Relevance of Design Patterns Today

Term Paper

Design patterns are blueprints to solve specific problems. It is a set of optimal guidelines that can be used in new implementations. Developers are not expected to reinvent the wheel each time. Instead they use previously gained valuable experiences. Patterns fit well in the "up-front design" style of development. In today's Agile environment, where the design emerges through an iterative process, where is the relevance for patterns? In the modern web and mobile world, where development life cycles have shrunk considerably, is there a place for design patterns? What challenges do present day software developers face? Do design patterns have answers for these new problems? The scope of this term paper is to look at answers for the above questions.

Grace Ramamoorthy 10278389

4/20/2011

Contents

Relevance of Design Patterns Today

Term Paper

Design Patterns: Description

Design pattern are well known solutions for recurring problems. By defining and categorizing patterns, a catalogue of patterns is created for use when building large and complex software systems. Patterns can be seen as a directorial repository that can be re-used in the later development processes.

Patterns are used in various domains ranging from organizations and processes to teaching and architecture. The software community implements patterns for software architecture and design. They are supposed to improve the final quality of a product by using the insights and experiences of other developers.

According to Dirk Riehle and Heinz Zullighoven, a pattern is a named nugget of insight that conveys the essence of a proven solution to a recurring problem within a certain context amidst competing concerns. Each pattern is a three-part rule that expresses a relation between a context, a problem and a solution. However, not every algorithm, solution or insight constitute a pattern. Patterns need to be tested for recurrence. According to Cope, good patterns solve a problem consistently, capture solutions that are not so apparent, describe a relationship and have a human component.

> Design patterns are documentation of best practices and lessons learned about recurring problems that are encountered in software development.

Most developers are pressured to build reliable software that meets all customer requirements within a short period of time. Most requirements are repetitive and therefore the challenges faced by developers are not new. So developers rely on proven practices and methodologies. Design patterns allow them to exploit the experiences of other developers.

Benefits of knowing and using design patterns are several. They reduce development time as patterns are known solutions for building software systems. They improve software quality as the solutions are tried and tested. Patterns also improve the communication between development teams by identifying names and providing structures to the challenges faced by the developers.

Design Patterns: Comparison

Design Patterns and OO concepts

Design patterns are parallel to object oriented design. They subscribe to reuse without relying on platform or language. Design patterns rely on decomposing larger tasks into smaller pieces; isolating and encapsulating the varying part of the system; extending the behaviour of objects through inheritance; treating objects with common base class in a similar fashion using polymorphism; and loosely coupling the dependent objects to reduce their dependency. Design patterns are design equivalent of object oriented programming.

> Design patterns are design equivalent of object oriented programming

Design Patterns and Algorithms

Algorithms address complex computation hurdles or optimize space, time or resource and provide an efficient way to overcome these challenges. Algorithms are fine-grained and used to implement one or more patterns whereas patterns are broader architectural issues and have large-scale effects. While algorithms are not concerned with maintainability or adaptability, patterns are predominantly concerned about maintainability and reuse. Therefore software developers need good algorithms as well as patterns.

> Algorithms are fine-grained and used to implement one or more patterns

Design Patterns and Frameworks

A framework is a set of cooperating classes that make up a reusable design for a specific class of software. It is a form of implementation of a system of design patterns and design reuse.

Various C++, C#, and Java libraries and frameworks are implementation of design patterns. Objects with names that include proxy, adapter, iterator, visitor, and command in a framework implement the design patterns of the same name.

> Framework is a piece of software that can be executed and patterns are insight and experience about the software

While a framework is a piece of software that can be executed, design patterns are not pieces of code but insight and experience about the software. Frameworks are physical in nature and patterns are logical. Patterns are more abstract, smaller architectural elements, and less specialized than frameworks. While design patterns

identify and name the common concerns of a developer, framework provides a reusable component for solving the problem. Therefore, with correct use of design patterns and building of reusable frameworks, greater productivity in software development can be achieved.

Design Patterns: Relevance

We defined patterns as "a solution to a problem in context". The common problems of web and mobile development are not the same as the problems in traditional development. Design patterns are catalogued by scope and purpose. Since the 1990's, the scope and complexity of software development has changed. Implementation of the patterns depends on the language used for development. With the advent of new emerging development scenario, what is the recurring situation for the developer? Can design patterns be extended?

Design Patterns and Agile Development

Agile development is a collaborative and quick turn-around methodology for developing software. This method requires active user involvement all through the development and empowers the team to make decisions. It allows requirements to evolve but fixes the timescale. In this development methodology, high level requirements are captured visually. Small, incremental releases are developed and focus is on frequent delivery of products. Each feature is completed before moving on to the next feature and testing is an integral part of the development process and happens frequently unlike the traditional system. In this scenario with little or no design stage, is there any scope for patterns? If Agile believes in delivering the most useful 80% of the product features in 20%time, when will refactoring to pattern take place? With Agile's YAGNI corner stone principle, will patterns be considered as extra-baggage?

> Active user involvement; team decisions; evolving user requirements; small incremental releases; frequent delivery of products are hallmarks of Agile development.

The basic difference between Agile and the waterfall method is the recurrent design stage in Agile. Some of the recurring challenges in traditional development are identified and solutions listed in the patterns catalogue. Most challenges faced by the Agile developers are same as in traditional methods and the frequent change in user requirements is also applicable in Agile. This makes it easier for Agile developers to look for existing frameworks that implement design patterns or to create one for their use. While Agile believes in just developing the functionality that is needed, there is no rule to prescribe how that functionality should be coded. If

patterns are a way of thinking then they apply to agile development equally. To take it further, patterns would be a good base on which Agile can ride smoothly. In Agile, if short development cycle time and frequent delivery of products is a challenge, then refactoring to patterns may be a good alternative. Most well-formed solutions to recurrent problems are integrated into frameworks and used extensively in Agile development methodology. Whatever programming language is adapted for development, frameworks that incorporate design patterns are the basic building blocks. Developers use these patterns in their day-to-day work as they incorporate frameworks in their code.

Design Patterns and Mobile Applications

Theories on design patterns contribute to the development of significant tools to standardize solutions. In addition, they assist in the interaction among separate system design concerns and thereby improve the structural design practices. In a new field like Mobile Development, the application of patterns has stretched to finally reach a specific field problem.

Developing mobile applications is both demanding and challenging task for software designers. Architectures that serve portable devices must have special characteristics in order to meet the end-users expectations. These characteristics facilitate the integration between computer-like applications and the operating systems of portable devices. While mobile applications are fast evolving and very useful, they have limitations such as, device size, processor speed, memory size, and battery capacity. These devices are designed in various sizes and display resolutions. They use various operating systems, adapt different technologies and differ in performance. Unpredictability or loss of standardization is a major challenge in creating universal mobile software or testing such software. Programming for such devices must take into account these boundaries and challenges. Can design patterns meet these modern challenges? It is imperative to find appropriate patterns and to tune them to fulfil the needs of mobile-application design.

> Mobile devices are designed in different sizes and display resolutions. Unpredictability and loss of standardization are major challenges.

Design Patterns in Android Development

Some patterns such as the Factory Method, Template Method, Command and Observer are used in the mobile development without any tweaking. The MediaPlayerService class in Android implements the Factory Method to create different types of concrete media players, Activity class in Android development uses the Template Method pattern, Intent uses the Command pattern and Cursor, Adapter and View classes are examples of the implementation of the Observer

pattern in Android development. Android View and Widgets are implementations of Composite pattern. Both Views and nested Views are derived from View base class interface that treats individual Views and composition Views uniformly. The draw() method will draw the individual widget or the composite widget.

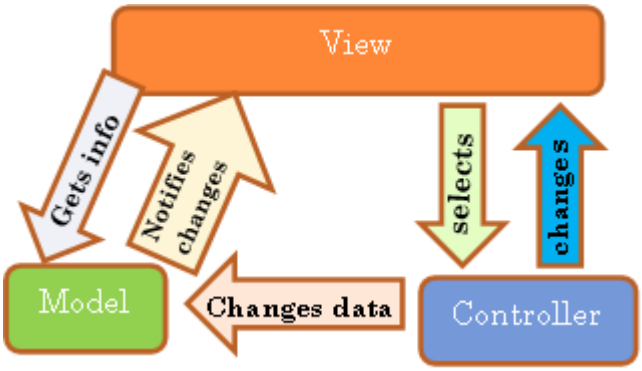
MediaPlayer: Factory Method

Activity class: Template Method

Intent: Command

Cursor, Adapter, View: Observers

Some patterns such as Model-View-Controller are modified to meet the new requirements. In MVC, the Model, View and Controller work together. Supporting dynamic properties is a major challenge while creating Views. For example, a menu could be a simple list of items, or can have a scrollbar, a title, images or a combination of any or all of these. One option is to create many menu objects with different options. This would make the menu complex and not flexible, the other option is to use a Decorator pattern in the View. The scrollbar, title, border and images can be created as decorators and the View could be created so that decorators can be added or removed at runtime.

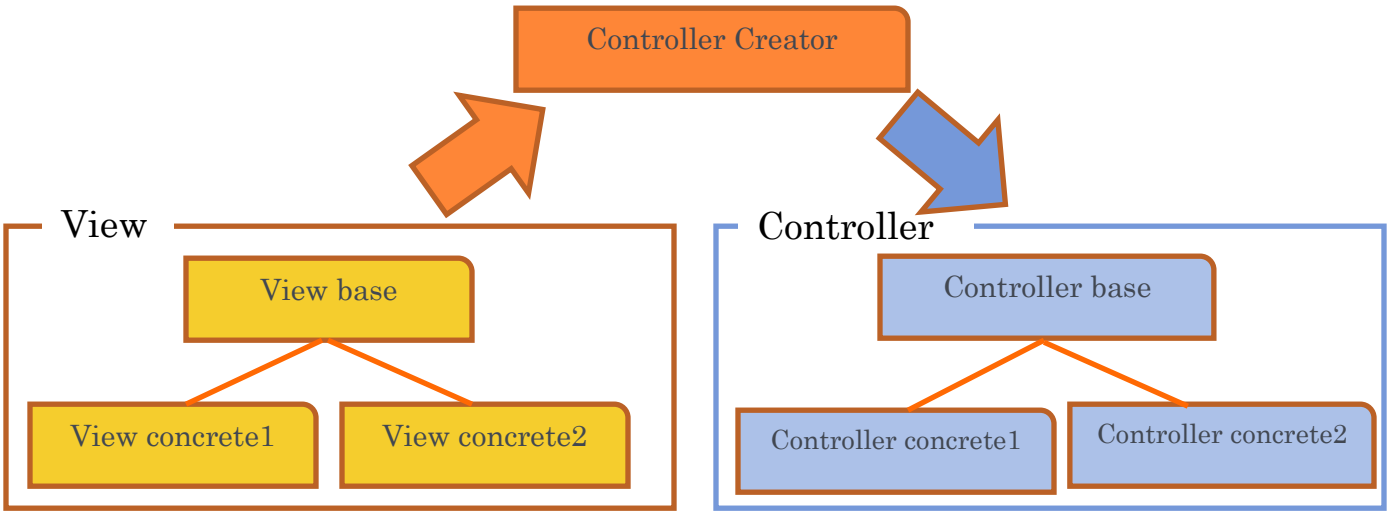


While creating a Controller, mobile applications sometimes require complex multiple algorithms. For example, when the user clicks number 3 on the keypad, user may expect to go the third item on the menu or to go to an item that starts with 'd' or 'D'. In such a situation, different algorithms can be designed and the View can select an algorithm at runtime. Here a Strategy pattern can be used between the Controller and the View. This pattern will encapsulate each algorithm and make them selectable at runtime. They also allow the algorithms to be replaced statically or dynamically and decouple the algorithm from the host.

When the Controller changes the data, the Model notifies the View. However, tight coupling of Model and View is not good practice. The Observer pattern fits in well in

this scenario. The observer notifies the associated Views when the Model changes without coupling the Model and View.

Multiple screen sizes and display resolutions of mobiles lead to multiple look and feel which implies multiple Model, View and Controller objects. To avoid complexity and to improve flexibility, in Android development, the MVC uses Factory Method to create multiple Views and multiple Controllers. It also abstracts the creation of Controllers from the View. The View class just knows when to create a Controller but not what kind of Controller. The controller creator creates the Controller concrete classes but passes the Controller base class information to the View base class. The View base class creates the concrete View classes and allows them to decide the type of Controller class they want to instantiate. This is an example of tweaking the existing patterns to meet new requirements.



Patterns emerge as a by-product of the design process. In mobile applications user interface (UI) is an important aspect of programming. The small screen size and higher user expectations have posed challenges to optimize and standardize space utilization.This has led to new emerging patterns that address UI issues: patterns to create simple user interfaces, dashboards, interactive title bars, action bars, quick action menus, auto-complete search bars etc.,

While most patterns from traditional development can be used in mobile development, there are some patterns like Dependency Injection that do not have the same impact as in large projects. There are yet other design patterns that are not applicable at all for mobile development as the recurring problems those design patterns tend to resolve do not exist in the mobile scenario. For example in the light weight mobile development scenario, using a Singleton pattern may not be ideal.

Design Patterns in Cocoa Development

Implementations of design patterns in Cocoa come in various forms. Some of the patterns are features of Objective-C language, some patterns are implemented in one class or a group of related classes, in some cases, pattern adaptation is part of a framework. In any case, Cocoa development uses patterns wherever it is possible.

Class cluster is an implementation of the Abstract Factory Method. A class cluster groups a number of private concrete subclasses under a public abstract superclass. The abstract superclass declares methods for creating instances of its private subclasses. When mutable or immutable objects are needed, a public class of a class cluster is used to create them. The superclass dispenses an object of the proper concrete subclass based on the creation method invoked. Each object returned may belong to a different private concrete subclass.

Protocols are implementations of Adapter pattern. This is a feature of Objective-C. A protocol is a set of methods that are not associated with any class. When a client object is unable to communicate with another object due to incompatible interfaces, a protocol is defined in the object. Then the client object adopts the protocol and implements a few methods of the protocol. The client object can then use the protocol to send messages to the object. Therefore, protocols implement the Adapter pattern by allowing two incompatible classes to communicate and work together.

Class cluster : Abstract Factory
Protocols : Adapter
AppKit, UIKit : Chain of Responsibility
View hierarchy: Composite

The Chain of Responsibility design pattern is at work in the AppKit framework for error handling and in UIKit for event handling.

The View hierarchy uses Composite patterns. When the window prepares to display the content, the superviews render themselves first and then pass on the message to the subviews to render themselves as well. Therefore, the view hierarchy has a unified view and the part-whole tree structure is seen in their implementation.

NSAttributedString, NSScrollView, and UIDatePicker use Decorator pattern to attach additional functionality to the object dynamically. Delegate pattern is common in Cocoa frameworks. Many classes in AppKit, UIKit and Foundation framework maintain delegates.

The NSImage class of AppKit framework provides a unified interface to loading and using, bitmap-based or vector-based images. NSImage allows more than one representation of the same image and automatically displays the appropriate

representation based on the type of the display device. This is a typical use of the Façade pattern.

NSEnumerator uses the Iterator pattern to traverse arrays, sets and dictionaries. The Controller object in the MVC framework that mediates between the Model and the View fits the Mediator pattern. The NSController abstract class and its concrete subclasses in the AppKit framework automatically synchronize the data in the Model objects as the data displayed in the View object is modified.

The notification mechanism of Cocoa implements one-to-many broadcast of messages based on the Observer pattern. Any object that wants to notify other objects creates a notification object and posts it to a notification centre. Objects in a program add themselves or other objects to a list of observers of one or more notifications. Each notification is identified by a global string. The notification center determines the observers of a particular notification and sends the notification to them via a message. The parameter of the notification method is the notification object, which contains the notification name, the observed object, and a dictionary containing any supplemental information.

> Decorator, Façade, Iterator, Observer and Proxy are some of the patterns implemented in Cocoa.

Concrete subclasses of NSProxy class accomplish the goals set by the Proxy pattern. They act as placeholder for another remote or secure object and control access to that object. Several Cocoa framework classes, such as NSFileManager, NSWorkspace, NSApplication, and, in UIKit, UIApplication, are singletons. A process is limited to one instance of these classes. When a client asks the class for an instance, it gets a shared instance, which is lazily created upon the first request. Template pattern is the fundamental design of Cocoa and other frameworks. The interfaces of Cocoa classes include methods that are meant to be overridden in subclasses.

The Cocoa version of MVC is a compound pattern that includes Composite patterns in the View objects, Strategy pattern between Controller objects and View objects and Observer pattern in the Controller object.

Conclusion

Patterns are examples of following the wisdom that guides good design. They emphasize the importance of creating encapsulation and using delegation to segregate systems to protect them from cascading change. When they were first suggested, computers were fundamentally slower, and technology resources, such as

memory and disk space were more expensive than they are today. The style of design suggested by patterns is far more realistic now.

Patterns are thought of as part of the old "up-front design" style of development, but we now realize that they are more powerful in an Agile environment, where much is assumed to be unknown at the beginning of a project, and where the design is expected to emerge through an iterative process.

Patterns are more useful as collection of best practices, rather than simply "reusable solutions". Patterns must be used as part of a thought process that guides analysis, using "pattern orientation" as a way to understand an ever-changing problem domain. So it really does not matter if it is Agile, Web or Mobile development. Design patterns have wound their way into frameworks and code components and are used more extensively now. Design pattern is a thought process that guides good programming practice and it is as relevant today as it was 20 years ago.

References

http://www.techrepublic.com/article/reap-the-benefits-of-design-patterns-in-software-development/5173591

http://ezinearticles.com/?Design-Patterns-for-Software-Engineering&id=550329

Importance of Design Patterns and Frameworks for Software Development by Ólafur Andri Ragnarsson, Adjunct, Reykjavik University, Kringlan, Reykjavík

Design Patterns for Multiagent Systems to Elevate Pocket Device Applications by Sameh Abdel-Naby Paolo Giorgini of University of Trento, Trento, Italy and Michael Weiss of Carleton University. Ottawa, Ontario, Canada.

Design Patterns in Mobile Architectures, Tomáš Chlouba

http://dl.google.com/googleio/2010/android-android-ui-design-patterns.pdf

http://kcchao.wikidot.com/design-patterns-in-android

http://andyjeffries.co.uk/articles/design-patterns-vs-agile-development

http://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/CocoaFundamentals/CocoaDesignPatterns/CocoaDesignPatterns.html

http://www.allaboutagile.com/10-key-principles-of-agile-software-development/

http://andyjeffries.co.uk/articles/design-patterns-vs-agile-development