CHAPTER 4

REQUIREMENTS ANALYSIS AND SPECIFICATION

Before starting to design a software product, it is extremely important to understand and document the exact requirements of the customer. In the past, many projects have suffered because the developers started implementing something without determining whether they were building what the customers exactly wanted. Starting development activities without properly analyzing and documenting the requirements is possibly the biggest mistake that one can commit during product development. Improper requirements increase the number of iterative changes required during the life cycle phases, and thereby push up the development costs alarmingly. This also sets ground for customer dissatisfaction and bitter customer-developer disputes and protracted legal battles.

When software is developed in a contract mode outside an organization (outsourced projects), the crucial role played by documentation of the precise requirements cannot be overstated. Even when an organization develops software for its internal use, proper formulation of requirements and their documentation is vital, since it helps to form a clear understanding of various features required to be supported. Further, the requirements specification document constitutes an important artifact that serves as the basis for various activities carried out during all later phases. For example, the activities in the design and testing phases are carried out largely based on the SRS document (we shall discuss this aspect later in this book). Therefore, requirements analysis and specification is considered to be a very important phase of software development and has to be undertaken with utmost care. Even experienced developers take considerable time to understand the exact requirements of the customer and to document them. They know that without a clear understanding of the problem and proper documentation of the same, it is impossible to develop a satisfactory solution.

The requirements analysis and specification phase starts after the feasibility study phase is complete and the project has been found to be financially sound and technically feasible. The goal of the requirements analysis and specification phase can be stated as follows.

> The goal of the requirements analysis and specification phase is to clearly understand the customer requirements and to systematically organize the requirements into a specification document.

The requirements analysis and specification phase ends with the production and validation of the requirements specification document that is usually called the Software Requirements Specification (SRS) document.

However, the question remains that who carries out requirements analysis and specification? A few experienced members of the development team usually visit the customer site to carry out the requirements analysis and specification activity. The engineers who gather and analyze customer requirements and then write the requirements specification document are known as *system analysts* in the software industry parlance. System analysts collect data pertaining to the product to be developed and analyze the collected data to conceptualize what exactly needs to be done. After understanding the precise user requirements, the analysts analyze the requirements to weed out inconsistencies, anomalies and incompleteness. They then proceed to write the Software Requirements Specification (SRS) document.

The SRS document is the final outcome of the requirements analysis and specification phase.

Once the SRS document is ready, it is first reviewed internally by the project team to ensure that it precisely captures all the user requirements, and that it is understandable, consistent, unambiguous, and complete. The SRS document is then given to the customer for review. After the customer has reviewed the SRS document and agrees to it, it forms the basis for all future development activities and also serves as a contract document between the customer and the development organization.

The main activities carried out during requirements analysis and specification phase are of two types as follows.

- Requirements gathering and analysis
- Requirements specification

These two activities are discussed in the following sections.

4.1 REQUIREMENTS GATHERING AND ANALYSIS

The requirements are almost never available in the form of a single document from the customer, neither is it completely obtainable from any single customer representative. In fact, it would be unrealistic to expect the customers to produce a comprehensive document containing a precise description of what he wants. Therefore, the requirements have to be gathered by the analyst from several sources in bits and pieces. These gathered requirements are then analyzed to remove several types of problems that frequently occur in the requirements gathered piecemeal from different sources.

We can broadly divide the requirements gathering and analysis activity into two separate tasks as requirements gathering and requirements analysis.

We discuss these two activities as follows.

4.1.1 Requirements Gathering

Requirements gathering is also popularly known as **requirements elicitation**. The analyst starts requirements gathering activity by collecting all information that could be useful to develop the system. Requirements gathering may sound like a simple task. However, in practice it is very difficult to gather all the necessary information from a large number of people and documents, and to form a clear understanding of a problem. This is especially so, if there are no working models of the problem. When the customer wants to automate

4.1.2 Requirements Analysis

After requirements gathering is complete, the analyst analyzes the gathered requirements to clearly understand the exact customer requirements and to weed out any problems in the gathered requirements.

> The main purpose of the requirements analysis activity is to analyze the collected information to obtain a clear understanding of the product to be developed, with a view to removing all ambiguities, incompleteness, and inconsistencies from the initial customer perception of the problem.

The following basic questions pertaining to the project should be clearly understood by the analyst in order to obtain a good grasp of the problem:

- What is the problem?
- Why is it important to solve the problem?
- What exactly are the data input to the system and what exactly are the data output by the system?
- What are the possible procedures that need to be followed to solve the problem?
- What are the likely complexities that might arise while solving the problem?
- If there are external software or hardware with which the developed software has to interface, then what exactly would the data interchange formats with the external system be?

After the analyst has understood the exact customer requirements, he proceeds to identify and resolve the various problems that he detects in the gathered requirements.

> There are three main types of problems in the requirements that the analyst needs to identify and resolve:
>
> - Ambiguity
> - Inconsistency
> - Incompleteness

Let us examine these different types of requirements problems in some detail.

Anomaly

An anomaly is an ambiguity in the requirement. When a requirement is anomalous, several interpretations of that requirement are possible. Any anomaly in the requirements can lead

to the development of incorrect systems, since an anomalous requirement can be interpreted in the several ways. The following are two examples of anomalous requirements:

Example 4.1: In a process control

4.2 SOFTWARE REQUIREMENTS SPECIFICATION (SRS)

After the analyst has gathered all the required information regarding the software to be developed, and has removed all incompleteness, inconsistencies, and anomalies from the specification, he starts to systematically organize the requirements in the form of an SRS document. The SRS document usually contains all the user requirements in structured though informal form.

Among all the documents produced during a software development life cycle, writing the SRS document is probably the toughest. One reason behind this difficulty is that the SRS document is expected to cater to the needs of a wide variety of audience. Different people need the SRS document for very different purposes. Some of the important categories of users of the SRS document and their needs are as follows:

Users, customers, and marketing personnel

The goal of this set of audience in referring to the SRS document is to ensure that the system as described will meet their needs. Remember that the customer may not be the user of the software, but someone employed or designated by the user. For generic products, the marketing personnel need to understand the requirements that they can explain to the customers.

[2] A safety-critical system is one whose improper working can result in financial loss, loss of property, or life.

Software developers

The software developers refer to the SRS document to make sure that they are developing exactly what is required by the customer.

Test engineers

Their goal is to ensure that the requirements are understandable from a functionality point of view, so that they can test the software and validate its working. They need that the functionality be clearly described, and the input and output data identified precisely.

User documentation writers

Their goal in reading the SRS document is to ensure that they understand the features of the product well enough to be able to write the users' manuals.

Project managers

They want to ensure that they can estimate the cost of the project easily by referring to the SRS document and that it contains all the information required to plan the project.

Maintenance engineers

The SRS document helps the maintenance engineers to understand the functionalities of the system. A clear knowledge of the functionalities can help them to understand the design and code. Also, a proper understanding of the customer's requirements would enable them to determine what modifications to the system's functionalities would be needed for a specific purpose.

Many software engineers in a project consider SRS document as a reference document. However, it is often more appropriate to think of the SRS document as the documentation of a contract between the development team and the customer. In fact, the SRS document can be used to resolve any disagreements between the developers and the customers that may arise in the future. The SRS document can even be used as a legal document to settle disputes between the customers and the developers. Once the customer agrees to the SRS document, the development team proceeds to develop the product ensuring that it conforms to all the requirements mentioned in the SRS document.

Before we discuss how to write a SRS document, we first discuss the characteristics of a good SRS document and the pitfalls that one must consciously avoid while writing an SRS document.

6.3 DATA FLOW DIAGRAMS (DFDs)

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. The main reason why the DFD technique is so popular is probably because of the fact that DFD is a very simple formalism—it is simple to understand and use. A DFD model uses a very limited number of primitive

symbols (shown in Figure 6.2) to represent the functions performed by a system and the data flow among these functions.

Starting with a set of high-level functions that a system performs, a DFD model represents the subfunctions performed by the functions using a hierarchy of diagrams. We had pointed out while discussing the principle of abstraction in section 1.3.2 that any hierarchical representation is an effective means to tackle complexity. Human mind is such that it can easily understand any hierarchical model of a system—because in a hierarchical model, starting with a very abstract model of a system, various details of the system are slowly introduced through different levels of the hierarchy. The DFD technique is also based on a very simple set of intuitive concepts and rules. We now elaborate the different concepts associated with building a DFD model of a system.

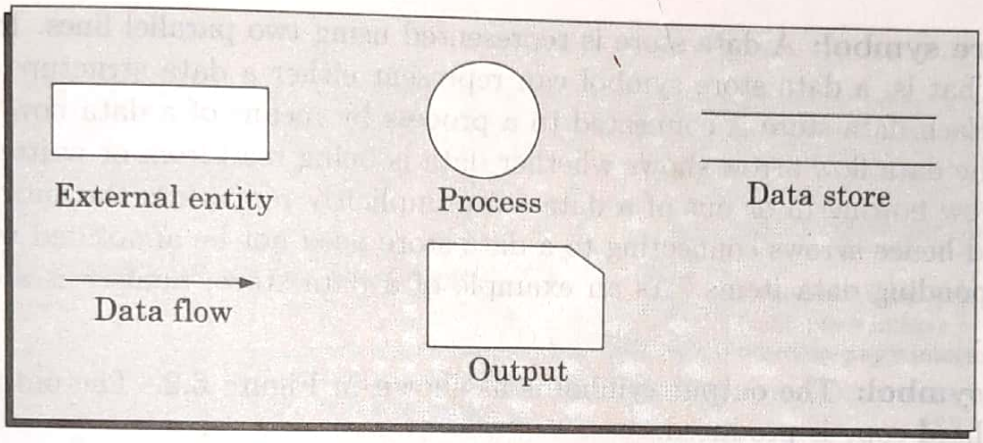

Figure 6.2: Symbols used for designing DFDs.

6.3.1 Primitive Symbols Used for Constructing DFDs

There are essentially five different types of symbols used for constructing DFDs. These primitive symbols are depicted in Figure 6.2. The meaning of these symbols are explained as follows:

1. **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 (see Figure 6.3).

2. **External entity symbol:** An external entity such as a librarian, a library member, etc. 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.

3. **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. For example, the DFD in Figure 6.3(a) shows three data flows: the data item `number` flowing from the process `read-number` to `validate-number`, `data-item` flowing into `read-number`, and `valid-number` flowing out of `validate-number`.

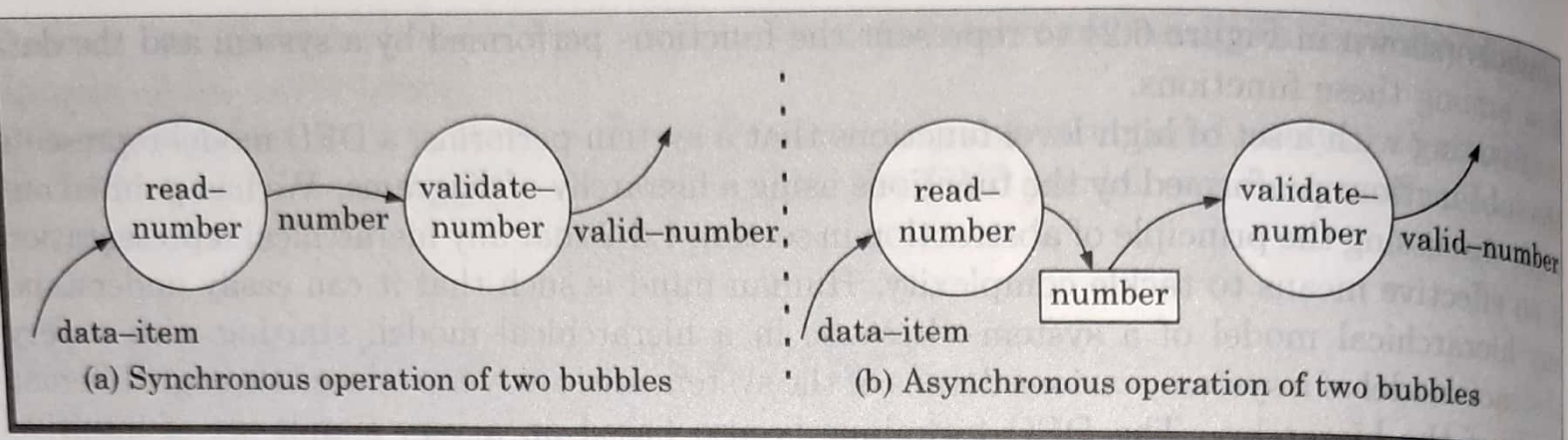

Figure 6.3: Synchronous and asynchronous data flow.

4. Data store symbol: A data store is represented using two parallel lines. It represents a logical file. That is, 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. As an example of a data store, `number` is a data store in Figure 6.3(b).

5. Output symbol: The output symbol is as shown in Figure 6.2. The output symbol is used when a hard copy is produced.

The notations that we are following in this text are closer to the Yourdon's notations than to the other notations. You may sometimes find notations in other books that are slightly different than those discussed here. For example, the data store may look like a box with one end open. That is because, they may be following notations such as those of Gane and Sarson [1979].

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. Please remember that the DFD model of a system typically consists of several DFDs, viz. level 0 DFD, level 1 DFD, level 2 DFDs, etc., as shown in Figure 6.4. However, a single data dictionary should capture all the data appearing in all the DFDs constituting the DFD model of a system.

A data dictionary lists the purpose of all data items and the definition of all composite data items in terms of their component data items.