Importance of Design Patterns and Frameworks for Software Development

Ólafur Andri Ragnarsson, Adjunct
Reykjavik University
Kringlan 1, 103 Reykjavík
andri@ru.is

Abstract

Design patterns are among the most powerful methods for building large software system. Patterns provide well-know solutions to reoccurring problems that developers are facing. There are several benefits of using patterns if applied correctly. Although design patterns are only over decade old, the science of patterns is becoming established, allowing for consistent communication. By using well-known patterns reusable components can be built in frameworks. Providing frameworks for reusability and separation of concerns is key to software development today.

Introduction

Software system can be among the most complex constructions. As an example, it is believed that Microsoft's Vista operating system is over 50 million lines of code (Ganssle, 2005). Although most software systems are not of this size, complexity of software development can be quick to increase.

Fortunately, computer scientists have developed several methods to make construction of large system easier. Among these methods that are the most important is the use of design patterns and frameworks.

What are Design Patterns?

But what are design patterns? Image we are building system's user interface. How to do we connect the presentation with the business processing and with data? Our course this is not a new problem. Developers have solved problems like this for decades. When we think about it, development is partly about solving the same similar problems again and again. For this given task it turns out that there is a well know solution – a design pattern called *Model-View-Controller* (Fowler, Model View Controller, 2002). Originating from the world of Smalltalk in the 70s, the pattern is about separating the presentation from the data model. Since then this pattern has been used in some form again and again for these kinds of projects.

Design pattern are well know solutions for reoccurring problems. By defining and categorizing patterns, a catalog of patterns can be created for use when building large and complex software systems. This was exactly what the pioneers in the field did. Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, also known as the Gang of Four (GoF), published in 1995 the book *Design Patterns: Elements of*

Reusable Object-Oriented Software. This influential book defines several patters for object oriented programming languages.

Using Patterns

Let’s take an example of a pattern that is very useful. Say we have an e-commerce shopping software and one of the components is a sale component. This component handles sales transactions. This includes financial transaction (connection to bank or a financial institution), stock changes and so on. In the system it turns out that many other components need to know when a sale transaction takes place, for example, components that show sales statistics or stock information. One obvious way of course is to use the database and have all components get the information from the database. This causes unnecessary load on the database. What if the sale component informs the other components? This can be done by having the sale component call the other components upon each sale transaction. This might sound like a nice design and is easy to understand and implement. However, this is horrid design. Every time some new component needs sale information, we must update the sale component. The sale component will become dependent on many other totally unrelated components. Systems like that can cause horrible maintenance problems.

The solution to this problem is actually very simple and elegant. The pattern *Observer* is designed for cases like this (WikiPedia, 2007). The pattern describes a sort for publish subscribe system, where a consumer or observer, becomes a subscriber to specific information by listening to events. The sale component can define a listener interface implemented by components interested in getting information about a sale. Upon a sale transaction, the sale component will iterate through the listeners and call them. Adding new subscriber component will not change the sale component.

Benefits of Patterns

Benefits of knowing and using design patters are several. Patterns are known solution for building software systems. They can reduce development time as known solutions are used instead of reinventing the wheel. It is also a benefit to use known solutions that are tried and tested. Finally, patterns make the communication of development teams easier.

Patterns for Communication

Communication is of course important in software development, especially when discussing the design of how to build the software. In my work as software architect, I have attended many meeting where people come together to discuss design. On multiple occasions much time is used on explaining different ways and proposals. In some cases people who at first seem to vigorously disagree are in fact talking about the same design but are using different way to visualize and express their thoughts. By establishing a common and known vocabulary these discussions can be reduced and misunderstanding can be avoided.

It has taken years to establish common vocabulary on patterns. One of the drawbacks to design patterns is that the same names are used for different patterns. An example is the pattern *Value Object* (Fowler, Value Object, 2002) that for object oriented programming languages describes a class with a particular state. Martin Fowler writes in his book, *Patterns of Enterprise Application Architecture* (Fowler, Patterns of Enterprise Application Architecture , 2002), that this pattern is in many cases used to storing mutable data, for example when moving data from one layer to the next. Fowler prefers using the pattern *Data Transfer Object* (Fowler, Data Transfer Object, 2002) for that purpose and *Value Object* for static and immutable data, such as a date. The *Date* class in Java API is an example of this.

Even with ambiguities like these, a common understanding on patterns is emerging and also more understanding on the benefits and drawbacks. Patterns are usually dependent on particular programming languages and are different between languages as programming languages are different. What is simple in one language can be very difficult in another. Due to this it is important to understand the context of the patterns. For development in C++, Java and C# there are several patterns that are designed for object oriented languages.

From Patterns to Frameworks

Although the benefits of patterns are clear, building collections of reusable components for solving common problems can provide even more productivity. This can be done by building frameworks.

The basic idea of frameworks is to place all common and reusable functionality in specific classes that can be reused for specific applications. The key here is to know the concern of the component begin built. This is perhaps the most complex thing with development. Let's take a shopping solution as an example. In this solution a web page will display product information that is retrieved from the database. There are several different concerns within this simple task. When reading from the database the concern is data query. When calling the database and processing the results the concern is business logic, and when displaying the result, the concern is the presentation. Very likely all these tasks will be done by the same programmer. But to do this, the programmer needs to jump from one concern to the next – and this can be difficult. Most likely the developer will make each concern dependent on this particular application of retrieving product information. This will most likely reduce reusability.

By creating frameworks, each concern is solved with reusability in mind. Using the above example, a framework for retrieving data is created. In such framework data access is the only concern. For this the developer is only concerned with data and is not concerned with particular type of data or specific product information. All components of the application can use the data framework.

There are several patterns for data retrieval and among them *Table Data Gateway* (Fowler, Table Data Gateway, 2002) also know as *Data Access Objects* or *DAO*. For the presentation and the business logic a framework using the *Model-View-Controller* can be used.

With correct use of design patterns and building of reusable frameworks, great productivity in software development can be achieved.

Biography

Erich Gamma, R. H. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley Professional.

Fowler, M. (2002). *Data Transfer Object*. Retrieved Maí 21, 2007, from P of EAA Catalog: http://www.martinfowler.com/eaaCatalog/dataTransferObject.html

Fowler, M. (2002). *Model View Controller*. Retrieved Maí 21, 2007, from P of EAA Catalog: http://www.martinfowler.com/eaaCatalog/modelViewController.html

Fowler, M. (2002). *Patterns of Enterprise Application Architecture* . Addison-Wesley Professional.

Fowler, M. (2002). *Table Data Gateway*. Retrieved maí 21, 2007, from P of EAA Catalog: http://www.martinfowler.com/eaaCatalog/tableDataGateway.html

Fowler, M. (2002). *Value Object*. Retrieved Maí 21, 2007, from P of EAA Catalog: http://www.martinfowler.com/eaaCatalog/valueObject.html

Ganssle, J. (2005, 05 10). *Big Code*. Retrieved 05 19, 2007, from Embedded.com: http://www.embedded.com/showArticle.jhtml?articleID=171203287

WikiPedia. (2007). *Observer Pattern*. Retrieved Maí 21, 2007, from Wikipedia: http://en.wikipedia.org/wiki/Observer_pattern