Architecture: First Steps

How do business requirements dictate architectural decisions? Lets understand this through a quick and small example. Assume that a software company, Takshila Inc., has recently bagged the contract for building a new inventory management system for a local cosmetics manufacturing firm. After the initial talks with the stakeholders, the business analyst from Takshila comes up with high-level specifications, which are:

- The system should be accessible from any online location
- The system should be able to process multiple orders at the same time
- The system should be able to interact and process information from different locations having different databases
- The system should interact with other software packages (such as financial software) already in use by the company
- The system should be easy to customize later by the internal development team

With these requirements in mind, and after detailed discussions with team members, the software architect has come up with the following architectural specifications for the proposed inventory management software:

- The system should be web based, using a thin-client architecture.
- The system should have built-in multithreading capabilities.
- The system should be database-independent, which means that the system should be able to work with multiple types of databases without changing the code—probable use of dependency injection.
- The system should expose a set of functions as an API, and should also be able to import data from other sources and process this data in its own tables.
- The system should have loosely-coupled tiers, so that each individual tier has no dependency on the other and can be used with any other tier.

Note how the business requirements have been translated into architectural specifications, and still there is not a word about a programming or development platform! So the architecture has nothing to do with development platforms, programming languages, design and so on. We can create a system satisfying the above requirements in many ways, using different designs and probably using different platforms too (for example, one could either use ASP.NET or JSP/J2EE). In short, the architecture does not care whether you use LINQ, AJAX, or Ruby on Rails. As long as you are meeting the architectural specifications, you are free to choose your own technology and tools.

Design Patterns

The word *pattern* means a guide or a model that is to be followed when making things. In software development, we often use programming techniques and solutions developed by others that prove to be credible over time. Solutions to software problems were not developed overnight, and most of these problems were common across the development world, so these time-tested solutions were grouped to be re-used by others.

So a design pattern is a re-usable solution that can be used in our projects so that time-tested programming techniques can be employed to solve similar kinds of problems.

The main difference between architecture and design patterns is that design patterns deal with implementation-level issues, and are more close to the programming and development platform, whereas architecture is at a more abstract level and is independent of the implementation. Design patterns tell us how we can achieve a solution in terms of implementation. But the patterns themselves are independent of the programming language and technology platform. We can implement a particular design pattern in any language we want: JAVA, C# or PHP. We cannot use design patterns as-it-is in our projects. They show us the right path to take in order to solve a problem, but they are not a complete solution. We cannot simply copy-paste a particular design pattern's code directly into our project. We will need to modify it to suit our own unique needs and implementation platform.

In the coming chapters, we will learn some of the famous design and commonly used patterns, with sample code in ASP.NET.

Project Life Cycle

From an idea to a fully functional binary or DLL, a project passes through a varied range of activities and processes. The project life cycle refers to the different logical and physical stages that a project goes through from inception to completion. The life cycle starts with gathering the business requirements and ends when the final product is delivered after complete testing. The following are the major stages of a generic project life cycle:

1. Project Initiation
2. Planning and Prototyping
3. Project Construction
4. Project Transition and Release