Patterns of Learning

When people begin to learn to play chess they first learn all the rules and physical requirements of the game. They learn the names of all the pieces, the way that pieces move move and capture, and the geometry and orientation of the board.

At this point, people can play chess, although they will probably not a very good players. But as they progress, they will learn the *principles* of the game. They learn the value of protecting their pieces, and the relative value of those pieces. They learn the strategic value of the center squares and the power of a threat. They learn how the king can oppose the enemy king and how even one passed pawn can win the game.

At this point, people can play a good game of chess. They know how to reason through the game and can recognize ``stupid'' mistakes.

However, to become a master of chess, one must study the games of other masters. Buried in those games are patterns that must be understood, memorized, and applied repeatedly until they become second nature. There are thousands upon thousands of these patterns. Opening patterns are so numerous that there are books dedicated to their variations. Midgame patterns and ending patterns are also prevalent, and the master must be familiar with them all.

So it is with software. First one learns the rules. The algorithms, data structures and languages of software. At this point, one can write programs, albeit not very good ones. Later, one learns the principles of software design. Structured programming, modular programming, object oriented programming. One learns the the importance of cohesion and coupling, of information hiding and dependency management.

But to truly master software design, one must study the designs of other masters. Deep within those designs are patterns that can be used in other designs. Those patterns must be understood, memorized, and applied repeatedly until they become second nature.

Robert C. Martin