Software Engineering

Chapter six

4. Analysis Modeling

At a technical level, software engineering begins with a series of modeling tasks that lead to a complete specification of requirements and a comprehensive design representation for the software to be built. The analysis model, actually a set of models, is the first technical representation of a system. Over the years many methods have been proposed for analysis modeling. However, two now dominate. The first; structured analysis is a classical modeling method and object oriented analysis. Analysis is a model building activity. Applying the operational analysis principles (The information domain of a problem must be represented and understood, the functions that the software is to perform must be defined, the behavior of the software (as a consequence of external events) must be represented, the models that depict information, function, and behavior must be partitioned in a manner that uncovers detail in a layered (or hierarchical) fashion, the analysis process should move from essential information toward implementation detail.) We create and partition data, functional, and behavioral models that depict the essence of what must built.

4.1 The Elements of the Analysis Model

The analysis model must achieve three primary objectives: (1) to describe what the customer requires, (2) to establish a basis for the creation of a software design, and (3) to define a set of requirements that can be validated once the software is built. To accomplish these objectives, the analysis model derived during structured analysis takes the form illustrated in Figure 4.1. At the core of the model lies the data dictionary—a repository that contains descriptions of all data objects consumed or produced by the software. Three different diagrams surround the core. The entity relation diagram (ERD) depicts relationships between data objects. The ERD is the notation that is used to conduct the data modeling activity. The attributes of each data object noted in the ERD can be described using a data object description. The data flow diagram (DFD) serves two purposes: (1) to provide an indication of how data are transformed as they move through the system and (2) to depict the functions (and sub functions) that transform the data flow. The DFD provides additional information that is used during the analysis of the information domain and serves as a basis for the modeling of function. A description of each function presented in the DFD is

contained in a process specification (PSPEC). The state transition diagram (STD) indicates how the system behaves as a consequence of external events. To accomplish this, the STD represents the various modes of behavior (called states) of the system and the manner in which transitions are made from state lo state. The STD serves as the basis for behavioral modeling. Additional information about the control aspects of the software is contained in the control specification (CSPEC). The analysis model encompasses each of the diagrams, specifications, descriptions, and the dictionary noted in Figure 4.1.

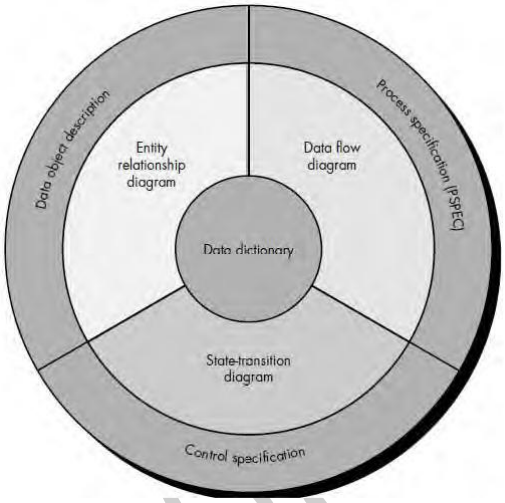

Figure 4.1 the structure of the analysis model

The ERD enables software engineers to identify data objects and their relationships using a graphical notation.

4.2 Data modeling

Data modeling answers a set of specific questions that are relevant to any data processing application. What are the primary data objects to be processed by the system? What is the composition of each data object and what attributes describe the object? Where do the objects currently reside? What are the relationships between each object and other objects? What are the relationships between the objects and the processes that transform them? To answer these questions, data modeling methods make use of the entity relationship diagram. The ERD described in detail later in this section, enables software engineers to identify data objects and their relationships using a graphical notation. In the context of structured analysis, the ERD defines all data that are entered, stored, transformed, and produced within an application. The entity relationship diagram focuses only on data (and therefore satisfies the first operational analysis

principles), representing a "data network" that exists for a given system. The ERD is especially useful for applications in which data and the relationships that govern data are complex. Unlike the data flow diagram (used to represent how data are transformed), data modeling considers data independent of the processing that transforms the data.

4.2.1 Data Objects, Attributes, and Relationships

The data model consists of three interrelated pieces of information: the data object, the attributes that describe the data object, and the relationships that connect data objects to one another.

Data objects: A data object is a representation of almost any composite information that must be understood by software. By composite information, we mean something that has a number of different properties or attributes. Therefore, width (a single value) would not be a valid data object, but dimensions (incorporating height, width, and depth) could be defined as an object. For example, a person or a car (Figure 4.2) can be viewed as a data object in the sense that either can be defined in terms of a set of attributes. The data object description incorporates the data object and all of its attributes. Data objects (represented in bold) are related to one another. For example, person can own car, where the relationship own connotes a specific "connection" between person and car.

Attributes: Attributes define the properties of a data object and take on one of three different characteristics. They can be used to (1) name an instance of the data object, (2) describe the instance, or (3) make reference to another instance in another table. In addition, one or more of the attributes must be defined as an identifier—that is, the identifier attribute becomes a "key" when we want to find an instance of the data object. In some cases, values for the identifier(s) are unique, although this is not a requirement. Referring to the data object car, a reasonable identifier might be the ID number.

Relationships: Data objects are connected to one another in different ways. Consider two data objects, book and bookstore. These objects can be represented using the simple notation illustrated in Figure 4.3. A connection is established between book and bookstore because the two objects are related.

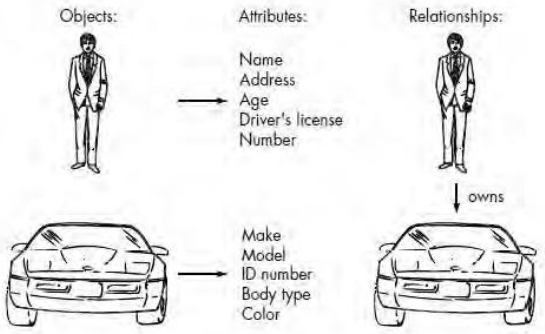

Figure 4.2 Data objects, attributes and relationships

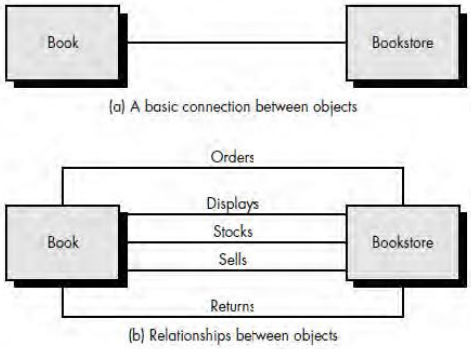

Figure 4.3 Relationships

4.2.2 Cardinality and Modality

The elements of data modeling—data objects, attributes, and relationships— provide the basis for understanding the information domain of a problem. However, additional information related to these basic elements must also be understood. We have defined a set of objects and represented the object/relationship pairs that bind them. But a simple pair that states: object X relates to object Y does not provide enough information for software engineering purposes.

We must understand how many occurrences of object X are related to how many occurrences of object Y. This leads to a data modeling concept called **cardinality**.

Cardinality: The data model must be capable of representing the number of occurrences objects in a given relationship. Cardinality is the specification of the number of occurrences of one [object] that can be related to the number of occurrences of another [object]. Cardinality is usually expressed as simply 'one' or 'many.' For example, a husband can have only one wife, while a parent can have many children. Taking into consideration all combinations of 'one' and 'many,' two [objects] can be related as

- One-to-one (1:1)—An occurrence of [object] 'A' can relate to one and only one occurrence of [object] 'B,' and an occurrence of ‘B' can relate to only one occurrence of 'A.'
- One-to-many (1: N)—One occurrence of [object] 'A' can relate to one or many occurrences of [object] 'B, but an occurrence of ' B’ can relate to only one occurrence of 'A.' For example, a mother can have many children, but a child can have only one mother.
- Many-to-many (M:N)—An occurrence of [object] 'A' can relate to one or more occurrences of 'B,' while an occurrence of 'B' can relate to one or more occurrences of 'A.' For example, an uncle can have many nephews, while a nephew can have many uncles.

Modality: The modality of a relationship is 0 if there is no explicit need for the relationship to occur or the relationship is optional. The modality is 1 if an occurrence of the relationship is mandatory. Figure 4.4 illustrates the relationship, cardinality, and modality between the data objects customer and repair action.

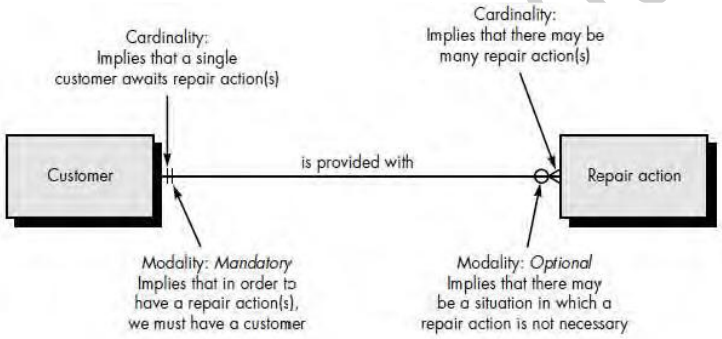

Figure 4.4 Cardinality and modality

4.2.3 Entity/Relationship Diagrams (ERD)

The object/relationship pair is the cornerstone of the data model. These pairs can be represented graphically by using the entity/relationship diagram, figure 4.5 shows an expand ERD.

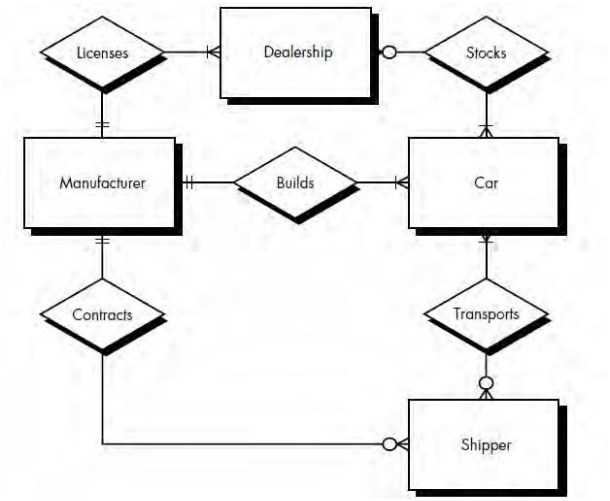

Figure 4.5 an expanded ERD

4.3 Functional Modeling And Information Flow

Information is transformed as it flows through a computer-based system. The system accepts input in a variety of forms; applies hardware, software, and human elements to transform it; and produces output in a variety of forms. Input may be a control signal transmitted by a transducer a series of numbers typed by a human operator, a packet of information transmitted on a network link, or a voluminous data-file retrieved from secondary storage. The transform(s) may comprise a signal logical comparison, a complex numerical algorithm, or a rule-inference approach of an expert system. Output may light a single LED or produce a 200 page report. In effect, we can create a flaw model for any computer based -system, regardless of size and complexity.

4.3.1 Data Flow Diagrams

As information moves through software, it is modified by a series of transformations; A data flow diagram is a graphical representation that depicts information flow and the transforms that are applied as data move from input to output. The basic form of a data flow diagram, also known as a **data flow graph** or a **bubble chart**, is illustrated in Figure 4.1.

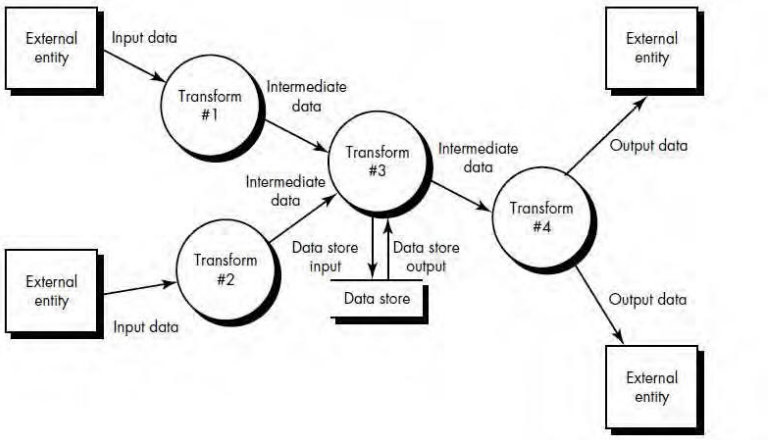

Figure 4.6 Information Flow Model

A rectangle is used to represent an external entity, that is a system element (e.g., hardware, a person, another program) or another system that produces information for transformation by the software or receives information produced by the software. A circle (sometimes called a bubble) represents a process or transform that is applied to data (or control) and changes it in some way. An arrow represents one or more data items (data objects). All arrows on a data flow diagram should be labeled. The double line represents a data store—stored information that is used by the software. The data flow diagram may be used to represent a system or software at any level of abstraction. In fact, DFDs may be partitioned into levels that represent increasing information flow and functional detail. Therefore, the DFD provides a mechanism for functional modeling as well as information flow modeling.

A level 0 DFD also called a fundamental system model or a context model represents the entire software element as a single bubble with input and output data indicated by incoming and outgoing arrows, respectively additional processes (bubbles) and information flow paths are represented as the level 0 DFD is partitioned to reveal more detail. For example, a level 1 DFD might contain five or six bubbles with interconnecting arrows each of the processes represented at level 1 is a sub function of the overall system depicted in the context model. As we noted earlier, each of the bubbles may be refined or layered to depict more detail, Figure 4.7 illustrates this concept. A fundamental model for system F indicates the primary input is A and ultimate output is B. We refine the F model into transform f1 to f7. Note that information flow continuity must be maintained; that is, input and output to each refinement must remain the same. This concept, sometimes called balancing, is essential for the development of consistent models. Further

refinement of f4 depicts detail in the form of transforms f41 to f45. Again, the input (X, y) and output (Z) remain unchanged.

DFD graphical notation must be augmented with descriptive text. A process specification (PSPEC) can be used to specify the processing details implied by a bubble within a DFD. The process specification describes the input to a function, the algorithm that is applied to transform the input, and the output that is produced in addition, the PSPEC indicates restrictions and limitations imposed on the process (function). Performance characteristics that are relevant to the process, and design constraints that may influence the way in which the Process will be implemented.

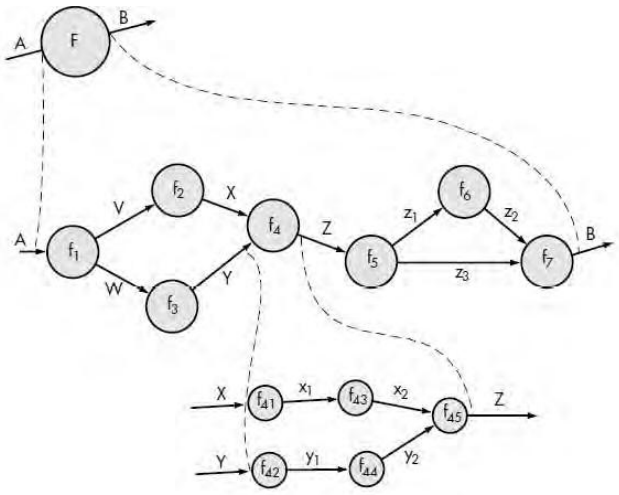

Figure 4.7 Information flow refinement

4.3.2 Control Flow Diagram

The control flow diagram (CFD) is superimposed on the DFD and shows events that control the processes noted in the DFD. CFD is used to determine the flow of control objects and processes that act on these objects shows how events move through a system. The CFD contains the same processes as the DFD, but shows control flow, rather than data flow. Control flow, events and control items is represented using a dashed or shaded arrow. A process that handles only control flows, called a **control process**, is represented using a dashed bubble and the dashed double arrow is used to represent the control store and a notational reference (a solid bar) to a control specification (**CSPEC**) is used. The CSPEC is used to indicate (1) how the software behaves when an event or control signal is sensed and (2) which processes are invoked as a consequence of the occurrence of the event. A process specification is used to describe the inner workings of a process

represented in a flow diagram. Control flow can be input directly to a conventional process or into a control process; figure 4.8 illustrates the data and control flow.

4.3.3 Relationship between data and Control Flow

Data flow diagrams are used to represent data and the processes that manipulate it. Control flow diagrams show how events flow among processes and illustrate those external events that cause various processes to be activated. The interrelationship between the process and control models is shown schematically in Figure 4.8. The process model is "connected" to the control model through **data conditions**. The control model is "connected" to the process model through **process activation** information contained in the CSPEC. A data condition occurs whenever data input to a process result in control output.

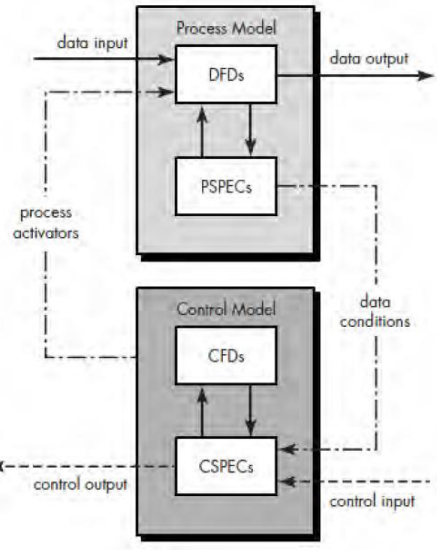

Figure 4.8 the Relationship between Data and Control models

4.4 Behavioral Modeling

Behavioral modeling is used to describe the behavior of the system according to different control objects Behavioral modeling is an operational principle for all requirements analysis methods.

4.4.1 State-Transition Diagram (STD)

The state transition diagram represents the behavior of a system by depicting its states and the events that cause the system to change state. In addition, the STD indicates what actions (e.g,

process activation) are taken as a consequence of a particular event. A state is any observable mode of behavior. A state transition diagram indicates how the system moves from state to state. The basic form of the STD is:-

1. Labeled rectangle represents system state.
2. Arrow represents transition between states.
3. Each arrow is labeled with a ruled expression; the top value indicates the event (control object) that causes the transition to occur and the bottom value indicates the action that occurs as consequences of the event.

4.5 Object-Oriented Analysis (OOA)

OO model of computer software must exhibit data and procedural abstractions that lead to effective modularity. A **class** is an OO concept that encapsulates the data and procedural abstractions required to describe the content and behavior of some real world entity. The data abstractions (attributes) that describe the class are enclosed by a "wall" of procedural abstractions (called **operations**, **methods**, or **services**) that are capable of manipulating the data in some way. The only way to reach the attributes (and operate on them) is to go through one of the methods that form the wall. Therefore, the class encapsulates data (inside the wall) and the processing that manipulates the data (the methods that make up the wall). An object encapsulates data (represented as a collection of attributes) and the algorithms that process the data. These algorithms are called operations, methods, or services and can be viewed as modules in a conventional sense. **Messages** are the means by which objects interact. A message stimulates some behavior to occur in the receiving object. The behavior is accomplished when an operation is executed. Figure 4.9 shows the object oriented classes and message passing between objects.

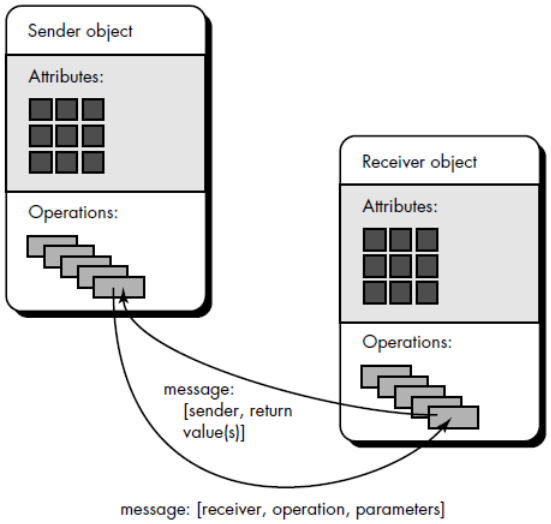

Figure 4.9 Object-oriented classes and message passing between objects.

Inheritance is one of the key differentiators between conventional and OO systems. A subclass Y inherits all of the **attributes** and **operations** associated with its superclass, X. This means that all data structures and algorithms originally designed and implemented for X are immediately available for Y—no further work need be done. Reuse has been accomplished directly. **Polymorphism** is a characteristic that greatly reduces the effort required to extend an existing OO system.

OOA is grounded in a set of basic principles that were introduced. In order to build an analysis model, five basic principles were applied: (1) the information domain is modeled; (2) function is described; (3) behavior is represented; (4) data, functional, and behavioral models are partitioned to expose greater detail; and (5) early models represent the essence of the problem while later models provide implementation details.

The intent of OOA is to define all classes that are relevant to the problem to be solved-the operations and attributes associated with them, the relationships between them, and behavior they exhibit. To accomplish this, a number of tasks must occur:

1. Basic user requirements must be collected through communication between the customer and the software engineer.
2. Classes must be identified (i.e., attributes and methods are defined).
3. A class hierarchy must be specified.
4. Object-to-object relationships (object connections) should be represented.
5. Object behavior must be modeled.
6. Tasks 1 through 5 are reapplied iteratively until the model is complete.

4.5.1 DOMAIN ANALYSIS

The objective of domain analysis is to define a set of classes (objects) that are encountered throughout an application domain. These can be reused in many applications. This activity, called domain analysis, is performed when an organization wants to create a library of reusable classes (components) that will be broadly applicable to an entire category of applications. During domain analysis, object (and class) extraction occurs.

i. Reuse and Domain Analysis

Object-technologies are leveraged through reuse. Consider a simple example. The analysis of requirements for a new application indicates that 100 classes are needed. Two teams are assigned to build the application. Each will design and construct a final product. Each team is populated by people with the same skill levels and experience. Team A does not have access to a class library, and therefore, it must develop all 100 classes from scratch. Team B uses a robust class library and finds that 55 classes already exist. It is highly likely that

1. Team B will finish the project much sooner than Team A.
2. The cost of Team B's product will be significantly lower than the cost of Team A's product.
3. The product produced by Team B will have fewer delivered defects than Team A’s product.

ii. The Domain Analysis Process

The domain analysis process can be characterized by a series of activities that begin with the identification of the domain to be investigated and end with a specification of the objects and classes that characterize the domain.