DATE 07/11/2023
PAGE

The Process & Stages of System Design:

System design is a solution, a "How-to" approach to the creation of a new system.

- Operational translation
- detailed implementation of the recommended system.

Logical and physical design:

data flow diagram (DFD): Logical flow of a system and defines the boundaries of the system. It describes inputs (source) output (destination), databases (data stores) and procedures (data flow).

The design covers the following:

1. Reviews the current physical system
2. Prepare output specifications
3. Prepare input specifications.
4. Prepare edit, security and control specifications.
5. Specifies the implementation plan.
6. Prepare a logical design walkthrough
7. Reviews benefits, costs, targets, dates and system constraints.

UNIT IV

Module Specification

Module is the way to improve the structure design by break down the problem for solving it into independent task.

Advantages of Module –

I. It breakdown the problem into independent modules so the complexity of the problem can be minimized.

II. Each independent module can be easily assigned to the various members of the development team.

III. Module can be easily run and tested independently from another.

Top-Down design approach:-

It is a technique of breakdown a problem into major tasks to be performed. Each task is then further broken down into separate sub task and so on until each sub task is sufficiently simple to be written as a self contained module.

In Top-Down design we initially describes the problem at the highest level that descript what must be done and It does not show how it must be done. Top-Down methods are used throughout the system analysis and design process. The value of using top-down approach, starting at general level and to understand and gain the system and moving down to the levels of greater details.

Advantages of Top-Down Approach –

I. By dividing of the problem into number of sub problems we have made it easier to share problem development.

II. It is easy to debug a large program as a number of smaller units rather than one big problem.

III. It is good way to delay decision on problems whose solution is not readily prepared.

IV. It allows a programmer to remain on top of a problem and view the developing solutions. The solution always proceeds from the highest level to the lowest level.

V. It becomes an ideal structure for managing the implementation of a computer program using team of programmers.

Bottom-Up design approach:-

When we face a large and complex problem , it is difficult to see how the whole thing can be done so it may easier to solve the part of the problem individual, taking the common and easy aspects first and then more difficult task and finally gather them all together to form complete solution, this is called bottom-up approach.

The bottom-up approach suffers from disadvantage that the part of the program may not fit together very easily and there may be a lack of consistency between modules and reprogramming have to be done.

Module Cohesion & Coupling:-

i) Modules:

Module is represented by a rectangle box with module name. After defining general purpose of each program it is divided into modules. Each of which performs a single function.

For Ex:-

An accounting system consists of many separate modules that are invoked one at a time as user wish to perform particular function. Each upper level module leads to using one or more lower level modules until the desired function is performed.

ii) Connectors:

Connections between modules are represented using links, modules. It is drown by on arrow line in downward direction.

iii) Coupling:

It refers to the strength of relationship between modules in the system. Coupling measures the degree to which two distinct modules are bound together.

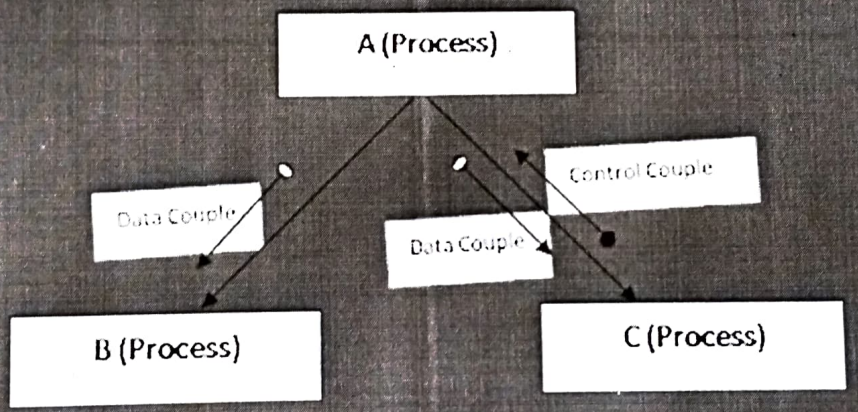
It can be of two types –

a) Data Coupling:

It carries data between two modules and it can have upward or downward direction. Data flows are shown using named arrow with a small circle at one end. It is shown by an arrow with blank circle at the tail.

b) Control Coupling:

It carries info about data couple and does not carry any data. It is shown by an arrow with filled circle at the tail.



iv) Cohesion:

It is used to denote the intra module strength. It is a relationship among elements within a module. A system designer should aim at minimizing coupling and maximizing cohesion to get modular design. Designer insures that each module performs a specific function and can be developed independently. A good design has low coupling i.e. low interdependence between modules and high cohesion i.e. high interrelationship within the elements of a single module.

The worst degree of cohesion

Coincidental cohesion is found in a component whose parts are unrelated to another.

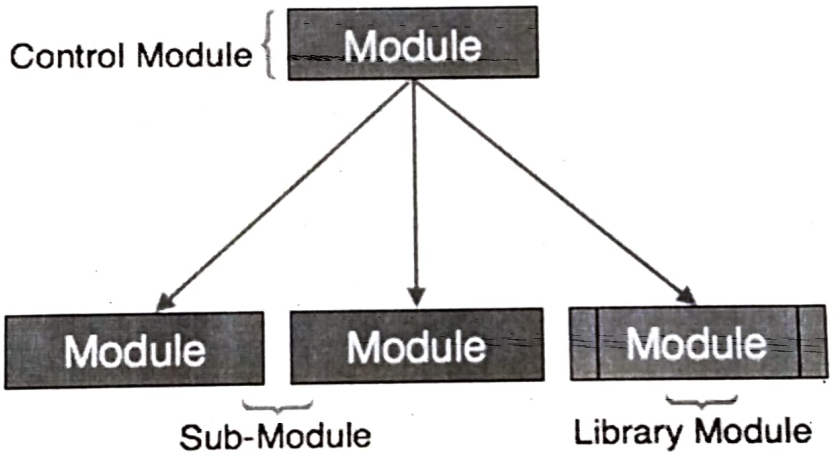
- **Logical Cohesion** – It is where several logically related functions or data elements are placed in same component.

- **Temporal Cohesion** – It is when a component that is used to initialize a system or set variables performs several functions in sequence, but the functions are related by timing involved.
- **Procedurally Cohesion** – It is when functions are grouped together in a component just to ensure this order.
- **Sequential Cohesion** – It is when the output from one part of a component is the input to the next part of it.

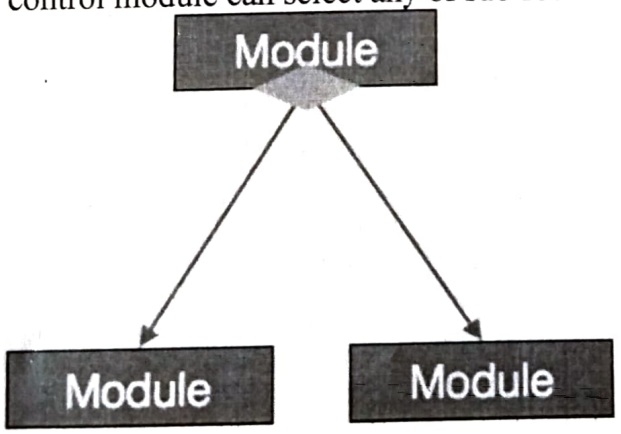
Structure Charts

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 -

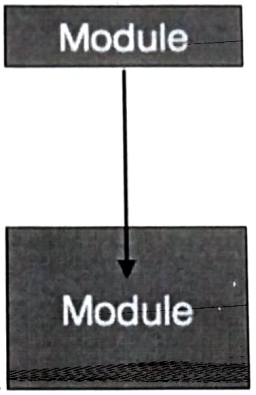
- **Module** - It represents process or subroutine or task. A control module branches to more than one sub-module. Library Modules are re-usable and invokable from any module.



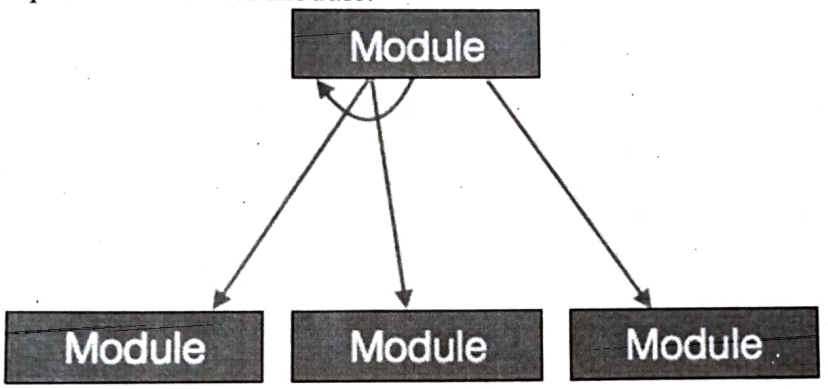
- **Condition** - It is represented by small diamond at the base of module. It depicts that control module can select any of sub-routine based on some condition.



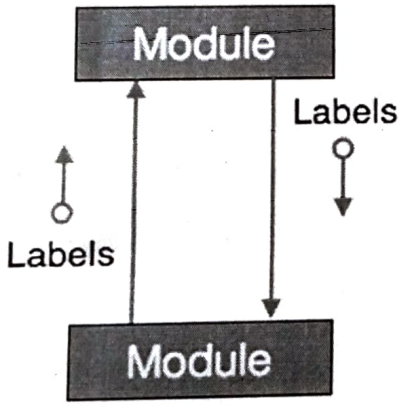
- **Jump** - An arrow is shown pointing inside the module to depict that the control will jump in the middle of the sub-module.



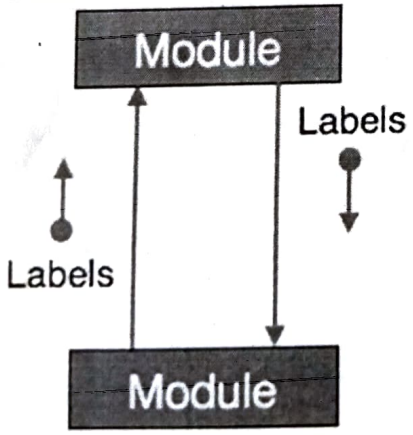
- **Loop** - A curved arrow represents loop in the module. All sub-modules covered by loop repeat execution of module.



- **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.



PROCESS MODELLING

Process modeling is the graphical representation of business processes or workflows. Like a flow chart, individual steps of the process are drawn out so there is an end-to-end overview of the tasks in the process within the context of the business environment.

A process model allows visualization of business processes so organizations can better understand their internal business procedures so that they can be managed and made more efficient. This is usually an agile exercise for continuous improvement.

Process modeling is a vital component of process automation, as a process model needs to be created first to define tasks and optimize the workflow before it is automated.

Benefits of using process modelling

The act of process modeling provides a visualization of business processes, which allows them to be inspected more easily, so users can understand how the processes work in their current state and how they can be improved. Other benefits from process modeling include:

Improve efficiency – process modeling helps to improve the process, helping business workers to be more productive by saving time

Gain transparency – modeling provides a clear overview of the process, identifying the start and end point and all the steps in between

Ensure best practice – using process models ensures consistency and standardization across the organization

Create understanding – by using the common language of process, it makes it easier for users across the organization to communicate with each other

Business orchestration – supports the coordination of people, systems and information across the organization to support business strategy

Difference between process modeling and data modeling

Process modeling describes what must be done(requirements) or what will be done(design) where as Data modeling describes what must be known or what data will be stored. You cannot do one without considerable understanding the other.

Logical Design

Logical design pertains to an abstract representation of the data flow, inputs, and outputs of the system. It describes the inputs (sources), outputs (destinations), databases (data stores), procedures (data flows) all in a format that meets the user requirements.

While preparing the logical design of a system, the system analyst specifies the user needs at level of detail that virtually determines the information flow into and out of the system and the required data sources. Data flow diagram, E-R diagram modeling are used.

Physical Design

Physical design relates to the actual input and output processes of the system. It focuses on how data is entered into a system, verified, processed, and displayed as output.

It produces the working system by defining the design specification that specifies exactly what the candidate system does. It is concerned with user interface design, process design, and data design.

It consists of the following steps –

- Specifying the input/output media, designing the database, and specifying backup procedures.
- Planning system implementation.
- Devising a test and implementation plan, and specifying any new hardware and software.
- Updating costs, benefits, conversion dates, and system constraints.

Architectural Design

It is also known as high level design that focuses on the design of system architecture. It describes the structure and behavior of the system. It defines the structure and relationship between various modules of system development process.

Detailed Design

It follows Architectural design and focuses on development of each module.

Conceptual Data Modeling

It is representation of organizational data which includes all the major entities and relationship. System analysts develop a conceptual data model for the current system that supports the scope and requirement for the proposed system.

The main aim of conceptual data modeling is to capture as much meaning of data as possible. Most organization today use conceptual data modeling using E-R model which uses special notation to represent as much meaning about data as possible.

Entity Relationship Model

It is a technique used in database design that helps describe the relationship between various entities of an organization.

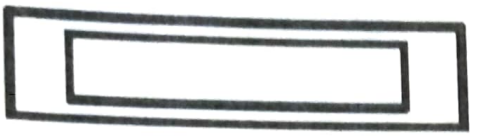
Terms used in E-R model

- **ENTITY** – It specifies distinct real world items in an application. For example: vendor, item, student, course, teachers, etc.
- **RELATIONSHIP** – They are the meaningful dependencies between entities. For example, vendor supplies items, teacher teaches courses, then supplies and course are relationship.
- **ATTRIBUTES** – It specifies the properties of relationships. For example, vendor code, student name. Symbols used in E-R model and their respective meanings –

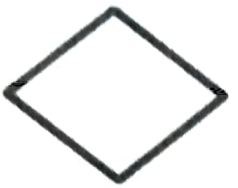
The following table shows the symbols used in E-R model and their significance –

Symbol	Meaning

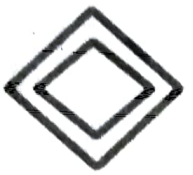
Entity



Weak Entity

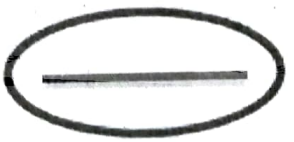


Relationship

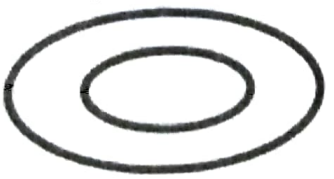


Identity Relationship

Attributes



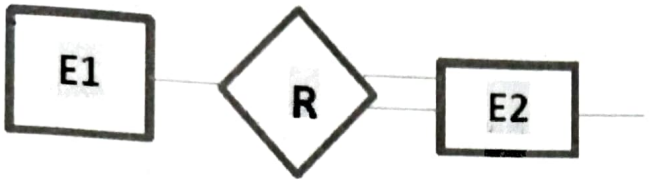
Key Attributes



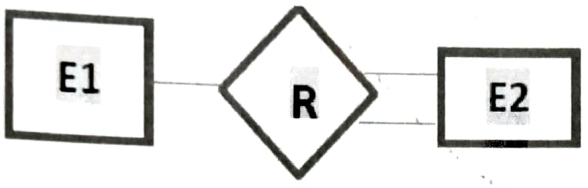
Multivalued

Composite Attribute

Derived Attributes



Total Participation of E2 in R



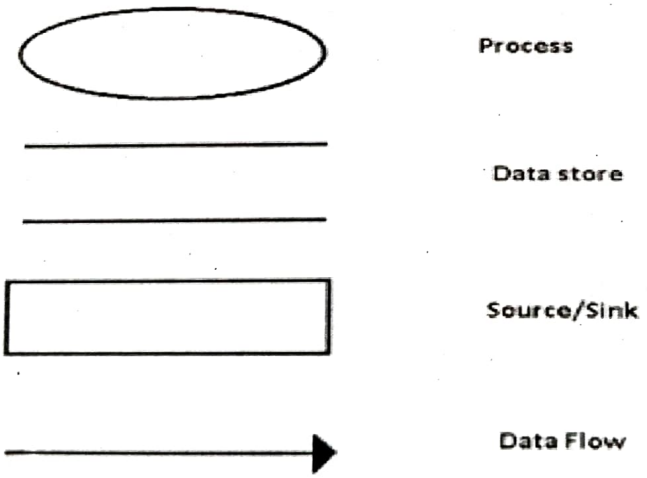
Cardinality Ratio 1:N for E1:E2 in R

Three types of relationships can exist between two sets of data: one-to-one, one-to-many, and many-to-many.

Data Flow Diagram

The Data Flow Diagram is a hierarchical graphical model of a system. It specifies various activities or functions to be performed by the system and the data interchange between these functions. A function in simple words, accepts data input, processes it and generates output. A DFD reveals information flow in a system, storage location and transformation of data through a system. DFD's are extremely useful in bridging the gap between the informal descriptions of a system and how information flows through a system. A system consists of number of functions.

Following is a list of basic symbols used in DFD:



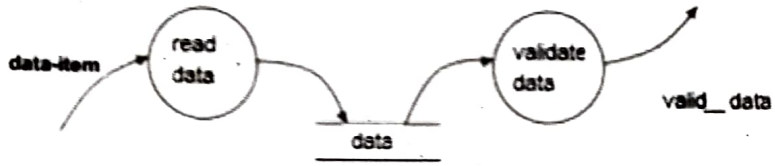
These symbols are used to represent the functions performed by a system and the flow of data between these functions.

- **Circle with names** is used to define the transformations or functions.
- **An arrow** is used to specify the direction of data flow, and its label identifies the data.
- **Rectangles** are used to indicate the source and destination of data.
- **Parallel lines** are used to indicate the permanent storage like files, databases, etc.

A DFD with single bubble or circle is called context level and it specifies the highest level of abstraction. Initially a DFD consists of single bubble only. A DFD is then decomposed into child-DFD's,and the process is called levelling. The decomposition process is then repeated for the child DFD's and it is re-iterated until the data flow for a software process is clearly understood.

Data flow between the functions can either be synchronous or asynchronous.

Observe the figure above; both the functions are directly connected by a data flow. It means that 'validate data' function can begin only after the read data function has successfully supplied data to it. This type of flow is called synchronous data flow.



Asynchronous data flow

In this figure, the two functions are connected through a data store and hence the two functions are independent. This type of flow is called asynchronous data flow

Importance of DFDs in a good software design

DFD is east to construct and easy to understand. It decomposes the high-level functions of a system into number of sub-functions. A DFD follows a very simple set of rules to specify the flow of data between the functions. DFD is also used for representing variety of documents, other than structured analysis.

Structured English

Structured English is an attempt to allow the use of natural language, stripped of ambiguity to express actions to be taken under particular conditions. This is accomplished by choosing a simple subset of natural language verbs and nouns, and defining constructs to express sequence, selection and iteration.

Structured English is derived from structured programming and its use of logical construction and imperative statements. This process is designed to carry out instructions for actions by creating decision statements that use structured programming terms, such as "IF", "ELSE" and "THEN".

The following guidelines are used when writing Structured English

- ✓ All logic should be expressed in operational, conditional, and repetition blocks.
- ✓ Logical blocks should be indented to show relationship and hierarchy.

- ✓ Keywords should be capitalized.
- ✓ Group blocks of statements together, with a capitalized name that describes their function and end with an EXIT.
- ✓ Mark comment lines with an asterisk

For example, Consider the following problem and its specification using Structured English.
PROBLEM- Granting a loan by bank.

Solution- Loan will be granted by bank only under the following conditions:
1. If a customer has an account with the bank and has no loan outstanding; loan will be granted.
2. If a customer has an account with the bank but some amount is outstanding from previous loans then loan will be granted if special approval is given. For all other cases, reject the loan application.

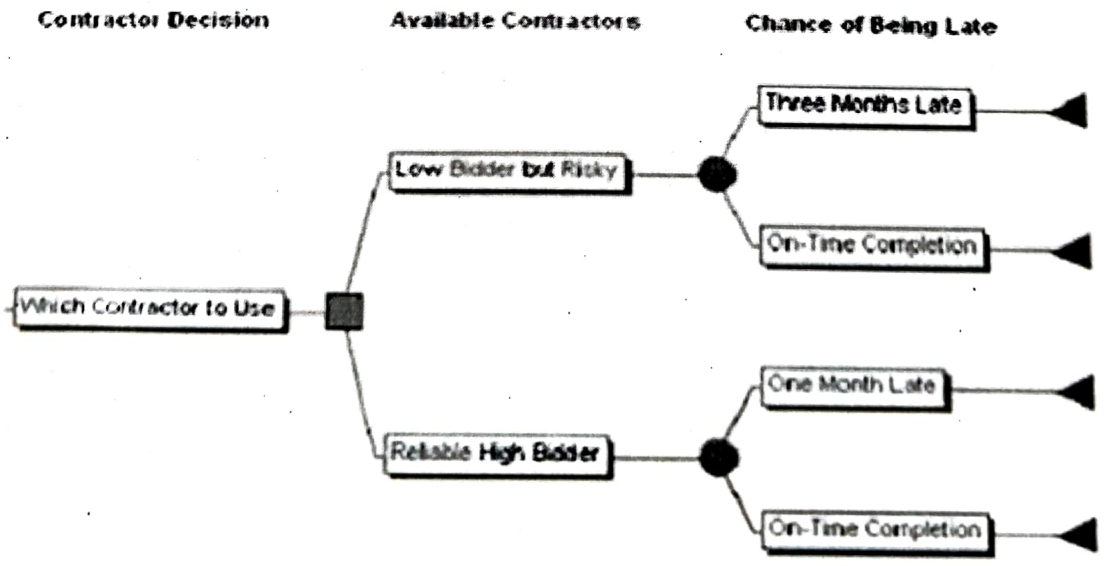
Decision tree

A decision tree is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. A decision tree consists of nodes and branches. It grows as it moves from left to right. At the top it has root decision node (square, also called a choice node) the branches of which represent two or more competing options available to the decision makers. At the end of these initial branches, there is either an end node (triangle, also called a value node) or an uncertainty node (circle, also called a chance node). The end node represents a fixed value. The circled branches represent the possible outcomes along with their respective probabilities.

Decision rules represent the linear form of decision trees in which the outcome is the contents of the leaf node and the conditions along the path form a conjunction in the 'if' clause. In general, the rules have the form:

if condition1 and condition2 and condition3 then outcome.

Decision rules can be generated by constructing association rules with the target variable on the right. They can also denote temporal or causal relations. For example,



Decision table

It is a powerful tool to debug and prevent errors. It helps group similar information into a single table and then by combining tables it delivers easy and convenient decision-making. Decision table can be used to perform certain action associated with some condition just like flowcharts, if-then-else, and switch-case statements. Each decision corresponds to a variable, relation or predicate whose possible values are listed among the condition alternatives. A decision table may also include don't care symbol as condition alternative. Using don't cares can simplify decision tables, especially when a given condition has little influence on the actions to be performed.

To create the decision table, the developer must follow basic four steps:

- Identify all possible conditions to be addressed.
- Determine actions for all identified conditions.
- Create Maximum possible rules.
- Define action for each rule.

Components of a Decision Table

- **Condition Stub** – It is in the upper left quadrant which lists all the condition to be checked.
- **Action Stub** – It is in the lower left quadrant which outlines all the action to be carried out to meet such condition.
- **Condition Entry** – It is in upper right quadrant which provides answers to questions asked in condition stub quadrant.
- **Action Entry** – It is in lower right quadrant which indicates the appropriate action resulting from the answers to the conditions in the condition entry quadrant.

For example,

	Conditions/Actions	Rules							
Conditions	Shows Connected	N	N	N	N	Y	Y	Y	Y
	Ping is Working	N	N	Y	Y	N	N	Y	Y
	Opens Website	Y	N	Y	N	Y	N	Y	N
Actions	Check network cable	X							
	Check internet router	X				X	X	X	
	Restart Web Browser							X	
	Contact Service provider		X	X	X	X	X	X	

The decision table above represents some conditions and the appropriate action that must be taken in case you are facing problem with your internet connection.

Data dictionary

A data dictionary is used to list all data items that appear in a DFD of a system, i.e. all the data flows and data stores appearing on the DFD. It is also used to list the purpose of each data item along with the definition of all composite data items. For example, an entry

practical_assessment in the data dictionary consists of the components internal_assessment and external_assessment.

practical_assessment = internal_assessment + external_assessment

A data dictionary also lists the name and type of simple data item. Typical information included in data dictionary is as follows:

Name- It is used to identify the data item.
Alias- It is used to identify other names used to identify a data item.
Data Structure- It specifies the type of data item, i.e. integer, char, etc.
Description- Indicates how a data item is used and purpose of using the data item.
Duration- It specifies the life span of the data item.
Accuracy- It can be high, medium or low.
Range- Domain or allowable values of the data item.
Data flows- Identify the processes that generate and receive the data item.

Consider the following entry in data dictionary:

Name	Type	Description	Range	Accuracy	Data Flows
Average_marks	Real	Average marks of the student	0<= Average_marks <= 100	0.2	get_marks

Composite data items are defined using the following data definition operators:

'+' denotes composition of two data items. For example, X + Y represent composition data items X And Y.

'[,,]' represents selection. For example, [X,Y] represents occurrence of either X or Y.

'()' it refers to the optional data item that may or may not appear. For example, X + (Y) represents either occurrence of either X or X + Y.

'{}' it is used to represent iterative data definition, for example, {name}5 represents five name data and {name}*represents zero or more instances of name data.

'=' represents equivalence. For example, X= Y + Z means that X represents Y and Z.