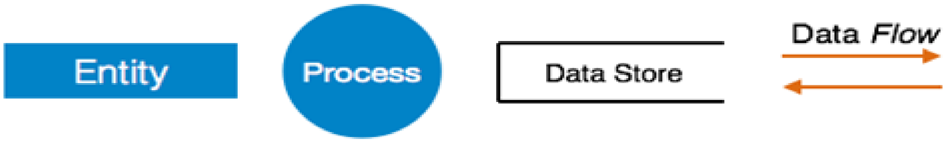


Figure 14: DFD components

- **Entities** -Entities are source and destination of information data. Entities are represented by rectangles.
- **Process** -Activities and action taken on the data are represented by Circle or Round-edged rectangles.
- **Data Storage**- There are two variants of data storage, it can either be represented as a rectangle with absence of both smaller sides or as an open-sided rectangle with only one side missing.
- **Data Flow**- Movement of data is shown by pointed arrows. Data movement is shown from the base of arrow as its source towards head of the arrow as destination.

Levels of DFD

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

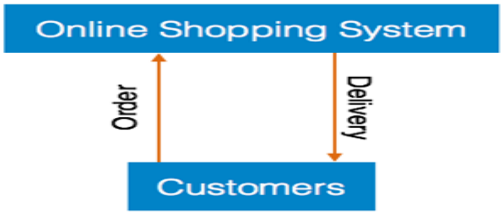


Figure 15: 0 Level DFD

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

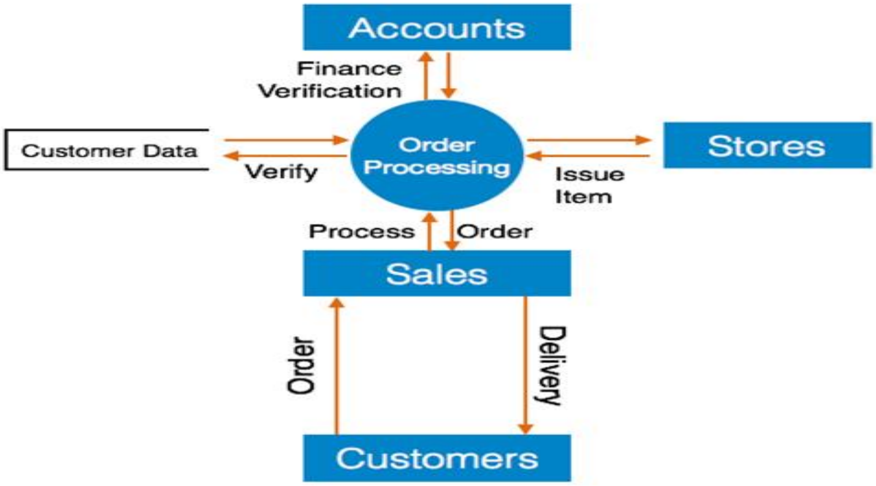


Figure 16: Level 1 DFD

- **Level** 2 - At this level, DFD shows how data flows inside the modules mentioned in Level 1.

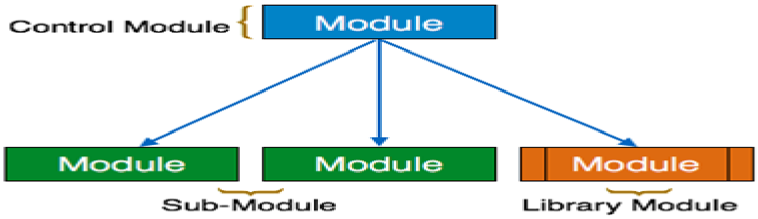
Higher level DFDs can be transformed into more specific lower level DFDs with deeper level of understanding unless the desired level of specification is achieved.

Structure Charts

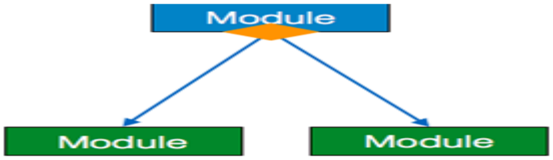
Structure chart is a chart derived from Data Flow Diagram. It represents the system in more detail than DFD. It breaks down the entire system into lowest functional modules, describes functions and sub-functions of each module of the system to a greater detail than DFD. Structure chart represents hierarchical structure of modules. At each layer a specific task is performed.

Here are the symbols used in construction of structure charts.

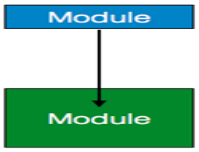
Module - It represents process or subroutine or task. A control module branches to more than one sub-module. Library Modules are re-usable and invoke-able from any module.



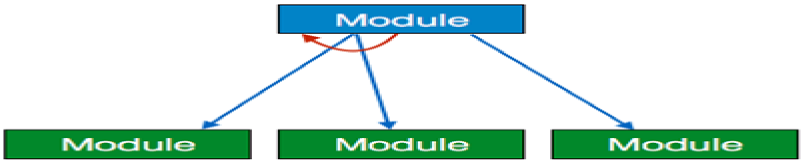
Condition - It is represented by small diamond at the base of module. It depicts that control module can select any of sub-routine based on some condition.



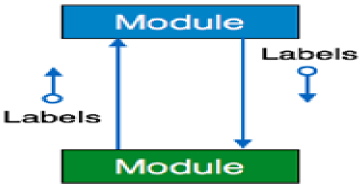
Jump - An arrow is shown pointing inside the module to depict that the control will jump in the middle of the sub-module.



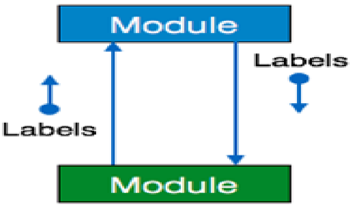
Loop - A curved arrow represents loop in the module. All sub-modules covered by loop repeat execution of module.



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

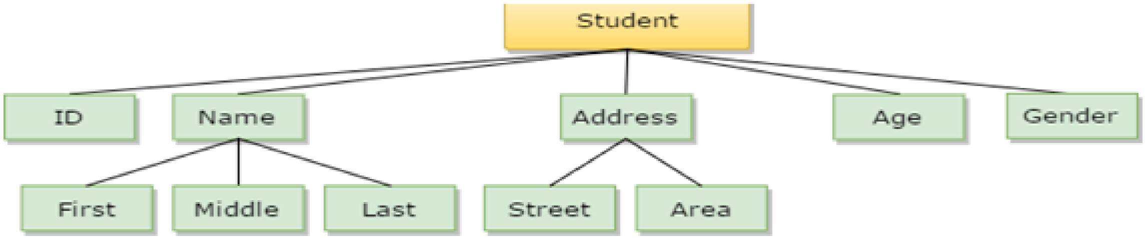


Control flow - A directed arrow with filled circle at the end represents control flow.



1.3.6 Data structure

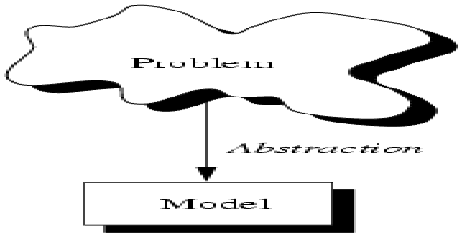
Data are just a collection of facts and figures or you can say data are values or a set of values that are in a particular format. A data item refers to a single set of values. In the modern world, Data and its information is an essential part and various implementations are being made to store in different ways. Data items are then further categorized into sub-items which are the group of items which are not being called a plain elementary form of items. Example, the name of the student may be divided into three sub items namely: first name, middle name and last name. But the ID that is assigned to a student would normally be considered as a single item.



In the example mentioned above such as ID, Age, Gender, First, Middle, Last, Street, Area, etc. are elementary data items, whereas (Name, Address) is group data items.

Data Structure

A data structure is a Specific way to store and organize data in a computer's memory so that these data can be used efficiently later. Data may be arranged in many different ways such as the logical or mathematical model for a particular organization of data is termed as a data structure. The way data are organized in a computer memory is said to be **data structure.** In computer science, a **data structure** is a particular way of storing and organizing data in a computer so that it can be used efficiently. A data structure is a group of data elements grouped together under one name. These data elements, known as *members*, can have different types and different lengths. Given a problem, the first step to solve the problem is obtaining one's own abstract view or *model* of the problem. This process of modeling is called *abstraction.*



.

Figure 17: Process of Modeling

The model defines an abstract view to the problem. This implies that the model focuses only on problem related stuff and that a programmer tries to define the properties of the problem. These properties include the data which are affected and the operations that are involved in the problem. With abstraction you create a well-defined entity that can be properly handled. These entities define the *data structure* of the program. An entity with the properties just described is called an abstract data type.

Abstract Data Types (ADT): An ADT consists of an abstract data structure and operations. Put in other terms, an ADT is an abstraction of a data structure.

The ADT specifies:

1. What can be stored in the Abstract Data Type
2. What operations can be done by the Abstract Data Type.

For example, if we are going to model employees of an organization:

☞ This ADT stores employees with their relevant attributes and discarding irrelevant attributes.

☞ This ADT supports hiring, firing, retiring, etc, operations.

A data structure is a language construct that the programmer has defined in order to implement an abstract data type.

Abstraction

Abstraction is a process of classifying characteristics as relevant and irrelevant for the particular purpose at hand and ignoring the irrelevant ones. Applying abstraction correctly is the essence of successful programming.

1.3.7 RAD (Rapid Application Development)

The RAD model is based on prototyping and iterative development with no specific planning involved. The process of writing the software itself involves the planning required for developing the product. RAD focuses on gathering customer requirements through workshops or focus groups, early testing of the prototypes by the customer using iterative concept, reuse of the existing prototypes (components), continuous integration and rapid delivery.

Rapid application development is a software development methodology that uses minimal planning in favor of rapid prototyping. A prototype is a working model that is functionally equivalent to a component of the product. In the RAD model, the functional modules are developed in parallel as prototypes and are integrated to make the complete product for faster product delivery. Since there is no detailed preplanning, it makes it easier to incorporate the changes within the development process.

RAD projects follow iterative and incremental model and have small teams comprising of developers, domain experts, customer representatives and other IT resources working progressively on their component or prototype. The most important aspect for this model to be successful is to make sure that the prototypes developed are reusable.

RAD Model Design

RAD model distributes the analysis, design, build and test phases into a series of short, iterative development cycles.

The various phases of the RAD Model

- **Business Modeling:** A complete business analysis is performed to find the vital information for business, how it can be obtained, how and when the information processed and what is the factors driving successful flow of information.
- **Data Modeling:** The information gathered in the Business Modeling phase is reviewed and analyzed to form sets of data objects vital for the business. The attributes of all data sets is identified and defined. The relation between these data objects are established and defined in detail in relevance to the business model.
- **Process Modeling:** The data object sets defined in the Data Modeling phase is converted to establish the business information flow needed to achieve specific business objectives as per the business model. The process model for any changes or enhancements to the data object sets is defined in this phase. Process descriptions for adding, deleting, retrieving or modifying a data object are given.
- **Application Generation**: The actual system is built and coding is done by using automation tools to convert process and data models into actual prototypes.
- **Testing and Turnover:** The overall testing time is reduced in the RAD model as the prototypes are independently tested during every iteration. However, the data flow and the interfaces between all the components need to be thoroughly tested with complete test coverage. Since most of the programming components have already been tested, it reduces the risk of any major issues.

RAD Model Vs Traditional SDLC

The **traditional SDLC** follows a rigid process models with high emphasis on requirement analysis and gathering before the coding starts. It puts pressure on the customer to sign off the requirements before the project starts and the customer doesn't get the feel of the product as there is no working build available for a long time.

The customer may need some changes after he/she gets to see the software. However, the change process is quite rigid and it may not be feasible to incorporate major changes in the product in the traditional SDLC. The RAD model focuses on iterative and incremental delivery of working models to the customer. This results in rapid delivery to the customer and customer involvement during the complete development cycle of product reducing the risk of non-conformance with the actual user requirements.

RAD Model - Application

RAD model can be applied successfully to the projects in which clear modularization is possible. If the project cannot be broken into modules, RAD may fail. The following pointers describe the typical scenarios where RAD can be used.

- RAD should be used only when a system can be modularized to be delivered in an incremental manner.
- It should be used if there is a high availability of designers for modeling.
- It should be used only if the budget permits use of automated code generating tools.
- RAD SDLC model should be chosen only if domain experts are available with relevant business knowledge.
- Should be used where the requirements change during the project and working prototypes are to be presented to customer in small iterations of 2-3 months.

RAD model enables rapid delivery as it reduces the overall development time due to the reusability of the components and parallel development. RAD works well only if high skilled engineers are available and the customer is also committed to achieve the targeted prototype in the given time frame. If there is commitment lacking on either side the model may fail.

1.3.8 Case tools

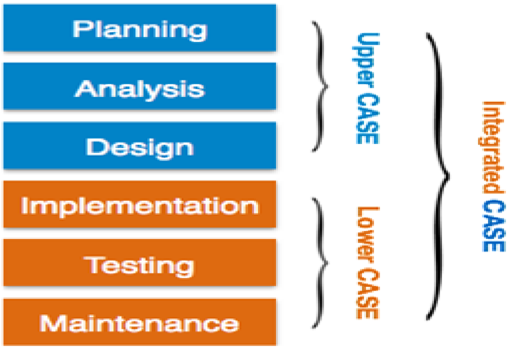
CASE stands for **Computer Aided Software Engineering**. It means development and maintenance of software projects with help of various automated software tools. CASE tools are set of software application programs, which are used to automate Software Development Life Cycle (SDLC) activities. CASE tools are used by software project managers, analysts and engineers to develop software system. There are number of CASE tools available to simplify various stages of SDLC such as Analysis tools, Design tools, Project management tools, Database Management tools, Documentation tools are to name a few. A CASE tool is software that can be used to mean any computer-based tool for software planning, development and evolution. CASE tool is software that assists with software development. The main purpose of using a CASE tool is to produce error-free, easy to maintain program code.

CASE systems provide tools to automate, to manage and simplify the development process. These can include tools for Summarizing initial requirements, Developing flow diagrams, Scheduling development tasks, Preparing documentation, Controlling software versions and Developing program code Use of CASE tools accelerates the development of project to produce desired result and helps to uncover flaws before moving ahead with next stage in software development.

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.



- **Upper Case Tools** - Upper CASE tools are used in planning, analysis and design stages of SDLC.
- **Lower Case Tools** - Lower CASE tools are used in implementation, testing and maintenance.

- **Integrated Case Tools** - Integrated CASE tools are helpful in all the stages of SDLC, from Requirement gathering to Testing and documentation.

CASE tools can be grouped together if they have similar functionality, process activities and capability of getting integrated with other tools.

Case Tools Types

- **Diagram tools:** These tools are used to represent system components, data and control flow among various software components and system structure in a graphical form. For example, Flow Chart Maker tool for creating state-of-the-art flowcharts.
- **Process Modeling Tools:** It is a method to create software process model, which is used to develop the software. Process modeling tools help the managers to choose a process model or modify it as per the requirement of software product. For example, EPF Composer
- **Project Management Tools**: These tools are used for project planning, cost and effort estimation, project scheduling and resource planning. Managers have to strictly comply project execution with every mentioned step in software project management. Project management tools help in storing and sharing project information in real-time throughout the organization. E.g., Creative Pro Office, Trac Project, Basecamp.
- **Documentation Tools**: Documentation in a software project starts prior to the software process, goes throughout all phases of SDLC and after the completion of the project. Documentation tools generate documents for technical users and end users. Technical users are mostly in-house professionals of the development team who refer to system manual, reference manual, training manual, installation manuals etc. The end user documents describe the functioning and how-to of the system such as user manual. For example, Doxygen, DrExplain, Adobe RoboHelp for documentation.
- **Analysis Tools**: These tools help to gather requirements, automatically check for any inconsistency, inaccuracy in the diagrams, data redundancies or erroneous omissions. For example, Accept 360, Accompa, CaseComplete for requirement analysis, Visible Analyst for total analysis.
- **Design Tools**: These tools help software designers to design the block structure of the software, which may further be broken down in smaller modules using refinement techniques. These tools provide detailing of each module and interconnections among modules. For example, Animated Software Design.

- **Configuration Management Tools**: An instance of software is released under one version. Configuration Management tools deal with Version and revision management, Baseline configuration management and Change control management.
- **Change Control Tools**: These tools are considered as a part of configuration management tools. They deal with changes made to the software after its baseline is fixed or when the software is first released. CASE tools automate change tracking, file management, code management and more. It also helps in enforcing change policy of the organization.
- **Prototyping Tools:** Software prototype is simulated version of the intended software product. Prototype provides initial look and feel of the product and simulates few aspect of actual product. Prototyping CASE tools essentially come with graphical libraries. They can create hardware independent user interfaces and design. These tools help us to build rapid prototypes based on existing information. In addition, they provide simulation of software prototype. For example, Serena prototype composer, Mockup Builder.
- **Web Development Tools**: These tools assist in designing web pages with all allied elements like forms, text, script, graphic and so on. Web tools also provide live preview of what is being developed and how will it look after completion. For example, Fontello, Adobe Edge Inspect, Foundation 3, Brackets.
- **Quality Assurance Tools**: Quality assurance in a software organization is monitoring the engineering process and methods adopted to develop the software product in order to ensure conformance of quality as per organization standards. QA tools consist of configuration and change control tools and software testing tools. For example, SoapTest, AppsWatch, JMeter.
- **Maintenance Tools:** It includes modifications in the software product after it is delivered. Automatic logging and error reporting techniques, automatic error ticket generation and root cause Analysis are few CASE tools, which help software organization in maintenance phase of SDLC.

1.3.9 Prototyping

Prototyping is the process of quickly putting together a working model (a prototype) in order to test various aspects of a design, illustrate ideas or features and gather early user feedback. Prototyping is often treated as an integral part of the system design process, where it is believed to reduce project risk and cost. It can also be a method used by designers to acquire feedback from users about future designs. The goal of prototyping is to support requirements determination to develop concrete specifications for the ultimate (final) system. Prototyping is most useful in the following circumstances/situation

- User requirements are not clear or well understood
- Only one or a few users involved
- Possible designs are complex
- Communication problems have existed in the past, between users and analysts

1.3.10 Modular programming

Modular programming also called "top-down design" and "stepwise refinement" is a software design technique that emphasizes separating the functionality of a program into independent, interchangeable modules, such that each contains everything necessary to execute only one aspect of the desired functionality. A module is a separate unit of software or hardware. **Modular programming means** break a large program into smaller independent modules (process of subdividing a computer program into separate sub-programs). So that modular programming is a solution to the problem of very large programs that are difficult to debug and maintain. By segmenting the program into modules that perform clearly defined functions, you can determine the source of program errors more easily. In modular programming, similar functions are grouped in the same unit of programming code and separate functions are developed as separate units of code. Modular programming enables multiple programmers to divide up the work and debug pieces of the program independently. The benefits of using modular programming include:

- Less code has to be written.
- A single procedure can be developed for reuse, eliminating the need to retype the code many times.
- Programs can be designed more easily because a small team deals with only a small part of the entire code.
- It allows many programmers to collaborate on the same application.
- The code is stored across multiple files.
- Code is short, simple and easy to understand.
- Errors can easily be identified, as they are localized to a subroutine or function.
- The same code can be used in many applications.
- The scoping of variables can easily be controlled.

- **Implementation phase:** After the design activity is completed, the next phase is the implementation or development of the software. In this phase, developers start coding according to the requirements and the design. Database administrators create the necessary data in the database, front-end developers create the necessary interfaces and GUI to interact with the back-end all based on guidelines and procedures defined by the company. Developers also write unit tests for each component to test the new code that they have written, review each other's code, create builds and deploy software to an environment. This cycle of development is repeated until the requirements are met.
- **Testing phase**: During testing, experienced testers start to test the system against the requirements. The testers aim to find defects within the system as well as verifying whether the application behaves as expected and according to what was documented in the requirements analysis phase. Testers can either use a test script to execute each test and verify the results, or use exploratory testing which is more of an experience based approach. It is possible that defects are identified in the testing phase. Once a defect is found, testers inform the developers about the details of the issue and if it is a valid defect, developers will fix and create a new version of the software which needs to be verified again. This cycle is repeated until all requirements have been tested and all the defects have been fixed and the software is ready to be shipped.
- **Maintenance phase**: The maintenance phase involves making changes to hardware, software, and documentation to support its operational effectiveness. It includes making changes to improve a system's performance, correct problems, enhance security, or address user requirements. Once a version of the software is released to production, there is usually a maintenance team that looks after any post-production issues. If an issue is encountered in the production the development team is informed and depending on how severe the issue is, it might either require a hot-fix which is created and shipped in a short period of time or if not very severe, it can wait until the next version of the software.

Self-Check 1.3	**Written Test**

Directions: Answer all the questions listed below. Use the Answer sheet provided in the next page:

I. Choose the best answer (each 1 point)

1. software development method is called____________
   - A. Preliminary investigation
   - B. Programming design
   - C. Software life cycle
   - D. Maintenance of program
2. In which of the following you defining the problem and Suggesting a solution?
   - A. Preliminary investigation
   - B. Design phase
   - C. Implementation phase
   - D. Testing phase
3. Which one is not a primary activities involved in the analysis phase are as follows:
   - A. Gathering business requirement
   - B. Creating process diagrams
   - C. Performing a detailed analysis
   - D. Documentation of the project
4. In which phase we describe the desired features and operations of the system including business rules, pseudo-code, screen layouts, and other necessary documentation?
   - A. Preliminary investigation
   - B. Design phase
   - C. Implementation phase
   - D. Testing phase
5. The sequence of computational steps to solve a problem is said to be _________?
   - A. Algorithm
   - B. Data Structure
   - C. Pseudo-code
   - D. D. Procedures
6. The way data are organized in a computer's memory is said to be_______?
   - A. Algorithm
   - B. Data Structure

C. Pseudo code

D. Data gathering technique

7. ____is used in planning an algorithm with sketching out the structure of the program before the actual coding takes place.

A. Narrative

B. Flowchart

C. Algorithm

D. Pseudo-code

8. Which tools is used to analyze a problem and visualize a solution using the top down design approach?

A. RAD

B. HIPO chart

C. DFD

D. Data structure

9. This process of modeling is called _________.

A. Data structuring

B. abstraction

C. prototyping

D. All

10. Which one is **Not** true about DFD?

A. DFDs depict flow of data in the system at various levels

B. DFD does not contain any control or branch elements.

C. DFDs depict flow of control in the system at various levels

D. All

Part II: Matching (each 1 point)

Match from column B to column A.

Column A	**Column B**
1. Algorism	A. Cardinality
2. Graphical representation for an algorithm	B. flowchart
3. Entity	C. Diamond
4. It describes relationship	D. Person
5. It defines the relationship between the entities in terms of numbers	E. Solve problem
6. Computer Aided Software Engineering	F. CASE

Part III: **Write short answer** (**5 point each**)

1. Draw the flow chart that calculates grade for ten students based on the scale: >80-A, >60-B, > 50-C, >40-D, <40-F.

2. Write a pseudo-code that calculates grade hint mark >=80 grade A, mark >=60 grade B, mark >=50 grade C, mark >=40 grade D ,F<40.

Note: **Satisfactory rating - 16 points** **Unsatisfactory - below 16 points**

You can ask you teacher for the copy of the correct answers.

Score = ____________

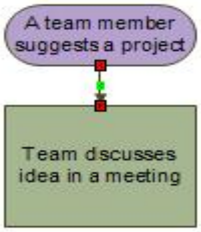
Rating: _____________

Operation Sheet 1: Creating a flowchart

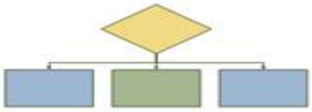
1. On the **File** menu, point to **New**, point to **Flowchart** and then click **Basic Flowchart,** then **click Ok**.
2. For each step in the process that you are documenting, drag a flowchart shape onto your drawing. First see the section what the flowchart shapes represent for information.
3. Connect 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.

Connect two shapes together.

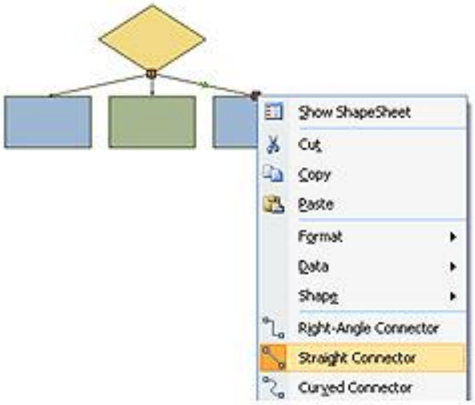
4. Click the **Connector** tool on the **Standard** toolbar.
5. Drag from a connection point ✕ on the first shape to a connection point on the second shape. The connector endpoints turn red when the shapes are connected.


A team member suggests a project
Team dscusses idea in a meeting

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.



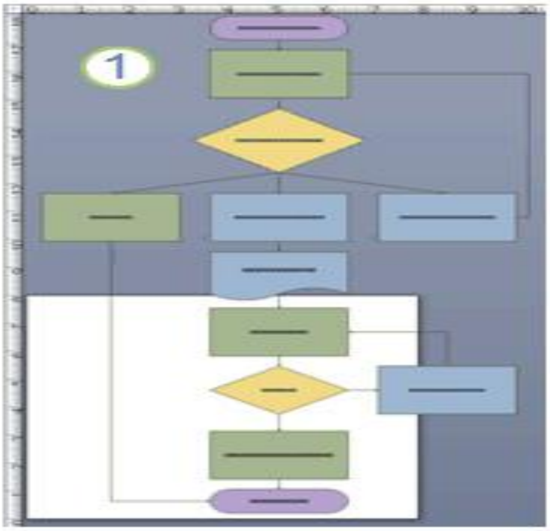
To have each connector radiate straight from the central point on the first shape to points on each of the other shapes you need to set the connectors to **Straight Connector** as shown in the following figure.

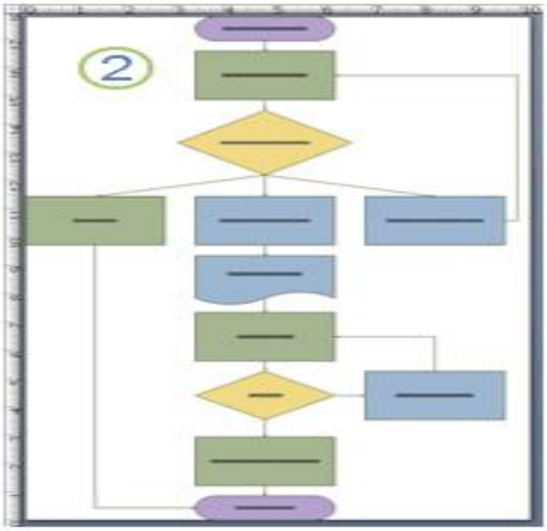


7. Click the **Connector** tool on the **Standard** toolbar.
8. For each shape you want to connect to, drag from the same connection point on the first shape to a connection point on each of the other shapes.
9. Right-click each connector and click **Straight Connector**.
10. Click the **Pointer** tool 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**.

Printing large flowcharts

The easiest way to print out a flowchart that is larger than your printer paper is to print it onto multiple pieces of paper and then tape the pieces together. Before you start printing, however, it's important to make sure that the drawing page, as it appears in Visio, contains the entire flowchart. Any shapes that hang off the edge of the Visio drawing page will not print. You can see whether the drawing page is large enough for the flowchart by checking the preview on the **Page Setup** dialog box (**File** menu, **Page Setup**, **Print Setup** tab).





1. A flowchart that is too large for the Visio drawing page.
2. A flowchart that fits the Visio drawing page.

Make your Visio drawing page fit your flowchart

1. With your flowchart open, on the File menu click **Page Setup**.
2. Click the **Page Size** tab.
3. Under **Page size** click **Size to fit drawing contents**.

To see how the flowchart will print, look at the **Print Preview**, which is on the **File** menu. The figure below shows a flowchart that prints on four pieces of letter-sized paper.

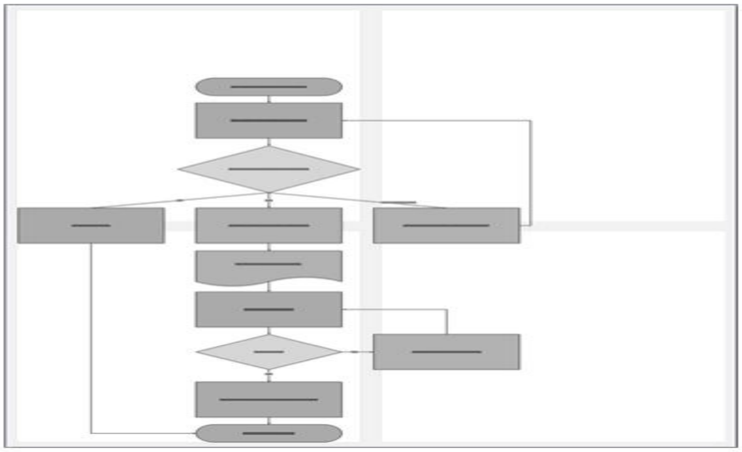


Figure 18: flowchart that prints on four pieces of letter-sized paper

Print a large flowchart onto multiple pieces of paper

1. On the **File** menu, click **Page Setup**.
2. On the **Print Setup** tab, in the **Printer paper** box, select the paper size you want if it isn't already selected. Don't click **OK** yet.
3. On the **Page Size** tab, click **Size to fit drawing contents**. The preview now shows the difference between the new page and the printer paper.
4. Click **OK**.
5. On the **File** menu, click **Print Preview** to see how the flowchart will print.

   Note If there are shaded margins between the pages, they mark areas that print on both pieces of paper so that when you assemble the pieces there are no gaps in the flowchart.
6. After the drawing is printed, you can trim the margins, overlap the pages, and tape them together.

Lap Test 1	Practical Demonstration

Name: ______________________________ Date: ________________________

Time started: _________________________ Time finished: _________________

Instructions: Given necessary templates, tools and materials you are required to perform the following tasks within **1:30** hour.

Task 1: database design, case scenario

Ambo TVET College needs a database to handle its trainee records. The trainee's asked to analyze and design the database. The college's trainee record management activities include, registering the trainee personal information, department information, trainer information, competency information, competencies given by a department, competencies attended by each trainee, and competencies given by a trainer. In order to perform these activities the college needs to keep records of:

1. Trainees identified by their id, full name, sex, birth date, department, year of entry to the college
2. Trainers identified by their id, full name, sex, educational level, field of study, and experience, department in which the trainer is working
3. Departments (like ICT, Manufacturing, Construction, Drafting, …) identified by their name, sector, and department establishment date
4. Competencies identified by unit code, title, total hours to complete the competency, competency's owner department
5. Trainee's competency records which contain records of competencies taken by trainee, the trainer who gives the competency to the trainee, and whether the trainee's result is satisfactory or unsatisfactory.

Based on the above scenario we should:

- Identify Entities, Attributes, Primary Keys (PK) and Foreign Keys (FK).
- Draw entity relationship(ER) diagram for the identified entities using Ms Visio application.

L #22	LO #2- Document the program logic or design

Instruction sheet

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

- project standards
- Structuring Diagrams of program flow and modules
- Documenting Program scope and limits
- Documenting or referencing Special routines or procedures
- Identifying and revising References for tables, files, inputs, and outputs
- Using Templates

This guide 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:

- Project standards
- Structure diagrams of program flow and modules
- Document Program scope and limits
- Document or referencing Special routines or procedures
- Identify and revising References for tables, files, inputs, and outputs,
- Use Templates

Learning Instructions:

Read the specific objectives of this Learning Guide.

1. Follow the instructions described below.
2. Read the information written in the “Information Sheets”. Try to understand what are being discussed. Ask your trainer for assistance if you have hard time understanding them.
3. Accomplish the “Self-checks” which are placed following all information sheets.
4. Ask from your trainer the key to correction (key answers) or you can request your trainer to correct your work. (You are to get the key answer only after you finished answering the Self-checks).
5. If you earned a satisfactory evaluation proceed to “Operation sheets”.
6. Perform “the Learning activity performance test” which is placed following “Operation sheets” ,
7. If your performance is satisfactory proceed to the next learning guide,
8. If your performance is unsatisfactory, see your trainer for further instructions or go back to “Operation sheets”.

Information Sheet 2.1: Project standards

2.1 Project 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 time-frame 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.

Self-Check 2.1	**Written Test**

Directions: Answer all the questions listed below. Use the Answer sheet provided in the next page:

3. Define project planning standards means

____

____

____

4. Project planning standards should include

____

____

____

____

____

Note: **Satisfactory rating - 6 points** **Unsatisfactory - below 6 points**

You can ask you teacher for the copy of the correct answers.

Score = ____________

Rating: _____________

Information Sheet 2.2: Structuring Diagrams of program flow and modules

2.2 Structuring Diagrams of program flow and modules

Program Structure Diagram

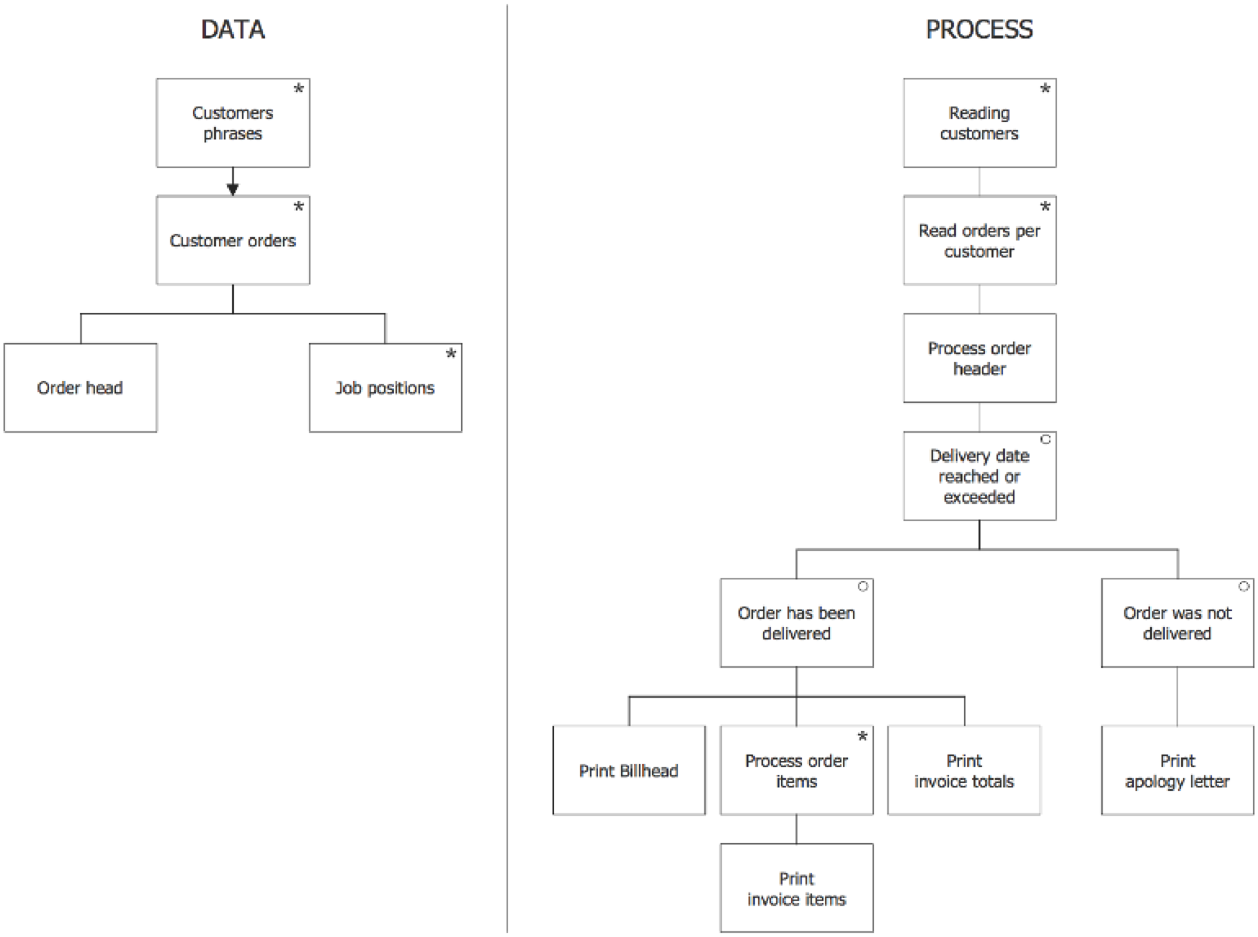
A Structure Chart in organizational theory as well as in the software engineering is known to be a chart used for showing the breakdown of some particular system to its lowest manageable levels. Such levels may be used in a structured programming for arranging the program modules into a “tree”. Each of the modules is represented in a way of a box. Such box contains the module's name and the whole tree structure visualizes the relationships between the mentioned modules. Any structure diagram or chart is widely used as a top-down modular design tool. It is constructed of squares that all represent the different modules in the system. There are also the lines that connect all these modules. The described lines represent the connections as well as the ownership of different activities. The sub-activities may be also represented with the help of the lines same way they are used in the organization charts.

In any structured analysis, structure charts are commonly used in order to specify the high-level design of some particular computer program. As a design tool, these diagrams aid the programmer in conquering and dividing some large and complicated software problem. It allows breaking a problem down into small parts so they can become well understood by a human brain.

The process is called “top-down design”. It may be also described as a so-called “functional decomposition”. Many programmers use a structure chart for building a program in a manner that is similar to the way an architect uses a blueprint in terms of building a house. The program structure diagram can be drawn and used in the design stage as a way for both the clients and the software designers to communicate. A structure diagram depicts the complexity and the size of the system, the number of the readily identifiable modules and functions as well as whether each of such functions is a manageable entity. Unless the functions are manageable entities, they should be broken down into some smaller components.

Any program structure diagram may be also used to diagram some associated elements. Such elements should be able to comprise a thread or a run stream. Being often developed as a hierarchical diagram, such elements may be represented in some other drawing way as well. Any representation must describe the breakdown of the configuration system into subsystems. A complete and accurate structure diagram may be the key to the process of determining the configuration items.

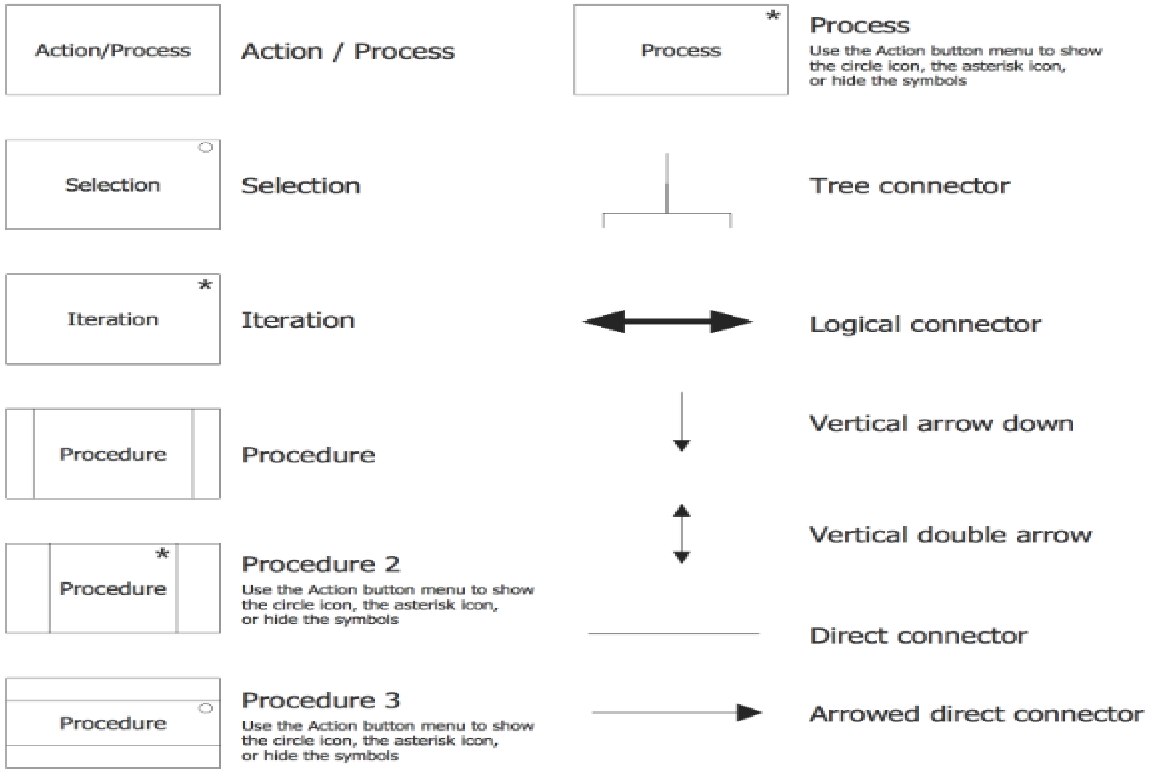
A visual representation of the configuration system, as well as the internal interfaces among such drawing's Construction Industry Scheme (CIs), may be always done with the help of the professional tool, such as the Concept Draw DIAGRAM diagramming software. The structure diagram may be used for identifying the CIs as well as their associated artifacts. Doing it with the aid of Concept Draw DIAGRAM as well as its extension (the Jackson Structured Programming (JSP) Diagrams solution) may simplify anyone's work of making such diagrams.



Example 1. Program Structure Diagram

Any structure chart can be developed by, first of all, creating its structure. It is expected to place the root of an upside-down tree. The last-mentioned tree is the one known to be forming the needed program structure diagram. Next, the conceptualization of the main sub-tasks must be performed by the program in order to solve the given problem. While creating the program structure diagrams, any programmer must be focused on each of the sub-tasks individually. We should conceptualize the way each of the tasks being broken down into even smaller ones. Having the finally broken-down program to a point where the leaves of the "tree" become representing simple methods, they can be coded with only a few program statements. Usually, first of all, there is a need for checking whether a Structure Diagram has been already developed before. In case it was, any expert needs to review such drawing in order to ensure that it represents the current structure. Unless it does, the updates need to be done to the diagram where needed.

Any data structure in computer science is known as a way of organizing and storing data in a computer. Making this data more efficiently in this way seems to be one of the best options of what to do with it. Different kinds of data structures may be well-known for fitting different applications. Some of them may be highly specialized to some definite, specific tasks. Data structures are known to be the best widely used tool for providing the needed means for managing the large amounts of data in an efficient way. These amounts of data may be internet indexing services and large databases. The efficient data structures may be a key to designing some efficient algorithms to be used in a computer science. Some programming languages and formal design methods may outline the data structures, not the algorithms, being the main thing that helps with organizing in software design. Both retrieving and storing can be carried out on data stored in secondary and in main memory.


Action/Process
Action / Process
Process
*
Process
Use the Action button menu to show the circle icon, the asterisk icon, or hide the symbols
Selection
Selection
Tree connector
Iteration
*
Iteration
Logical connector
Procedure
Procedure
Vertical arrow down
Vertical double arrow
Procedure
*
Procedure 2
Use the Action button menu to show the circle icon, the asterisk icon, or hide the symbols
Direct connector
Procedure
Procedure 3
Use the Action button menu to show the circle icon, the asterisk icon, or hide the symbols
Arrowed direct connector

Example 2. Jackson Structured Programming Symbols

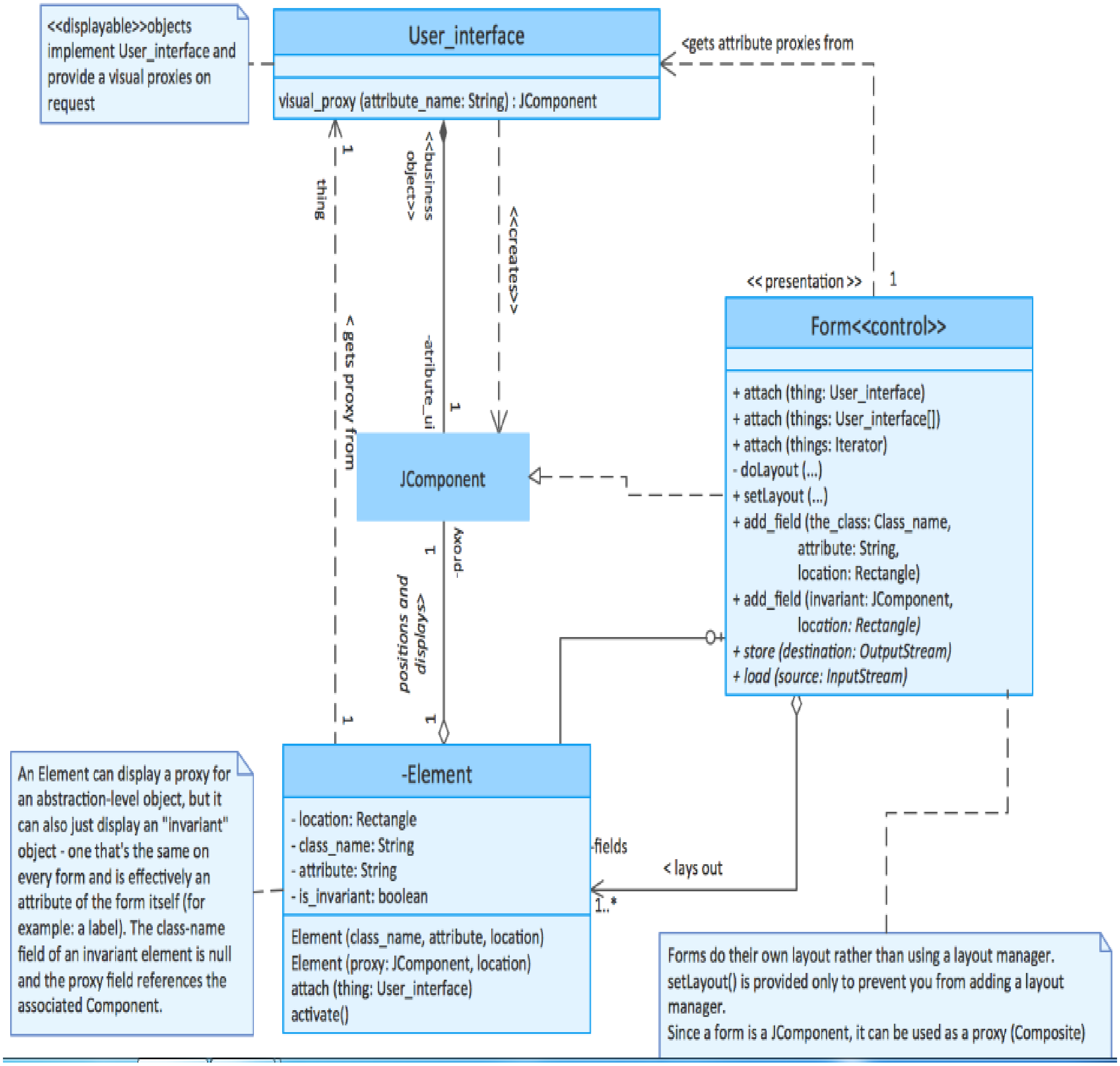
The Jackson Structured Programming (JSP) Diagrams solution from Concept Draw Solution Park provides the stencil library of design elements as well as multiple pre-made templates of the diagrams. They all can be used for quick drawing the needed diagrams of programs. The design objects from this solution may be also used while working in the Concept Draw DIAGRAM diagramming and vector drawing software.

For example Software Diagram and Templates

Concept Draw DIAGRAM is a powerful tool for business and technical diagramming. Software Development area of Concept Draw Solution Park provides 5 solutions:

- Data Flow Diagrams,
- Entity-Relationship Diagram (ERD),
- Graphic User Interface,
- IDEFO Diagrams,
- Rapid UML.

A wide collection of ready-to-use predesigned objects, templates and samples is included in these solutions to make your drawing the Software Diagrams quick, easy and effective. The examples that you see below were created in ConceptDraw DIAGRAM diagramming and vector drawing software using the solutions of Software Development area. You can choose from ConceptDraw STORE to create your Software Diagram in one moment or from Ms-Visio.


<<displayable>>objects implement User_interface and provide a visual proxies on request
User_interface
visual_proxy (attribute_name: String) : JComponent
<gets attribute proxies from
1
thing
<<business object>>
<<creates>>
<< presentation >>
1
< gets proxy from
-atribute_ui
1
JComponent
Form<<control>>
+ attach (thing: User_interface)
+ attach (things: User_interface[])
+ attach (things: Iterator)
- doLayout (...)
+ setLayout (...)
+ add_field (the_class: Class_name, attribute: String, location: Rectangle)
+ add_field (invariant: JComponent, location: Rectangle)
+ store (destination: OutputStream)
+ load (source: InputStream)
1
-proxy
<displays positions and
1
1
-Element
- location: Rectangle
- class_name: String
- attribute: String
- is_invariant: boolean
Element (class_name, attribute, location)
Element (proxy: JComponent, location)
attach (thing: User_interface)
activate()
-fields
< lays out
1..*
An Element can display a proxy for an abstraction-level object, but it can also just display an "invariant" object - one that's the same on every form and is effectively an attribute of the form itself (for example: a label). The class-name field of an invariant element is null and the proxy field references the associated Component.
Forms do their own layout rather than using a layout manager. setLayout() is provided only to prevent you from adding a layout manager.
Since a form is a JComponent, it can be used as a proxy (Composite)

UML (Unified Modeling Language)

UML is stands for **Unified Modeling Language.** UML is a standard language for specifying, visualizing, constructing and documenting the object (artifacts) of software systems. UML was created by the Object Management Group (OMG) and UML 1.0 specification draft was proposed to the OMG in January 1997.

UML was initially started to capture the behavior of complex software and non software system and now it has become an OMG standard. It captures decisions and understanding about systems that must be constructed. The UML gives you a standard way to write a system's blueprints, covering conceptual things, such as business processes and system functions, as well as concrete things, such as classes written in a specific programming language, database schemas, and reusable software components. UML is a pictorial language used to make software blueprints. OMG is continuously making efforts to create a truly industry standard. **UML** is different from other common programming languages such as C++, Java, COBOL, etc.

UML can be described as a general purpose visual modeling language to visualize, specify, construct and document software system. Although UML is generally used to model software systems, it is not limited within this boundary. It is also used to model non-software systems as well. For example, the process flows in a manufacturing unit, etc. UML is not a programming language but tools can be used to generate code in various languages using UML diagrams. The goal of UML can be defined as a simple modeling mechanism to model all possible practical systems in today's complex environment.

A conceptual model of UML

A conceptual model is the 1st step before drawing a UML diagram. It helps to understand the entities in the real world and how they interact with each other. As UML describes the real time systems, it is very important to make a conceptual model.

Class diagram

Class represents a set of objects having similar responsibilities. **Class diagram** is a static diagram. It represents the static view of an application. **Class diagram** is used for visualizing, describing and documenting different aspects of a system and also used for constructing executable code of the software application. Class diagram describes the attributes and operations of a class and also the constraints imposed on the system. **Class diagram** shows a collection of classes, interfaces, associations, collaborations and constraints. It is also known as a structural diagram. UML Class Diagram gives an overview of a software system by displaying classes, attributes, operations, and their relationships. This Diagram includes the class name, attributes, and operation in separate designated compartments. Class Diagram helps construct the code for the software application development.

How to Draw a Class Diagram?

Class diagrams have a lot of properties to consider while drawing but here the diagram will be considered from a top level view. A collection of class diagrams represent the whole system.

Remember the following points to draw a class diagram

- The name of the class diagram should be meaningful to describe the aspect of the system.
- Each element and their relationships should be identified in advance.
- Responsibility (attributes and methods) of each class should be clearly identified
- For each class, minimum number of properties should be specified, as unnecessary properties will make the diagram complicated.
- Use notes whenever required to describe some aspect of the diagram. At the end of the drawing it should be understandable to the developer/coder.
- Finally, before making the final version, the diagram should be drawn on plain paper and reworked as many times as possible to make it correct.

Class: A class is a set of objects that share the same attributes, operations, and relationships. So a class is similar to an entity type only operations are added. It is the most important building block of any object-oriented system. A class represents a relevant concept from the domain, a set of persons, objects, or ideas.

Essential elements of UML class diagram are:

- Class Name
- Attributes
- Operations

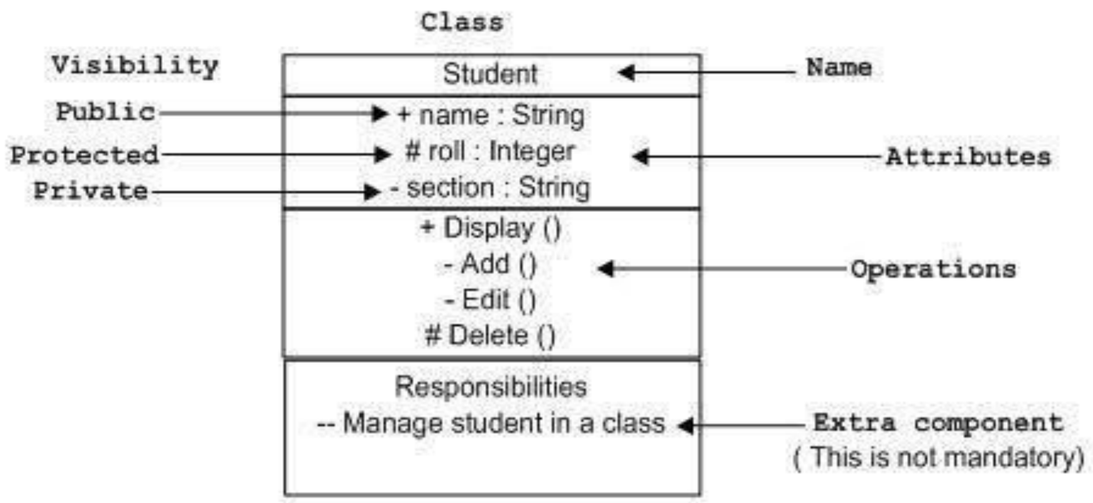
A class is symbolized by a rectangle with normally contains three compartments/sections and sometimes with one additional component. The section includes class name, attributes and operations in separate section.

Class
Attributes
Operations

- The top section is used to **name class.**
- The second one (sometime the middle section) is used to show the **attributes of the class**.

- The third (Bottom) section is used to describe the **operations** performed by the class that can be applied to individual objects.
- The fourth section is optional to show any additional components.

For example, see the figure below:



Attributes: An attribute is named property of a class which describes the object being modeled. In the class diagram, this component is placed just below the class name. An attribute of a class represents a characteristic of a class.

Student
Name: String Address: String Birthrate: Date

A derived attribute is computed from other attributes. For example, an age of the student can be easily computed from birth date.

Student
Name: String Address: String Birthrate: Date Age: date

Attributes characteristics

- The attributes are generally written along with the visibility factor.

- Public, private, protected and package are the four visibilities which are denoted by +, -, #, or ~ signs respectively.
- Visibility describes the accessibility of an attribute of a class.
- Attributes must have a meaningful name that describes the use of it in a class.

Operation: An operation is a specification of a transformation or query that an object may be called to execute. It has a name and a list of parameters.

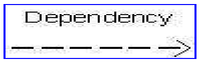
Relationships

The kinds of relationships in UML:

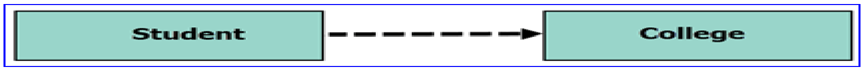
1. Dependencies
2. Generalizations
3. Associations
4. Realization

Dependency

Dependency is a relationship between two things in which change in one element also affects the other. A dependency means the relation between two or more classes in which a change in one may force changes in the other. However, it will always create a weaker relationship. Dependency indicates that one class depends on another. In the following example, Student has a dependency on College

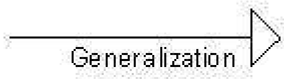


e.g.

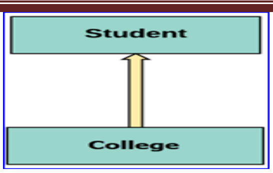


Generalization:

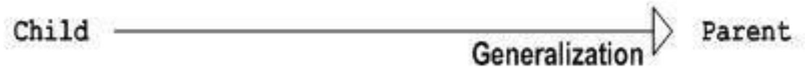
Generalization can be defined as a relationship which connects a specialized element with a generalized element. It basically describes the inheritance relationship in the world of objects. It is a parent and child relationship. A generalization helps to connect a subclass to its super class. A sub-class is inherited from its super class. Generalization relationship can't be used to model interface implementation. **Generalization** is the process of extracting shared characteristics from two or more classes, and combining them into a generalized super class. Class diagram allows inheriting from multiple super classes. In this example, the class Student is generalized from Person Class.



e.g.



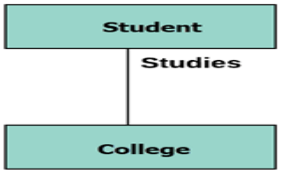
Generalization is represented by an arrow with a hollow arrow head as shown in the following figure. One end represents the parent element and the other end represents the child element. Generalization is used to describe parent-child relationship of two elements of a system.



Association

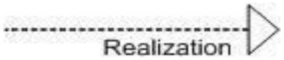
Relationships in UML are called "**Associations**". An association represents a relationship between two classes. An association indicates that objects of one class have a relationship with objects of another class, in which this connection has a specifically defined meaning. Association is basically a set of links that connects the elements of a UML model. It also describes how many objects are taking part in that relationship. This kind of relationship represents static relationships between classes A and B. For example; an employee works for an organization. Association is mostly use verb or a verb phrase or noun or noun phrase between classes. In this example, the relationship between student and college is shown by studies indicate relationship.

Association



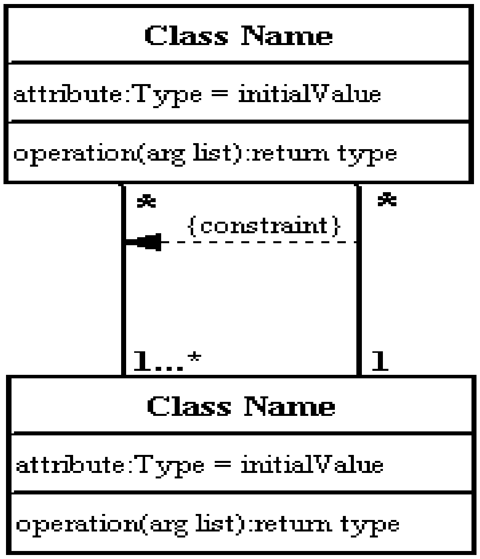
Realization

Realization can be defined as a relationship in which two elements are connected. One element describes some responsibility, which is not implemented and the other one implements them. This relationship exists in case of interfaces.

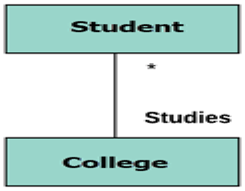


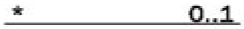
Multiplicity

Cardinalities in UML are called **multiplicities**. A multiplicity is a factor associated with an attribute. It specifies how many instances of attributes are created when a class is initialized. If a multiplicity is not specified, by default one is considered as a default multiplicity. Place multiplicity notations (association names) above, on or below the association line near the ends of an association in the form of (min, max). These symbols indicate the number of instances of one class linked to one instance of the other class.

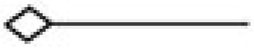

Class Name
attribute:Type = initialValue
operation(arg list):return type
*
{constraint}
*
1...*
1
Class Name
attribute:Type = initialValue
operation(arg list):return type

Let's say that that there are 100 students in one college. The college can have multiple students.

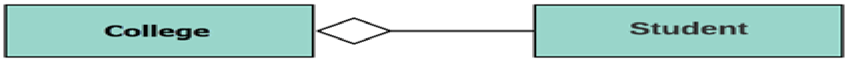


A multiplicity allows for statements about the number of objects that are involved in an association: 

Aggregation: Aggregation is a special type of association that models a whole- part relationship between aggregate and its parts. An aggregation is a special case of an association meaning “consists of”:

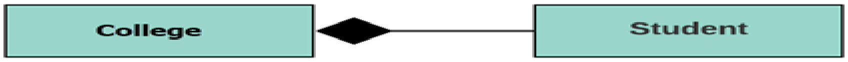


For example, the class college is made up of one or more student. In aggregation, the contained classes are never totally dependent on the lifecycle of the container. Here, the college class will remain even if the student is not available.



Aggregation indicates a relationship where the child can exist separately from their parent class. Example: Automobile (Parent) and Car (Child). So, if you delete the Automobile, the child Car still exists.

Composition: The composition is a special type of aggregation which denotes strong ownership between two classes when one class is a part of another class. For example, if college is composed of classes student. The college could contain many students, while each student belongs to only one college. So, if college is not functioning all the students also removed.



Composition display relationship where the child will never exist independent of the parent. Example: House (parent) and Room (child). Rooms will never separate into a House.

Enhanced ER diagram and UML modeling

Enhanced entity relationship diagrams are advanced database diagrams very similar to ER diagrams which represent requirements and complexities of complex databases. It is a diagrammatic technique for displaying:

- Subclasses and super classes
- Specialization and generalization
- Category or union type
- Aggregation
- Attribute, and relationship inheritance

Specialization and generalization

There are very common relationships found in real entities. Specialized classes are often called as subclass while generalized classes are called super class. A sub class is best understood by "IS A analysis". For example Technician **IS A** Employee, Laptop **Is A** computer.

An entity is specialized class of other entity. For example, Technician is special Employee in a university system; Faculty is special class of Employee. We call this phenomenon as generalization/ specialization. In the example here Employee is generalized entity class while Technician and Faculty are specialized class of Employee.

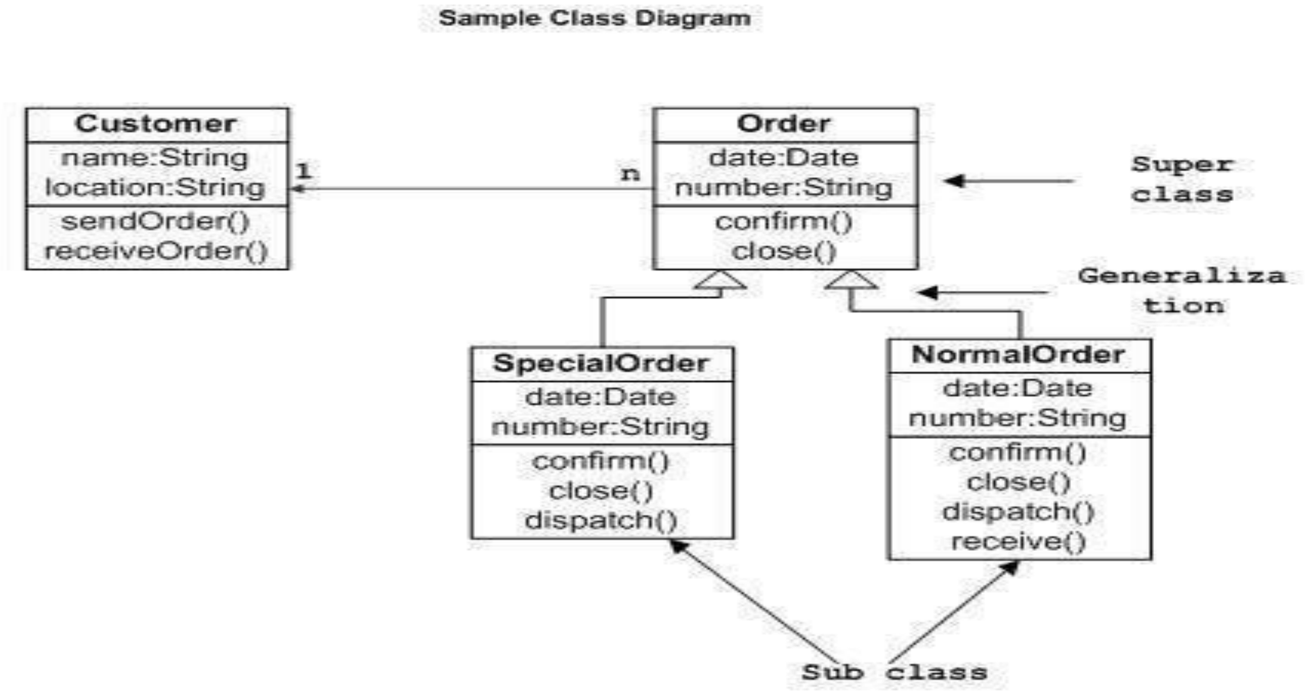
Example: This example instance of "Sub class" relationships. Here we have four sets of employee: Secretary, Technician and Engineer. Employee is super class of rest three set of individual subclass of Employee set.

The following diagram is an example of an Order System of an application.

- First of all, Order and Customer are identified as the two elements of the system. They have a one-to-many relationship because a customer can have multiple orders.

- Order class is an abstract class and it has two concrete classes (inheritance relationship) SpecialOrder and NormalOrder.
- The two inherited classes have all the properties as the Order class. In addition, they have additional functions like dispatch () and receive ().

The following class diagram has been drawn considering all the points mentioned above.



Self-Check 2.2	**Written Test**

Directions: Answer all the questions listed below. Use the Answer sheet provided in the next page.(each 3 point)

1. Why we need program structure diagram?

____

____

____

____

2. List the software Development area of Concept Draw Solution Park provides

____

____

____

____

____

3. List the kinds of relationships in UML

____

____

____

____

Note: **Satisfactory rating - 6 points** **Unsatisfactory - below 6 points**

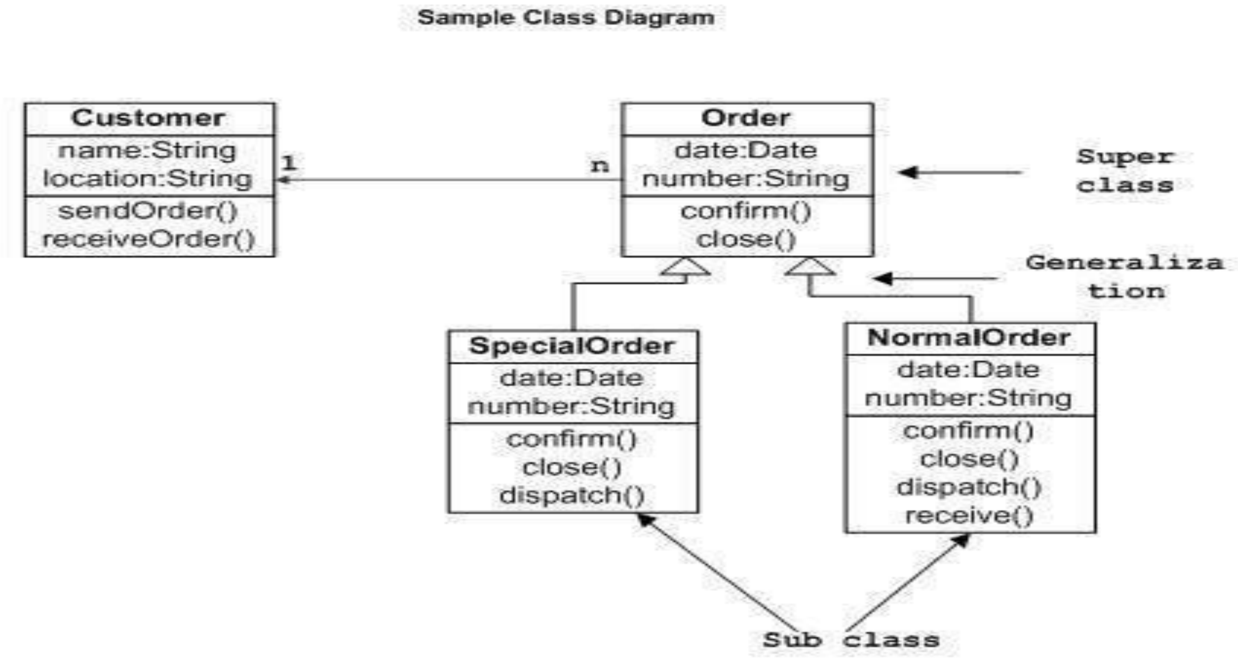
You can ask you teacher for the copy of the correct answers.

Score = ___________

Rating: ____________

Operation Sheet 2: Crating UML diagram

The following class diagram considering the entire points customer, Order, SpecialOrder and NormalOrder. Draw the class diagram as it is.



The steps you need to follow to create a class diagram.

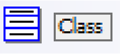
Step 1: Identify the class names: The first step is to identify the primary objects of the system.

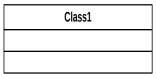
Step 2: Distinguish relationships: Next step is to determine how each of the classes or objects is related to one another. Look out for commonalities and abstractions among them; this will help you when grouping them when drawing the class diagram.

Step 3. Select the software template to create for example ms-Visio 2007

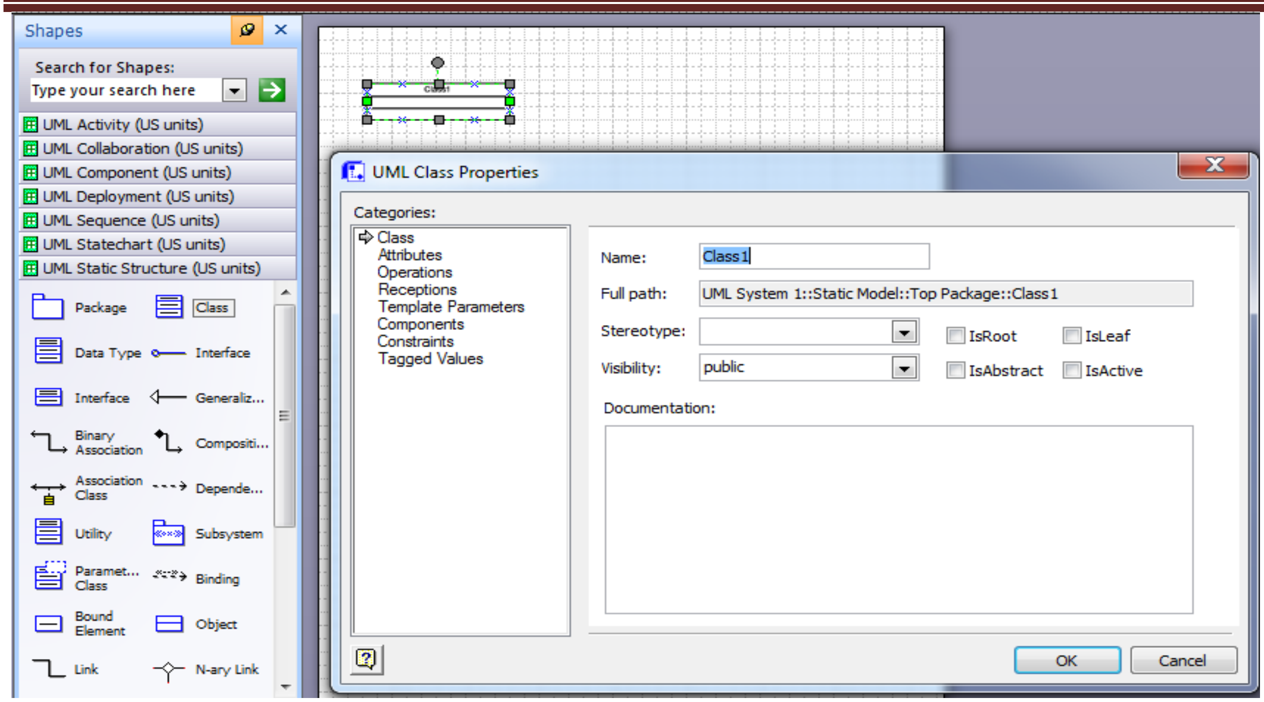
Step 4. Open Ms-Office 2007 point to software and Database and click on UML Model Diagram and click create.

Step 5: Create the Structure: First, add the class names and link them with the appropriate connectors. You can add attributes and functions/ methods/ operations.

Example: Click on 

and drag to the place you want and Right click on 

Pont to properties and click on it.



Name: Class1

Full path: UML System 1::Static Model::Top Package::Class1

Stereotype:

Visibility: public

□ IsRoot □ IsLeaf

□ IsAbstract □ IsActive

Documentation:

Change the class1 to Customer then click on Attributes to change it and click New, then click properties and change the attribute 1 to Name, click ok and follow the same steps for the rest attribute and for the other classes. At the end your class diagram must similar the first figure.

Lap Test 2	**Practical Demonstration**

Name: ______________________________ Date: ________________________

Time started: _________________________ Time finished: ________________

Instructions: Given necessary templates, tools and materials you are required to perform the following tasks within **1:00** hour.

Case Scenarios

Draw a UML class diagram representing the following elements from the problem domain for Ethiopian Football League. An Ethiopian Football League made up of least 14 clubs. Each football clubs is composed of 23 to 25 players and one player captains the team. A team has a name and a record. Players have a number and a position. All football teams play games against each other. Each game has a score and a location. Teams are sometimes lead by a coach. A coach has a level of accreditation and a number of years of experience. Coaches and players are people, and people have names and addresses. Draw a class diagram for this information, and be sure to label all associations with appropriate multiplicities.

☞ Draw the corresponding (Design a) UML class diagram for the above requirements.

Information sheet 2.3: Documenting Program scope and limits

2.3 Documenting Program scope and limits

Project scope document

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.

Self-Check 2.3	**Written Test**

Directions: Answer all the questions listed below. Use the Answer sheet provided in the next page:

1. List scope document elements you should define in project. (**6 point**)

____

____

____

Note: **Satisfactory rating - 3 points** **Unsatisfactory - below 3 points**

You can ask you teacher for the copy of the correct answers.

Score = ____________
Rating: ______________

Information sheet 2.4: Documenting or referencing Special routines or procedures

2.4 Documenting or referencing Special routines or procedures

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. It’s a simple description of what you’re working to deliver, but it also spells out small details like the quantity, quality, amount, length, or whatever other aspects accurately describe the project.
- **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.

Self-Check 2.4	**Written Test**

Directions: Answer all the questions listed below. Use the Answer sheet provided in the next page: (5 point each)

1. The list of possible elements you should consider adding to your scope statement.

____

____

____

____

____

2. What a scope of work document?

____

____

____

____

Note: **Satisfactory rating - 6 points Unsatisfactory - below 6 points**

You can ask you teacher for the copy of the correct answers.

Score = ____________

Rating: _____________

Information sheet 2.5: Identifying and revising references for tables, files, inputs, and outputs

2.5 Identifying and revising references for tables, files, inputs, outputs and other program functionalities

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

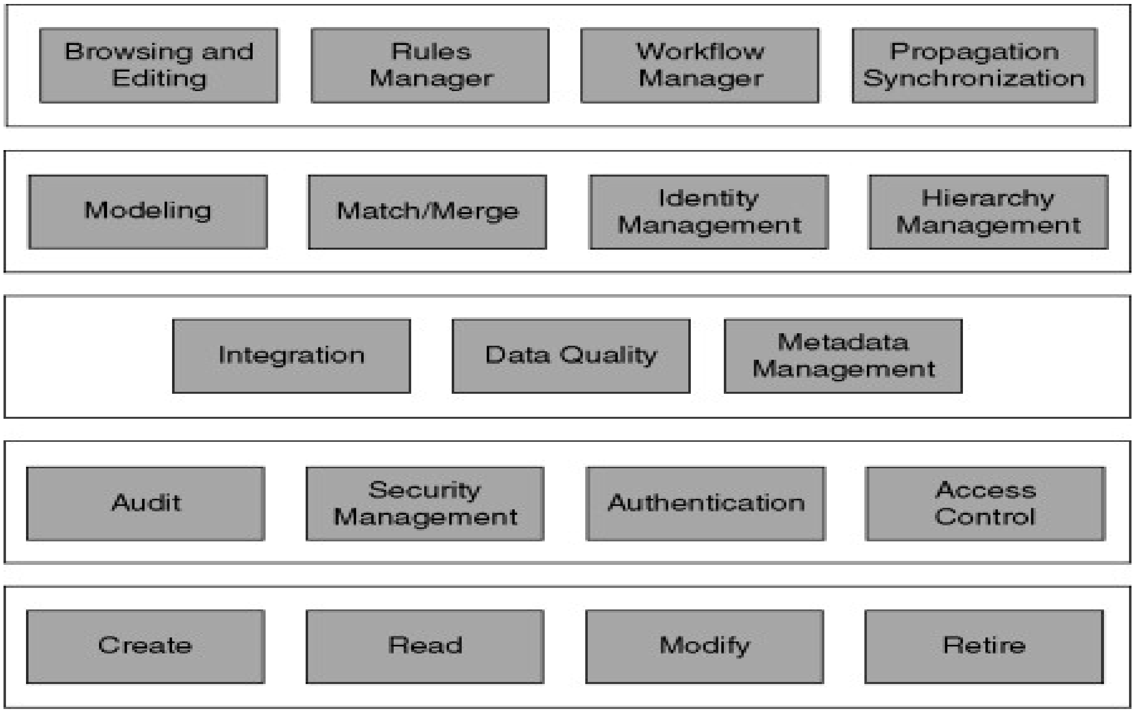


Figure 19: This Figure shows some example core services, such as the object create, update, and retire services, among others; access control; data consolidation; integration; and the components supporting consolidation, business process workflows, rule-directed operations, and so on.

Although the differences inherent across many organizations will reflect slight differences in the business's catalog of services, this framework provides a reasonable starting point for evaluation and design.

Reference Tables

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.

Note that it depends on the actual project: e.g., the simple no-history reference table has no begin-date and no end-date because there are no changes in the data. Therefore, the structure is very simple, as in the figure below.

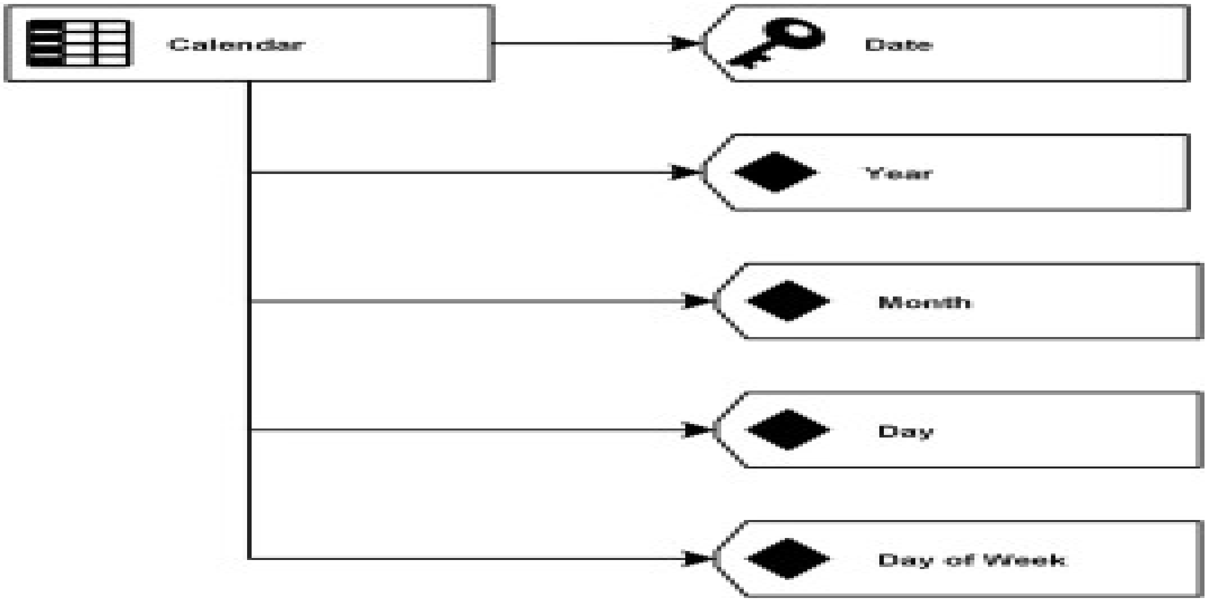


Figure 20: Figure for a non histories reference table for calendar (logical design)

This logical model shows a reference table to store a simple calendar in the Business Vault. The data is identified by the Date key, which is a Date field in the database. Other attributes in this example are the Year, Month, and Day, which store the corresponding whole numbers. *Day of* Week is the text representation of the week day, e.g. "Monday." There is no need for keeping a history of changes because there will be no need to track those in most businesses. It doesn’t mean that there are no changes to the data in this structure. However, most changes are bug-fixes or should update all information marts, including historical data. Examples for the latter include translations of the *Day of Week* attribute or abbreviating the text.

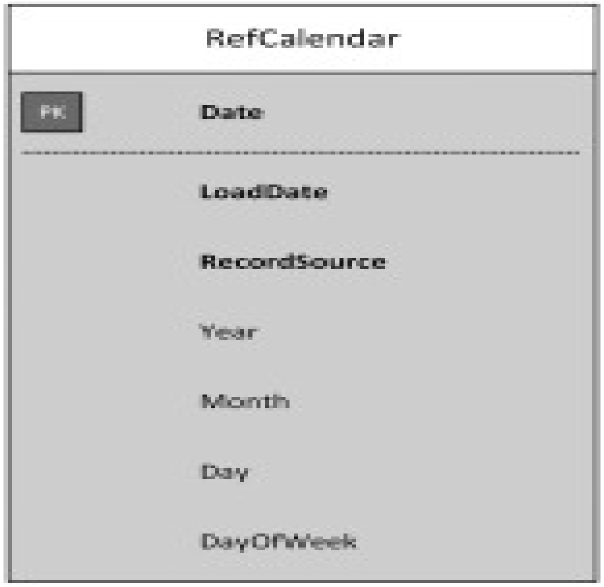


Figure 21: The Figure shows the ER model for reference table **(Physical design)**

The descriptive business key is used as the primary key of the table. The reason for this is that the key is used in satellites and Business Vault entities to reference the data in this table. That way, it becomes more readable and ensures audit ability over time. If a business key is used as the primary key of the reference table, it has the advantage that it can be used in ER models or in referential integrity, if turned on, for example for debugging purposes. The following table shows an excerpt of the reference data in the calendar table.

Date	**Load Date**	**Record Source**	**Year**	**Month**	**Day**	**Day of Week**
2000-01-01	2014-06-20 04:30:21.333	MDS	2000	1	1	Saturday
2000-01-02	2014-06-20 04:30:21.333	MDS	2000	1	2	Sunday
2000-01-03	2014-06-20 04:30:21.333	MDS	2000	1	3	Monday
2000-01-04	2014-06-20 04:30:21.333	MDS	2000	1	4	Tuesday
2000-01-05	2014-06-20 04:30:21.333	MDS	2000	1	5	Wednesday
2000-01-06	2014-06-20 04:30:21.333	MDS	2000	1	6	Thursday
2000-01-07	2014-06-20 04:30:21.333	MDS	2000	1	7	Friday

This example uses a *Record Source* attribute again because the data is sourced from Master Data Services (MDS). If the data in MDS is changed by the user, it will overwrite the content of the reference table because there is no history tracking. In other cases, the data is not sourced from anywhere. Then, the *Load Date* and the *Record Source* attributes are not needed. However, it is good practice to source the data from analytical master data because it becomes editable by the business user without the need for IT. This is a prerequisite for managed self-service business intelligence. Once the reference table has been created in the model, it can be integrated into the rest of the model by using the primary key of the reference table wherever appropriate: the biggest use is in satellites, but they are also used in Business Vault entities. The following figure shows a typical use case where a satellite on a *Passenger* hub is referencing the primary key of a reference table.

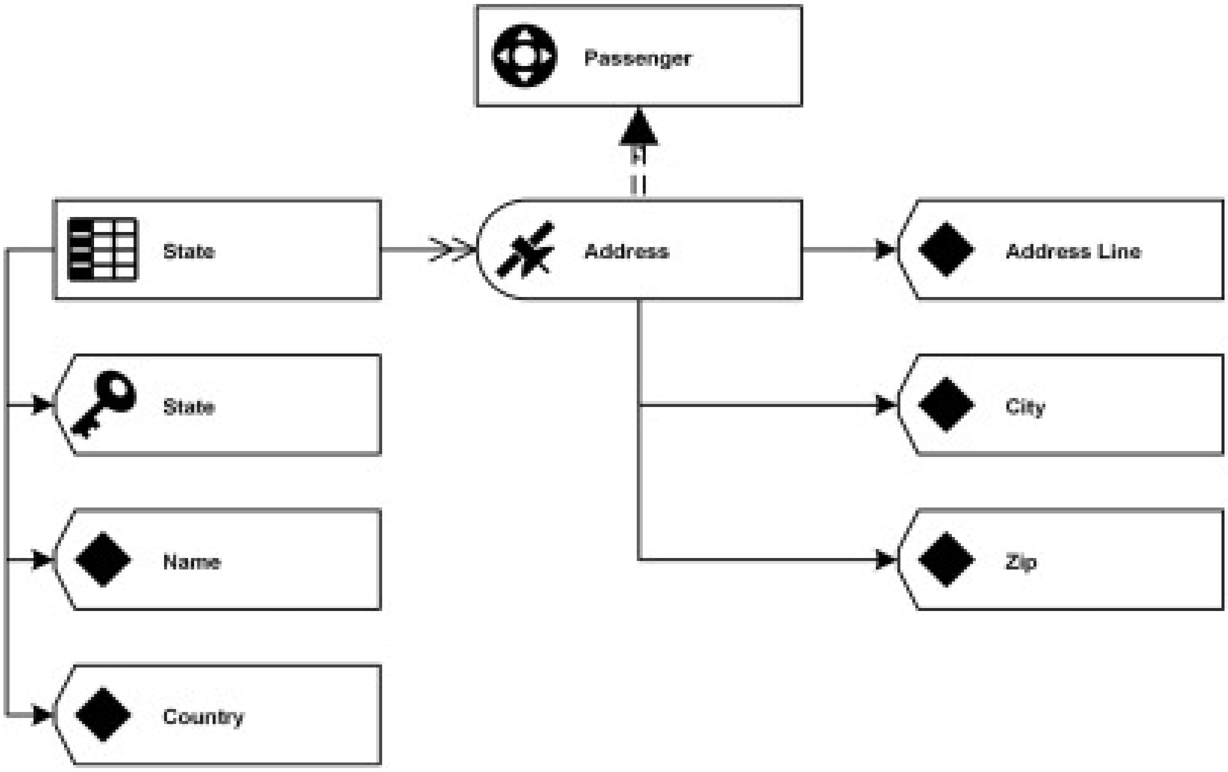


Figure 22: Passenger hub referencing the primary key

Self-Check 2.5	Written Test

Directions: Answer all the questions listed below. Use the Answer sheet provided in the next page(5 point each):

1. The three layers of master data service implementations are characterized with ____

2. The most basic reference table is just a typical table in _____________________ or ____________________ normal form.

Note: **Satisfactory rating - 6 points** **Unsatisfactory - below 6 points**

You can ask you teacher for the copy of the correct answers.

Score = ____________

Rating: _____________

Information sheet 2.6: Using Templates

2.6 Using Templates

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

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

Assumptions	***External Factors***

L #23	LO #3- Validate the design

Instruction sheet

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

- Checking Program flow, states or conditions for interfaces and compliance to design documentation requirements
- Gaining Feedback/input from appropriate person

This guide 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:

- Check program flow,
- states conditions for interfaces
- compliance to design documentation requirements
- Gain Feedback/input from appropriate person

Learning Instructions:

1. Read the specific objectives of this Learning Guide.
2. Follow the instructions described below.
3. Read the information written in the "Information Sheets". Try to understand what are being discussed. Ask your trainer for assistance if you have hard time understanding them.
4. Accomplish the "Self-checks" which are placed following all information sheets.
5. Ask from your trainer the key to correction (key answers) or you can request your trainer to correct your work. (You are to get the key answer only after you finished answering the Self-checks).

Instruction Sheet 3.1: Checking Program flow, states or conditions

3.1 Checking Program flow, states or conditions

Verification and Validation

Software verification provides objective evidence that the design outputs of a particular phase of the software development life cycle meet all the specified requirements phases. 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 project, but is not separately defined in the Quality System regulation. For purposes of this guidance, consider 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.**"** 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. The validation of software typically includes evidences that all software requirements have been implemented correctly and completely and 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 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 sending to clients. The level of confidence, and therefore the level of software validation, verification, and testing effort needed, will vary depending upon the type of project.

Self-Check 3.1	**Written Test**

Directions: Answer all the questions listed below. Use the Answer sheet provided in the next page(3 point each)

1. In practice, software validation activities occurs at

   ____

   ____

   ____

2. Software verification looks for ______________________, ________________ and __________________________________ of the software and its supporting documentation.
3. Software _________________ and _________________ are difficult because a developer cannot test forever, and it is hard to know how much evidence is enough.

Note: **Satisfactory rating - 6 points Unsatisfactory - below 6 points**

You can ask you teacher for the copy of the correct answers.

Score = ___________

Rating: ____________

Instruction Sheet 3.2: Gaining Feedback/input from appropriate person

3.2 Gaining Feedback/input from appropriate person

Getting feedback early and often is a critical component of your success. If requires organizing your work in a way that you can get feedback, being transparent so people can see what to give feedback on and incorporating that feedback into your work and getting continually better. By increasing the role of iteration and feedback in your development process, you can build better software and make happier customers. Feedback helps you learn. Feedback makes you and your work better. Whether you follow specific Agile practices or not, feedback early and often is a critical component of being more successful.

In gaining feedback focused on optimizing the following kinds of feedback workflows:

Feedback on Priorities: Am I building the most important capabilities that you want? Am I doing it in the right order? If I don't get to everything you want, have I delivered enough of the critical capabilities that the solution is useful to you and I can continue to iterate? This is all about making sure that the stakeholders and team are aligned on how the work will be organized and what capabilities (user stories, if you will) will be delivered.

Feedback on Design Does the software deliver the capability you need? Have I properly captured the requirements? Is it going to be easy enough for the customer to use? If I haven't understood what you wanted and translated that into a design that will meet your needs, then the best developers in the world aren't going to succeed. Ask development teams "Have you every built what the customer asked for and not what they wanted?" You may get a lot of people nodding vigorously. It's a symptom of a problem where poor communication both in the inability to conceptualize and communicate the requirements and in the diligence to validate the proposed solution. It is said that a picture is worth 1,000 words. Use a picture to enable the development team to work together more easily with stakeholders to ensure that before they go build a user story/scenario, they had a high fidelity conversation about what the user experience is going to be and how the scenario is going to work.

Feedback on Working Software: Does the software actually do what I think we said it would? Now that I see it in action, is the solution actually going to work? I can't tell you how many times I've built something that I thought was going to be great only to sit down and use it and realize it actually didn't mesh well with my workflow. Back to the drawing board (or storyboard) for another round of design. There's nothing like actually sitting down and using the software to get a feel for whether or not it is "good". This is one of the biggest cultural changes in teams adopting Agile practices and yet, one of the most important changes. To make sure you are building the right thing, you MUST build it in reasonably small consumable increments and solicit feedback early and often. It will save you countless weeks of wasted work developing down a path that ultimately turns out to have been a dead end that you could have avoided much earlier simply by asking "What do you think of what I have so far?" The Feedback tool and associated workflow enables a product owner/business analyst/dev lead to easily solicit feedback on a specific set of user stories/requirements on a recent build of the software. The stakeholder can easily review the user stories and the associated storyboards. They can explore the software, capture screen shots and comment on what they see. They can optionally capture video, audio and easily bookmark them to draw attention to areas where they have feedback. All of this in a feedback works item that the project owner can use to make any necessary plan updates.

Feedback on Code: While the first 3 of these are primarily focused in enabling stakeholder collaboration, let's not forget that collaboration between developers is important too. Code reviews are becoming a more and more popular way to enable feedback among developers. They can provide:

1. A fresh set of eyes to make sure you didn't miss anything.
2. A good way to stay up to date on changes happening in a body of code.
3. A good way to learn a code base and understand why changes are being made.

Good customer feedback can help to do a number of things, continually seeking feedback through a variety of sources can help you identify trend, 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 3.2	Written Test

Directions: Answer all the questions listed below. Use the Answer sheet provided in the next page :

1. Why we need gaining feedback in program development?

____

____

____

____

2. **In gaining feedback you should focuses on ;**

____

____

____

____

Note: **Satisfactory rating - 6 points** **Unsatisfactory - below 6 points**

You can ask you teacher for the copy of the correct answers.

Score = ____________

Rating: _____________

AKNOWLEDGEMENT

We wish to extend thanks and appreciation to the many representatives of TVET instructors who donated their time and expertise to the development of this TTLM.

We would like also to express our appreciation to the TVET instructors and Oromia TVET Bureaus, and Federal Technical and Vocational Education and Training Agency (FTVET) who made the development of this curriculum with required standards and quality possible.

This TTLM developed on December 2020 at Bishoftu Ethiopia.

The trainers who developed the TTLM

No	Name	Organization	Educational background & Level	Profession /Job title	Email address	Mobile
1.	Ayansa Ergiba	Ambo TVETC	Msc.	Instructor	aergiba@gmail.com	0917851343
2.	Ejigu Birhanu	Nekemte TVETC	Msc.	Instructor	ejigubiranu2011@gmail.com	0906566892
3.	Endale Lema	Adama PTC	Msc.	Instructor	endhiywet@gmai.com	0913292212
4.	Keresa Gadisa	Nekemte TVETC	Msc.	Instructor	keresag2010@gmail.com	0920420664
5.	Meseret Tezera	Atilet Kenanisa PTC	Msc.	Instructor	YafetMes@gmail.com	0911751285

REFENCES

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**

Department of Computer Science Engineering University of Texas at Arlington **Shamkant B. N avathe**

Beginning SQL Server 2005 For Developers (2006) **Robin Dewson**,

Beginning SQL (2005) **Paul Wilton and John W. Colby**

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**

Answer Key

LO #1- Select the Program Design Approach	
Self-Check 1.1	**Written Test**

Directions: Answer all the questions listed below. Use the Answer sheet provided in the next page:

I. Choose the best answer (1 point each)

1. A 2. D 3. C 4. D 5. A

Self-Check 1.2	**Written Test**

Directions: Answer all the questions listed below. Use the Answer sheet provided in the next page:

I. Choose the best answer (1 point each)

1. A 2. C 3. D 4. D 5. C

Self-Check 1.3	**Written Test**

Directions: Answer all the questions listed below. Use the Answer sheet provided in the next page:

I. Choose the best answer (each 1 point)

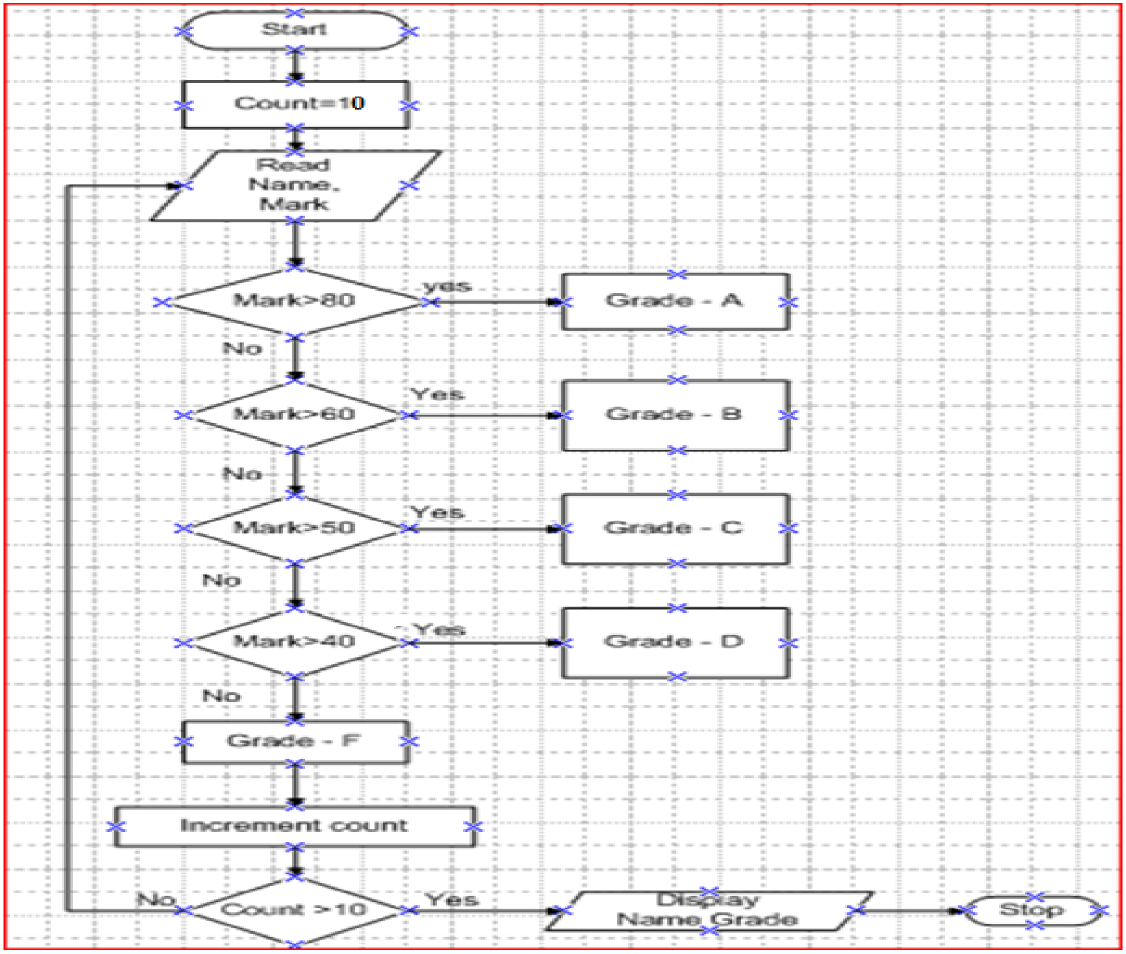
1. C 2. A 3. D 4.B 5. A 6. B 7.D 8.B 9.B 10.C

II. Part II: Matching (each 1 point)

1. E 2. B 3.D 4.C 5.A 6.F

III. **Part III**: **Write short answer** (**5 point each**)

1. Draw the flow chart that calculates grade for ten students based on the scale: >80-A, >60-B, > 50-C, >40-D, <40-F.



2. Write a pseudo-code that calculates grade hint mark >=80 grade A, mark >=60 grade B, mark >=50 grade C, mark >=40 grade D, 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
```

Lap Test 1	**Practical Demonstration**

Recommended entities with attributes and the primary keys of each entity are underlined.

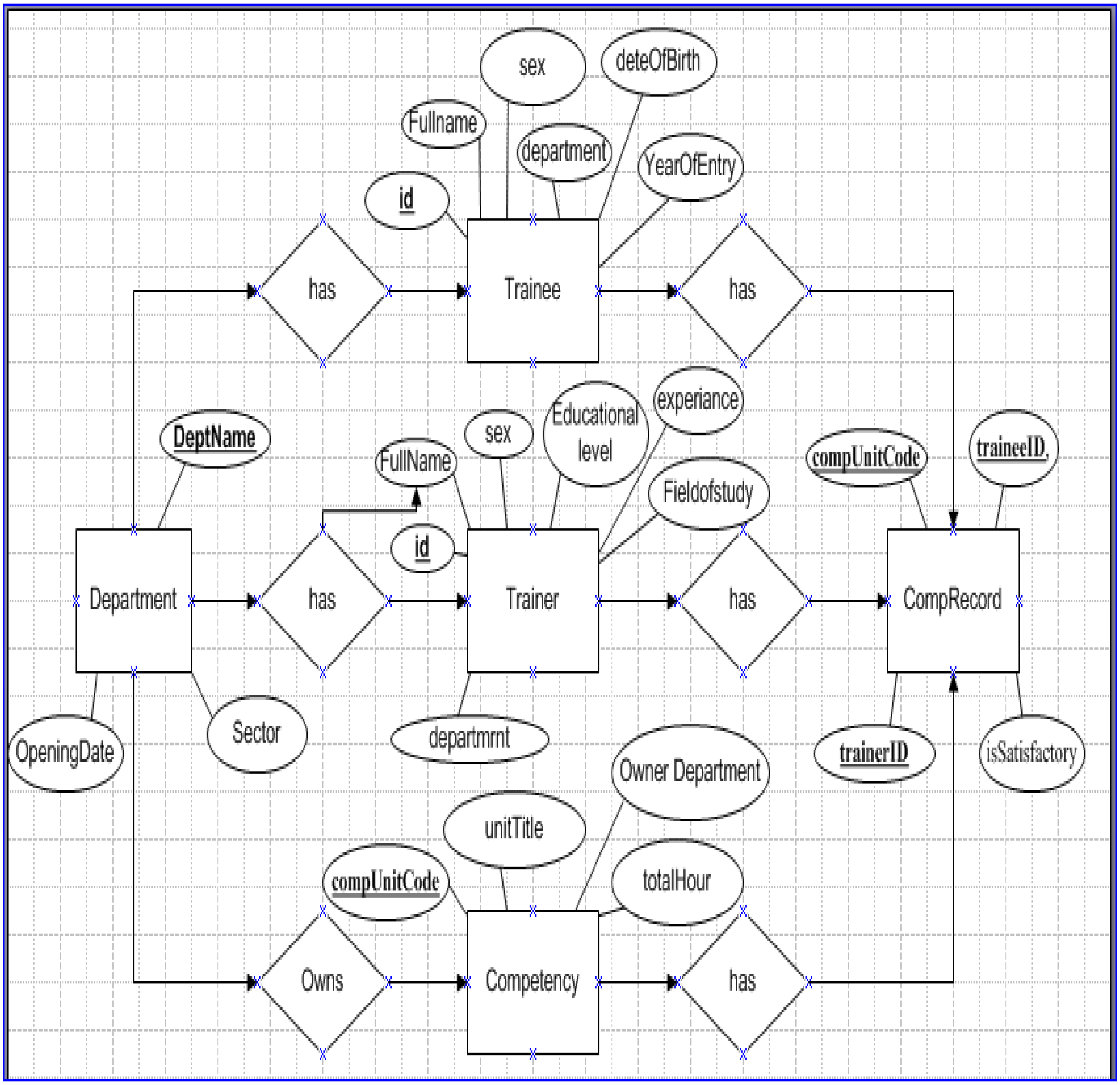
Trainee (**id**, fullName, sex, dateOfBirth, *department*, yearOfEntry, …)

Trainer (**id**, fullName, sex, educationLevel, fieldOfStudy, experience, *department*, …)

Department (**departmentName**, sector, OpeningDate, …)

Competency (**compUnitCode**, unitTitle, totalHour, *ownerDepartment*, …)

CompRecord (**compUnitCode**, **traineeID**, **trainerID**, isSatisfactory, …)



LO #2- Document the program logic or design	
Self-Check 2.1	**Written Test**

Directions: Answer all the questions listed below. Use the Answer sheet provided in the next page:

1. Define project planning standards means
   - Project planning standards may obviously vary from project to project, but the goals are usually the same to complete the project within the time-frame and without exceeding the allotted resources.
2. Project planning standards should include
   - Quality Assurance Necessities
   - Risk Management Plan
   - Security Measures
   - Testing Techniques
   - Documentation and Portfolio Requirements

Self-Check 2.2	**Written Test**

Directions: Answer all the questions listed below. Use the Answer sheet provided in the next page. (Each 3 point)

1. Why we need program structure diagram?

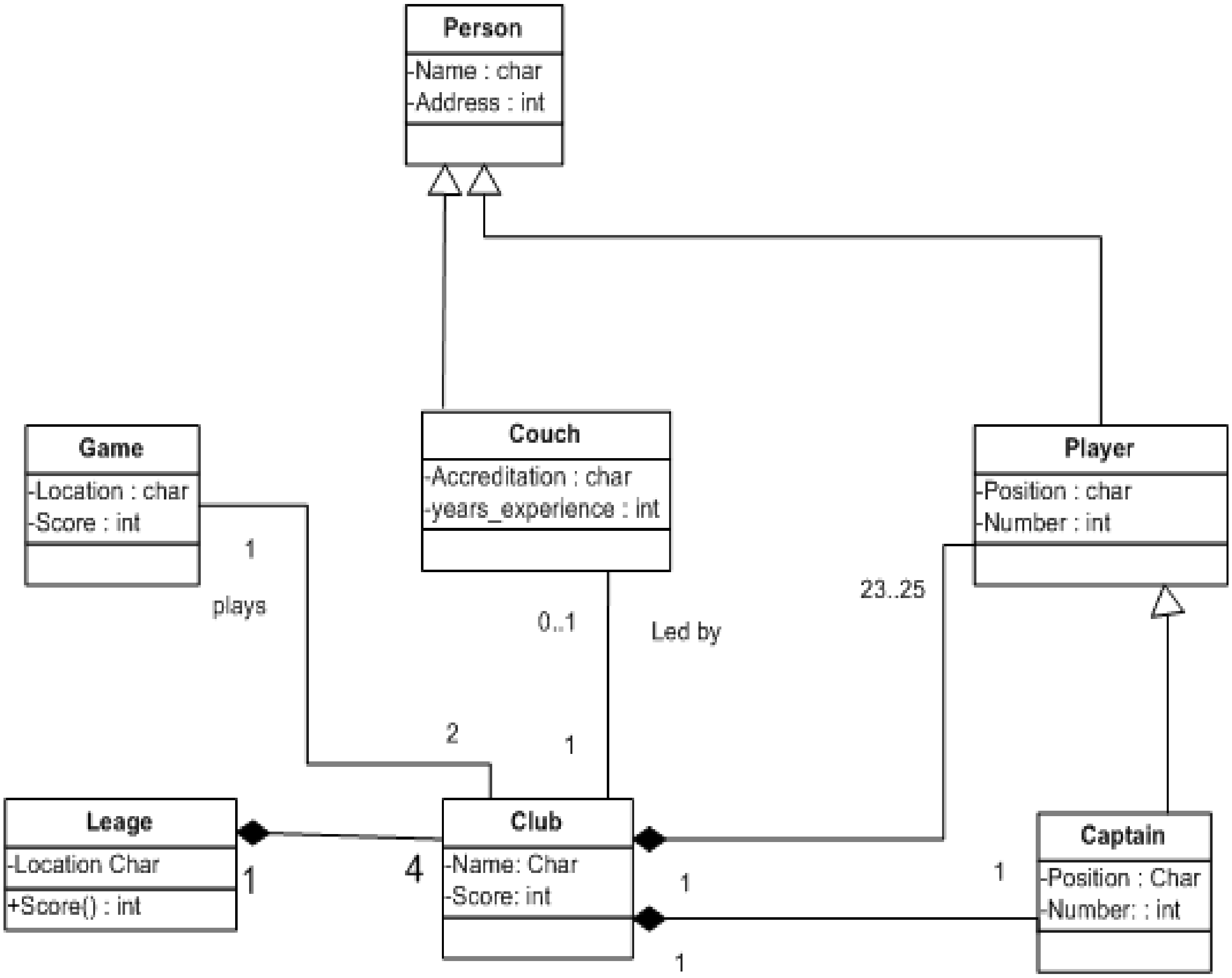
   Program structure diagram is used for showing the breakdown of some particular system to its lowest manageable levels. Such levels may be used in a structured programming for arranging the program modules into a "tree".

2. List the type of software you can use to draw structural diagram
   - Data Flow Diagrams,
   - Entity-Relationship Diagram (ERD),
   - Graphic User Interface,
   - IDEFO Diagrams,
   - Rapid UML.

3. List the kinds of relationships in UML
   - Dependencies
   - Generalizations
   - Associations
   - Realization

Lap Test 2	Practical Demonstration

Answer for Draw the corresponding (Design a) UML class diagram for the above requirements.



Self-Check 2.3	Written Test

Directions: Answer all the questions listed below. Use the Answer sheet provided in the next page:

1. List scope document elements you should define in project. (**6 point**)

   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

Self-Check 2.4	**Written Test**

Directions: Answer all the questions listed below. Use the Answer sheet provided in the next page: (5 point each)

1. The list of possible elements you should consider adding to your scope statement.

The list of possible elements you should consider adding to your project scope statement are

- Business case and goals
- Project description and deliverables
- Acceptance criteria
- Limitations
- Assumptions
- Exclusions
- Costs
- Agreement

2. What a scope of work document?

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.

Self-Check 2.5	**Written Test**

Directions: Answer all the questions listed below. Use the Answer sheet provided in the next page(5 point each):

1. The three layers of master data service implementations are characterized with

- Core services,
- Object services,
- Business application services,

2. The most basic reference table is just a typical table in **third** or **second normal form**.

LO #3- Validate the design	
Self-Check 3.1	**Written Test**

Directions: Answer all the questions listed below. Use the Answer sheet provided in the next page (3 point each)

1. 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.
2. Software verification looks for **consistency**, **completeness**, and **correctness** of the software and its supporting documentation.
3. Software **verification** and **validation** are difficult because a developer cannot test forever, and it is hard to know how much evidence is enough.

Self-Check 3.2	Written Test

Directions: Answer all the questions listed below. Use the Answer sheet provided in the next page (3 point each)

1. Feedback helps you learn. Feedback makes you and your work better. Whether you follow specific Agile practices or not, feedback early and often is a critical component of being more successful.
2. In gaining feedback focused on optimizing the following kinds of feedback workflows:
   - Feedback on Priorities
   - Feedback on Design
   - Feedback on Working Software:
   - Feedback on Code