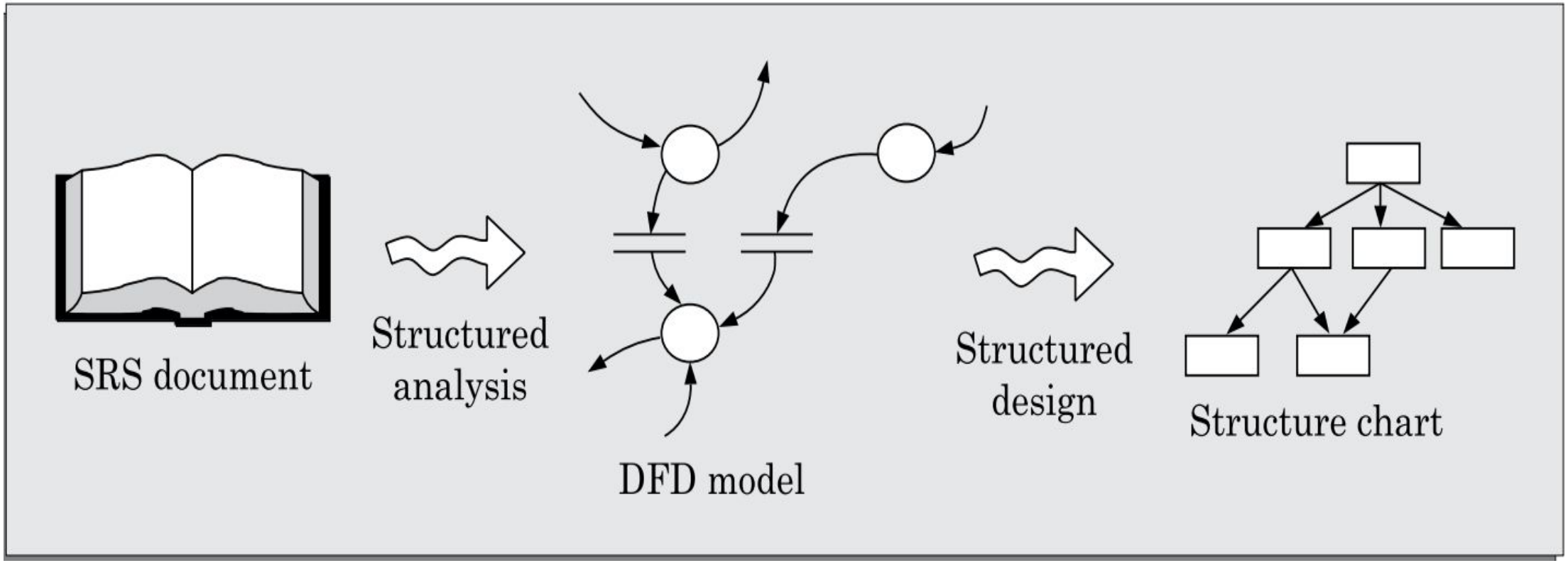
Function-Oriented Software Design

Function-Oriented Software Design

- Function-oriented design techniques were proposed nearly four decades ago.
- These techniques still very popular and are currently being used in many are at the present time software development projects.
- These techniques, to start with, view a system as a black-box that provides a set of services to the users of the software.
- These services provided by a software to its users are also known as the high-level functions supported by the software. During the design process, these high-level functions are successively decomposed into more detailed functions.
- After top-down decomposition has been carried out, the different identified functions are mapped to modules and a module structure is created.

SA/SD Design Methodology



- During structured analysis, the SRS document is transformed into a *data flow diagram* (DFD) model.
- During structured design, the DFD model is transformed into a structure chart.

Structured Analysis

- The structured analysis activity transforms the SRS document into a graphic model called the DFD model.
- During structured analysis, the major processing tasks (high-level functions) of the system are analyzed, and the data flow among these processing tasks are represented graphically.
- The structured analysis technique is based on the following underlying principles:
  - Top-down decomposition approach.
  - Application of divide and conquer principle. Through this each high-level function is independently decomposed into detailed functions.
  - Graphical representation of the analysis results using *data flow diagrams* (DFDs).

DFD

- The DFD (also known as the *bubble chart*) is a simple graphical formalism that can be used to represent a system in terms of the input data to the system, various processing carried out on those data, and the output data generated by the system.
- DFD representation of a problem is very easy to construct.
- It is a very powerful tool to tackle the complexity of industry standard problems.
- DFD model only represents the data flow aspects and does not show the sequence of execution of the different functions and the conditions based on which a function may or may not be executed.
- It completely ignores aspects such as control flow, the specific algorithms used by the functions, etc. In the DFD terminology, each function is called a *process* or a *bubble*.
- 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.

DFD

Symbols used for constructing DFDs

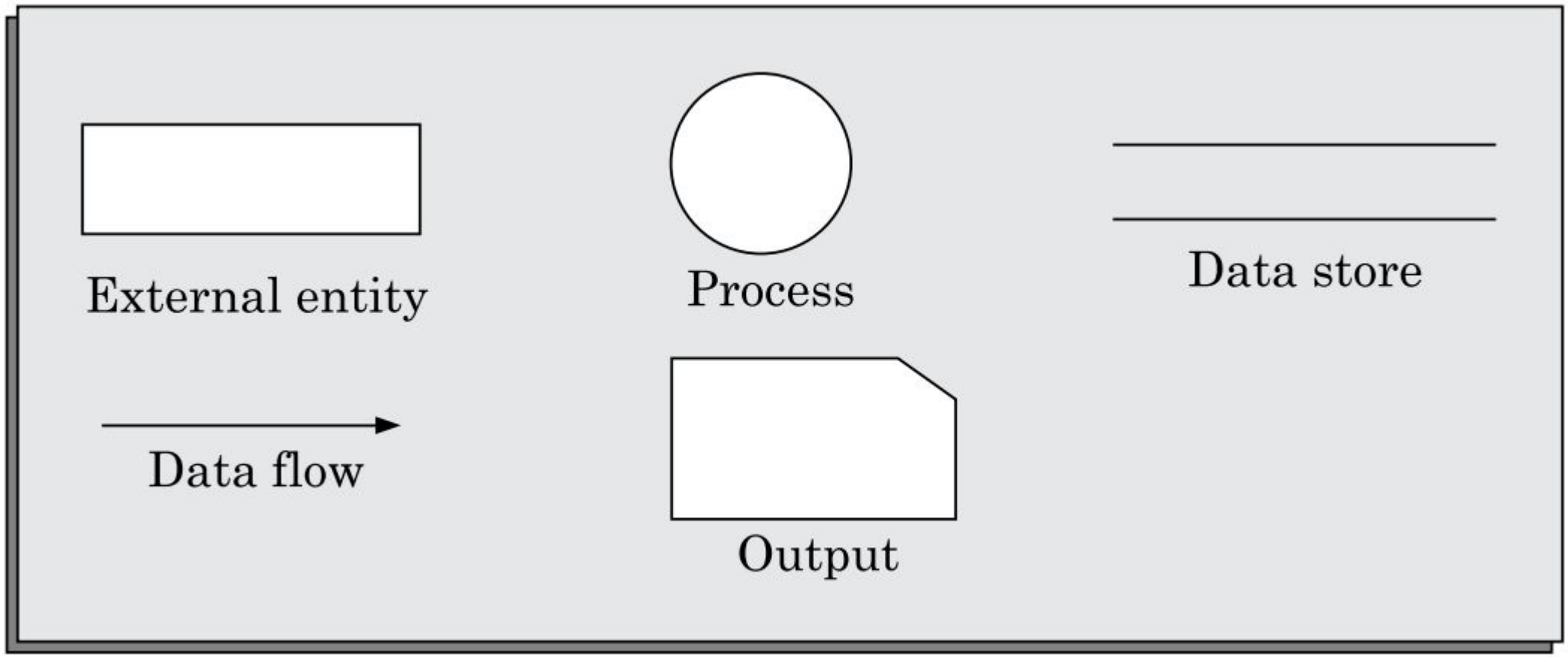
Function symbol: A function is represented using a circle. This symbol is called a *process* or a *bubble*. Bubbles are annotated with the names of the corresponding functions.

External entity symbol: An external entity is represented by a rectangle. The external entities are essentially those physical entities external to the software system which interact with the system by inputting data to the system or by consuming the data produced by the system. In addition to the human users, the external entity symbols can be used to represent external hardware and software such as another application software that would interact with the software being modelled.

Data flow symbol: A directed arc (or an arrow) is used as a data flow symbol. A data flow symbol represents the data flow occurring between two processes or between an external entity and a process in the direction of the data flow arrow. Data flow symbols are usually annotated with the corresponding data names.

DFD

Symbols used for constructing DFDs



DFD

Data store symbol: A data store is represented using two parallel lines. A data store symbol can represent either a data structure or a physical file on disk. Each data store is connected to a process by means of a data flow symbol. The direction of the data flow arrow shows whether data is being read from or written into a data store. An arrow flowing in or out of a data store implicitly represents the entire data of the data store and hence arrows connecting to a data store need not be annotated with the name of the corresponding data items.

Output symbol: The output symbol is as shown in figure. The output symbol is used when a hard copy is produced.

Synchronous VS. Asynchronous Operations

Synchronous: If two bubbles are directly connected by a data flow arrow, then they are synchronous. This means that they operate at the same speed. An example of such an arrangement is shown in Figure-(a). Here, the validate-number bubble can start processing only after the read-number bubble has supplied data to it; and the read-number bubble has to wait until the validate-number bubble has consumed its data.

Asynchronous: if two bubbles are connected through a data store, as in Figure-(b) then the speed of operation of the bubbles are independent. This statement can be explained using the following reasoning. The data produced by a producer bubble gets stored in the data store. It is therefore possible that the producer bubble stores several pieces of data items, even before the consumer bubble consumes any of them.

Synchronous VS. Asynchronous Operations

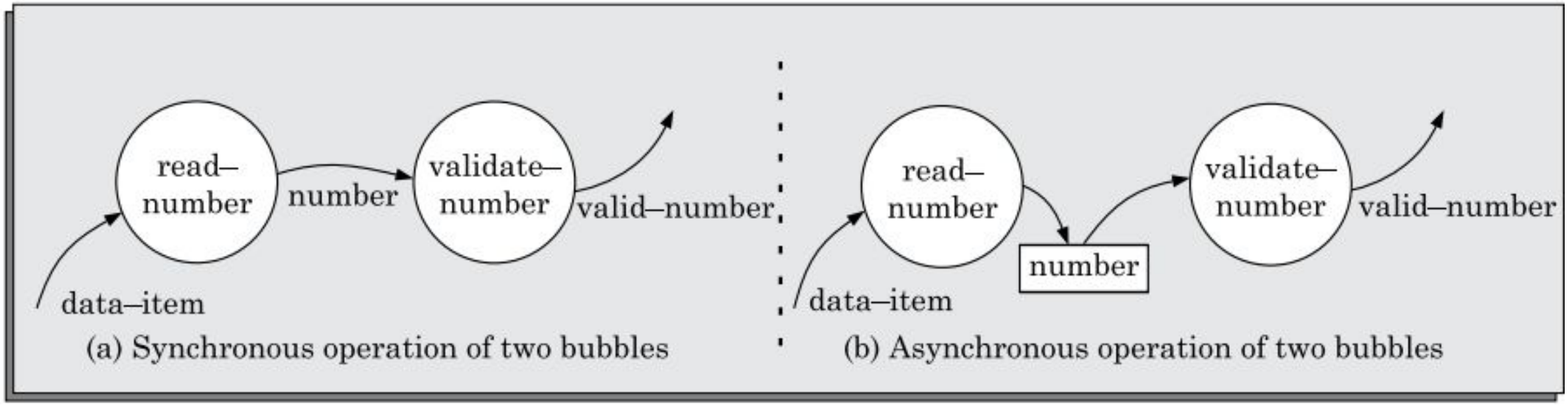


Figure-(a).

Figure-(b).

DFD

Importance of DFD

DFD is a very simple formalism – it is simple to understand and use.

Starting with a set of high-level functions that a system performs, a DFD model hierarchically represents various simple to understand sub-functions.

Human mind is such that it can easily understand any hierarchical model of a system – because in a hierarchical model, starting with a very simple and abstract model of a system, different details of the system are slowly introduced through different hierarchies.

DFD is an elegant modeling technique that turns out to be useful not only to represent the results of structured analysis of a software problem, but also for several other applications such as showing the flow of documents or items in an organization.

Data dictionary

Every DFD model of a system must be accompanied by a data dictionary. A data dictionary lists all data items that appear in a DFD model. The data items listed include all data flows and the contents of all data stores appearing on all the DFDs in a DFD model. However, a single data dictionary should capture all the data appearing in all the DFDs constituting the DFD model of a system.

Importance of Data Dictionary

- A data dictionary provides a standard terminology for all relevant data for use by the developers working in a project. A consistent vocabulary for data items is very important, since in large projects different developers of the project have a tendency to use different terms to refer to the same data.
- The data dictionary helps the developers to determine the definition of different data structures in terms of their component elements while implementing the design.
- The data dictionary helps to perform impact analysis. That is, it is possible to determine the effect of some data on various processing activities and *vice versa*.

Data Definition

Composite data items can be defined in terms of primitive data items using the following data definition operators.

+: denotes composition of two data items, e.g. **a+b** represents data **a** and **b**.

[,,]: represents selection, i.e. any one of the data items listed in the brackets can occur. For example, **[a,b]** represents either **a** occurs or **b** occurs.

(): the contents inside the bracket represent optional data which may or may not appear. e.g. **a+(b)** represents either **a** occurs or **a+b** occurs.

{}: represents iterative data definition, e.g. **{name}5** represents five **name** data. **{name}*** represents zero or more instances of **name** data.

=: represents equivalence, e.g. **a=b+c** means that **a** is composite data item comprising both **b** and **c.**

/* */: Anything appearing within /* and */ is considered as a comment.