Chapter 2
Software Design

Rachita Misra
C.V. Raman College of Engineering, India

Bijayalaxmi Panda
C.V. Raman College of Engineering, India

Chhabi Rani Panigrahi
C.V. Raman College of Engineering, India

Bibudhendu Pati
C.V. Raman College of Engineering, India

ABSTRACT

This chapter on "Software Design" emphasizes the role of modeling, prototyping, and simulation in software design. The chapter introduces the principles of software design, issues and challenges. Modeling techniques used in procedural and object oriented methodologies is presented along with the Unified Modeling Language (UML). The suitability of prototyping, as a design artifact and a simulation method is briefly discussed. Software processes such as Rapid Application Development (RAD), Rational Unified Process (RUP) and Agile methodologies which influence the design process have been discussed and recommended. The chapter then deals with Design Metrics for Quality Analysis, Software Risk Analysis and Threat Modeling for design of secure software. Finally, some of the recent research topics such as Model Driven Architecture (MDA), Model Driven Development (MDD), Meta Object Facility (MOF), and Model Driven Testing (MDT) have been covered.

INTRODUCTION

Software design plays an important role in Software Development Life Cycle (SDLC). According to the IEEE definition, design is both "*the process of defining the architecture, components, interfaces, and other characteristics of a system* and *the result of that process"* (IEEE, 1990). More specifically, a software design describes the architecture of the system that is how the system is decomposed and organized into components and also describes the interfaces between these components.

During software design phase various models corresponding to the system are developed and these form a kind of blueprint of the solution to be implemented. These models are useful in a number of ways. First, these are used as input to the *coding phase* in the SDLC and used for planning subsequent activities.

The models are also *analyzed & reviewed* to check the traceability (that is whether if each identified functional requirements of the system can be traced to some model element and vice versa), maintain-

DOI: 10.4018/978-1-5225-3422-8.ch002

ability, correctness, and implementation. Lastly, these models can also be used to evaluate various alternative solutions and trade-offs.

In this chapter, first the key principles of design are identified. Next, the various design techniques available in the literature starting from *process oriented modeling to object oriented modeling* are described. Subsequently, several factors such as *design patterns, design metrics and software process* which help creating good design are discussed. Various techniques used for *quality analysis, risk analysis, and threat modeling* have been given to ensure the quality of the design. Finally, we highlight current trend and future directions in software engineering including *model driven development, meta-object facility, model based testing,* and *process automation.*

BACKGROUND

In this section, we discuss the basic principles which guide the software design for traditional procedure oriented and object oriented systems. It may be noted that the concepts behind these two principles are similar but they are tuned to the two development methodologies.

Design Principles for Traditional Software Design

The principles of design for traditional software are stated as follows (James, 1989; Roger, 2014): Modularity and partitioning, Coupling, Cohesion, Span of control, Size, and Shared modules.

- **Modularity and Partitioning:** Partitioning is a top down approach and starts during system study moving down gradually to understand the system in detail. A process is exploded to detail level processes showing data flows in the Data Flow Diagrams (DFDs). Each function of the system is identified from the system design specifications and developed to greater detail with step-wise refinement creating a hierarchy of separate modules.
- **Coupling:** Coupling refers to the strength of the relationship between modules in a system. While partitioning, analysts should ensure that interdependence between modules is minimum. A good design seeks a loose coupling among modules. If modules have high coupling it is difficult to understand, use and test in isolation, change in one module forces changes to related modules. Coupling can be categorized as:
    - **Content Coupling:** When one module relies on the internal workings of another module, they are said to possess content coupling. This creates high coupling among modules and should be avoided.
    - **Common Coupling:** When two modules share global data then this dependence is called common coupling.
    - **External Coupling:** When two modules share an external data, it is said to have external coupling.
    - **Control Coupling:** Control coupling occurs when one module controls the logic of other modules by passing information on what to do.
    - **Data Coupling:** Two modules are data coupled if they communicate through some elementary data item which is passed as a parameter. This category displays the lowest form of coupling and is preferable.