5.3 SOFTWARE DESIGN AND SOFTWARE ENGINEERING

Software design sits at the technical kernel of the software engineering process and is applied regardless of the software process model that is used. Beginning once, software requirements have been analyzed and specified. Software design is the first of three technical activities design, code generation and testing- which are required to build and verify the software. Each activity transforms information in a manner that ultimately results in validated computer software.

Software design is an iterative process through which requirements are translated to a "blue print" for constructing the software. Throughout the design process, the quality of the evolving design is assumed with a series of formal technical reviews. In order to evaluate the quality of a design representation, we must establish technical criteria for good design. The design should provide a complete picture of the software. The design must implement all the explicit requirements contained in the analysis model.

Table 5.1: The Design Phase: What are the plans?

Phase	Deliverable
Design	▪ Architecture Document ▪ Implementation Plan ▪ Critical Priority Analysis ▪ Performance Analysis ▪ Test Plan

In the design phase the architecture is established. This phase starts with the requirement document delivered by the requirement phase and maps the requirements into architecture. The architecture defines the components, their interfaces and behaviours. The deliverable design document is the architecture. The design document describes a plan to implement the requirements. This phase represents the "how" phase. Details on computer programming languages and environments, machines, packages, application architecture, distributed architecture layering, memory size, platform, algorithms, data structures, global type definitions, interfaces, and many other engineering details are established. The design may include the usage of existing components. The design phase is summarized in Table 5.1.

The architectural team can now expand upon the information established in the requirement document. Using the typical and a typical scenarios provided from the requirement document, performance trade-offs can be accomplished as well as complexity of implementation trade-offs.

Obviously, if an action is done many times, it needs to be done correctly and efficiently. A seldom used action needs to be implemented correctly, but it is not obvious what level of performance is required. The requirement document must guide this decision process. An example of a seldom used action which must be done with high performance is the emergency shutdown of a nuclear reactor.

Analyzing the trade-offs of necessary complexity allows for many things to remain simple which, in turn, will eventually lead to a higher quality product. The architecture team also converts the typical scenarios into a test plan.