FACULTAD DE INGENIERÍA
TECNOLOGÍA & DESARROLLO

Vol. 11, Nº 1, Enero - Diciembre 2013, Pág.059-066

Tecnología y Desarrollo

El patrón de arquitectura n-capas con orientación al dominio como solución en el diseño de aplicaciones empresariales.

The pattern of N-Layers architecture with domain orientation as a solution in Enterprise Application design.

Lain Cárdenas Escalante
Universidad César Vallejo, Trujillo, Perú
laincardenas@gmail.com

Recepción: 21-05-2013 **/ Aceptación:** 20-07-2013

RESUMEN

El artículo presenta un Modelo de Diseño de Software aplicando el patrón de arquitectura N-Capas con Orientación al Dominio y el patrón de diseño Fábrica Abstracta. El propósito de aplicar estos patrones en el diseño de Aplicaciones Empresariales complejas es para garantizar que sean flexibles y fácilmente mantenibles.

Palabras clave: *Patrón de Arquitectura, Patrón de Diseño, Desacoplamiento.*

ABSTRACT

The paper presents a Model of Software Design using the pattern N-Layers architecture with domain orientation and the Abstract Factory design pattern. The purpose of applying these patterns in complex enterprise application design is to ensure they are robust and easily maintainable.

Key words: *Architecture pattern, Design Pattern, Decoupling.*

1. INTRODUCCIÓN

Las empresas luchan por mantener y conseguir más clientes para garantizar su estabilidad y crecimiento económico y para lograrlo deben brindar buenos servicios. Brindar un buen servicio va a depender mucho de cómo se llevan a cabo los procesos de negocio. Para esto, las empresas necesitan y deben automatizar de la mejor forma sus procesos para ser agiles y productivos y así poder ofrecer buenos servicios a sus clientes.

Automatizar los procesos de negocio requiere la implementación de sistemas informáticos. Pero principalmente las empresas medianas y grandes suelen tener procesos complejos que requieren de implementaciones de sistemas complejos. Estos tipos de empresas tienen procesos complejos porque sus procesos manejan o aplican muchas reglas de negocio, condiciones, o políticas y que pueden ir variando en el tiempo. Además, estas empresas están creciendo constantemente, lo cual va a demandar más usuarios, más volumen de información, y cambios tecnológicos. Por tanto, los sistemas informáticos que se desarrollen para estos tipos de empresas van a requerir que sean flexibles ante los cambios, es decir que no genere trauma o costos elevados cuando se hagan cambios, también van a requerir que sean sistemas robustos, seguros y escalables.

En consecuencia, el problema a solucionar es: ¿Cómo estructurar una aplicación para soportar requerimientos complejos operacionales y disponer de una buena mantenibilidad, reusabilidad, escalabilidad, robustez y seguridad?

Entonces, para dar solución a este problema se plantea lo siguiente: Organizar la estructura lógica de gran escala en un sistema en capas separadas de responsabilidades distintas y relacionadas con una separación clara cohesiva de intereses.

Para el desarrollo y explicación de este trabajo, el artículo se ha organizado de la siguiente manera: en la Sección 2 se muestra un breve marco teórico; en la Sección 3 se describe la solución con un Modelo de Diseño aplicando patrones; y por último, en la Sección 4 se presentan algunas las ventajas y recomendaciones.

2. MARCO TEÓRICO

2.1 Arquitectura de Software

Una arquitectura de software es una descripción de los subsistemas y componentes de un sistema de software y las relaciones entre ellos. Subsistemas y componentes se especifican normalmente en diferentes vistas para mostrar las propiedades funcionales y no funcionales relevantes de un sistema de software. La arquitectura de software de un sistema es un artefacto. Es el resultado de la actividad de diseño de software [1].

2.2 Componente

Un componente es una parte encapsulada de un sistema de software. Un componente tiene una interfaz. Los componentes sirven como bloques de construcción para la estructura de un sistema. A nivel de lenguaje de programación, los componentes pueden ser representados en forma de módulos, clases, objetos o un conjunto de funciones relacionadas [1].

2.3 Vista

Una vista representa un aspecto parcial de una arquitectura de software que muestra propiedades específicas de un sistema de software [1].

2.4 Mantenibilidad

Esta se ocupa principalmente de la fijación de un problema, "reparar" un sistema de software después de que ocurran errores. Una arquitectura de software que está bien preparado para el mantenimiento tiende a localizar cambios y minimizar sus efectos secundarios en otros componentes [1].

2.5 Extensibilidad

Este se centra en la extensión de un sistema de software con nuevas características, así como la sustitución de los componentes con las versiones mejoradas y la eliminación de las características y componentes no deseados o innecesarios. Para lograr extensibilidad un sistema de software requiere componentes débilmente acoplados. Se

trata de una estructura que le permite intercambiar componentes sin afectar a sus clientes. El apoyo a la integración de nuevos componentes en una arquitectura existente también es necesario [1].

2.6 ¿Qué es un Patrón?

Un patrón es una regla de tres partes, que expresa una relación entre un cierto contexto, un problema, y una solución.
Un patrón aborda un problema de diseño recurrente que surge en situaciones específicas de diseño, y presenta una solución a ella.
Los patrones identifican y especifican abstracciones que están por encima del nivel de las clases individuales e instancias, o de los componentes. Por lo general, un patrón describe varios componentes, clases u objetos, y detalles de sus responsabilidades y relaciones, así como su cooperación. Los patrones son una forma de documentar arquitecturas software. Los patrones proporcionan un esqueleto del comportamiento funcional y por lo tanto ayudan a implementar la funcionalidad de la aplicación.
Un patrón arquitectónico expresa un esquema de organización estructural fundamental de los sistemas de software. Proporciona un conjunto de subsistemas predefinidos, especifica sus responsabilidades, e incluye reglas y directrices para la organización de las relaciones entre ellos.
Un patrón de diseño proporciona un esquema para refinar los subsistemas o componentes de un sistema de software, o las relaciones entre ellos. En él se describe una estructura común recurrente de comunicar componentes que resuelve un problema de diseño general en un contexto particular [1].

2.7 Patrón Capas

El patrón arquitectónico Capas ayuda a estructurar las aplicaciones que se pueden descomponer en grupos de subtareas en la que cada grupo de subtareas está en un nivel particular de abstracción.

Contexto: Un sistema de gran tamaño que requiere la descomposición.

Problema: Imagínese que usted está diseñando un sistema cuya característica predominante es una mezcla de temas de alto y bajo nivel, donde las operaciones de alto nivel se basan en los niveles inferiores.

Tales sistemas a menudo también requieren alguna estructuración horizontal que es ortogonal a su subdivisión vertical. Este es el caso en el que varias operaciones están en el mismo nivel de abstracción, pero son en gran parte independientes entre sí.

En tal caso, es necesario equilibrar las fuerzas siguientes:
Cambios tardes en el código fuente no debe enredar a través del sistema. Ellos deben limitarse a un componente y no afectar a los demás.

Las partes del sistema deben ser intercambiables. Los componentes deben ser capaces de ser reemplazado por implementaciones alternativas sin afectar al resto del sistema. Diseño para el cambio, en general, es un importante facilitador de la evolución normal del sistema.

Solución: Desde el punto de vista de alto nivel de la solución es muy simple. Estructura tu sistema en un número apropiado de capas y colocarlas una encima de otra. Comience en el nivel más bajo de abstracción, llámalo capa 1. Esta es la base de tu sistema. Trabaja la abstracción poniendo la capa J en la parte superior de la capa de J-1 hasta llegar al nivel superior de la funcionalidad, llámalo capa N.
La principal característica estructural del patrón de capas es que los servicios de la capa J sólo son utilizados por la capa J+1, no hay más dependencias directas entre capas. La responsabilidad de la capa J es proveer servicios usados por la capa J+1 y delegar sub tareas a la capa J-1 [1].

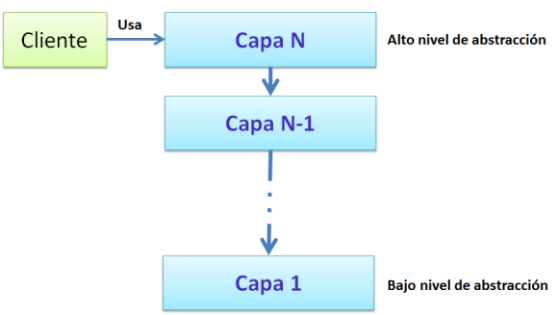

Figura 1: Patrón Capas.

2.8 Patrón N-Capas con Orientación al Dominio

Este patrón es una extensión al patrón Capas tradicional explicado en el punto anterior. En el nivel más alto y abstracto, la vista de arquitectura lógica de un sistema puede considerarse como un conjunto de servicios relacionados agrupados en diversas capas, similar a la figura 2.

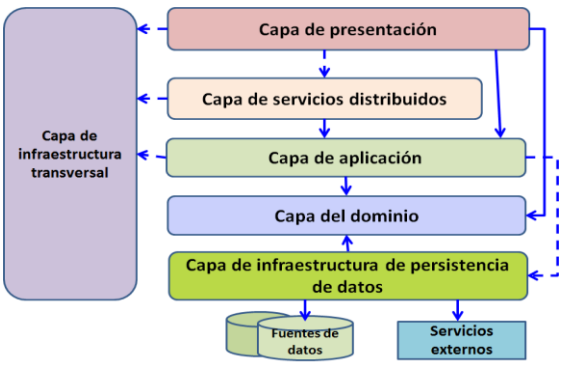

Figura 2: Arquitectura N-Capas orientada al Dominio.

En Arquitecturas "Orientadas al Dominio" es crucial la clara delimitación y separación de la capa del Dominio del resto de capas. Es realmente un pre-requisito para DDD (Domain Driven Design). "Todo debe girar alrededor del Dominio".

En las aplicaciones complejas, el comportamiento de las reglas de negocio (lógica del Dominio) está sujeto a muchos cambios y es muy importante poder modificar, construir y realizar pruebas sobre dichas capas de lógica del dominio de una forma fácil e independiente. Debido a esto, un objetivo importante es tener el mínimo acoplamiento entre el Modelo del Dominio (lógica y reglas de negocio) y el resto de capas del sistema (capas de presentación, capas de infraestructura, persistencia de datos, etc.). Así pues, las razones por las que es importante hacer uso de una "Arquitectura N-Capas Orientada al Dominio" es especialmente en los casos donde el comportamiento del negocio a automatizar (lógica del dominio) está sujeto a muchos cambios y evoluciones [2].

En concreto, las capas y sub-capas propuestas para aplicaciones "N-Capas con Orientación al Dominio" son:

a. Capa de Presentación

Esta capa es responsable de mostrar información al usuario e interpretar sus acciones.

b. **Capa de Servicios Distribuidos**
Cuando una aplicación actúa como proveedor de servicios para otras aplicaciones remotas, o incluso si la capa de presentación esta también localizada físicamente en localizaciones remotas, normalmente se publica la lógica de negocio (capas de negocio internas) mediante una capa de servicios. Esta capa de servicios (habitualmente Servicios Web) proporciona un medio de acceso remoto basado en canales de comunicación y mensajes de datos.

c. Capa de Aplicación
Define los trabajos que la aplicación como tal debe de realizar y redirige a los objetos del dominio y de infraestructura (persistencia, etc.) que son los que internamente deben resolver los problemas. Sirve principalmente para coordinar la lógica del flujo del caso de uso. También permite implementar conversores de formatos o adaptadores.

d. Capa del Dominio
Esta capa es responsable de representar conceptos de negocio e implementación de las reglas del dominio. Esta capa, Dominio, es el corazón del software. Así pues, estos componentes implementan la funcionalidad principal del sistema y encapsulan toda la lógica de negocio relevante (genéricamente llamado lógica del Dominio según nomenclatura DDD).

e. Capa de Persistencia de Datos
Esta capa proporciona la capacidad de persistir datos así como lógicamente acceder a ellos. Pueden ser datos propios del sistema o incluso acceder a datos expuestos por sistemas externos (Servicios Web externos, etc.). Así pues, esta capa de persistencia de datos expone el acceso a datos a las capas superiores. Esta exposición deberá realizarse de una forma desacoplada (esto se puede lograr aplicando el patrón fábrica abstracta).

2.8 Patrón Fábrica Abstracta

En términos simples, una fábrica abstracta es una clase que proporciona una interfaz para producir una familia de objetos [3]. El diseño general del patrón se muestra en la figura 3.

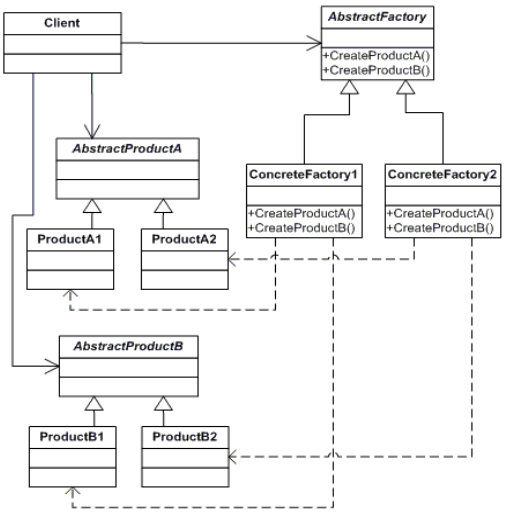

Figura 3: Patrón Fábrica Abstracta.

Las clases que participan en este patrón son:

- **Fabrica Abstracta** (de la figura 3: AbstractFactory): declara una interfaz para operaciones que crean objetos productos abstractos.

- **Fabrica Concreta** (de la figura 3: ConcreteFactory1, ConcreteFactory2): implementa las operaciones para crear objetos productos concretos.

- **Producto Abstracto** (de la figura 3: AbstractProductA, AbstractProductB): declara una interfaz para un tipo de objeto producto.

- **Producto Concreto** (de la figura 3: ProductA1, ProductA2, ProductB1, ProductB2): define un objeto producto para que sea creado por la fábrica correspondiente; implementa la interfaz producto abstracto.

- **Cliente** (de la figura 3: Client): sólo usa interfaces declaradas por las clases Fabrica Abstracta y Producto Abstracto.

Úsese el patrón principalmente cuando:

- Un sistema debe ser independiente de cómo se crean, componen y representan sus productos.
- Quiere proporcionar una familia de clases de productos, y sólo quiere revelar sus interfaces, no sus implementaciones.

3. MODELO DE DISEÑO

El modelo de diseño que se presenta en esta sección permite demostrar la aplicación del patrón de arquitectura N-Capas con Orientación al Dominio y la aplicación del patrón de diseño Fábrica Abstracta. El uso del patrón de diseño Fábrica Abstracta va a permitir lograr el desacoplamiento entre la Capa de Aplicación y la Capa de Persistencia de Datos.

El modelo de diseño representa la vista lógica de la arquitectura, el cual permite mostrar las capas, subcapas y clases que participan en la realización de un caso de uso significativo o de alta prioridad.
Para el caso de ejemplo, se diseña el caso de uso "Generar pago de trabajador". El propósito del caso de uso es calcular y registrar el pago a un trabajador que se ha contratado por un periodo de tiempo para realizar un conjunto de tareas. El pago dependerá de cuantas tareas haya realizado según lo programado y en cuantas horas. La hora de trabajo programada y la hora de trabajo extra tiene fijado un valor en su contrato.

A continuación se explica el diseño del caso de uso.

3.1. Diseño de la Arquitectura

A continuación se muestra un diagrama de paquetes de UML que representa la Arquitectura N-Capas. Cada capa contiene subcapas y cada subcapa contiene las clases de diseño necesarias para la realización del caso de uso. La figura 4 muestra la estructura general de la arquitectura con capas y subcapas.

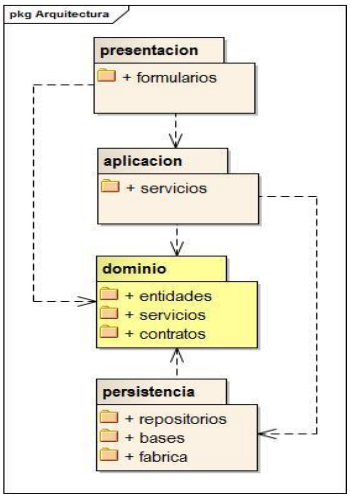

Figura 4: Arquitectura N-Capas orientado al Dominio.

3.2 Capa de Presentación

Esta capa contiene la subcapa "formularios" en donde en encuentra la clase FormPagoContrato que representa la interfaz gráfica de usuario que le permite al actor interactuar con la funcionalidad del sistema relacionado al caso de uso. La figura 5 representa la capa de presentación con la subcapa y su clase.

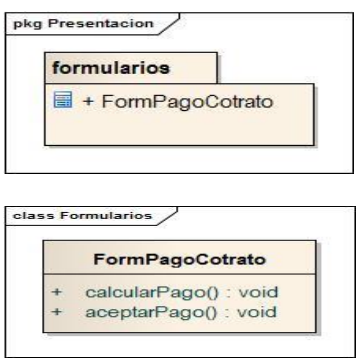

Figura 5: Capa de Presentación.

3.3 Capa de Aplicación

Esta capa contiene la subcapa "servicios" en donde se encuentra la clase PagoContratoServicio que permite coordinar las peticiones que vienen de la capa de presentación y delegando tareas a las capas del dominio y persistencia. La figura 6 representa la capa de aplicación con la subcapa y su clase.

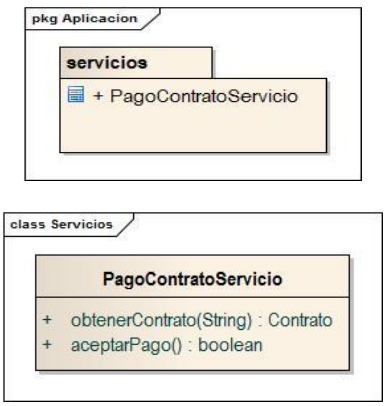

Figura 6: Capa de Aplicación.

3.4 Capa de Dominio

Esta capa contiene la subcapa "entidades" en donde se encuentran las clases que representan la información del dominio o del negocio y que tienen la responsabilidad de manejar las reglas o lógica de negocio. También contiene la subcapa "contratos" en donde se encuentran las interfaces que serán implementadas por las clases de la capa de persistencia. Las interfaces definen los contratos que se deben respetar para lograr el desacoplamiento con la capa de persistencia. La figura 7 representa la capa de dominio con las subcapas y sus clases.

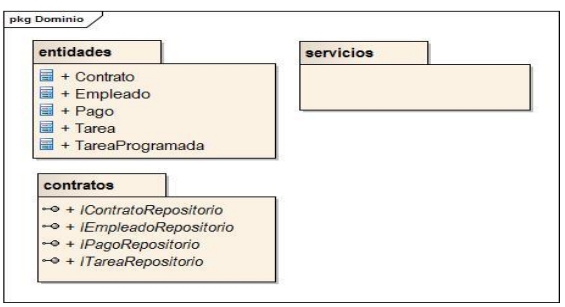

Figura 7: Capa de Dominio.

La figura 8 muestra el diagrama de clases de la subcapa "entidades" de la capa de Dominio. En esta figura se muestran sólo los atributos de las clases y

no las operaciones por motivo de mejor visualización.

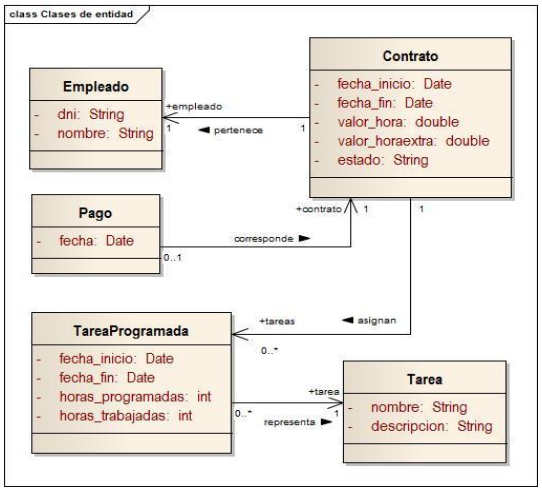

Figura 8: Clases de la Subcapa Entidades.

La clase entidad que contiene operaciones más relevantes en este modelo es la clase Contrato. La figura 9 muestra la clase Contrato con todos sus atributos y operaciones.

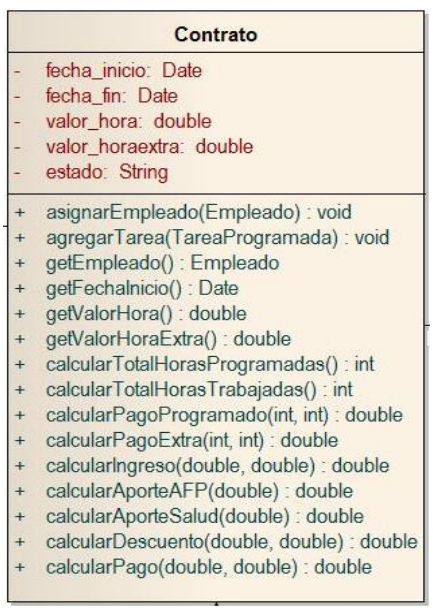

Figura 9: Clase Contrato.

3.5 Capa de Persistencia

Esta capa contiene la subcapa "repositorios" en donde se encuentran las clases que implementan las interfaces de la subcapa "contratos" de la capa de Dominio. Estas clases cumplen lógica de acceso a datos, es decir consultan o actualizan las fuentes de datos. También contiene la subcapa "fabrica" en donde se encuentran las clases que implementan el patrón Fabrica Abstracta. La figura 10 representa la capa de persistencia con las subcapas y sus clases.

Por otra parte, la figura 11 muestra el diagrama de clases de la subcapa "repositorios" de la capa de Persistencia. En este diagrama de clases se muestran las relaciones de realización entre las clases repositorios y las interfaces.

Así mismo, la figura 12 muestra el diagrama de clases de la subcapa "fabrica" de la capa de Persistencia. En este diagrama de clases se muestra la relación de herencia entre las clases, en donde la clase FabricaRepositorioSQL implementa los métodos abstractos de la clase abstracta FabricaRepositorio. FabricaRepositorioSQL es la clase que crea los objetos repositorio que acceden a una base de datos SQL.

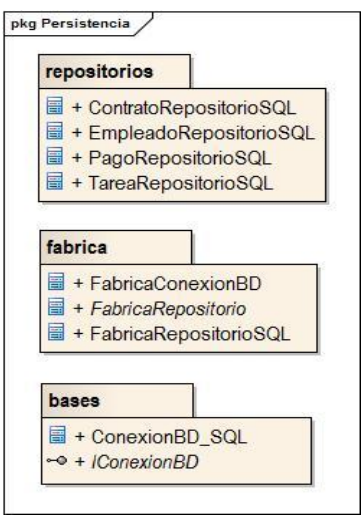

Figura 10: Capa de Persistencia.

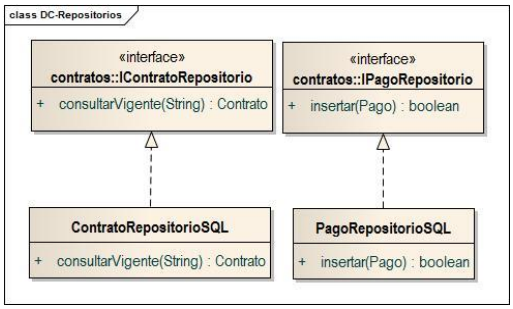

Figura 11: Clases de la Subcapa Repositorios.

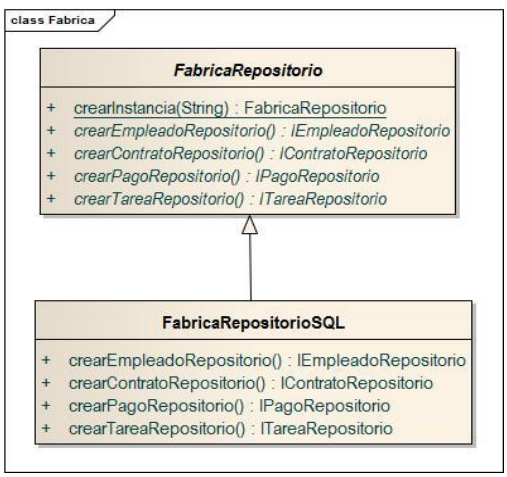

Figura 12: Clases de la Subcapa Fábrica.

4. VENTAJAS Y RECOMENDACIONES

4.1 Ventajas

El mantenimiento de mejoras en una solución será mucho más fácil porque las funciones están localizadas. Además las capas deben estar débilmente acopladas entre ellas y con alta cohesión internamente, lo cual posibilita variar de una forma sencilla diferentes implementaciones/combinaciones de capas.

4.2 Recomendaciones

Este tipo de arquitectura deberá implementarse en las aplicaciones empresariales complejas cuya lógica de negocio cambie bastante y la aplicación vaya a sufrir cambios y mantenimientos posteriores durante una vida de aplicación, como mínimo, relativamente larga.
Por otra parte, este tipo de arquitectura no debería implementarse en aplicaciones pequeñas que una vez finalizadas se prevén pocos cambios, la vida de la aplicación será relativamente corta y donde prima la velocidad en el desarrollo de la aplicación.

REFERENCIAS BIBLIOGRÁFICAS

[1] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sornmerla, Michael Stal. "PATTERN - ORIENTED SOFTWARE ARCHITECTURE, Volumen 1: A System of Patterns". John Wiley & Sons, 1996.

[2] Cesar de la Torre Llorente, Unai Zorrilla castro, Javier Calvarro Nelson, Miguel Àngel Ramos Barroso. "Guía de Arquitectura N-Capas orientada al Dominio con .Net 4.0". Primera edición, Krasis Press, 2010. http://msdn.microsoft.com/es-es/architecture

[3] Erich Gamma. "Patrones de Diseño". Addison Wesley, 1995.