Chapter 6

Function-Oriented Design

The design activity begins when the requirements document for the software to be developed is available and the architecture has been designed. During design we determine what modules should the system have and which have to be developed. The design exercise determines the module structure of the components. We have to ensure that the module view created in design is consistent with the architecture.

The design of a system is a plan for a solution for the system. A system is a set of modules with clearly defined behavior which interact with each other in a defined manner to produce some behavior or services for its environment. ***A module of a system can be considered a system, with its own modules.***

The design process has two levels.

1. First level
    - Focus is on deciding which modules are needed for the system, the specifications of these modules, and how the modules should be interconnected.
    - Known as system design or top-level design.
2. Second level
    - The internal design of the modules, or how the specifications of the module can be satisfied, is decided.
    - Known as detailed design or logic design,
    - **Expands the system design to contain a more detailed description of the processing logic and data structures so that the design is sufficiently complete for coding.**

In a function-oriented design approach, a system is viewed as a transformation function, transforming the inputs to the desired outputs.

Each module in design supports a functional abstraction. The basic output of the system design phase is the definition of all the major data structures in the system, all the major modules of the system, and how the modules interact with each other.

6.1 Design Principles

The design of a system is correct if a system built precisely according to the requirements of that system. The goal during the design phase is to produce not only the correct designs but also best possible design within the limitations imposed by the requirements and the physical and social environment in which the system will operate.

Specify some properties and criteria to evaluate a design. We can precisely evaluate the "goodness" of a design and determine the best design. However, criteria for quality of software design is often subjective or non-quantifiable. In such a situation, **criteria are essentially thumb rules that aid design evaluation.**

A design should clearly be

a) verifiable,
b) complete (implements all the specifications), and
c) Traceable (all design elements can be traced to some requirements).

The two most important properties that concern designers are **efficiency and simplicity**.

- ✓ Efficiency of any system is concerned with the **proper use of scarce resources by the system** due to cost considerations. Scarce and expensive resources should be used efficiently. **In computer systems, the resources that are most often considered for efficiency are processor time and memory.** An efficient system is one that consumes less processor time and requires less memory.
- ✓ Simplicity is important quality criteria for software systems. Maintenance of software is expensive. The design of a system is one of the most important factors affecting the maintainability of a system. Maintainer must have a thorough understanding of the different modules of the system, how they are interconnected, and how modifying one will affect the others. A simple and understandable design will go a long way in making the job of the maintainer easier.

Simplicity is the primary property of interest, and therefore the objective of the design process is to produce designs that are simple to understand.

Creating a simple (and efficient) design of a large system can be an extremely complex task that requires good engineering judgment. Effectively handling the complexity will not only reduce the effort needed for design (i.e., reduce the design cost), but can also reduce the scope of introducing errors during design.

The principles that can be used in design are the same as those used in problem analysis. However, there are some fundamental differences between design and analysis.

i. In problem analysis, we are constructing a model of the problem domain, while in design we are constructing a model for the solution domain.
ii. In problem analysis, the analyst has limited degrees of freedom in selecting the models as the problem is given, and modeling has to represent it. In design, the designer has a great deal of freedom in deciding the models, as the system the designer is modeling does not exist.
iii. In design, the system depends on the model, while in problem analysis the model depends on the system.
iv. The basic aim of modeling in problem analysis is to understand, while the basic aim of modeling in design is to optimize (in our case, simplicity and performance).

6.1.1 Problem Partitioning and Hierarchy

When solving a small problem, the entire problem can be tackled at once. For solving larger problems, the basic principle is the time-tested principle of "divide and conquer." For software design, therefore, the goal is to divide the problem into manageably small pieces that can be solved separately. **The basic rationale behind this strategy is the belief that if the pieces of a problem are solvable separately, the cost of solving the entire problem is more than the sum of the cost of solving all the pieces.**

The different pieces have to cooperate and communicate to solve the larger problem. This communication adds complexity. As the number of components increases, the cost of partitioning, together with the cost of this added complexity, may become more than the savings achieved by partitioning. The designer has to make the judgment about when to stop partitioning.

Dependence between modules in a software system is one of the reasons for high maintenance costs. Clearly, proper partitioning will make the system easier to maintain by making the design easier to understand. **Problem partitioning also aids design verification.** Design produced by using problem partitioning can be represented as a hierarchy of components.

6.1.2 Abstraction

It is a tool that permits a designer to consider a component at an abstract level without worrying about the details of the implementation of the component. An abstraction of a component describes the external behavior of that component without bothering with the internal details that produce the behavior.

To decide how a component interacts with other components, the designer has to know, at the very least, the external behavior of other components. To allow the designer to concentrate on one component at a time, abstraction of other components is used.

Abstraction of existing components plays an important role in the maintenance phase. To modify a system, the first step is understanding what the system does and how. During design, the components do not exist, and in the design the designer specifies only the abstract specifications of the different components. There are two common abstraction mechanisms for software systems: ***functional abstraction and data abstraction.***

In functional abstraction, a module is specified by the function it performs. For example, a module to compute the log of a value can be abstractly represented by the function log. Functional abstraction is the basis of partitioning in function-oriented approaches. The decomposition of the system is in terms of functional modules.

In data abstraction, any entity in the real world provides some services to the environment to which it belongs. Entities provide some fixed predefined services. Certain operations are required from a data object, depending on the object and the environment in which it is used. Data is not treated simply as objects, but is treated as objects with some predefined operations on them. From outside an object, the internals of the object are hidden; only the operations on the object are visible. **Data abstraction forms the basis for object-oriented design**.

In abstraction, a system is viewed as a set of objects providing some services. Hence, the decomposition of the system is done with respect to the objects the system contains.