1

Introduction to Architecture and Design

Almost every software developer I know is fascinated by software architecture and design. High-level architecture and design patterns are concepts that beginner developers least understand. For most of us, programming is relatively easier to learn; usually good aptitude and decent logical skills are enough to be a good programmer. But architecture is altogether a different beast to handle. It is more of an art, and usually takes years of experience to master.

In this chapter, we will focus on:

- Understanding architecture and design from a practical viewpoint
- Architectural styles
- What Design patterns are
- Different stages of a project lifecycle
- Difference between tiers and layers

Software Architecture

There are many different definitions of software architecture scattered across the web, in reference materials, and in books. In the wide world of programming, many of the definitions you may find are most likely going to be extremely technical in the language they use, and can be difficult for a beginner to fully grasp and understand. There are even places on the web that list thousands and thousands of different definitions by leading software architects, engineers, doctors, philosophers, and professors. (Reference: `http://www.sei.cmu.edu/architecture/community_definitions.html`).

To begin with, let's start with a technical definition:

> *Software architecture is an abstraction, or a high-level view of the system. It focuses on aspects of the system that are most helpful in accomplishing major goals, such as reliability, scalability, and changeability. The architecture explains how you go about accomplishing those goals.*

Now we will translate this definition into something simple, generic, and easy to understand:

```
Software architecture is a blueprint of your application.
```

To elaborate more on the "blueprint" part, let us try to understand software architecture with a simple analogy—the process of casting.

Casting is a manufacturing process in which a liquid material is poured into a mold that contains a hollow cavity of a desired shape. The liquid is then allowed to cool and solidify, taking the shape of the mold it was poured into. The mold is the guide that shapes the liquid into the intended result. Keep in mind that the mold can be of any shape, size, or dimension, and is separate or unrelated to the liquid that is poured in.

Now, think of software architecture as the mold and think of your project as the liquid that is poured into this mold. Just like casting, software architecture is the guide that shapes your project into the intended result. The architecture of a software system has no strict relation to the actual code that is written for this system. The architecture simply makes sure that the development process stays within certain defined limits.

Software Design

Software design refers to the thought process involved in planning and providing for a better solution during problem solving. Software design comes after the architecture is decided upon. Architecture is more closely related to the business needs of the project, and theoretically it does not concern the actual technology platform (such as J2EE or Microsoft .NET or PHP) on which the application will be built (although practically we can decide the platform either in parallel with working on the architecture of the application or before doing so). Software design deals with the high-level concepts related to the actual implementation of the architecture in our projects, which include tasks such as usability studies to make sure our project targets the right kind of users, deciding which design patterns to use to make our application scalable, secure and robust. During the design phase, we also decide on the implementation methodology to be used in the actual development phase (which comes after design and involves actual coding). The following diagram shows how architecture and design fit together and relate to each other: