Universidad de las Ciencias Informáticas

Facultad 15

Propuesta de una guía de principios de buenas prácticas para el Desarrollo de Software Orientado a Aspectos.

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autor: Danise Torres Matos

Tutor: Ing. C. Inf. Jorge Luis Valdés González

Junio 2010

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo a la Facultad 15 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Danise Torres Matos

_____

Firma de Autor

Jorge Luis Valdés González

_____

Firma de Tutor

AGRADECIMIENTOS

No resulta fácil agradecer a todas las personas que han estado conmigo brindándome su apoyo en cada momento en estos cinco cursos aquí en la UCI, pero de una manera u otra quiero agradecerles de todo corazón por haber estado conmigo en los momentos buenos y malos.

Especialmente quiero agradecer a mi mamá y a mi papá y mi hermanita por ser la luz de mi vida, mi máxima inspiración, por no haber dudado ni un segundo y haberme apoyado siempre, sin ellos no hubiese sido posible llegar hasta aquí. Los quiero mucho.

A mi tutor Ing. Jorge Luis Valdés por su esfuerzo y confianza, por hacer de mí una mejor profesional. Por todos los conocimientos que me ha aportado. Por ser más que un guía, por ser un autentico compañero y amigo. Por todo el apoyo que me brindó para lograr con éxitos este trabajo de diploma.

A todos mis compañeros y amigos que siempre estuvieron presente en las buenas y en las malas. En especial a mis mejores amigas, a las que siempre han estado junto a mí, dándome cada una lo mejor de sí, las quiero mucho: Yude, Baby, Lisy y Vivi.

A mi familia por su ayuda, comprensión, y apoyo en todo momento a lo largo de estos cinco años.

A todos los profesores que dejaron algo suyo en mí, y que se esforzaron para que aprendiera y fuera una mejor persona y profesional.

A la Revolución y a nuestro Comandante Fidel Castro, por habernos dado la oportunidad de estudiar en esta universidad.

A todos ustedes muchas gracias.

Danise

DEDICATORIA

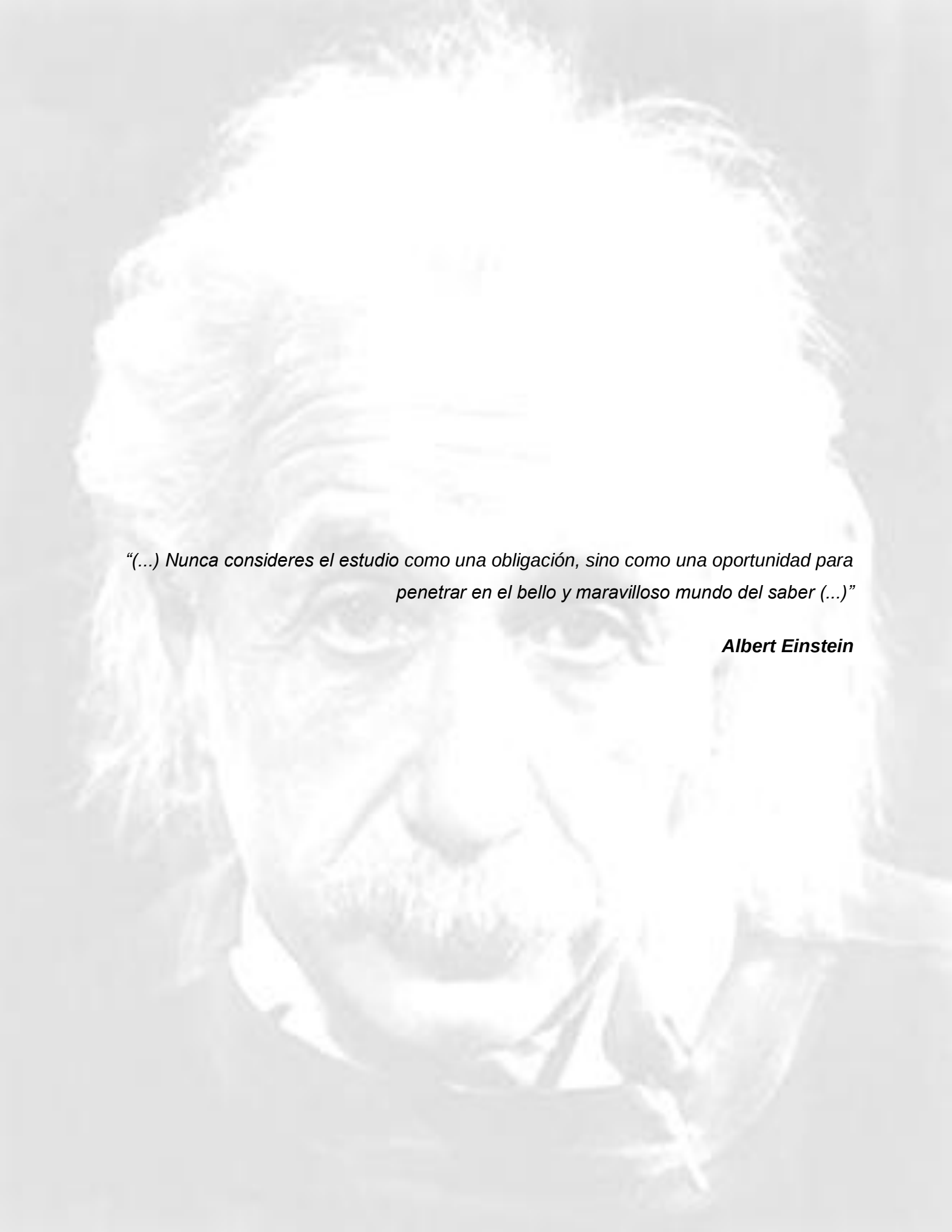
A la flor más linda que existe, la razón de mí empeño, el faro que guía cada día que vivo, la persona más importante en mi vida, la que con sus consejos, desvelos, ejemplo y sacrificio ha hecho de mi lo que soy hoy, la que ha batallado tanto para lograr que este sueño se hiciera realidad: **mi mamita querida**.

Al hombre que he admirado toda mi vida, al que me enseño a sacar de todo lo malo la parte buena, aunque ya no estés conmigo: a ti mi abuelito lindo.

A mi papá por su ejemplo, sacrificio y amor, al que siempre he admirado y respetado.

A mi hermanita Susana que aún estando lejos se mantuvo siempre junto a mí, sigue así cómo vas, espero verte pronto realizando también tus sueños.

A mis grandes amigas Viviana, Yudelkys, Lisset y Bárbara, juntas hemos pasado por mucho, puedo decir en estos momentos que soy una persona rica, pues ustedes me han dado lo más valioso que un ser humano puede tener: su amistad.



“(...) Nunca consideres el estudio como una obligación, sino como una oportunidad para penetrar en el bello y maravilloso mundo del saber (...)”

Albert Einstein

RESUMEN

El presente trabajo realiza un estudio sobre el estado de aplicación del paradigma de Programación Orientada a Aspectos (POA) en los proyectos productivos de la Universidad de las Ciencias Informáticas (UCI), a partir del cual se identifica una creciente necesidad de definir una guía de principios de buenas prácticas para apoyar el proceso de toma de decisiones en la inserción de técnicas de POA en el desarrollo de aplicaciones informáticas. Se analizan principios conceptuales del paradigma orientado a aspectos reflejados en la evaluación sintáctico-semántica del lenguaje de propósito general *AspectJ* y se exponen un conjunto de principios de buenas prácticas enfocados a apoyar el proceso de introducción de la POA en el desarrollo tradicional, así como una evaluación cualitativa de la factibilidad de separar las incumbencias mediante técnicas de POA y un análisis cuantitativo a partir de modelos matemáticos propuestos, que se enfocan en el favorecimiento de los indicadores de productividad mediante la reducción del esfuerzo de desarrollo, si se logra aplicar adecuadamente el paradigma antes mencionado.

Palabras claves: aspectos, incumbencias

ÍNDICE

DECLARACIÓN DE AUTORÍA I

AGRADECIMIENTOS 1

DEDICATORIA 2

RESUMEN 4

Capítulo 1

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA 15

INTRODUCCIÓN 15

1.1. LAS INCUMBENCIAS EN EL DESARROLLO DE APLICACIONES INFORMÁTICAS. 17

1.1.1 IDENTIFICACIÓN DE INCUMBENCIAS. 17

1.1.2 DESCOMPOSICIÓN ASPECTUAL DE UN SISTEMA. 19

1.1.2.1 DESCOMPOSICIÓN ASPECTUAL MEDIANTE LA POO. 20

1.2. LA PROGRAMACIÓN ORIENTADA A ASPECTOS. UNA NUEVA FILOSOFÍA. 21

1.2.1 FUNDAMENTOS DE LA PROGRAMACIÓN ORIENTADA A ASPECTOS 21

1.2.2 RESEÑA HISTÓRICA. 22

1.2.3 ANATOMÍA DEL PARADIGMA ORIENTADO A ASPECTOS. 23

1.2.4 EL TEJEDOR DE ASPECTOS Y LOS TIPOS DE TEJIDOS. 24

1.2.4.1 TEJIDO ESTÁTICO. 25

1.2.4.2 TEJIDO DINÁMICO. 26

1.2.4.3 ¿TEJIDO ESTÁTICO VS TEJIDO DINÁMICO? 26

1.2.5EL LENGUAJE BASE 27

1.3 ALGUNAS EXPERIENCIAS DE APLICACIÓN DE LA POA EN EL MUNDO. 28

1.4 ANÁLISIS SOBRE EL EMPLEO Y CONOCIMIENTO DE LA POA EN LA UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS 31

1.4.1 APLICACIÓN DE UN MÉTODO DE RECOLECCIÓN DE DATOS 31

1.4.1.1 REALIZACIÓN DE UN CUESTIONARIO..... 31

1.4.1.2 ENTREVISTA A DIRECTIVOS DE LA ACTIVIDAD PRODUCTIVA DE LA UCI..... 33

1.4.2 ANÁLISIS DE LOS RESULTADOS DEL ESTUDIO DE CAMPO..... 34

1.4.2.1 RESULTADOS DEL CUESTIONARIO 34

1.4.2.2 RESULTADOS DE LAS ENTREVISTAS A DIRECTIVOS DE LA INFRAESTRUCTURA PRODUCTIVA DE LA UCI..... 36

Capítulo 2

CAPÍTULO 2: ANÁLISIS DEL LENGUAJE ORIENTADO A ASPECTOS DE PROPÓSITO GENERAL ASPECTJ.SINTAXIS Y SEMÁNTICA 42

INTRODUCCIÓN..... 42

2.1 ELEMENTOS BÁSICOS DE ASPECTJ..... 43

2.2 PUNTOS DE CORTE..... 44

2.2.1 TIPOS DE PUNTOS DE CORTE..... 44

2.2.2 PUNTOS DE CORTES BASADOS EN CONTROL-FLOW. 45

2.2.3 PUNTOS DE CORTES BASADOS EN LOCALIZACIÓN (WITHIN)..... 47

2.2.4 PUNTOS DE CORTES BASADOS EN OBJETOS (THIS, TARGET)..... 48

2.2.5 PUNTOS DE CORTES BASADOS EN ARGUMENTOS (ARGS). 48

2.2.6 PUNTOS DE CORTES CON CONDICIONES (IF). 48

2.2.7 COMBINAR PUNTOS DE CORTES..... 48

2.3 SINTAXIS Y SEMÁNTICAS DE LOS PUNTOS DE ENLACE..... 49

2.3.1 EL MODELO DINÁMICO DE PUNTOS DE ENLACES..... 49

2.4 AVISOS..... 51

2.5 INTER-TIPOS (INTER-TYPE). 51

CONCLUSIONES PARCIALES..... 53

Capítulo 3

CAPÍTULO 3: GUÍA DE PRINCIPIOS DE BUENAS PRÁCTICAS PARA EL DESARROLLO ORIENTADO A ASPECTOS 54

INTRODUCCIÓN 54

3.1 CAPTURA Y MODELACIÓN DE INCUMBENCIAS EMPLEANDO CASOS DE USO 55

3.2 MANTENER LA SEPARACIÓN DE LAS INCUMBENCIAS MEDIANTE MODELOS DE CASOS DE USO 59

3.3 ESTRUCTURACIÓN DE EQUIPOS DE DESARROLLO ENFOCADA EN LA SEPARACIÓN DE INCUMBENCIAS. 62

3.3.1 CONFORMACIÓN DE UN EQUIPO BASADO EN EL DESARROLLO ORIENTADO A ASPECTOS 67

3.3.1.1 DESARROLLO DE LA LÓGICA DE NEGOCIO 68

3.3.1.2 DESARROLLO DE LA LÓGICA DE ASPECTOS 69

3.4 PRINCIPIO DE "CERO ACOPLAMIENTO" 71

3.5 DISEÑOS MÁS FLEXIBLES 75

3.6 VALORACIONES CONSENSUADAS 76

CONCLUSIONES GENERALES 78

RECOMENDACIONES 79

TRABAJOS FUTUROS 79

BIBLIOGRAFÍA 80

ANEXO 1: RESULTADOS DEL CUESTIONARIO 82

ANEXO 2: EJEMPLO DE ARQUITECTURA SOBRE PLATAFORMA .NET QUE IMPLEMENTA LOS PATRONES IOC E IOD. 87

INTRODUCCIÓN

La sociedad actual se encuentra en un proceso de transformación donde la información es un recurso decisivo, ya que la creación, distribución y manipulación de la misma forma parte importante de las actividades culturales, políticas, sociales y económicas. Se requieren entonces soluciones informáticas capaces de estar en consonancia con las especificaciones del entorno y las demandas del mercado y la sociedad en general. A raíz de esto, ha alcanzado gran importancia la calidad de los productos informáticos que son utilizados por las instituciones, organizaciones o individuos en particular. ¿Viajaría en un avión controlado por una computadora?

Al analizar la calidad de un sistema informático suele enfocarse en el producto en sí, cuando este análisis debiera extenderse incluso hasta el propio proceso, dado a que, en cierta medida, la calidad de este último incide significativamente en la calidad del primero. Actualmente, en el contexto del desarrollo de aplicaciones informáticas, convergen un sinnúmero de dificultades, que provocan dilación en los tiempos de entrega de los productos, los presupuestos asignados al proyecto no sean suficientes, que el costo de producción sea difícil de pronosticar o muchas veces sobrepase las estimaciones iníciales que se hacen del producto, entre otros tantos, debido, en gran medida, a que no se sigue un proceso bien definido, correctamente estructurado y en correspondencia con las legítimas necesidades de los clientes. Algunas de estos problemas están reflejados en la incorrecta concepción y aplicación de los procedimientos y métodos; inadecuada estimación del esfuerzo de realización incidiendo negativamente en la planificación; incoherencia o poca correspondencia entre las necesidades de los clientes y la solución final, entre otros aspectos.

Un proceso no es más que la definición de "*quién*" está haciendo "*qué*", "*cuándo*" y "*cómo*" para alcanzar el objetivo propuesto. Las empresas no han adquirido una disciplina en el desarrollo de soluciones informáticas, pero cabe señalar que sea cual sea el camino por el que opte la misma para mejorar la calidad de sus productos informáticos, implicaría mejoras sustanciales en la forma en que se desarrollan estos y la manera de utilizar los métodos y procedimientos de la Ingeniería y Gestión de Software (JACOBSON, y otros, 2000).

Para el correcto desenvolvimiento de un proceso de desarrollo de aplicaciones informáticas, es necesario ir indicando paso a paso las actividades que se van a realizar, qué personas deben

participar en el desenvolvimiento de las mismas y qué papel deben desempeñar con el objetivo de lograr un producto final.

Implantar una metodología no es una labor de inmediatos resultados, cuesta tiempo para el equipo de trabajo adaptarse a ella, esto hace necesario ajustarlas a las necesidades de cada proyecto y de su equipo de realización. Actualmente entre las metodologías de desarrollo más utilizadas están: el *Proceso Unificado de Desarrollo de Software* (RUP[1] por sus siglas en inglés), Microsoft Solution Framework (MSF), Extreme Programming (XP[2]), Open, Métrica 3 y Crystal; agrupadas en metodologías *ágiles* y metodologías *pesadas,* en dependencia de la naturaleza de su concepción general.

Diversos son los elementos que componen una metodología de desarrollo de sistemas informáticos, dentro de los cuales es preciso analizar la existencia de un componente en particular, el cual determina en gran medida, no sólo la dinámica de desarrollo sino también la garantía de obtener una alta calidad del producto final, la reducción en los tiempos de entrega e incluso un efectivo uso de los recursos humanos: el paradigma de desarrollo que se emplee. Este es apenas un elemento de todo el engrane. De acuerdo al propio avance tecnológico han surgido nuevos paradigmas y con ellos nuevas tendencias de desarrollo. Algunos de estos paradigmas son: el Paradigma Orientado a Procedimientos o Estructurado; el Paradigma Orientado a Datos; el Paradigma Orientado a Aspectos, el Paradigma Orientado a Agentes y el Paradigma Orientado a Objetos.

En Cuba, debido a la creciente informatización de la sociedad, han surgido numerosos proyectos en los cuales son puestos en práctica las metodologías antes mencionadas y por consiguiente, las filosofías o paradigmas para el desarrollo de soluciones informáticas.

[1] En el argot informático de los países de habla hispana, se emplea con mayor frecuencia las siglas RUP para referirse al Proceso Unificado de Desarrollo de Software muy a pesar de que su traducción sea precisamente la inglesa, por tal motivo utilizará RUP en el contexto del presente trabajo.

[2] También conocida en la literatura hispana como “Programación Extrema”.

La UCI es un ejemplo de un macro-proyecto nacional donde la versatilidad de metodologías y paradigmas de desarrollo (Ver ANEXO 1) constituye no solo una característica de la institución para la realización de los procesos sustantivos, sino también una necesidad concreta para dar respuesta a solicitudes de clientes tanto nacionales como internacionales. En ella, y desde una perspectiva general, los procesos ingenieriles para el desarrollo de productos informáticos carecen actualmente de mecanismos que permitan abstraer o encapsular conceptos que son ajenos a los requerimientos funcionales de los sistemas, más conocido como lógica de negocio. Estos conceptos no pueden encapsularse dentro de una unidad funcional debido a que atraviesan todo o gran parte del sistema afectando su rendimiento y *performance*, entre los que se pueden encontrar: la sincronización de procesos, validación, manejo de excepciones, procesos de registro en bitácoras, manejo de memoria, distribución, chequeo de errores, seguridad, entre otros. Dichos conceptos pueden ser agrupados en una sola terminología, que viene siendo objeto de investigación hace ya unos años, conocida como *incumbencias*[3].

Muchas veces, a la hora de programar, se encuentran problemas que se pueden solucionar de la mejor forma con las técnicas habituales usadas en la programación por procedimientos o en la programación orientada a objetos, con las que se obliga a los desarrolladores a tomar decisiones de diseño que repercuten de manera importante en el desarrollo de la aplicación y que se alejan con frecuencia de otras posibilidades.

A menudo hace falta escribir líneas de código para definir la lógica de cierta propiedad o comportamiento del sistema, que se encuentran por toda o gran parte de la aplicación, con las consecuentes dificultades de mantenimiento y desarrollo, incurriendo en un problema que se conoce en el contexto del desarrollo de aplicaciones informáticas, como *código disperso* o *código "espaguetis[4]"*. El hecho es que hay ciertas decisiones de diseño que son difíciles de ejecutar,

[3] En la literatura asociada a esta temática, el término de "incumbencias" es manejado desde su propia definición del idioma inglés, como "*concerns*".

[4] Es una manera de tipificar aquellas implementaciones que además de la funcionalidad básica, se encargan de embeber segmentos de códigos para complementar y cumplimentar algún requerimiento funcional específico pero que no tienen que ver directamente con la definición de la lógica de funcionamiento propiamente dicha.

debido a que determinados elementos no se pueden encapsular de igual forma que los que habitualmente se resuelven con funciones u objetos.

El resultado de esta insuficiente abstracción es una notable disminución de la calidad del producto final, reflejado principalmente en el alto grado de acoplamiento, dificultad en la comprensión del código, la baja cohesión, dificultando de este modo el mantenimiento del sistema y su extensibilidad.

Aunque pudiera pensarse que decidir si desarrollar un sistema informático empleando Programación Orientada a Objeto (POO) o realizar la implementación de manera estructurada no determinaría o al menos incidiría en la calidad del producto final, el correcto uso de los recursos, entre otras variables que pudieran tenerse en cuenta, es un análisis correcto, en verdad se estaría cayendo en un grave y posiblemente irreversible error.

Es muy común que en muchos proyectos de desarrollo que se realizan en la UCI no se tenga en cuenta la separación o aislamiento de las incumbencias que entrecruzan toda la lógica de negocio del sistema. Es sumamente importante y en gran medida beneficioso que se logre separar de la implementación de los requerimientos funcionales todos aquellos conceptos que aunque complementan la definición de la lógica de funcionamiento básica, no tienen relación directa en la concepción general del dominio, al menos no desde la perspectiva del cliente.

Supóngase un sistema de gestión empresarial, pudiera ser un ERP (del inglés *Enterprise Resource Planning*), en el que por su propia naturaleza se tenga que realizar en cortos plazos de tiempos un sinnúmero de conexiones a una base de datos. Es evidente que se impone, para un correcto funcionamiento, la validación de este proceso dado que sería caótico el hecho de que no se tengan en cuenta variables como el tiempo de respuesta, la gestión de la concurrencia, la disponibilidad de la conexión, entre otras. Y como es lógico, estos son elementos que no tienen que ver directamente con lo que se pretende resolver, lo cual podría ser, por ejemplo, listar todos los estados financieros de las cuentas contables en determinado centro de costo de la empresa.

Aún más simple: ¿por qué preocuparse de validar el resultado de dividir un número X entre un número Y cuando solo se precisa implementar el modelo matemático Z=X/Y? Claro, ciertamente

es importante tener en cuenta la división entre cero, pero por qué no dejarle la responsabilidad a alguien más que no se encargue de programar la lógica de negocio del sistema en construcción.

Algunos de los paradigmas actuales no soportan una completa separación de conceptos, sino que quedan diseminados por todo el sistema "*enmarañando"* el código que implementa la funcionalidad básica, y yendo en contra de la claridad del mismo, la cual es clave para manejar un sistema informático entendible y de fácil evolución. Se puede afirmar entonces que las técnicas tradicionales no soportan de una manera adecuada la separación de las propiedades de aspectos distintos a la funcionalidad básica, y que esta situación tiene un impacto negativo en la calidad de las aplicaciones informáticas que se desarrollen bajo estas técnicas.

Existen un sinnúmero de metodologías de desarrollo de soluciones informáticas que hacen posible seguir una serie de pasos para llevar a cabo la programación de un sistema informático, sin embargo, ninguna metodología es ni única ni universal, ya que siempre hay que adaptarlas al contexto del proyecto según recursos humanos y técnicos, tiempo de desarrollo, tipo de sistema y tamaño del proyecto[5].

Cada uno de los proyectos de sistemas informáticos que se desarrollan en el mundo usa alguna metodología en particular. Lo real es que los de la UCI siguen este mismo principio; en cada uno de ellos se aplica alguna metodología que dicta *cuándo* hacer *qué* y por *quién;* y adicionalmente, en un significativo número de estos se ha asumido en el desarrollo de soluciones informáticas, la incorporación del paradigma de POA independientemente de la metodología empleada. Sin embargo, ninguna de las metodologías para el desarrollo de soluciones informáticas en la institución garantiza que se apliquen adecuadamente los fundamentos del paradigma antes mencionado.

Un estudio realizado (Ver 1.4) arrojó resultados que indican que las metodologías de desarrollo que actualmente se emplean en la UCI para la construcción de aplicaciones informáticas no garantizan la adecuada incorporación de la POA. Esto no implica necesariamente que estén conceptualmente mal definidas estas metodologías sino que desde su concepción, no incorporan

[5] Entiéndase como una unidad organizacional.

elementos, principios o procedimientos que guíen el proceso de aplicación del paradigma de POA o al menos apoyen la toma de decisiones en este sentido. Lo cierto es que algunas metodologías se especializan en el uso de determinado paradigma de programación, como el orientado a objetos, y como es lógico no tienen en cuenta el empleo de otras filosofías o paradigmas de desarrollo. Sin embargo existe una realidad respecto al caso específico del paradigma de POA de acuerdo con las metodologías de desarrollo tradicional, y es que el no existir pautas o principios para la adecuada incorporación del mismo, complejiza y en algunos casos da al traste, con el desarrollo satisfactorio de la solución final.

Partiendo de esto, el presente trabajo de diploma se encarga de dar solución al siguiente **problema:** *las metodologías para el desarrollo tradicional de aplicaciones informáticas que se emplean en la UCI no garantizan una adecuada aplicación del paradigma de Programación Orientada a Aspectos en la construcción de sistemas informáticos.*

Por consiguiente, el **objeto de estudio** de este trabajo es la Ingeniería de Software, cuyo **campo de acción** es el Proceso Desarrollo de Software.

En consecuencia, el **objetivo general** es proponer una guía de principios de buenas prácticas enfocada a apoyar una adecuada toma de decisiones en la introducción del paradigma de POA en el proceso de desarrollo de aplicaciones informáticas en proyectos productivos de la UCI.

Para alcanzar este objetivo, se dará cumplimiento a las siguientes **tareas de investigación**:

1. Elaboración del marco teórico.
2. Estudio del estado del arte de la POA.
3. Diagnóstico sobre la utilización de la POA en las metodologías de desarrollo de soluciones informáticas en la UCI.
4. Propuesta de una guía de principios de buenas prácticas para la introducción de la POA en el desarrollo de aplicaciones informáticas en la UCI.

El trabajo se encuentra estructurado en tres capítulos de la siguiente manera: en el **primer capítulo** se define el marco teórico de la investigación, así como una fundamentación exhaustiva sobre el tema que anima el desarrollo de la misma. Una particularidad distintiva en este capítulo

es que se desarrolla una evaluación del estado de aplicación de la POA en los proyectos productivos de la UCI, a partir de los resultados de una investigación realizada y que además sirvió para identificar y fundamentar el problema enunciado. En un **segundo capítulo** se presenta un análisis de los fundamentos de la POA mediante el uso del lenguaje de propósito general orientado a aspectos, AspecJ, debido a que dada su condición de lenguaje extendido de Java, su sintaxis y semántica deviene en un excelente componente didáctico para un mejor entendimiento del paradigma. Finalmente, un **tercer capítulo** donde se presenta una guía de principios de buenas prácticas para apoyar la adecuada toma de decisiones en el proceso de integración del paradigma de POA en el desarrollo de aplicaciones informáticas, enfocada en la definición de un conjunto de principios, análisis de factibilidad de uso y observaciones generales sobre la pertinencia de incorporar la POA al desarrollo tradicional. Dentro de los aportes particulares que se realizan en este capítulo, se encuentra la propuesta de un modelo matemático para la evaluación cuantitativa, desde un enfoque comparativo, del favorecimiento de la productividad a partir de una adecuada aplicación del paradigma de orientación a aspectos. Además de que el propio modelo lleva acompañado una fundamentación que tributa a la evaluación cualitativa del mejoramiento de la productividad en los equipos de desarrollo que implementan adecuadamente los principios propuestos.

CAPÍTULO 1

Capítulo 1: FUNDAMENTACIÓN TEÓRICA

INTRODUCCIÓN

En el desarrollo de aplicaciones informáticas intervienen una serie de etapas en las que se realizan un conjunto de actividades en función de obtener un resultado concreto. En este sentido, la Ingeniería de Software fue evolucionando a medida en que se fueron introduciendo conceptos que enriquecieron la concepción general de esta ciencia en tanto permitió que el desarrollo de sistemas no solo fuese más coherente, sino que se alcanzara un alto nivel en el análisis sobre cómo construirlos.

Los progresos más importantes se han podido percibir gracias a la aplicación de tres principios fundamentales: abstracción, encapsulamiento y modularidad; los que presentan una estrecha interrelación inherente a sus propias definiciones. Sin embargo, no fue hasta la introducción del paradigma de POO en donde estos tres principios adquirían una verdadera connotación en función de lograr coherencia en el desarrollo de sistemas informáticos y calidad en los productos resultantes de cada etapa.

Evidentemente los principios de abstracción, encapsulamiento y modularidad, fueron en sus inicios un complemento a lo que ya existía. Gradualmente, y durante la propia evolución de los procesos ingenieriles, se introdujeron conceptos que llevaron a una programación de más alto nivel, tales como la noción de tipos, bloques estructurados, agrupamiento de instrucciones a través de procedimientos y funciones como una forma primitiva de abstracción, módulos, tipos abstractos de datos, entre otros.

Indiscutiblemente un avance importante lo introdujo la POO al forzar el encapsulamiento y la abstracción, mediante una unidad modular que captura tanto funcionalidad o comportamiento, estado e identidad: el *objeto*[6].

Durante décadas bastó con la introducción de la POO para garantizar altos niveles de abstracción, encapsulamiento y modularidad en la modelación de las funcionalidades y conceptos básicos en la construcción de un sistema informático. Sin embargo existen otros conceptos que no tienen que ver directamente con esas funcionalidades básicas o lógica de negocio, como comúnmente se le suele llamar, de los sistemas informáticos. Conceptos que entrecruzan horizontal o transversalmente la concepción general de esos sistemas que no pueden ser encapsulados modularmente mediante las técnicas de la POO. A esos conceptos se les conocen como *incumbencias* o *concerns* (en el idioma inglés) y que hasta el momento pueden ser separados únicamente a través de la introducción de los principios de la POA.

Precisamente en este capítulo se estará haciendo una revisión sobre esos conceptos que atraviesan la lógica de funcionamiento básico de una aplicación informática, así como un estudio de los fundamentos del paradigma orientado a aspectos, además de un análisis de los resultados de una investigación de campo, asociada a la aplicación del paradigma de orientación a aspectos en los proyectos productivos de la UCI.

[6] Es la unidad básica en la POO que presenta estados, comportamiento e identidad y son modelados mediante la definición e implementación de clases.

1.1. LAS INCUMBENCIAS EN EL DESARROLLO DE APLICACIONES INFORMÁTICAS.

¿Cuán cercanas están las actuales metodologías de desarrollo de sistemas informáticos a la realidad de una empresa o proyecto, e incluso de un desarrollador independiente, para obtener un producto que responda a las exigencias y/o necesidades del cliente, en un corto plazo de tiempo, con una alta calidad y de forma eficiente? La respuesta a esta interrogante es razón de polémicas discusiones en el mundo.

Las *incumbencias* aparecen como un elemento crítico en el proceso de desarrollo de un sistema informático dado a que estas tipifican un conjunto de conceptos que se encuentran diseminados por toda la lógica de funcionamiento del sistema. Retomando el ejemplo de la introducción del presente trabajo, el caso del sistema ERP y la realización sistemática de conexiones a una base de datos, se podría identificar la "*validación de conexiones a bases de datos"* como una incumbencia que debe ser encapsulada modularmente con el fin de lograr una verdadera separación entre aquellos conceptos que pertenecen al dominio de funcionalidades básicas del sistema y los que por su naturaleza conceptual no, o sea, las incumbencias.

A partir de este análisis es preciso tipificar las incumbencias de acuerdo a su naturaleza e identificar su nivel de tratamiento a partir de su alcance.

1.1.1 IDENTIFICACIÓN DE INCUMBENCIAS.

Las incumbencias pueden identificarse desde etapas tempranas del proceso de desarrollo o en su defecto, comenzado el proceso de diseño e implementación del sistema. Lo más importante es que se conozca que sea cual sea la incumbencia, siempre responderá a requerimientos funcionales o no funcionales del sistema, directa o indirectamente, ya que aunque alguna esté asociada a necesidades concretas del equipo de desarrollo en particular, el interés por identificarla, clasificarla y tratarla siempre estará en correspondencia con los legítimos intereses de los clientes.

La relación que se establece entre los requerimientos funcionales y los no funcionales respecto a las incumbencias, evidentemente no es una relación inyectiva sino que se basa en el

cumplimiento de los objetivos fundamentales de la organización. Es decir, una incumbencia no tiene por qué responder necesariamente a un requerimiento del sistema ni un requerimiento del sistema tiene por qué dar lugar a una incumbencia. De modo que la identificación de *concerns* debe realizarse desde la óptica del equipo de desarrollo pero en función del cumplimiento de los requisitos del sistema establecidos por ambas partes, clientes y desarrolladores.

¿Cuándo se está en presencia de una incumbencia?, pues cuando se ha identificado un concepto atómico que atraviesa parte o la totalidad de la lógica funcional del sistema y que su aislamiento, desde el punto de vista analítico primeramente y luego de diseño e implementación, no afecte o provoque traumas a la dinámica de funcionamiento del sistema y el correcto cumplimiento de los requerimientos funcionales y no funcionales. Básicamente, cuando se requiere introducir funcionalidades adicionales o complementarias para hacer cumplir estos requerimientos.

Lógicamente, el aislamiento al que anteriormente se hacía referencia es relativo al paradigma de programación que se emplee. La identificación de incumbencias, propiamente dicho, no tiene por qué estar asociada a un paradigma u otro, pero el aislamiento de estas será total o parcial, en dependencia de si se usa o no, respectivamente, un paradigma orientado a aspectos. Para el caso específico de la POO, por citar uno de los paradigmas de programación más usados, el aislamiento o separación de incumbencias será puramente conceptual, ya que dicho paradigma no cuenta con las herramientas necesarias para desligar, desde el punto de vista programático, las incumbencias o *concerns,* de la implementación de la lógica funcional básica del sistema. Por lo que muy a pesar del uso de buenas prácticas, como la utilización de patrones de diseño, no podrían evitarse las dependencias entre componentes de sistemas, y por consiguiente, el aislamiento de las incumbencias sería parcial.

La identificación de las incumbencias permite establecer pautas de desarrollos más específicas y centradas en la productividad de los equipos, dado a que se logrará un mejor entendimiento de lo que se debe hacer en función del cumplimiento de los requerimientos del sistema. Esto evidentemente será más visible para aquellas organizaciones que asuman la introducción de la POA como paradigma para aislar o separar las incumbencias identificadas, dando lugar a la

posibilidad de dividir los equipos de desarrollo en desarrolladores de la lógica de negocio y desarrolladores de *aspectos* (Ver 3.3.1).

1.1.2 DESCOMPOSICIÓN ASPECTUAL DE UN SISTEMA.

Cuando se realiza el análisis para la identificación de las incumbencias se está haciendo básicamente el primer paso para la descomposición aspectual del sistema en construcción. De modo que se puede establecer que una incumbencia es un *aspecto* desde el punto de vista de la descomposición aspectual.

Como se exponía anteriormente, en el caso de los paradigmas tradicionales para el desarrollo de aplicaciones informáticas, la identificación de incumbencias era una labor que no traía grandes[7] frutos dado a que estos paradigmas no cuentan con las herramientas necesarias para aislar cada uno de los aspectos identificados. De modo que un aspecto no puede ser encapsulado en un procedimiento o función implementada en algún lenguaje de estos paradigmas tradicionales ya que estos no suelen ser unidades de descomposición funcional del sistema, sino propiedades que afectan al rendimiento o la semántica de los componentes (REINA QUINTERO, 2000).

Un importante elemento dentro del proceso de descomposición aspectual de un sistema, es el lenguaje de modelado que se emplee para representar gráficamente esta descomposición, y que sea perfectamente entendible para aquellos que tendrán la responsabilidad de implementarla. Por lo tanto, es preciso contar con un lenguaje de modelado orientado a aspectos que brinde las herramientas necesarias para representar en un entorno gráfico las relaciones que se establecerán entre bloques o unidades funcionales y los aspectos. En este sentido, a la hora de modelar un sistema, se deberá tener en cuenta que el mismo se divide en componentes y aspectos. Los componentes pueden encapsularse dentro de una unidad modular funcional y los aspectos, por el contrario, no podrán ser encapsulados en unidades funcionales dado a que

[7] Desde el punto de vista de la magnitud alcanzada en los niveles de desacoplamiento mediante la separación de incumbencias aplicando técnicas de POA.

constituyen propiedades que afectan el *performance*[8] o la semántica de los componentes y que se encuentran diseminados por toda o parte de la lógica funcional del sistema, lo que implica la utilización de arquetipos de modelado capaces de representar una nueva abstracción.

1.1.2.1 DESCOMPOSICIÓN ASPECTUAL MEDIANTE LA POO.

Por su parte, la descomposición aspectual en la POO no va más allá de la identificación de las incumbencias y de la dispersión embebida de las mismas por la implementación de toda la funcionalidad básica del sistema. En otras palabras, no hay tal *separación de incumbencias.*

En el intento de aislar los conceptos que entrecruzan la lógica de un sistema, se introducen nuevos componentes, como clases, que tratan de encapsular o más bien de gestionar las incumbencias. Sin embargo esto no garantiza una completa separación ya que de alguna manera se producirían nuevas dependencias. Por ejemplo: ¿de qué manera mediante la POO podría separarse la validación de las conexiones a bases de datos en el caso del sistema ERP para el reporte de los estados financieros de las cuentas contables? La solución no iría más allá de análisis como estos:

- Por una parte, desde la óptica de la programación defensiva, se trataría de validar la conexión a través de bloques condicionales antes de establecer la carga de la información.
- Otro análisis podría estar enfocado en el empleo de bloques para el tratamiento de excepciones, en función de garantizar un adecuado funcionamiento del sistema en caso de que la conexión colapse.
- Quizás otro enfoque sea la implementación del patrón *Observer*[9] para que de alguna manera se pudiese encapsular la acción de validar el evento de conexión.

[8] En este contexto se refiere a los tiempos de respuestas y empleo eficiente de los recursos de un sistema. Es un préstamo lingüístico que ha permitido a la comunidad hispana para referirse a estos elementos de funcionamiento y rendimiento.

[9] Patrón de diseño “Observador”, en el idioma español.

En definitiva, sea cual sea el análisis, se caería en la misma situación: tener que implementar las incumbencias embebidas en el código, o en el mejor de los casos, introducir nuevos componentes que por medio de dependencias garantizaran el cumplimiento del objetivo principal, lo que traería como consecuencia continuar con la problemática del acoplamiento entre elementos del sistema.

En conclusión, y como bien plantea la bibliografía consultada, los paradigmas tradicionales, como la POO, carecen de la instrumentación sintáctico-semántica capaz de lograr una verdadera separación de incumbencias.

1.2. LA PROGRAMACIÓN ORIENTADA A ASPECTOS. UNA NUEVA FILOSOFÍA.

La POA ha revolucionado en gran medida la forma de desarrollar aplicaciones informáticas. Al igual que otros tantos paradigmas, marca una manera diferente de ver el proceso de construcción de un sistema. Deviene además en una excelente opción cuando se desea ganar en flexibilidad y facilidad de mantenimiento en los diseños de sistemas. De modo que para muchos constituye una mejor vía de obtener un producto informático, en tanto para otros es solo un paradigma o técnica más de programación. Lo cierto es que es una útil herramienta y que, como es lógico, la decisión de emplearla o no está en función del análisis de otros elementos que permitan evaluar los beneficios de su introducción.

1.2.1 FUNDAMENTOS DE LA PROGRAMACIÓN ORIENTADA A ASPECTOS.

Lo más interesante de los paradigmas de programación durante estas últimas décadas ha sido su versatilidad. Unos se encargan de “entender” los sistemas como un conjunto de funcionalidades, rutinas o procedimientos que interactúan entre sí; otros como los diferentes tipos de relaciones que se establecen entre objetos de un dominio; o un último caso pero no menos interesante, es la concepción de separar de esas funcionalidades, rutinas, procedimientos u objetos, aspectos que se encuentran diseminados como complemento general, pero que en definitiva hacen más engorroso, inentendible y muy difícil de mantener las mismas.

Evidentemente no puede hablarse de paradigma de programación sin hacer referencia a un lenguaje base que lo implemente y para el caso muy específico de la POA, existen lenguajes que se encargan de implementar los conceptos que atraviesan la lógica básica del sistema. Cuando

se refiere a lenguajes orientados a aspectos se habla de aquellos lenguajes que definen una nueva unidad de programación para encapsular las propiedades que entrecruzan todo el código, denominada convenientemente *aspecto*. Estos conceptos no son totalmente independientes, y es evidente que existe una relación entre los componentes y los aspectos, lo que trae consigo que el código de los componentes y de estas nuevas unidades de programación interactúe de alguna manera. Una condición necesaria para que los aspectos y los componentes se puedan mezclar, es que deben tener algunos puntos comunes a los que se le llama *puntos de enlace* (***joinpoints***).

1.2.2 RESEÑA HISTÓRICA.

Como la mayoría de los paradigmas y lenguajes de programación, la Programación Orientada a Aspectos se basa en conceptos preexistentes en otros paradigmas de desarrollo. Aún antes de que surgiera este concepto, el equipo de investigación Demeter, había considerado la aplicación de ideas similares y con el mismo objetivo. El concepto principal en el resultado del grupo Demeter es la Programación Adaptativa (PA) (DEITERS, 2005), en el año 1991, la cual constituyó un gran avance para el proceso de desarrollo de soluciones informáticas. Esta se basaba en el uso de autómatas finitos y una teoría formal de lenguaje para expresar concisamente y procesar eficientemente conjuntos de caminos en un grafo arquitectónico, como pueden ser los diagramas de clases en el lenguaje de modelado UML.

La relación entre la POA y la PA surge de la ley de Demeter: "*(…) solo conversa con tus amigos inmediatos (…)"*. Esta ley formulada en el 1987 en la Northeastern University y popularizada en los libros de Booch, Budd, Coleman, Larman, Page-Jones, Rumbaugh, entre otros, es una simple regla de estilo en el diseño orientado a objetos. Por lo tanto, la separación completa de conceptos fue área de interés de este grupo de investigación antes de que la POA existiera como tal.

En 1995 dos miembros de este grupo, Cristina López, actualmente integrante del grupo Xerox PARC, y Walter Hersch, presentaron un reporte técnico sobre separación de conceptos incluyendo varias técnicas como la de filtros composicionales y la de PA para tratar con los conceptos entrecruzados. Este reporte identificó el tema general de separación de incumbencias (cross-cutting concerns) y su implementación, y lo propuso como uno de los ***problemas a resolver más importantes en el diseño y desarrollo de sistemas informáticos***.

La definición formal de PA puede considerarse como una definición inicial y general de la POA, dado a que en ella se planteaba que los programas se descomponían en varios bloques de cortes (REINA QUINTERO, 2000).

La primera definición del concepto de aspecto fue publicada por el grupo Demeter en 1995 y planteaba que *un aspecto es una unidad que se define en términos de información parcial de otras unidades.* Se comenzaba a establecer los principios generales del paradigma de orientación a aspectos, donde precisamente la unidad básica de información eran los aspectos, como son los objetos en el paradigma de POO.

Muy a pesar de los más de quince años de "antigüedad", la POA es aún un paradigma de desarrollo en pleno nacimiento. Muy a pesar de la estabilidad de sus principios y conceptos y de la variedad de lenguajes y plataformas que lo soportan, aún existen posiciones escépticas respecto a la factibilidad de su empleo e integración con otros paradigmas; probablemente por el desconocimiento de esta "nueva" filosofía o por el mero efecto del rechazo al cambio. Lo cierto es que existe y se ha aplicado de manera satisfactoria. Ya son varias las experiencias que existen en el mundo en cuanto al uso de este paradigma y con resultados positivos.

1.2.3 ANATOMÍA DEL PARADIGMA ORIENTADO A ASPECTOS.

La interacción que existe entre componentes y aspectos no es la misma que ocurre entre los módulos del lenguaje base, donde la comunicación está basada en declaraciones de tipo y llamadas a procedimientos y funciones. Analizando esta situación se puede decir que: *Los puntos de enlace (join points) son una clase especial de interfaz entre los aspectos y los módulos del lenguaje de componentes. Son espacios dentro de la ejecución del código donde es posible agregar el comportamiento adicional que destaca a la POA, el cual se especifica en los aspectos* (GREGOR, y otros, 1997).

El encargado principal en el proceso enlace entre los aspectos y los componentes se le conoce como *tejedor de aspectos* (*aspect weaver*); el cual realiza la parte final y de mayor importancia: mezclar los diferentes mecanismos de abstracción y composición que aparecen tanto en los lenguajes de aspectos como en los lenguajes de componentes, auxiliándose en los puntos de enlace. El tejedor, además de los aspectos, los módulos y los puntos de enlace, recibe unas

reglas que le indican cómo debe realizar el tejido. Estas reglas son los llamados *puntos de corte* (*pointcuts*), que le especifican al tejedor *qué aspecto tiene que enlazar a qué módulo a través de qué punto de enlace* (GREGOR, y otros, 1997).

Tradicionalmente el proceso de generar un programa ha consistido en pasar el código a través de un compilador o un intérprete que lo traduzca de un lenguaje de alto nivel a un código directamente entendible por la máquina, para así disponer de un ejecutable. Sin embargo en las aplicaciones orientadas a aspectos no se tiene un único código del programa, sino uno que se encarga de implementar la funcionalidad básica del sistema y otro para la implementación de los aspectos. Ambos se encuentran físicamente separados.

Todo este código, tanto el de componentes como el de aspectos, debe pasar no sólo a través del compilador, sino que debe ser tratado por el tejedor, que es el que se encarga de crear un único programa con toda la funcionalidad básica y los distintos módulos que implementan los aspectos, como bien se mencionaba anteriormente, donde además cada aspecto se pueden codificar con un lenguaje distinto. Se puede decir entonces que en las aplicaciones tradicionales los bloques de funcionalidad quedan repartidos por todo el código, lo cual no ocurre de igual manera en las aplicaciones orientadas a aspectos, ya que se cuenta con un programa modular, con un bajo acoplamiento y altamente cohesivo.

Concretamente, la estructura de un programa orientado a aspectos está compuesta por los siguientes tres elementos:

- Un ***lenguaje base***, el cual, se encarga de definir la funcionalidad básica. Suele ser un lenguaje de propósito general, como C++ o Java, aunque también se podrían utilizar lenguajes no imperativos.
- Uno o varios ***lenguajes de aspectos,*** los que definen la sintaxis de los aspectos que encapsulan las incumbencias identificadas.
- Un ***tejedor de aspectos***, el que se encargará de combinar los lenguajes y en donde el proceso de mezcla se puede hacer en tiempo de ejecución o en tiempo de compilación, lo que se conoce como tejido dinámico o tejido estático, respectivamente.

1.2.4 EL TEJEDOR DE ASPECTOS Y LOS TIPOS DE TEJIDOS.

El tejido de aspectos es el proceso que tiene el objetivo de integrar o mezclar el código funcional con el código de los aspectos generando código ejecutable y es llevado a cabo precisamente por el *tejedor*. Para realizar esto utiliza técnicas de transformación de programas. Es necesario entonces saber cuándo o en qué momento se produce el proceso de tejido, lo que constituye una de las principales diferencias entre los lenguajes orientados a aspectos.

Si el tejido ocurre en tiempo de compilación, es usualmente denominado *tejido estático* y si el tejido ocurre durante la ejecución, se lo conoce como *tejido dinámico*. También se da el caso del tejido en tiempo de carga, el que consiste en aplicar técnicas basadas en la transformación de código a nivel binario.

1.2.4.1 TEJIDO ESTÁTICO.

Actualmente existen una gran cantidad de implementaciones de POA que se encuentran basadas en el tejido estático, el cual consiste en la modificación en tiempo de compilación del código fuente, insertando llamadas a las rutinas específicas de los aspectos. Primeramente se parte del programa con la funcionalidad básica implementada en un lenguaje base (C++, Java, u otros) y además se tiene uno o varios aspectos escritos en un *lenguaje de aspectos* que puede ser una extensión de algún lenguaje “normal”, o uno específicamente definido con este fin.

Se entiende por aspecto, desde el punto de vista declarativo, al código que se debe añadir (*advice*) y las instrucciones que recibe el tejedor sobre dónde y cómo (*pointcut*) insertar el código de los aspectos en el código principal. Se puede dar el caso de que estas instrucciones vengan expresadas en el mismo lenguaje o en otro particular, además se pueden encontrar en el mismo fichero o en otro. La composición del código es realizada por el tejedor de la siguiente forma: se inserta código proveniente de los aspectos en el de la funcionalidad básica, siguiendo las instrucciones recibidas; y como resultado se genera un nuevo código fuente, el cual pasará a través de un compilador, para generar el programa ejecutable (VINUESA, 2007).

Cuando se usa un tejedor estático se debe modificar el código fuente escrito en el lenguaje base, insertando sentencias en los puntos de enlace; de manera que el código de aspectos se introduce en el código fuente. Se tiene como resultado entonces que el programa final se genere

tejiendo el código de la funcionalidad básica y el de los aspectos seleccionados en la fase de compilación.

1.2.4.2 TEJIDO DINÁMICO.

No todas las aplicaciones necesitan ser adaptadas mediante aspectos en tiempo de ejecución. Existen aspectos específicos que obtienen beneficios de un sistema con tejido dinámico; en este sentido suele darse el caso de aplicaciones que necesitan adaptar sus competencias específicas en respuesta a cambios en el entorno de ejecución, como la gestión de recursos y el equilibrio de carga en sistemas distribuidos, entre otros (VINUESA, 2007).

Un programa que funcione con tejido dinámico, requiere que los aspectos existan y estén presentes de forma explícita tanto en tiempo de compilación como en tiempo de ejecución. Para conseguir esto, tanto los aspectos como las estructuras entrelazadas se deben modelar como objetos y deben mantenerse en el ejecutable. Primeramente, se cuenta con un programa escrito en un lenguaje normal, el que utiliza el compilador para generar un programa que se pone en ejecución y que no tiene por qué saber que va a ser adaptado. Cuando se necesita adaptar, añadir o modificar, es cuando surgen nuevos requerimientos o en respuesta a cambios en el entorno. Luego sin detenerlo se procesa el programa en ejecución, aquel que se encuentra en ese momento en memoria, junto a los aspectos que van a adaptarlo, y por medio del tejedor se completa la funcionalidad de estos, dando lugar a una nueva versión del programa que continúa la ejecución. El programa ejecutable inicial no ha sido modificado, con lo que se puede utilizar de nuevo sin las modificaciones realizadas en memoria. (VINUESA, 2007)

En sistemas que usan tejido dinámico de aspectos, la funcionalidad básica permanece separada de los aspectos en todo el ciclo de vida, incluso en la ejecución del sistema. El código resultante es más adaptable y reutilizable, y los aspectos y la funcionalidad básica pueden evolucionar de forma independiente.

1.2.4.3 ¿TEJIDO ESTÁTICO vs TEJIDO DINÁMICO?

El tejido estático resulta más fácil de implementar y consume menor cantidad de recursos, además se evita que el nivel de abstracción introducido por la POA derive de manera negativa en la eficiencia de la aplicación, esto se debe a que no existe sobrecarga en ejecución, debido a que

todo el trabajo se realiza en tiempo de compilación, hay menos flexibilidad, y los aspectos no pueden ser modificados dado a que se encuentran en tiempo de ejecución, de manera que quedan fijos. Además no existe la posibilidad de agregar o remover nuevos aspectos, de esta forma se llega a una mayor seguridad, que se obtiene efectuando controles en compilación, evitando así que surjan errores catastróficos o fatales en la ejecución.

En tanto en el tejido dinámico el proceso de composición se realiza en tiempo de ejecución, de forma que disminuye la eficiencia de la aplicación. Existe una mayor flexibilidad, se le da al programador la libertad de modificar un aspecto según información generada en ejecución, además se pueden introducir o remover dinámicamente aspectos. Sin embargo se pone en riesgo la seguridad de la aplicación producto a que se puede eliminar dinámicamente un comportamiento de un aspecto que quizás luego se quiera usar, o aun más grave, se puede eliminar de forma total el aspecto. La implementación de los tejedores dinámicos es más compleja, debido a que lleva una mayor información en ejecución y considera más detalles.

1.2.5 EL LENGUAJE BASE.

Diseñar un sistema basado en aspectos requiere entender qué se debe incluir en el lenguaje base, qué se debe incluir dentro de los lenguajes de aspectos y qué debe compartirse entre ambos lenguajes. El lenguaje componente debe proveer la forma de implementar la funcionalidad básica y asegurar que los programas escritos en ese lenguaje componente no interfieran con los aspectos. Los lenguajes de aspectos tienen que proveer los medios para implementar los aspectos deseados de una manera intuitiva, natural y concisa (GREGOR, y otros, 1997).

En general, el desarrollo de una aplicación basada en aspectos consiste de tres pasos según (LADDAD, 2002):

1. Descomposición de aspectos: es clasificar los requerimientos para distinguir aquellos que son componentes de los que son aspectos.
2. Implementación de requerimientos: implementar cada requerimiento por separado.
3. Recomposición: dar las reglas de recomposición que permitan combinar el sistema completo (entrelazado).

Básicamente, el rol fundamental del lenguaje base es implementar toda la lógica de negocio propiamente dicha, desligada y sin ninguna responsabilidad de las incumbencias o aspectos que

hayan sido identificados en el proceso de desarrollo del sistema. Partiendo de esto, y sobre la base de los pasos anteriormente expuestos, es preciso señalar, que el rol del lenguaje base tiene un alcance bien establecido en el paso dos dado a que la descomposición aspectual es un proceso previo al diseño e implementación de los requerimientos del sistema, y de manera análoga, la recomposición o entrelazado es un paso que le corresponden al lenguaje de aspectos realizarlo.

1.3 ALGUNAS EXPERIENCIAS DE APLICACIÓN DE LA POA EN EL MUNDO.

Grandes compañías de desarrollo de sistemas informáticos en la actualidad, como la IBM (del inglés, *International Business Machines*) y la Microsoft, han introducido en su desarrollo el uso de la POA. Daniel Sabbah, vicepresidente del Grupo de Estrategia y Desarrollo de Software de la IBM, afirma que "*esta tecnología es lo suficientemente confiable para su uso comercial, de ahí que emplean este paradigma en el desarrollo de soluciones informáticas*" (IBM, 2000). Según plantea Robert Berry, ingeniero de la IBM, "*al introducir la POA en sus proyectos, la compañía ha obtenido grandes beneficios en la calidad del código y la velocidad con que los programadores escriben el mismo*".

La IBM ha desarrollado varios lenguajes, herramientas, aplicaciones, proxies y frameworks para la utilización de la POA en el desarrollo de soluciones informáticas**.** Entre los que se encuentran los lenguajes: *AspectJ, Aspect, PhpAspect, Aspyct, AspectC++, AspectS, Caesar, AspectWerkz, AspectScript, AspectMaps*. Además de un proxy basado en la POA llamado *Dynaop*; el framework *Aspect Nanning*, y el módulo del framework *Spring* (*Java*) para POA. También el servidor de aplicaciones J2EE (del inglés*, Java 2 Platform Enterprise Edition*) de código abierto implementado en Java, totalmente orientado a aspecto; Jboss AOP, entre otros.

Por su parte, la Microsoft ha desarrollado una versión del módulo del framework *Spring* en la plataforma .NET, el *Spring.NET*, muy usado actualmente, además del framework *SetPoint*. También los tejedores[10] *Loom.Net*, *Weave.Net* y el tejedor de aspectos multilenguaje *AspectDNG*; así como los lenguajes *Eos*, *PostSharp*, entre otros.

[10] Herramientas para la integración del código de aspectos con el código del lenguaje base.

Este desarrollo de sistemas informáticos que ha surgido usando la POA no se remite solamente a las grandes compañías de países como EE.UU., existen países en América que se destacan también en este ámbito, entre ellos se encuentra Argentina, en donde la introducción de la POA en soluciones informáticas se ha hecho inminente en los últimos años. Una de estas empresas es Hexacta, la que realiza un gran desarrollo y mantenimiento de aplicaciones en Java y .Net; donde se utilizan elementos del *DSOA*, en sus soluciones informáticas (HEXACTA, 2000).

Existen otras experiencias en el uso de la POA, entre las que se puede encontrar, el trabajo del laboratorio PLEIAD (*Programming Languages and Environments for Intelligent, Adaptable and Distributed Systems*). Este laboratorio está dedicado a la investigación de cómo los lenguajes de programación y sus ambientes de desarrollo, contribuyen a la construcción de un sistema informático de fácil evolución y adaptación. Su investigación abarca tres áreas principales: la POA, el debugging y comprensión de programas, y la programación de sistemas de computación previsiva (PLEIAD, 2004). Además se realizan investigaciones sobre la definición de lenguajes de la POA y la construcción de los mismos. PLEIAD está involucrado como miembro fundador de la Red Latinoamericana de Desarrollo de Software Orientado a Aspectos (*Latin AOSD)*.

En el marco de la *Latin AOSD*, se efectuó el Primer Taller Latinoamericano sobre AOSD, el que tuvo como objetivo principal reunir a la comunidad de investigadores de América Latina interesados en temas relacionados con este paradigma; y de esta forma compartir experiencias y discutir ideas, fortalecer la colaboración del grupo de investigación e identificar nuevas oportunidades de investigación. Este taller se dividió en tres exitosas ediciones del Taller Brasileño de AOSD que reunió a investigadores de todas partes de Brasil y de América Latina. Con la realización de estos talleres se logra propiciar el crecimiento de las investigaciones que se realizan usando paradigma orientado a aspectos en esta comunidad. Además de estos encuentros, se desarrolla cada año la Conferencia Internacional de AOSD, la que en el presente año se efectúo del 15 al 19 de Marzo en Rennes y Saint Malo, Francia.
Diversas universidades de América Latina también colaboran en el desarrollo de la POA, varios profesores que trabajan en ellas conforman esta red latinoamericana existente. Se pueden mencionar entre algunas de las más destacadas en la práctica de la POA a la Universidad Federal de Bahía, la Universidad de Pernambuco, Universidad Federal do Río Grande do Norte,

Pontificia Universidad Católica de Río de Janeiro, en Brasil; además de la Universidad de Chile; Universidad de los Andes, en Colombia; Universidad de Buenos Aires, en Argentina; entre otras.

En Europa también se pueden encontrar grandes compañías que se dedican al desarrollo de sistemas informáticos, algunas de las cuales se encuentran entre las más importantes en estos momentos en la industria. Las nuevas perspectivas que brinda la POA se evidencian en empresas como Sage (SAGE), la cual se encuentra entre las diez empresas más importantes en Europa actualmente según (TOP 100 RESEARCH FUNDATION), esta empresa ofrece soluciones de gestión empresarial (*ERP*), usando metodología Java. Otra empresa importante que ha tenido experiencias significativas en el uso de la POA en sus aplicaciones es UNIT4 Ibérica[11], uno de los primeros fabricantes internacionales de sistemas de gestión empresarial (*ERP y otros*).

Al igual que en América, en Europa existen también varias experiencias de trabajo con el uso de la POA. Numerosas universidades aplican este paradigma, ya sea para la enseñanza del mismo como para su utilización en la realización de soluciones informáticas que desarrollan. En estas universidades trabajan varios profesores de renombre internacional que son miembros de asociaciones de DSOA en Europa. Se puede mencionar a la Universidad de Málaga, la Universidad Nova de Lisboa, la Universidad de Extremadura, Universidad de do Minho, Universidad de Oviedo, Universidad Politécnica de Valencia, Universidad de Lancaster, Universidad de Potsdam y Duisburg-Essen, de Alemania.

Como se evidencia, el uso de la POA no queda simplemente en la teoría. En la práctica las experiencias del trabajo con la POA han sido y seguirán siendo muchas. La POA no compite con los paradigmas ya existentes, sino que se introduce en ellos complementando de esta manera el desarrollo de sistemas. De ahí proviene su importancia y aplicabilidad en grandes empresas desarrolladoras que lideran el mercado actualmente, así como otras instituciones que introducen este paradigma en su desarrollo de soluciones informáticas.

[11] Filial española del grupo multinacional de origen holandés del sector de las TIC, que ofrece soluciones ERP para la gestión global de las empresas y servicios asociados.

1.4 ANÁLISIS SOBRE EL EMPLEO Y CONOCIMIENTO DE LA POA EN LA UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

En este epígrafe, como bien lo sugiere el título, se estará realizando un análisis sobre el estado de aplicación del paradigma de programación orientada a aspectos en el desarrollo de sistemas informáticos, así como el nivel de conocimientos de este paradigma por miembros de proyectos con importantes roles de trabajo, tales como líderes, arquitectos, diseñadores principales, integradores de módulos, entre otros.

Habiéndose expuesto algunos elementos conceptuales sobre qué era la POA y en cierta medida cuál era su funcionamiento general, además de realizado una breve revisión sobre algunas experiencias en el empleo del paradigma de programación orientada a aspectos en el desarrollo de sistemas, se está en condiciones de centrar la atención en cómo se aplica el paradigma en el marco del desarrollo de aplicaciones informáticas en la UCI, y cuál es aproximadamente el estado de conocimiento de los principales roles de trabajo en el proceso de desarrollo de algunos proyectos, sobre dicho paradigma.

1.4.1 APLICACIÓN DE UN MÉTODO DE RECOLECCIÓN DE DATOS.

Para la recopilación de datos en el proceso investigativo se emplearon dos métodos científicos particulares (HERNÁNDEZ LEÓN, y otros, 2002), que sirvieron como soporte para la realización del análisis del estado de aplicación y conocimiento de la POA en la UCI.

1.4.1.1 REALIZACIÓN DE UN CUESTIONARIO.

El cuestionario pretende identificar el empleo o no del paradigma de POA en una muestra de proyectos productivos de la UCI. El mismo está realizado desde el anonimato para resguardar la identidad de los encuestados.

El cuestionario está dirigido a miembros de proyectos productivos de la UCI que cumplan roles que intervengan en la toma de decisiones para adoptar una u otra metodología de desarrollo en la construcción de un sistema, así como la introducción o no de técnicas o paradigmas de programación para elevar la calidad del producto final y realizar eficientemente la gestión de los recursos de la organización. Asimismo se pretende incluir explícitamente los roles de Programador y Diseñador para que brinden sus experiencias sobre el proceso de diseño e

implementación en el contexto de las metodologías y los paradigmas definidos por la gerencia del proyecto. Algunos de estos roles son:

- Gerentes de Proyectos
- Jefe de Proyecto
- Arquitecto
- Jefe de Módulo
- Programador
- Diseñador

El objetivo general del mismo es *identificar la importancia de introducir el paradigma de Programación Orientado a Aspectos en el desarrollo de sistemas informáticos*. Y más específicamente:

1. Identificar si se emplea o no la POA en los proyectos productivos de la UCI.
2. Identificar el nivel de conocimiento que tienen los encuestados sobre el paradigma de POA.
3. Identificar buenas prácticas en la introducción de la POA en el desarrollo tradicional de sistemas.
4. Identificar las metodologías empleadas en los proyectos encuestados y la forma en que son implementadas.
5. Evaluar la importancia de implementar una guía de principios de buenas prácticas orientada a apoyar el proceso de toma de decisiones en el desarrollo de aplicaciones informáticas.

La muestra fue seleccionada mediante un método de *muestreo intencional* ya que la experiencia demuestra que sin tener información previa para emitir juicios, esta técnica no ofrece resultados confiables (HERNÁNDEZ LEÓN, y otros, 2002) y para este caso se apoyó la decisión en los datos asociados a los polos de producción. Esta muestra comprende todos los proyectos de la facultad 15, el proyecto ERP y un proyecto de cada facultad seleccionado de acuerdo al alcance del mismo, impacto económico-social, tamaño e importancia a partir de los legítimos intereses de la institución.

La razón por la cual se seleccionan todos los proyectos de la facultad 15, es porque de acuerdo al perfil de la misma y el papel que juega en el desarrollo de sistemas de gestión, además de la magnitud poblacional de proyectos productivos debido al proceso de fusión de las facultades 3 y 4 para conformar la actual facultad 15, esta representa un 7% de todos los proyectos de la UCI que hasta el momento se han definido y se encuentran en alguna etapa de licitación, desarrollo, despliegue o soporte.

Se selecciona además el proyecto ERP por ser un macro-proyecto de gestión empresarial, con un desarrollo avanzado y resultados concretos en la obtención del producto final. Además, porque son precisamente los sistemas de gestión, los más proclives a introducir variedades de técnicas o paradigmas de programación y metodologías de desarrollo, para dar cumplimiento a los objetivos fundamentales de la organización.

Finalmente, seleccionar un proyecto de cada facultad complementa la muestra en un 17 % de la población total de proyectos de la UCI hasta el momento en que se aplica la recolección de datos. Otra razón por la que se decide realizar el cuestionario a estos proyectos, es porque cada facultad define un perfil que no necesariamente tiene que ver con sistemas de gestión, y por este motivo, la versatilidad de proyectos con diversos alcances y diferentes naturalezas le imprime heterogeneidad a la muestra enriqueciendo de esta manera el análisis de los resultados alcanzados.

De acuerdo a los principios metodológicos de (HERNÁNDEZ LEÓN, y otros, 2002), el cuestionario se sometió a un pilotaje que permitió garantizar su comprensión por los encuestados y prever el tratamiento estadístico más apropiado para la información que se obtiene. El mismo se aplicó a un proyecto de la muestra seleccionada.

1.4.1.2 ENTREVISTA A DIRECTIVOS DE LA ACTIVIDAD PRODUCTIVA DE LA UCI

Por su parte, la entrevista tuvo un carácter individual y estuvo orientada a evaluar la aplicación institucionalizada del paradigma de POA en los proyectos de desarrollo en la UCI. La misma forma parte de los instrumentos de recopilación de información para el desarrollo de la fundamentación del presente trabajo.

El objetivo general que perseguía la misma *es evaluar la aplicación institucionalizada del paradigma de POA en los proyectos de desarrollo aplicaciones informáticas en la UCI.* Y más específicamente:

1. Identificar el nivel de aplicación del paradigma de POA en los proyectos productivos de la UCI.
2. Conocer si existe alguna metodología certificada o autorizada por la Infraestructura Productiva (IP) de la UCI que guíe el PDSOA.
3. Evaluar la importancia que le confiere la institución a la existencia de una guía de principios de buenas prácticas que sirva de complemento y/o apoyo al PDSOA en la UCI.

Estuvo dirigida a directivos o asesores de la IP de la UCI que intervengan en los procesos de toma de decisiones o certificación de metodologías, métodos, procedimientos o instrumentación para el proceso de desarrollo de sistemas en los proyectos productivos de la institución.

1.4.2 ANÁLISIS DE LOS RESULTADOS DEL ESTUDIO DE CAMPO

Este es un epígrafe de significativa relevancia. En el mismo se realiza un análisis de cada uno de los aspectos e indicadores evaluados en el trabajo de campo desarrollado en la UCI, con el objetivo principal de identificar el estado de uso en los proyectos productivos de la POA.

Como se había mencionado, se emplearon dos instrumentos de recolección de datos: una encuesta y entrevistas a directivos de la IP de la UCI, donde los respectivos resultados y análisis son los que a continuación se presentan.

1.4.2.1 RESULTADOS DEL CUESTIONARIO

Se encuestaron un total de 21 miembros de proyectos con roles de Jefe de Proyecto, Arquitecto, Diseñador Principal o Programadores, vinculados a 21 proyectos de las 9 facultades con las que actualmente cuenta la sede central de la UCI (Ver *ANEXO 1*).

De los encuestados el 61,9% conocían qué era la POA, en tanto un 38.1% desconocía de su existencia.

El 71,4% de los encuestados consideró que la POA era un paradigma de programación, mientras un 28,6% la clasificó como una técnica de programación. De a cuerdo a la literatura revisada, la POA es un paradigma de programación, sin embargo, la clasificación de técnica de programación aunque simplifica en cierta medida su alcance, no es un término inapropiado para referirse a la misma. En cambio se sugiere que se trate como paradigma y no como técnica.

Solo un 23,8% de los encuestados ha tenido experiencia de trabajo con la POA en el desarrollo de algún sistema informático en el contexto de la actividad productiva de la UCI. En cambio el 76,19% manifestó no haber tenido experiencia de trabajo con el paradigma en algún proyecto productivo. Esto último no indica que ese porcentaje no haya realizado estudios sobre la POA o que no tengan algún dominio del mismo, por lo que partiendo del nivel de experiencia en una escala del 1 al 10, el 66,7% de los encuestados se atribuyó no tener ningún dominio del paradigma, en tanto el resto estuvieron por los niveles 1, 2, 3, 6, 7 y 8, para un dominio promedio de nivel 4.6; indicando una básica experticia de los encuestados en el ámbito de uso de la POA.

Sobre el empleo de la POA en los proyectos productivos, el 38,1% de los proyectos encuestados emplea el paradigma de orientación a aspectos en tanto un 61,9% no lo usan. En cuanto a este resultado, hay que agregar que el empleo o no de la POA en el desarrollo de un sistema, está determinado por los intereses particulares de la empresa, además de otros factores como el desarrollo de la curva de aprendizaje de los programadores, la evaluación de escenarios de la arquitectura para identificar posibles beneficios en la aplicación del paradigma, así como otros elementos asociados al mejoramiento de atributos de calidad como el acoplamiento, la cohesión y la facilidad de mantenimiento del sistema. No es obligatorio emplear el paradigma de orientación a aspectos para desarrollar un determinado producto informático; esto debe ser un análisis casuístico.

Otro importante resultado fue el de otorgar un nivel de importancia en el rango de 1 a 10, a la conveniencia de implementar de manera totalmente aislada, todos aquellos conceptos que entrecruzan un sistema y que hacen menos legible y de fácil mantenimiento el código de negocio. En este sentido, el 52,38% le atribuyó un nivel 10 de importancia, por otra parte el resto estuvo entre los niveles 5 y 9, mientras que solo un 4,762% de los encuestados no le asignó ningún nivel de importancia.

Uno de los resultados más importantes fue que en ninguno de los proyectos que emplea la POA, existe alguna guía o metodología, que les permita estar al alcance de principios de buenas prácticas para garantizar la adecuada aplicación del paradigma. Esto es un resultado que viene muy a tono con que el 80,95% de los encuestados le otorgó un nivel 10 de importancia a contar precisamente con una guía de principios de buenas prácticas para apoyar el proceso de toma de decisiones enfocado en la adecuada aplicación del paradigma de orientación a aspectos, mientras el resto estuvo entre los niveles 7 y 9, indicando un elevado interés por la existencia de alguna herramienta o instrumento que apoyase el correcto uso de la POA en el desarrollo de sistemas informáticos en la UCI.

1.4.2.2 RESULTADOS DE LAS ENTREVISTAS A DIRECTIVOS DE LA INFRAESTRUCTURA PRODUCTIVA DE LA UCI

Se realizaron cuatro entrevistas a ejecutivos de la Dirección de la IP de la UCI (Ver *ANEXO 1*) con el objetivo de ***evaluar la aplicación institucionalizada del paradigma de POA en los proyectos de desarrollo de la UCI*** y más específicamente:

1. Identificar el nivel de aplicación del paradigma de POA en los proyectos productivos de la UCI.
2. Conocer si existe alguna metodología certificada o autorizada por la Infraestructura Productiva de la UCI que guíe el proceso de DSOA.
3. Evaluar la importancia que le confiere la institución a la existencia de una guía de buenas prácticas que sirva de complemento y/o apoyo al proceso de DSOA en la UCI.

Los encuestados ocupaban importantes roles en la toma de decisiones de la actividad productiva en la UCI, e intervenían en disímiles procesos dentro del desarrollo de sistemas informáticos de la institución, los que a continuación se señalan:

Rol	Procesos en los que interviene
Director General de Producción de la UCI.	Interviene en todos los procesos asociados al desarrollo de sistemas en la UCI.
Jefe de la Línea de plataforma Java en el	Interviene en la estandarización de las

Centro de Consultaría UCI	tecnologías Java, en el asesoramiento en el proceso de desarrollo sobre esta plataforma, y en el desarrollo de sistemas J2EE en la UCI.
Director de la Dirección Técnica de la IP	Interviene en el Modelo de Producción de la UCI y en la identificación de las tecnologías que se deben usar en el desarrollo de soluciones informáticas en la institución.
Director del Centro de Informatización Universitaria.	Interviene en todos los proyectos del centro, de alguna manera en los proyectos de informatización que están en otras áreas productivas.

Tabla 1: Relación de los roles y responsabilidades de entrevistados.

El tiempo promedio de trabajo de estos directivos en la IP de la UCI es de tres años y medio aproximadamente, los que en ese tiempo en sentido general, han guiado el proceso de desarrollo en la institución en función de la aplicación de un determinado modelo productivo, el correcto y pertinente uso de tecnologías informáticas y el asesoramiento el proceso de desarrollo en cuestión.

La entrevista resultó ser un excelente espacio para reflexionar sobre el proceso de desarrollo de sistemas informáticos en la UCI y sobre las actuales tendencias en emplear uno u otro paradigma de desarrollo. El aporte de experiencia estuvo fundamentado por la diversidad de roles que han ocupado los entrevistados en el transcurso de sus carreras y la experiencia adquirida en cada uno de ellos. Dentro de las experiencias de trabajo en el desarrollo de sistemas, es preciso señalar que los mismos han desempeñado roles de Jefe de Proyecto, Arquitectos y Programadores por períodos de aproximadamente 3, 2 y 5 años, respectivamente.

Sobre el conocimiento de paradigmas de programación, los entrevistados tenían conocimiento de los siguientes:

- Paradigma de Programación Orientada a Aspectos
- Paradigma de Programación Orientada a Objetos
- Paradigma de Programación Orientada a Eventos
- Paradigma de Programación Orientada a Agentes
- Paradigma de Programación Estructurada
- Paradigma Funcional
- Paradigma de Programación Lógica
- Paradigma de Lineal u Orientado a Usuarios
- Paradigma de Programación Orientada a Servicios

Todos estuvieron de acuerdo con que el paradigma de POO era el que más se empleaba en la UCI, sin embargo mencionaron que también se emplean otros paradigmas como el de POA, Programación Lineal, el Orientado a Eventos, el Orientado a Agentes, el paradigma de Programación Lógica, entre otros.

Cada uno de los entrevistados conocía un conjunto de metodologías de desarrollo de sistemas informáticos, como las que se pueden citar: RUP, Open Up, la Metodología de la Programación Estructurada, la de Grady Booch, Jacobson, Rumbaugh, Análisis y Diseño Orientado a Objetos de Sistemas Informáticos (ADOOSI), Análisis y Diseño de Sistemas Automatizados (ADESA), Modelo de la Catedral y el Bazar, además la del diseño Orientado a Agente, la metodología de Sistemas de Expertos, metodología ágil XP y muchas otras. Coincidieron en que la metodología más usada en la UCI era RUP y algunas adaptaciones híbridas de la misma.

Sobre el empleo de la POA en proyectos productivos de la UCI, el Centro de Consultoría certificó el empleo de este paradigma en algunos proyectos como son: *Prisiones, CICPC, 171, Hospitales*; los que no han podido ser incluido en el estudio de campo realizado en el marco del desarrollo de esta investigación, puesto a que son proyectos de exportación y sus principales directivos[12] se encuentran fuera del país o cumpliendo alguna otra actividad muy específica.

[12] Referente a los roles de Jefe de Proyecto, Arquitecto Principal, Diseñador Principal, entre otros.

Sobre la conveniencia de separar los conceptos que entrecruzan la implementación de un sistema (*concerns*) de la lógica de negocio, predominaron dos criterios que aunque a simple vista pudieran tener marcadas diferencias, no se contraponen:

- ***Criterio 1***: los conceptos que entrecruzan todo el sistema, como el manejo de excepciones, gestión de errores, logging, gestión de transacciones, entre otros que cumplimentan los requerimientos, hacen menos entendible y más difícil de mantener el código de negocio, donde este último debe ser totalmente independiente de esos elementos.
- ***Criterio 2***: lo ideal sería desarrollar orientado a procesos, pero en caso que se emplearan paradigmas tradicionales para el desarrollo de la solución informática, el nivel de usabilidad aumentaría mediante la introducción del paradigma orientado a aspectos.

La totalidad de los entrevistados se pronunciaron por las ventajas de analizar y diseñar de manera aislada al negocio, los conceptos que entrecruzan el sistema y que en cierta medida no tienen relación con el dominio de negocio aunque lo complementa. Algunas de las ventajas que se mencionaba son: que esto permitía a analistas, arquitectos y diseñadores concentrarse en la lógica de negocio propiamente dicha; que la implementación aislada de estos conceptos hace que el negocio no solo sea mejor entendible, sino que pueda ser modelado y mantenido de manera más sencilla. Sobre esto último, el Centro de Consultoría está trabajando en una experiencia de desarrollo de un componente se seguridad sobre plataforma J2EE que está implementado haciendo uso del paradigma de orientación a aspectos y que puede ser acoplado a cualquier negocio independientemente de la naturaleza del mismo. Por último, se hacía referencia a la necesidad de que todas las decisiones asociadas al empleo de una u otra tecnología o uno u otro paradigma, estuvieran en función de la arquitectura y que esta, a su vez, se encargara de orquestar cada uno de estos elementos que se han modelado e implementado de manera aislada.

Se identificó, en la realización de las entrevistas a directivos de la IP, que en la UCI no se cuenta con ninguna metodología o guía de buenas prácticas que garanticen la adecuada introducción de la POA en el desarrollo de sistemas informáticos en la institución. Sin embargo se planteó que lo que más próximo está de apoyar este proceso de integración y aplicación de la POA, es la existencia de patrones de diseños que de alguna manera u otra dictan algunos principios, pero

que por otro lado no dicen cómo llevar a cabo el proceso de uso de la POA reflejado en otras buenas prácticas, ya que los patrones en sí mismos son buenas prácticas. Otro punto es que las metodologías guían el proceso de desarrollo pero no están condicionadas al estilo de programación. La introducción del paradigma de POA está más asociada a la madurez de la arquitectura que se emplee, no a la dependencia de la guía por parte de las metodologías de desarrollo, las que en definitiva carecen de herramientas para garantizar el correcto uso de este paradigma en particular.

Se identificó un total interés por la existencia de una guía de principios de buenas prácticas orientada a apoyar el proceso de integración de la POA al desarrollo de sistemas en la UCI; apostando por la conveniencia de un mecanismo de análisis que sirva para evaluar los beneficios de emplear el paradigma orientado a aspectos en el favorecimiento de indicadores de calidad en la arquitectura y productividad de los equipos de desarrollo. Se señaló que esta guía deberá estar enfocada al análisis de estos indicadores, en el contexto del desarrollo de la arquitectura y en cómo los arquitectos deberán centrar su actividad de guiar el proceso de construcción de un sistema informático. Y por último pero no menos importante, es que se identificó una tendencia de desarrollo orientado a componentes COST[13] (del inglés, *Commercial Off The Shelf*) en la UCI y en donde la aplicación de la POA tiene muchas ventajas, sobre todo para el tema de integración, dado a que el desarrollo de componentes COST está orientado al empleo de metodologías y técnicas de desarrollo orientado a componentes, donde la reutilización e integración son dos elementos determinantes y en donde la POA tiene un efecto muy positivo.

Se sugirió que la guía propuesta esté en función de las experiencias reales en la aplicación de la POA de proyectos productivos de la UCI. Que respondiera a situaciones reales más que ideales y que esté en función del análisis arquitectónico más que del análisis del paradigma en cuestión.

[13]En los últimos años se ha visto un aumento en el uso de componentes comerciales en prácticas de reutilización de componentes. Concretamente, estos componentes comerciales comúnmente conocidos con el nombre de COTS (*Commercial Off The Shelf*) están siendo considerados con mayor frecuencia para la construcción de sistemas complejos, distribuidos y abiertos.

CONCLUSIONES PARCIALES

Luego de un estudio sobre la evolución histórica de algunos paradigmas de desarrollo, así como el análisis conceptual de lo que es POA y cuál es el estado de aplicación de la misma en el contexto de desarrollo de sistemas informático en la UCI, se llega a las siguientes conclusiones:

1. Queda evidenciado que no existe una guía o metodología enfocada a la presentación de principios de buenas prácticas para garantizar la adecuada aplicación de la POA en el proceso de desarrollo de sistemas informáticos o en apoyo al proceso de toma de decisiones en este sentido.

2. Existe un marcado interés por el desarrollo de una guía de buenas prácticas que sirva de apoyo al proceso de integración de la POA con el desarrollo tradicional de sistemas informáticos.

3. Se identificó una tendencia en la UCI de desarrollo de *componentes COST* y en este sentido, un nicho de aplicación de la POA en beneficio de facilitar el proceso de integración y reutilización en el desarrollo; lo que le imprime mayor trascendencia al resultado de la presente investigación.

CAPÍTULO 2

Capítulo 2: ANÁLISIS DEL LENGUAJE ORIENTADO A ASPECTOS DE PROPÓSITO GENERAL ASPECTJ.SINTAXIS Y SEMÁNTICA

INTRODUCCIÓN

En este capítulo se analizará los fundamentos de la POA mediante el uso del lenguaje de propósito general orientado a aspectos, AspectJ. Su condición de *lenguaje extendido de Java* y su sintaxis y semántica, lo convierten en un excelente componente didáctico, logrando así una mejor comprensión de la POA desde el punto de vista de implementación. Cada elemento de la POA tiene una contrapartida en el lenguaje AspectJ; este lenguaje implementa los aspectos separados en un archivo independiente de los componentes Java.

En sentido general, AspectJ es construido como una extensión del lenguaje Java creado en Xerox PARC[14]. En este lenguaje los aspectos se escriben en *Java extendido* generándose un archivo java o compilado con código de máquina compatible con el generado por los compiladores del lenguaje base. Permite modelar referencias transversales, extendiendo así las posibilidades de POO.

[14] Centro de investigación de Palo Alto, California EE.UU. Es una división de investigación de Xerox Corporation.

2.1 ELEMENTOS BÁSICOS DE ASPECTJ

En el presente epígrafe se abordarán detalles sobre cómo implementar aspectos haciendo uso del lenguaje de propósito general para la implementación de la POA, AspectJ; tratando en primer lugar los conceptos y detalles sintácticos y semánticos de los puntos de enlace (join points), puntos de corte (point cuts), avisos (advice), el manejo de la información contextual, las reglas de entrelazado estático disponibles para AspectJ, extensión e instancias de los aspectos y por último, una visión rápida de la API de AspecttJ.

Resaltar que la extensión que hace AspectJ del lenguaje Java es exclusivamente para especificar reglas de entrelazado entre los aspectos y los componentes Java. Esta extensión es favorable a la hora de comprender el lenguaje de aspectos AspectJ dado a que la curva de aprendizaje se ve disminuida respecto a otros lenguajes de propósito general con total independencia del lenguaje base. Esto implica invertir menos tiempo para estudiar y entender, desde el punto de vista sintáctico y semántico, el lenguaje AspectJ.

AspectJ define cuatro constructores para especificar las reglas de entrelazado entre los aspectos y los componentes del lenguaje base (XEROX Corporation, 2003):

- **Puntos de enlace (Joint points):** dentro de la ejecución de una aplicación, es un lugar de fácil identificación, podría consistir en una llamada a algún método, en la declaración de alguna variable o en la asignación de algún valor a algún atributo de clase.
- **Punto de corte (Pointcuts):** especifican dónde se encuentran los puntos de enlaces y se captura información relacionada con el contexto de la aplicación. La diferencia entre puntos de cortes y puntos de enlace es que los primeros definen reglas de especificación y los segundos especifican las situaciones que satisfacen dichas reglas.
- **Avisos (Advice)**: Este es el código que se debe ejecutar en los puntos de enlaces especificados en los puntos de cortes. Los mismos se pueden ejecutar antes, durante o después. Los avisos se parecen a las funciones tradicionales de una aplicación y definen el "*qué hacer"* en los puntos de enlace.
- **Inter-tipos (inter-type):** permiten a un aspecto definir completamente los campos y los métodos implementados para una o múltiples clases. Es decir, permiten modificar la estructura estática de un programa.

2.2 PUNTOS DE CORTE.

Los puntos de cortes capturan o identifican los puntos de enlaces (Ver 2.3) dentro del flujo del programa. Después de estos puntos de cortes ser definidos por los constructores del lenguaje de aspectos que se esté utilizando, puede especificarse mediante reglas de entrelazamiento cómo cambiar los comportamientos definidos en los aspectos con el código original.

En AspectJ los puntos de corte se pueden declarar anónimos o nombrados. Al igual que en las clases. Un punto de corte anónimo se define en el lugar donde se usa, que bien podría ser en un aviso o en la definición de otro punto de corte. En caso de no ser anónimos, se definen con un nombre que pueda ser referenciado desde múltiples partes del código permitiendo su reusabilidad. Respecto a ellos, la sintaxis en AspectJ es la siguiente:

pointcut nombre(argumentos): <definición>

Por *definición de un pointcut* se entiende al conjunto de reglas que definen los *puntos de enlace* donde se quiere insertar cierto comportamiento. El nombre que se utilice para definir el punto de corte es el que luego ha de utilizar en los avisos para hacer referencia a ellos (Ver 2.4).

Se pueden definir puntos de cortes anónimos, aunque no es aconsejable por la sencilla razón de que no pueden ser reutilizados, por tanto, se sugiere que no sean usados cuando se está ante una expresión complicada o cuando se desea referenciar en varios avisos.

2.2.1 TIPOS DE PUNTOS DE CORTE

Luego de definir la sintaxis de representación de los puntos de cortes es preciso saber cómo van a ser implementados. Esto se debe hacer según el tipo de punto de enlace (llamada a métodos, constructores, definición de variables, accesos a atributos). En el siguiente esquema se presenta como *AspectJ* ofrece diferentes posibilidades (ASTEASUAIN, y otros, 2002):

Categoría	Sintaxis
Ejecución de métodos	execution(MethodSignature)

Llamada a métodos	call(MethodSignature)
Ejecución de constructores	execution(ConstructorSignature)
Llamada a constructores	call(ConstructorSignature)
Inicialización de clases	staticinitialization(TypeSignature)
Lectura de atributos	get(FieldSignature)
Escritura de atributos	set(FieldSignature)
Captura de excepciones	handler(TypeSignature)
Inicialización de objetos	initialization(ConstructorSignature)
Pre-inicialización de objetos	preinitialization(ConstructorSignature)
Ejecución de avisos	adviceexecution()

Tabla 2: Categorías de Puntos de Cortes.

2.2.2 PUNTOS DE CORTES BASADOS EN CONTROL-FLOW.

A continuación se muestran un conjunto de puntos de cortes definidos en **AspectJ** cuya particularidad consiste en capturar puntos de corte siempre y cuando estos ocurran en el contexto de puntos de corte. Por esta razón, la declaración de estos incluye el paso por parámetros de otros puntos de cortes. La siguiente tabla muestra un conjunto de ejemplos que pretenden reflejar la versatilidad de puntos de cortes basados en *control-flow* (GREGOR, y otros, 1997).

Definición de Puntos de Cortes	Descripción
cflow(**call** (* UnaClase.unaoperacion(..)))	Define los puntos de enlace durante la ejecución del método ***unaoperacion***() de la clase ***UnaClase***, incluyendo la llamada al propio método.

cflowbelow(**call**(*UnaClase. unaoperacion (..))	Define los puntos de enlace durante la ejecución del método ***unaoperacion***() de la clase ***UnaClase***, en este caso excluyendo la llamada al propio método de la clase ***UnaClase***.
cflow(Operaciones())	Define los puntos de enlace en el contexto de todos los puntos de enlace capturados por el punto de corte ***Operaciones****().*
cflow(**staticinitializer**(UnaClase))	Todos los puntos de enlace durante la inicialización de la clase ***UnaClase***.
cflowbelow(**execution**(UnaClase.**new**(..))	Todos los puntos de enlace durante la ejecución de alguno de los constructores de la clase ***UnaClase***, excluyendo la ejecución misma del constructor.

Tabla 3: Ejemplos de puntos de cortes basados en *control-flow*.

Para comprender mejor este útil punto de corte, es oportuno analizar los siguientes casos ejemplos:

Considere el punto de corte ***P***. En el siguiente ejemplo se capturan todos los puntos de enlaces en el flujo de control del punto de corte ***P***.

```
cflow(P)
```

Gráficamente esto sería del siguiente modo:

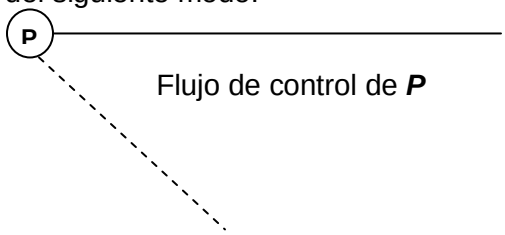


Figura 1. Captura de puntos de enlaces en el flujo de control de P

Un caso más interesante es cuando se desean capturar puntos de enlace en la intercepción de dos flujos de control, por ejemplo:

```
cflow(P)&& cflow(Q)
```

Efectivamente, en este caso se entraría a analizar solo aquellos puntos de cortes cuya ocurrencia es en la intersección de dos flujos de control, o sea:

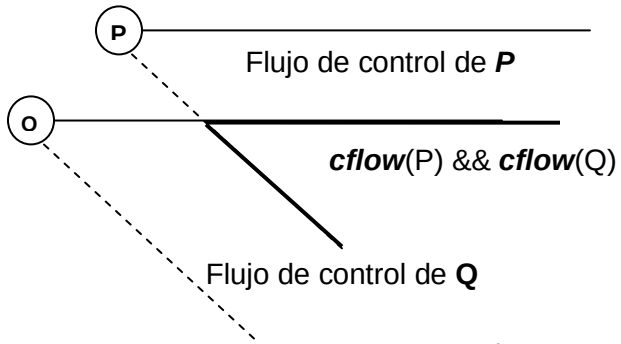


Figura 2. Captura de puntos de enlaces en la intercepción de los flujos de control de P y Q

Existe una diferencia *entre capturar la intercepción entre el flujo de control de un punto de corte y el flujo de control de otro* y *capturar el flujo de control de la intercepción de dos puntos de cortes*. Por ejemplo:

```
cflow (P && Q)
```

En este último caso se refiere a la ocurrencia de los puntos de enlace que intervienen dentro de la suma del flujo de control de ***P*** y el flujo de control de ***Q***, entonces:

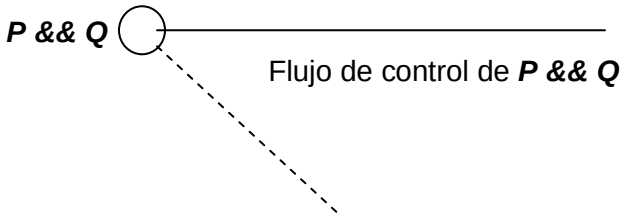


Figura 3. Captura de puntos de enlaces en el flujo de control de la intercepción de P y Q

2.2.3 PUNTOS DE CORTES BASADOS EN LOCALIZACIÓN (WITHIN).

El próximo grupo de puntos de cortes están relacionados con la localización de código referenciado. Existen dos tipos de clasificaciones para estos puntos de cortes:

- ***Within*** *(TipoPatrón)*: Se usan para capturar todos los puntos de enlaces en el cuerpo de las clases, aspectos o sub-clases especificados en el *TipoPatrón.*
- ***Withincode*** *(FirmaMétodo):* Se emplea para capturar los puntos de enlace dentro de la estructura léxica de una clase o un método.

Este tipo de punto de corte tiene como uso excluir los puntos de enlaces del aspecto donde es definido.

2.2.4 PUNTOS DE CORTES BASADOS EN OBJETOS (THIS, TARGET).

Los siguientes puntos de corte, ***this*** y ***target***, están basados en los tipos de los objetos en tiempo de ejecución. Con ***this*** se referencia al objeto actual, mientras que con ***target*** es al objeto sobre el que se llama al método. Estos puntos de corte sirven además para recoger el contexto en el punto de enlace. ***This*** toma la forma ***this(Type***) y así captura los puntos de enlace asociados con objetos del tipo especificado ***Type***. Al igual, target es también de la forma ***target(Type).***

2.2.5 PUNTOS DE CORTES BASADOS EN ARGUMENTOS (ARGS).

El siguiente tipo de *pointcouts* se basa en el tipo de argumentos del punto de enlace. Por argumentos ha de entenderse lo siguiente: para los puntos de cortes que capturan excepciones, la excepción capturada es considerada el argumento; y para los accesos a atributos en escritura, se considera como argumento el nuevo valor a escribir sobre el articulo. Este tipo de punto de corte sirve también para capturar la información contextual en el punto de enlace.

2.2.6 PUNTOS DE CORTES CON CONDICIONES (IF).

El último de los puntos de cortes que será analizado en el lenguaje AspectJ, permite usar expresiones condicionales, lo cual tendrá la consecuencia de capturar solo los puntos de enlace cuando dichas condiciones se evalúen a cierto. Estos puntos de cortes toman la forma ***if(BooleanExpression).*** La expresión lógica se evalúa en tiempo de ejecución, por lo que debe ser válida en el contexto donde se sitúe.

2.2.7 COMBINAR PUNTOS DE CORTES

Los puntos de cortes pueden combinarse dando lugar a una señalización más específica. Es decir, mediante la conjugación de varios puntos de cortes puede reunirse mayor cantidad de

información sobre dónde se desea tejer el código del aspecto. Estas combinaciones son posibles a través del empleo de operadores relaciones ***or*** y ***and***, y desde el punto de vista de su sintaxis, es muy similar al modo de emplear estos operadores en otros lenguajes de programación.

2.3 SINTAXIS Y SEMÁNTICAS DE LOS PUNTOS DE ENLACE

Los puntos de cortes, como se ha mencionado, harán referencia a una serie de ***puntos de enlaces*** en la aplicación, por lo que requiere de un lenguaje para definirlos. AspectJ emplea una gama de elementos comodines para capturar dichos puntos de enlaces que comparten características comunes. Los comodines son los siguientes:

Comodines
***** Denota cualquier número de caracteres excepto el punto.
.. Denota cualquier número de caracteres incluyendo el punto.
+ Denota los descendientes de una clase.
! Denota todos los puntos de enlace excepto aquellos que se especifiquen en la expresión que siga.

Tabla 4: Comodines de AspectJ para definir puntos de enlaces.

Pueden usarse los paréntesis combinados con estos operadores con objeto de alterar la precedencia por defecto de las expresiones y facilitar también la legibilidad. Los elementos tendrán diferentes significados según acompañen a paquetes, tipos (clases, interfaces o aspectos) métodos o atributos.

2.3.1 EL MODELO DINÁMICO DE PUNTOS DE ENLACES

El elemento crítico en el diseño de cualquier lenguaje orientado a aspectos es el modelo de puntos de unión. Este modelo provee un marco de referencia común para hacer posible la definición de la estructura dinámica de los conceptos que entrecruzan el sistema.

Los puntos de unión son puntos bien definidos en la ejecución de un programa. Cuando un método o constructor son corridos, existen dos tiempos asociados a ellos: "*cuando son llamados"*,

y "*cuando son ejecutados"*. Por lo tanto son dos puntos de enlace diferentes. Algunos ejemplos de puntos de unión, o puntos de enlace, son: llamadas a métodos, ejecución de métodos, instanciación de objetos, ejecución de constructores, referencias a campos, manejos de excepciones, entre otras. Es importante que se conozca que los puntos de unión no son posiciones en el código fuente, sino en la ejecución del programa. Son eventos que ocurren durante la ejecución del programa.

Un punto de unión de llamadas a métodos abarca las acciones que un objeto recibe cuando se hace una llamada a un método. Este incluye todas las acciones comprendidas en la llamada, comenzando desde la evaluación de los parámetros hasta el retorno de un valor o una excepción.

Cada llamada a un método en tiempo de ejecución es un punto de enlace o punto de unión diferente, incluso si la llamada proviene de la misma expresión de un programa; todos los puntos de unión ocurren mientras se ejecute el cuerpo del método.

Cuando se ejecuta un método o un constructor, están asociados a esta acción dos eventos importantes: uno cuando ha sido llamado y otro cuando se está ejecutando. En este sentido, AspectJ expone dos tipos de puntos de unión para cada uno de estos eventos; además de que brinda dos puntos de cortes que hacen posible capturar ambos puntos de enlaces: el punto de corte ***call*** para cuando se realiza una llamada y el punto de corte ***execution*** para cuando se ejecuta un método o un constructor.

Básicamente es muy simple darse cuenta de la diferencia entre llamar a un método o constructor, y ejecutar el cuerpo del mismo. Sin embargo se debe tener mucho cuidado con un caso en particular: la *recursividad*.

Para el caso de la recursividad en la implementación de métodos, la llamada al método se realiza incluso en el transcurso de su ejecución. Es decir, existe al menos una llamada en el cuerpo del método a sí mismo, lo que hace que el análisis vaya un poco más allá de lo simplista.

En AspectJ se presentan dos construcciones léxicas que permiten manejar esta situación: ***within*** y ***withincode*** (XEROX Corporation, 2003). Existe mucha similitud semántica en ambos puntos de

corte. De manera que a partir del concepto básico de *puntos de enlaces* pueden construirse puntos de cortes que capturen el comportamiento recursivo de algún algoritmo, muy útil cuando se desea conocer el estado de ejecución del mismo; haciendo posible, por ejemplo, el desarrollo de una pila de ejecución para controlar las llamadas recursivas de un determinado método.

2.4 AVISOS.

El cuerpo de un aviso se parece mucho al cuerpo de un método, ya que encapsula la lógica que debe ser ejecutada cuando se alcanza cierto punto de enlace en la ejecución. Los puntos de cortes identifican ***dónde*** y los *avisos* lo complementan indicando ***qué hacer.***

Los avisos (*advices*), en conjunto con los puntos de cortes (*pointcuts*), son las herramientas para implementar conceptos de entrelazamiento, dinámico o estático, dentro de un sistema.

Los avisos son constructores parecidos a los métodos donde se indica el código que se debe ejecutar en los puntos de enlace que se especifiquen en un punto de corte. Estos pueden ejecutarse antes (*before*), después (*after*) o "alrededor" (*around*) del punto de enlace. Alrededor quiere decir que se puede modificar el código en ejecución a la altura del punto de enlace, reemplazarlo o ignorarlo si se desea.

De manera general, los avisos definen el código adicional a ejecutar en los puntos de enlaces capturados por los puntos de cortes. Serían entonces los avisos los responsables de definir el comportamiento que entrecruza toda la funcionalidad de la aplicación. Estos pueden trabajar con puntos de cortes nombrados o anónimos.

2.5 INTER-TIPOS (inter-type).

Los inter-tipos permiten a un aspecto definir completamente los campos y los métodos implementados para una o múltiples clases (XEROX Corporation, 2003). Es decir, posibilitan modificar la estructura estática de un programa.

Considere el problema de agregarle una nueva funcionalidad a un conjunto de clases en una determinada jerarquía. En Java esto puede ser resuelto declarando una interfaz que capture esta funcionalidad e implementándola en dicha jerarquía de clases. AspectJ puede expresar esta

funcionalidad en un solo lugar, precisamente usando declaraciones de inter-tipos. En la declaración de un aspecto se pueden definir campos y funcionalidades necesarias para crear nuevas capacidades en un programa, y luego asociar esos campos y funcionalidades a las clases existentes en una determinada jerarquía sin tener que afectar el diseño original de las mismas.

Las declaraciones de inter-tipos favorece en gran medida la flexibilidad de los diseños de programas. La decisión de emplear o no declaraciones de inter-tipos para modificar la estructura estática de un programa, debe estar en función de la arquitectura sobre la que se esté desarrollando, así como también de los intereses de la organización por alcanzar mayor flexibilidad en los diseños de aplicaciones informáticas.

CONCLUSIONES PARCIALES

Con el afán de alcanzar soluciones informáticas altamente flexibles y de fácil mantenimiento, se introducen técnicas y se aplican principios de buenas prácticas, como el uso de patrones de diseño, que a pesar de su efectividad aún son insuficientes para lograr tal objetivo. Este parecería ser un criterio refutable, pero luego de un análisis de los fundamentos generales de la POA se evidenciaron vías que resuelven definitivamente el problema de la separación de incumbencias en el desarrollo de sistemas informáticos.

El estudio del lenguaje de aspectos de propósito general, *AspectJ*, sirvió para mostrar, desde el punto de vista sintáctico-semántico, los principales conceptos del paradigma de orientación a aspectos, además de un conjunto de instrumentos que hacen posible lograr aumentar la flexibilidad de los sistemas desde el cambio de su estructura estática hasta la disminución e incluso, completa eliminación, de las dependencias entre componentes de sistemas mediante el uso de puntos de cortes y avisos.

El capítulo dos constituyó en definitiva, un importante espacio para comprender los fundamentos del paradigma de POA a partir de la didáctica de la sintaxis y semántica de un lenguaje específico que implementa dicho paradigma, y que además está enriquecido dada su condición de *lenguaje extendido de Java,* lo que sirvió para integrar los conceptos de orientación a aspectos en el desarrollo de sistemas orientados a objetos.

CAPÍTULO 3

Capítulo 3: GUÍA DE PRINCIPIOS DE BUENAS PRÁCTICAS PARA EL DESARROLLO ORIENTADO A ASPECTOS

INTRODUCCIÓN

Asumir el desarrollo de un sistema presupone un elevado conocimiento de disímiles temáticas en diversas áreas que podrían enmarcar aspectos económicos, de comercialización, contables, financieros, entre otros; para poder disponer de al menos los elementos mínimos necesarios para llevar a cabo una adecuada toma de decisiones en el contexto del proceso de desarrollo de sistemas informáticos.

El presente capítulo tiene como encargo la conformación de un contexto de análisis en el que se exponen un conjunto de razonamientos y evidencias concretas, asociados a la evaluación de factibilidad al introducir el paradigma de POA en el desarrollo tradicional de un sistema informático.

Los principios de buenas prácticas propuestos exponen un marco teórico que sirve de referencia para el análisis en la toma de decisiones del proceso de desarrollo de un sistema informático, específicamente asociado a la evaluación de aplicar técnicas de POA para ganar en flexibilidad en los diseños, disminuir los niveles de acoplamiento de los componentes y favorecer la productividad de los equipos de desarrollo a partir de la disminución del esfuerzo de satisfacción de requerimientos del sistema.

No es una guía absoluta ni rígida, y tampoco define un procedimiento o metodología para la construcción de un producto informático; sino que se basa en la propuesta de principios de buenas prácticas orientados a complementar la instrumentación disponible en el proceso de toma de decisiones en el entorno de desarrollo.

3.1 CAPTURA Y MODELACIÓN DE INCUMBENCIAS EMPLEANDO CASOS DE USO

Al tener en cuenta la separación de incumbencias se logra una mejor comprensión de aquellos sistemas que son más complejos, ya que se obtiene un código más legible, facilitando en gran medida su mantenimiento. Además se favorece el mejoramiento de los indicadores de acoplamiento y cohesión de los componentes.

Como se ha venido defendiendo durante todo el presente trabajo, el proceso de mantener las incumbencias separadas de la lógica de negocio se debe comenzar desde etapas tempranas del proceso de desarrollo, modelando la lógica de negocio según la facilidad de validación y el desarrollo gradual del sistema. En este sentido, la modelación a través casos de uso aparece como una provechosa alternativa.

Los casos de uso no son más que fragmentos de funcionalidad que el sistema ofrece para aportar un resultado de valor para sus actores; los cuales son la representación de los usuarios y otros sistemas externos que pueden interactuar con el sistema en construcción (JACOBSON, y otros, 2004).

Una especificación de casos de uso contiene una serie de flujos que describen cómo se manejan las variaciones. Se puede comprobar fácilmente que se ha capturado las exigencias de los stakeholders[15], pasando a través de flujos. Un flujo de trabajo normal de un caso de uso de negocio puede tener flujos básicos de actividades y flujos alternativos de actividades. Utilizando los flujos se ayuda a visualizar las responsabilidades del caso de uso y evita que se caiga en la "trampa" de la descomposición funcional. Además es el punto de partida para el modelado de puntos de corte en los casos de uso.

[15] Personas u organizaciones que están activamente implicadas en el negocio ya sea porque participan en él o porque sus intereses se ven afectados con los resultados del proyecto.

Las relaciones que se establecen entre los casos de uso, proporcionan los medios para el modelado de incumbencias. Estas relaciones pueden ser de extensión, inclusión y generalización (JACOBSON, y otros, 2004).

La relación de extensión permite añadir nuevos comportamientos en los casos de uso existentes. Estos nuevos comportamientos describen de forma independiente el caso de uso extendido del caso de uso base. Permitiendo así que el caso de uso base no tenga que entrar en los detalles de la nueva conducta del caso de uso extendido. La relación de extensión en los casos de uso, se transforma durante la ejecución en los llamados avisos (*advices*) de POA.

Un caso de uso extendido puede contener uno o más flujos de extensión. Los flujos de extensión son como flujos alternos excepto que aumentan los comportamientos de diferentes casos de uso. En los flujos de extensión se encuentran los puntos de extensión. Un punto de extensión identifica un punto específico en el trayecto de ejecución de una instancia de un caso de uso, en dónde los comportamientos definidos del caso de uso se podrían volver a incluir. (JACOBSON, y otros, 2004) En este contexto aparecen los puntos de enlace, los cuales concuerdan con los puntos de extensión en los casos de uso y los puntos de corte, que no son más que una expresión que indica dónde serán introducidas las extensiones dentro de una operación ya existente.

Los puntos de extensión y puntos de cortes de extensión son dos elementos diferentes y ocurren con fines diferentes en una relación de extensión. Un punto de corte de extensión especifica dónde el caso de uso base será extendido. Los puntos de extensión están definidos en el caso de uso base, mientras que los puntos de cortes de extensión están definidos en el caso de uso de extensión. El responsable de nombrar los puntos de extensión es el actor del caso de uso base, el que establece además referencias para los puntos de cortes de extensión.

Existen varias formas de definir los puntos de extensión, entre ellas se encuentran:

- **Definición de puntos de extensión pasando por un solo paso**: Este punto de extensión ocurre en un punto específico, en un determinado paso dentro del flujo básico de determinado caso de uso.

- **Definición de puntos de extensión pasando por múltiples pasos**: Son aquellos puntos de extensión que ocurren dentro de una secuencia de puntos, es decir sobre varios puntos dentro del flujo básico.
- **Definición de puntos de extensión con propiedades de comportamiento en cada paso**: Estos puntos ocurren cuando aparecen algunas características conductuales, por ejemplo cuando el actor envía una petición al sistema. La aceptación de una solicitud es también una característica conductual, por lo tanto el punto de extensión puede ocurrir en cualquier paso dentro del caso de uso, por eso es que se identifican los pasos según su comportamiento. Lo cual es útil cuando se quiere extender las incumbencias dentro de un caso de uso existente como podría ser el registro de bitácora o la autorización.

Se pueden utilizar de dos formas diferentes los puntos de extensión y los puntos de corte de extensión. Una de ella es utilizando una referencia directa, donde el punto de corte de extensión hará referencia a un paso en el caso de uso extendido de forma directa. Y la referencia indirecta, donde un punto de corte de extensión hará referencia a un punto de extensión que luego hará referencia a un caso de uso extendido.

Además de los casos de uso generalizados, incluidos y extendidos, se pueden definir otro tipo de caso de uso llamado caso de uso de utilidad; los cuales no proporcionan funcionalidades que son de utilidad directa para los actores, por lo que no se podrá crear instancias de ellos. Como ejemplo de caso de uso de utilidad se tienen las siguientes funciones: abrir, archivar, cerrar y renombrar archivos. Estas funciones necesitan ser usadas dentro de un contexto más grande. De modo que los casos de uso de utilidad siempre se ejecutan en el contexto de otra instancia en un caso de uso. El propósito de los flujos básicos es especificar el comportamiento que aparece cuando un caso de uso es instanciado. Un caso de uso sin un flujo básico no puede ser instanciado.

Según (JACOBSON, y otros, 2004) existen dos clases de casos de uso de utilidad: las *inclusiones de utilidad* y las *extensiones de utilidad*. Los casos de uso de utilidad incluidos tienen sub-flujos donde se definen dónde serán incluidos en otros casos de uso, mientras que los casos de uso de utilidad extendidos tienen flujos alternos que los interconectan a otros casos de uso, lo que permite que sean sumamente reusables. Como ejemplo de los casos de uso de utilidad

incluidos se tienen las operaciones de crear, leer, actualizar y suprimir diferentes tipos de información almacenada dentro del sistema; tales operaciones son siempre realizadas como parte de una funcionalidad mayor.

Con una correcta identificación de los casos de uso se logra una mejor estructuración de los mismos, y de esta forma se garantiza una mejor comprensión del sistema. Si se han identificado correctamente los casos de uso incluido, excluido y la generalización no se caerá en la descomposición funcional. La idea de un correcto modelado y estructurado de los casos de uso es comprender y definir las exigencias y necesidades de los clientes, para así alcanzar una mejor comprensión del sistema.

La captura de requisitos debe definir el problema al que el sistema dará solución, es necesario entonces que queden claros los requisitos que desean los clientes, por lo que se debe comprender el negocio, pasando por cada uno de sus escenarios. Los casos de uso proporcionan técnicas para describir lo que el sistema debe hacer con el fin de satisfacer las necesidades de los clientes.

Existen dos tipos de requisitos según (JACOBSON, y otros, 2004): los funcionales, que no son más que las capacidades o condiciones que el sistema debe cumplir; y los no funcionales, que son las propiedades o cualidades que el producto debe tener. Los requisitos no funcionales usualmente se refieren a algunos mecanismos de soporte dentro la infraestructura del sistema y los que imponen comportamientos en el sistema.

Asociado a esto y estableciendo cierta analogía, se pueden identificar entonces dos tipos de casos de uso, los de aplicación y los de infraestructura. Los casos de uso de aplicación describen las funcionalidades básicas del sistema. En tanto los casos de uso de infraestructura describen cómo el sistema agrega cualidades como: facilidad de uso, confiabilidad, rendimiento y soporte para cada paso de un caso de uso de aplicación. Primeramente es necesario entender los intereses de los *stakeholders*, y como resultado de esta interacción entre clientes, usuarios y otros implicados, se obtiene una lista de características del sistema la cual incluye requisitos funcionales y no funcionales. Cada caso de uso de infraestructura se conecta a los de aplicación cuantas veces este se encuentre dentro de él. Los casos de uso de infraestructura al igual que

los de la aplicación tienen flujos básicos y alternos así como especificaciones de casos de uso. (JACOBSON, y otros, 2004)

Cuando se capturan los casos de uso de la aplicación, primeramente hay que identificar actores y casos de uso a partir de los requisitos funcionales de la lista de características, y se describen los casos de uso en las especificaciones de casos de uso contemplando posibles extensiones. La descripción de los casos de uso ayudará a identificar las incumbencias. Luego se capturan los casos de uso de infraestructura como extensiones modulares a los casos de uso de aplicación, Para ello, se revisa nuevamente la lista de características del sistema para identificar a los requisitos no funcionales que afectan algún paso de los casos de uso de aplicación, los cuáles serán tratados como casos de uso de infraestructura, si se tratan de requisitos de infraestructura para el sistema. Y se culmina este proceso con la descripción de los casos de uso de infraestructura en una especificación contemplando también sus flujos alternativos.

Los casos de uso de aplicación, definen el negocio del sistema y los casos de uso de infraestructura definen las incumbencias que hay que tener en cuenta en el desarrollo de un sistema, como por ejemplo las transacciones, seguridad, entre otras. Ambos tipos de casos de uso se relacionan mediante la extensión en la modelación del sistema. Al clasificar estos dos tipos de casos de uso se identifican las incumbencias desde la captura de requisitos, por lo que se logra separarlas de la lógica de negocio desde etapas tempranas del desarrollo de un sistema informático.

3.2 MANTENER LA SEPARACIÓN DE LAS INCUMBENCIAS MEDIANTE MODELOS DE CASOS DE USO

En el desarrollo de aplicaciones informáticas guiado por casos de uso, se comienza con el modelo de casos de uso para capturar necesidades de los stakeholders. Luego se refina el modelo de casos de uso en el modelo de análisis, el cual formula una descripción de alto nivel del sistema. Con el modelo de diseño se arman estrategias de cómo el sistema funcionará en la plataforma de ejecución. Y en el modelo de implementación, se tienen los códigos. Sin embargo, no se construye un modelo del sistema según un modelo en específico, esto se hará caso de uso por caso de uso, de manera que se toma cada caso de uso, se perfecciona y se lleva a cabo de forma progresiva a través de los diferentes modelos (JACOBSON, y otros, 2004).

Identificar y separar las incumbencias en etapas tempranas en el proceso de desarrollo, como es la captura de requerimientos y en la modelación del negocio, apenas es un primer paso. El siguiente paso sería lograr mantener la separación de incumbencias durante las etapas de análisis, diseño e implementación del sistema. En consecuencia aparece una nueva unidad modular llamada segmentos de caso de uso (use-case slice), los que contienen colaboraciones, clases y características específicas en la realización de casos de uso (JACOBSON, y otros, 2004).

Un modelo comprende una jerarquía de elementos. En él se pueden encontrar paquetes que contienen clases, clases que contienen operaciones, y así sucesivamente.

Los segmentos de casos de uso son independientes de la estructura de elementos. Se superponen sobre la estructura de elementos y definen su contenido, es decir, definen los comportamientos que se extienden en dicha estructura. Los segmentos de casos de uso tienen dependencias entre sí, que describen cuál se debe colocar en primer lugar, y cuál deberá ser el siguiente, y así sucesivamente. Esto lleva a otra estructura, llamada estructura de casos de uso. Por lo tanto, un modelo consta de dos estructuras:

- **Estructura de los elementos**: Identifica los elementos. Se analizan las clases organizadas en paquetes y capas.
- **Estructura de casos de uso**: Define el contenido de los elementos. Se analizan los segmentos de casos de uso que agregan el contenido en la estructura de elementos.

Los elementos como las clases, son tratados como cajas vacías. Su contenido, su comportamiento estará en los segmentos de casos de uso durante la composición. Los segmentos de casos de uso no son los paquetes típicos que se encuentran en un modelo de diseño, el mismo consta de una estructura de elementos, una organización jerárquica de los elementos en términos de capas, paquetes, subsistemas, y un conjunto de casos de uso. Estos casos de uso se superponen en la parte superior de esta estructura de elemento. Si se quita un segmento de caso de uso, todos los contenidos en él como clases y características son eliminados también de esta estructura de elementos.

Desde la perspectiva del diseño, estas estructuras de casos de uso identificadas en el modelo de diseño están conformadas por paquetes estereotipados con «*use-case slice*» y «*non-uc-specific slice*»[16]. Un segmento no específico contiene clases del dominio que se usan en muchos casos de uso, mientras que un segmento de caso de uso contiene elementos necesarios para un caso de uso específico.

Cada segmento de casos de uso mantiene los detalles de una realización de casos de uso en un modelo. Las partes genéricas y reutilizables se mantienen en los segmentos no específicos de casos de uso.

Se puede parametrizar cualquier tipo de elementos en un segmento de caso de uso, incluyendo los aspectos, lo que permite que un segmento de caso de uso sea reutilizable. Estos segmentos están compuestos por clases y extensiones de las clases existentes organizadas dentro de los aspectos, que son específicos en la realización de cada caso de uso en el modelo de diseño. Es necesario un mecanismo para poner el fragmento de la clase contenida en el segmento de caso de uso en la clase existente. Esto es logrado usando aspectos en POA. Se modelan los aspectos como un elemento estereotipado «*aspect*» propuesto en UML para el modelado de aspectos. Los aspectos tienen compartimiento de extensiones de clase para revestir las extensiones de clase sobre las clases existentes, lo cual se logra con POA usando declaraciones inter-tipos y avisos.

Mediante un conjunto de clases es posible llevar a cabo la realización de un caso de uso. Luego es necesario determinar qué partes de estas clases son específicas para el caso de uso y qué partes son comunes. De esta forma se define cada parte específica como extensiones de clases dentro de los segmentos de casos de uso. Estos emplean aspectos para crear las extensiones de clase en sus respectivas clases dentro de la estructura de elementos; por lo que se debe determinar el contexto estructural de la extensión: *a qué clase o operación de extensión serán añadidas*; y el contexto del comportamiento: *en qué momento en el transcurso de la ejecución de la operación será invocada la operación de extensión*.

[16] Son unidades modulares específicas y no específicas, respectivamente, que permiten una mejor organización del sistema. Es conveniente tratar el término original del inglés, para no distorsionar su significado semántico.

Correspondencia entre los elementos de un segmento de caso de uso durante el diseño (con UML) y durante la implementación (con POA) (JACOBSON, y otros, 2004):

Modelos UML	Modelo POA
Segmentos de casos de uso	Paquete
Colaboración	Ninguno
Aspecto	Aspecto
Extensión de clase	Agrupamiento de declaraciones de inter-tipos para cada clase

Tabla 5: Mapeando Modelos UML a Modelos POA

Hasta ahora, la estructura de casos de uso no podía ser separada de la estructura de elementos. Un desarrollador tenía que reunir el conjunto de responsabilidades en cada clase a partir de diferentes casos de uso y el desarrollo de la clase. Esto da lugar a enredos y dispersión, ya que las incumbencias están mezcladas con estos elementos. Con los segmentos de casos de uso, los desarrolladores pueden implementar cada caso de uso por separado, lo que permite que se preparen los segmentos de casos de uso en el entorno de desarrollo para formar un conjunto de elementos en el modelo de diseño; logrando así una separación de incumbencias mediante modelos de casos de uso en el sistema.

3.3 ESTRUCTURACIÓN DE EQUIPOS DE DESARROLLO ENFOCADA EN LA SEPARACIÓN DE INCUMBENCIAS.

La productividad es una variable cuya criticidad radica en los efectos, positivos o negativos, que se obtienen en los resultados y la calidad de los procesos y productos, respectivamente, de cualquier empresa. De modo que para el caso del desarrollo de sistemas informáticos este razonamiento no se encuentra ajeno a los intereses que subyacen en la organización en sí. En este sentido y con motivo de lograr favorecer los índices de productividad, la definición de una arquitectura que tenga en cuenta el “modus operandi” de los equipos de desarrollo a partir de su estructura, cantidad de personal, experticia, entre otros componentes, podría y de hecho es, una acertada estrategia de administración.

Aumentar la productividad en una empresa se traduce en obtener mayores ganancias, reducción de costos de producción y un aumento considerable de los ingresos reales para los empleados. Existen múltiples factores que intervienen en el aumento de la productividad, como es el caso especial del factor esfuerzo. Si bien es cierto que la disminución del esfuerzo de producción no garantiza completamente elevar los índices de productividad en una empresa para el desarrollo de sistemas informáticos, al menos contribuye en gran medida a la reducción de tiempos de entrega, el aprovechamiento laboral, y por tanto a una mejor gestión de cambios y planificación para una mejor asignación de los recursos humanos.

En el desarrollo de un sistema informático existen (n) requerimientos del cliente que los equipos de desarrollos deben acometer con el objetivo de alcanzar un producto final lo más completo posible. Para lograr cada uno de esos requerimientos se requiere de un esfuerzo de implementación (x). Partiendo de esta premisa, cada organización debe evaluar casuísticamente la conveniencia de lograr un determinado nivel de separación de incumbencias en función de reducir los esfuerzos de producción y por consiguiente, aumentar la productividad de los equipos de desarrollo. En este sentido téngase en cuenta los siguientes casos:

- **Caso 1 (Independencia total entre requerimientos del cliente):**
  Cada uno de los requerimientos del cliente es absolutamente independiente del resto de los requerimientos y por tanto, el desarrollo individual de cada uno de ellos está libre de conexiones con el desarrollo de los otros. Este es un caso muy **poco probable** dado a que la no existencia de dependencias entre requerimientos indica posibilidades nulas de reutilización entre componentes del sistema. Para este caso, el *esfuerzo total* ($\boldsymbol{E_t}$) para desarrollar todos los requerimientos es:
    - $\boldsymbol{E_t = nx}$ Para el caso en que el esfuerzo de realización de cada requerimiento sea el mismo o al menos las diferencias promedios sean despreciables.

    - Para el caso en que el esfuerzo base sea independiente del requerimiento, que es lo que generalmente pasaría:

$$\boldsymbol{E_t = \sum_{i=1}^{n} x_i}$$

- **Caso 2 (Dependencia entre requerimientos):**

  Es un caso en el que la realización de un requerimiento depende de la realización de los otros (h) requerimientos, o en el peor de los casos de los (n – 1) requerimientos. Para el análisis de este caso hay que tener en cuenta una tercera variable: el esfuerzo de integración (y) entre requerimientos. Por tanto:

  - El esfuerzo de realización específico ($\boldsymbol{E_{R_i}}$) de cualquier requerimiento es:

    $\boldsymbol{E_{R_i} = x + yh}$ Para un esfuerzo de integración constante.

  - Y por tanto, el esfuerzo total ($\boldsymbol{E_t}$) de realización del sistema es:

  Para el caso en que el esfuerzo de integración ($\boldsymbol{y}$) y el esfuerzo base ($\boldsymbol{x}$) sea constante para cada requerimiento:

  $\boldsymbol{E_t = n(x + yh)}$

  $\boldsymbol{E_t = n(x + y(n - 1))}$ En el peor de los casos $\boldsymbol{h = n - 1.}$

  Cuando el esfuerzo base ($\boldsymbol{x}$) y el esfuerzo de integración ($\boldsymbol{y}$) varían para cada requerimiento, se tiene:

Esf.		$\boldsymbol{R_1}$	$\boldsymbol{R_2}$	…	$\boldsymbol{R_h}$
$\boldsymbol{x_1}$	$\boldsymbol{R_1}$	0	$\boldsymbol{y_1}$	$\boldsymbol{y_2}$	$\boldsymbol{y_n}$
$\boldsymbol{x_2}$	$\boldsymbol{R_2}$	$\boldsymbol{y_1}$	0	$\boldsymbol{y_2}$	$\boldsymbol{y_n}$
…	…	…	…	0	…
$\boldsymbol{x_n}$	$\boldsymbol{R_h}$	$\boldsymbol{y_1}$	$\boldsymbol{y_2}$	$\boldsymbol{y_3}$	0

Tabla 6: Correlación de esfuerzo de integración entre requerimientos.

  El esfuerzo total de realización de un sistema donde no se ha realizado separación de las incumbencias, es:

$$\boldsymbol{E_t = \sum_{i=1}^{n} (x_i + \sum_{j=1}^{h} y_j)}$$

  Es preciso señalar que este en particular, es un caso donde el acoplamiento entre componentes es extremadamente elevado.

- **Caso 3 (Separación de Incumbencias):**

  A partir de la clasificación de ($\boldsymbol{q}$) requerimientos de aplicación y ($\boldsymbol{k}$) requerimientos de infraestructura, donde ($\boldsymbol{z}$) requerimientos de aplicación están asociados con ($\boldsymbol{h}$) requerimientos de infraestructura, el esfuerzo de realizar un requerimiento de aplicación sería $(x + \boldsymbol{h}y)$. Dentro de este contexto, en el peor de los casos, cada uno de los ($\boldsymbol{q}$) requerimientos de aplicación tiene una conexión o dependencia con los ($\boldsymbol{k}$) requerimientos de infraestructura; de modo que el esfuerzo total de realizar todos los requerimientos de aplicación es $\boldsymbol{q}(x + \mathbf{k}y)$ y para los de infraestructura es sencillamente $(\boldsymbol{k}x)$.

  El esfuerzo total de realizar todos los requerimientos de un sistema donde se ha logrado separar las incumbencias del negocio, es:

  - $\boldsymbol{E_t = q(x + hy) + kx}$ Para cuando el esfuerzo base y el esfuerzo de integración son los mismos para todos los requerimientos. En el peor de los casos, en el que cada requerimiento de aplicación se asocia con todos los requerimientos de infraestructura, se tiene que $(\boldsymbol{h = k})$**.**

  - Para el caso donde el esfuerzo base y el esfuerzo de integración son específicos para cada requerimiento.

$$\boldsymbol{E_t} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{q}} \left(\boldsymbol{x_i} + \sum_{\boldsymbol{j=1}}^{\boldsymbol{h}} \boldsymbol{y_j} \right) + \sum_{\boldsymbol{i=1}}^{\boldsymbol{k}} \boldsymbol{x_i}$$

Este es un modelo matemático que correlaciona los indicadores de esfuerzos de realización ($\boldsymbol{x}$) e integración ($\boldsymbol{y}$) de requerimientos de un sistema informático, haciendo posible un análisis comparativo a partir de la evaluación de dos contextos diferentes: cuando en un sistema se introducen las técnicas de orientación a aspectos para lograr la separación de incumbencias, y otro contexto en el que no son implementadas estas técnicas. En este sentido, el cálculo o estimación de ambos esfuerzos es irrelevante respecto a la semántica del modelo, dado a que no se alteran los resultados siempre y cuando se utilice el mismo criterio en la obtención de ambos indicadores.

El objetivo fundamental de esta representación matemática es evaluar el porciento de reducción de esfuerzo alcanzado al separar las incumbencias en el desarrollo de un sistema mediante la implementación de técnicas de POA.

Para la cuantificación porcentual de la reducción de esfuerzo lograda a partir de la separación de incumbencias en el desarrollo de un sistema, respecto al caso en que no se realice esta, se establece la siguiente relación:

$$Porciento\ de\ Disminución\ (PD) = 100 - \left(\frac{\boldsymbol{E_{t_{CI}}}}{\boldsymbol{E_{t_{SI}}}} \times 100\right)$$

En este sentido:

- Si ***(PD) < 0***: se disminuye el esfuerzo de desarrollo del sistema a partir de la aplicación de técnicas de POA para la separación de incumbencias.
- Si ***(PD) > 0***: no se favorece la productividad separando las incumbencias a partir de la aplicación de la POA.
- Si ***(PD)= 0***: es porque se deben analizar otros indicadores asociados a la conveniencia de aplicar o no la POA para el desarrollo del sistema, ya que no se registra una disminución del esfuerzo de desarrollo al aplicar técnicas de POA.

Haciendo una valoración de los resultados que se obtienen en los casos 2 y 3 (no será preciso evaluar el caso 1 por ser un caso poco probable) partiendo de una situación hipotética donde se tenga un sistema de 50 requerimientos y los valores de esfuerzo base y esfuerzo de integración sean de 10 y 1, respectivamente, se tiene que:

- **Caso 2:** $\boldsymbol{E_t = n\big(x + y(n-1)\big) = 50(10 + 49) = 2950}$

- **Caso 3:** Para este caso supóngase que se hayan identificado 20 requerimientos de aplicación y 30 de infraestructura (aunque pueden ser otros valores arbitrarios que sumen un total de 50 requerimientos)

$$\boldsymbol{E_t = q(x + ky) + \ kx = \ 20(10 + 30) + \ 30 = \ 1100}$$

A partir de la anterior situación, se logra reducir el esfuerzo de realización del sistema si se separan las incumbencias, en un 62.7% menos.

Lo más interesante de este análisis es que no se ha tenido en cuenta las ventajas que proveen los *tejedores de aspectos* respecto a la considerable disminución del esfuerzo de integración entre los requerimientos de aplicación y los requerimientos de infraestructura. Suponiendo una reducción a la mitad del esfuerzo de integración (y) (que puede ser incluso más gracias a la utilización de *tejedores*), los resultados serían aún más alentadores, alcanzando solo un esfuerzo total de 800 para el caso 3 y reduciendo 72.9% más el esfuerzo total.

Importante Conclusión: Separar las incumbencias del negocio y desarrollar a partir de esto una modelación del sistema donde se identifiquen y realicen por separado, requerimientos de aplicación y requerimientos de infraestructura, disminuye en gran medida el esfuerzo de desarrollo y por tanto favorece al incremento de la productividad de los equipos.

3.3.1 CONFORMACIÓN DE UN EQUIPO BASADO EN EL DESARROLLO ORIENTADO A ASPECTOS

Una adecuada estructuración de los equipos de desarrollo tributa a garantizar en cierta medida la efectividad del trabajo. Existen metodologías, como SCRUM, XP, RUP, que proponen principios y estructuras preconcebidas para definir los equipos a la hora de asumir el desarrollo de un sistema. En el contexto del desarrollo orientado a aspectos, la conformación de los equipos alcanza una trascendencia muy singular. De acuerdo a todo lo que se ha explicado asociado a la identificación de las incumbencias en etapas tempranas del desarrollo y la conveniencia de modelarlas separadas de la lógica de negocio (Ver 3.1), se propone lo siguiente: *conformación de equipos para el desarrollo de la lógica de negocio y equipos para el desarrollo de la lógica de aspectos.*

Básicamente la idea anterior consiste en la factibilidad de poder corresponder, desde el punto de vista estructural, al principio de disminución del esfuerzo para aumentar la productividad en el desarrollo de sistemas. A partir de que las incumbencias pueden ser identificadas y separadas en la modelación del sistema, y que por otro lado la realización o implementación de los casos de uso de infraestructura es totalmente independiente, gracias a los conceptos de entrelazado de

código de aspecto y código de componente (Ver capítulo 2), de los casos de uso de aplicación, se puede y es muy conveniente, diferenciar el desarrollo de la lógica de negocio al desarrollo de la lógica de aspectos.

Establecer equipos que asuman la realización de los casos de uso de aplicación, y equipos para desarrollar la lógica de aspectos, trae varias ventajas:

1. Una mejor gestión y control de las tareas asignadas.
2. Agilidad en el desarrollo, ya que ambas implementaciones, tanto el funcionamiento básico como la lógica de aspectos, avanzan independientemente.
3. Facilidad en la gestión de cambios, dado a que se logra reducir la repercusión de un cambio en los requerimientos en el trabajo realizado por un equipo específico.
4. Reducción del tiempo de entrega de versiones funcionales del sistema, ya que la implementación de la lógica de negocio es totalmente independiente de la lógica de aspectos.
5. Facilidad de mantenimiento del código del sistema.
6. Facilidad de realización de pruebas de caja blanca.

Estas son algunas de las ventajas más importantes de poder estructurar los equipos de desarrollo de acuerdo a los principios de desarrollo orientado a aspectos. Evidentemente las especificidades de asignación de roles estarían en correspondencia de la capacidad adaptativa de los equipos de desarrollo, de los intereses de la organización y quizás en algunos casos, en la metodología de desarrollo que se aplique. Por una parte, los equipos de lógica de negocio podrían continuar estableciendo los mismos principios estructurales que anteriormente, en el desarrollo tradicional, existían. En tanto los equipos de la lógica de aspectos aparecerían con una novedosa conformación y asignación de nuevos roles para el desarrollo (Ver epígrafe 3.3.1.2).

3.3.1.1 DESARROLLO DE LA LÓGICA DE NEGOCIO

El desarrollo de la lógica de negocio responde a la realización de los casos de uso de aplicación definidos en la modelación del sistema, donde estos pueden o no tener asociaciones (extensiones) con los casos de uso de infraestructura. Sin embargo, esas relaciones no condicionan el desarrollo de los equipos de la lógica de negocio por la sencilla razón que la dirección en que se establecen las dependencias entre estos dos tipos de casos de uso, es hacia

los casos de uso de infraestructura y de manera unidireccional. Diciéndolo de otro modo, los casos de uso de aplicación no “conocen” la existencia de la realización de los casos de uso de infraestructura, ya que son estos últimos los que establecen los fundamentos de asociación gracias a la definición reglas de entrecruzamiento funcional.

Aprovechando esta ventaja, los equipos de desarrollo de la lógica de negocio, pueden dar cumplimiento a la implementación de los casos de uso de aplicación, sin tener en cuenta la existencia o no de la realización de los casos de uso de infraestructura extendidos. Esto, como se especificaba en 3.3.1, garantiza reducción en los tiempos de entrega de versiones funcionales del sistema, permitiendo que los clientes obtengan resultados funcionales concretos, favoreciendo la realización temprana de pruebas de aceptación y por tanto, lograr disminuir la capacidad de respuesta ante las nuevas exigencias o no conformidades del cliente.

Evidentemente, el desarrollo de la lógica de negocio es independiente de la lógica de aspectos, no garantiza el cumplimiento de algún requisito del sistema, dado a que hasta tanto no se implementen las dependencias de infraestructura, el sistema no podrá *hacer* y *contar* con las funcionalidades y cualidades que espera el cliente.

3.3.1.2 DESARROLLO DE LA LÓGICA DE ASPECTOS

Por otra parte está el desarrollo de la lógica de aspectos, la que al igual que la lógica de negocio, se desarrolla de manera independiente en el contexto de la POA. Sin embargo, en este caso existe una sutil diferencia: muy a pesar de que para implementar un caso de uso de infraestructura pueden o no existir aquellos casos de uso de aplicación que los asocien, los desarrolladores de la lógica de aspectos sí necesitan conocer algunos elementos asociados a la modelación de los casos de uso de aplicación para establecer las reglas de entrelazado entre ambos.

De acuerdo a lo anteriormente planteado, la empresa, proyecto u organización en sentido general, que asuma el desarrollo de un sistema, debe garantizar lo siguiente:

- Definición y estandarización de estilos de diseño y codificación.
- Realización centralizada, bien sea por módulo, sub-sistema o sistema, de la modelación de los casos de uso de aplicación y los casos de uso de infraestructura,

garantizando conocimiento total de todos los equipos de desarrollo del dominio del problema.

- Establecer mecanismos de comunicación entre los equipos de lógica de negocio y lógica de aspectos, preferentemente para que estos últimos puedan obtener información valiosa y necesaria para llevar a cabo la implementación de los aspectos.

Para apoyar el proceso de conformación de un equipo para el desarrollo de la lógica de aspectos, la gerencia de las organizaciones pueden basarse en algunos de los siguientes roles que a continuación se proponen:

- **Jefe de Equipo:** Es el máximo responsable del equipo de lógica de aspecto. Define el cronograma de desarrollo de la lógica de aspectos a partir de la identificación de los requerimientos más críticos y responde directamente por su cumplimiento.
- **Diseñador de Aspectos:** Es el encargado de identificar y diseñar los aspectos que se derivan de la realización de un caso de uso de infraestructura, así como de establecer los estilos de codificación a seguir para implementar cada uno de los aspectos.
- **Diseñador de Reglas de Entrelazado:** Es responsable de diseñar y documentar las reglas de entrelazado del código de aspectos y el código de componente, útiles para la definición de los puntos de cortes.
- **Identificador de Puntos de Cortes:** Se encarga de identificar los puntos de cortes del sistema, a partir de los requerimientos y el modelo de casos de uso. Genera los puntos de cortes que deberá tener cada uno de los aspectos diseñados.
- **Programador de Aspectos:** Es responsable de implementar los aspectos del sistema, a partir del diseño realizado. Su mayor trabajo radica en la implementación de los *advice* que definen el comportamiento de aspectos.
- **Integrador:** Se encarga de establecer y controlar las políticas de integración entre la lógica de aspectos y la lógica de negocio. Documenta todo el proceso de integración y es responsable de velar por el cumplimiento de los estándares de codificación. Funge de enlace entre los equipos de lógica de negocio y lógica de aspectos, tramitando la realización de los casos de uso de infraestructura a partir de los intereses generales de la organización en la realización de uno o varios requerimientos en específico.

En sentido general, estos son algunos roles que pueden asignarse en la definición de un equipo de desarrollo de lógica de aspectos. Varía en función de los legítimos intereses de la organización y pueden adaptarse a las metodologías implementadas para asumir el desarrollo del sistema, las que quizás brinden otro enfoque de conformación, como es el caso de la metodología XP, en la que se puede particularizar la conveniencia de asociar en un dueto un programador de lógica de negocio y un programador de lógica de aspectos. Lo cierto es que con esta estructuración se lograría reducir el esfuerzo de implementación y con ello alcanzar todas las ventajas antes mencionadas.

PRINCIPIO DE "CERO ACOPLAMIENTO"

El acoplamiento es un indicador de calidad que define una medida de la "fuerza" con que un elemento está conectado a otro, tiene conocimiento de otro y confía[17] en otros elementos. Por consiguiente, un elemento con bajo o débil acoplamiento, no depende de demasiados[18] elementos (IEEE, 2004).

Reducir a cero el acoplamiento entre componentes en el desarrollo de un sistema informático, pudiera ser sencillamente una metáfora que sin importar cuán probable es lograrlo, motivaría a un razonamiento positivista de este fenómeno. Ciertamente alcanzar bajos niveles de acoplamiento es más que una buena noticia, un excelente camino para hacer más extensible los sistemas.

En aras de reducir al máximo el acoplamiento, han surgido técnicas y patrones, que a partir de la semántica de diferentes paradigmas de programación, brindan una solución en menor o mayor medida conveniente para resolver una problemática que motiva el estudio continuo de ávidos investigadores en esta área del conocimiento.

[17] La confianza a la que se refiere, está asociada a la dependencia que existe en el empleo de funcionalidades definidas por otros elementos.

[18] En dependencia del contexto en que se esté trabajando. Es una definición relativa que da a la medida de una cantidad determinada.

El caso de la introducción de la POA en el desarrollo tradicional, este análisis de reducción al máximo del acoplamiento adquiere una connotación muy singular gracias un detalle específico de este paradigma: la capacidad de separar las incumbencias y lograr entrelazar código de aspectos y código de componente sin que medien dependencias entre ambos. De manera que eliminando las dependencias se logra reducir el acoplamiento. Esto último podría parecer un análisis simplista cuya evidente conveniencia haría parecer que es una solución sencilla. Lo cierto es que hay resultados concretos en este sentido y el objetivo de este apartado es brindar un razonamiento a favor del uso de la POA para reducir minimizar el acoplamiento.

Existen un conjunto de formas en las que pueden manifestarse el acoplamiento en un diseño (LARMAN, 2003):

1. El tipo X tiene un atributo (miembro de datos, o variable de instancia) que hace referencia a una instancia de tipo Y, o al propio tipo Y.
2. Un objeto de tipo X invoca los servicios (métodos) de un objeto de tipo Y.
3. El tipo X tiene un método que referencia a una instancia de tipo Y, o al propio tipo Y, de algún modo. Esto, generalmente, comprende un parámetro o variable local de tipo Y, o que el objeto de retorno de un mensaje sea una instancia de tipo Y.
4. El tipo X es una subclase, directa o indirecta, del tipo Y.
5. El tipo Y es una interfaz y el tipo X implementa esa interfaz.

Para lograr resultados alentadores del indicador de acoplamiento, es factible encaminar los esfuerzos desde la propia definición de la arquitectura del sistema. En este sentido existe dos patrones que implementados con técnicas de POA favorecen extraordinariamente la disminución del acoplamiento entre componentes: patrón *Inversión de Control (IoC[19])* y patrón de *Inyección de Dependencia (IoD[20]),* siendo este último una modalidad de implementación del primero.

El patrón IoC está concebido para disminuir el acoplamiento entre componentes de una aplicación informática y fomentar de este modo la reutilización entre ellos. Básicamente, este

[19] Las siglas *IoC* corresponden a su significado en el idioma inglés (*Inversion of Control*) y su empleo en el contexto de la literatura hispana es muy común, por lo que en lo adelante se utilizarán las siglas para referirse a este patrón

[20] Ídem al caso de las siglas *IoC.* Específicamente se refiere al término *Inyection of Dependence.*

patrón resuelve el problema de las dependencias existentes entre componentes a partir de la aplicación de un principio de diseño denominado en el argot informático como "*Principio de Hollywood*[21]". Este patrón tiene como una de sus técnicas de implementación el patrón específico IoD, el que propone no instanciar las dependencias explícitamente en su clase, sino que declarativamente expresarlas en la definición de la clase. La esencia de la inyección de las dependencias es contar con un componente capaz de obtener instancias validas de las dependencias del objeto y pasárselas durante la creación o inicialización del objeto. En el ámbito de la POA, este componente responsable de inyectar las dependencias válidas, vendría siendo un contenedor de dependencias (FOWLER, 2004).

Estos patrones pueden usarse en algunas situaciones, como pueden ser:

- Cuando se desee desacoplar las clases de sus dependencias de manera de que las mismas puedan ser reemplazadas o actualizadas con muy pocos o casi ningún cambio en el código fuete de sus clases.
- Cuando existe la necesidad de probar las clases aisladamente, sin sus dependencias.
- Se necesitan escribir clases cuyas dependencias corresponden a implementaciones que no son conocidas en tiempo de compilación.
- Se desea desacoplar los componentes de una arquitectura.

Desde la perspectiva de la arquitectura, independientemente del estilo arquitectónico que se esté aplicando, la implementación de estos patrones mediante la POA repercutiría positivamente en los indicadores de calidad del producto final.

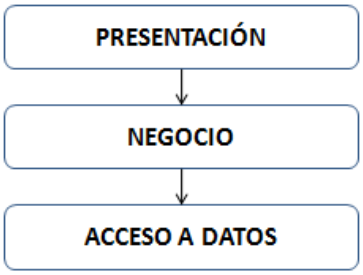


Figura 4: Ejemplo del modelo general de arquitectura en tres capas.

[21] Es un principio de diseño y tiene como fundamento la inversión del control de dependencias entre componentes a partir de slogan "*No nos llames, nosotros te llamaremos".*

Un ejemplo concreto de lo anteriormente planteado, basado en la existencia hipotética de una arquitectura que implemente el estilo arquitectónico *3-Capas (Presentación, Negocio y Acceso a Datos)* (Ver Figura 4), es la presentación de una situación (que además pudieran presentarse para el caso de *n-Capas*): *Las dependencias entre las capas están determinadas por la necesidad de utilización de servicios de las capas inferiores inmediatas;* a partir de de la cual pudiera pensarse en dos soluciones:

- **Solución 1 (Figura 4):** Dada la no existencia de un mecanismo único de comunicación, cada elemento de la capa superior que requiera de una funcionalidad de uno o varios elementos de la capa inferior, está obligado a definir una dependencia (instancia) de los últimos.
- **Solución 2 (Figura 5):** Definir interfaces de comunicación entre capas para reducir las dependencias.

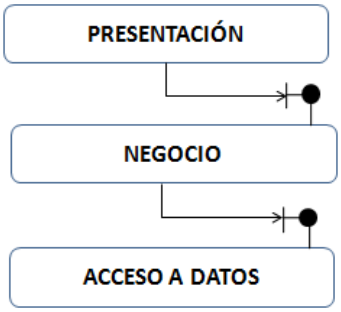


Figura 5: Implementación de Interfaces

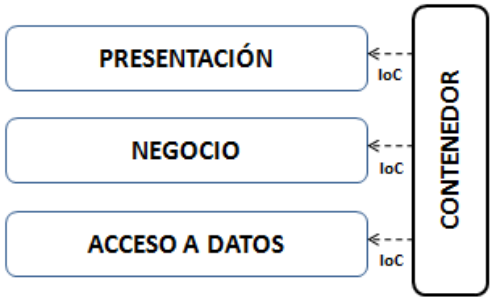


Figura 6: Definición de un Contenedor *IoC*

La primera solución, como es lógico, ni siquiera se aproxima a ser una solución factible producto a que la utilización de los servicios[22] de las capas inferiores estaría sujeta a la definición de las dependencias de componentes de clases. Para el caso de la segunda solución, con la definición de interfaces de comunicación, y la implementación de algún que otro patrón como el patrón de Bajo Acoplamiento, se logra reducir considerablemente las dependencia no solo entre componentes, sino entre capas, módulos o subsistemas; sin embargo aún sigue siendo un problema las dependencias porque a pesar de la existencia de interfaces, habría que continuar diciendo:

[22] Entiéndase en este contexto como las funcionalidades o métodos.

InterfazX ref = **new *ClasImplementInterfazX();***

Y aunque para muchos pudiera parecer imposible que esto pueda eliminarse completamente, existe una solución que lo logra, aplicando, evidentemente, el paradigma de orientación a aspectos en la implementación de los patrones *IoD* e *IoC.* De manera que para el ejemplo de la arquitectura en capas, las dependencias entre cada una de las capas dejarían de existir. Lo que se traduce en "**cero acoplamiento**", delegando la responsabilidad a una entidad externa al sistema, denominada "Contenedor", que se encargará de gestionar las instancias, así como sus creaciones y destrucciones; además de realizar la inyección de las sub-partes (dependencias); haciendo posible decir:

InterfazX ref = ***Container.getDependency();***

Esta solución sería la mejor vía para resolver el problema de las dependencias y el elevado acoplamiento entre elementos de un sistema informático, garantizando empleo de buenas prácticas de diseño de alto y bajo nivel en situaciones reales, como es el caso de Proyecto GKT (Ver ANEXO 2).

3.4 DISEÑOS MÁS FLEXIBLES

Las declaraciones inter-tipos, los puntos de cortes y los avisos, aportan grandes beneficios en el diseño de sistemas informáticos. Las declaraciones inter-tipos son un poderoso instrumento de la POA, que permiten a los desarrolladores (diseñadores y programadores) realizar cambios en la estructura estática de los sistemas a partir de la asignación de nuevos campos, responsabilidades de las clases, además de la definición de extensiones e implementaciones de clases o interfaces, sin tener que modificar los diseños originales de los sistemas. El principal beneficio que aporta la definición de inter-tipos desde la perspectiva de flexibilidad en el diseño, radica en la capacidad de incorporar o modificar comportamientos al sistema sin tener que comprometer la definición original del diseño.

La introducción de la POA en el desarrollo de sistemas informáticos, y en consecuencia, la introducción de mecanismos de entrelazado de código de aspectos y código de componente, permite mantener "limpios" los diseños de elementos que si bien son necesarios para la satisfacción de los requerimientos, hacen más acoplados y difícil de extender los componentes.

La magnitud con que se desee flexibilizar el diseño, deberá ser una cuantificación asociada a intereses particulares de la organización que acomete el desarrollo del sistema, en función de evaluar la pertinencia o no de incorporar elementos de la POA para no caer en la trampa de sobrevalorar las cualidades de este paradigma, o subvalorar las soluciones tradicionales. Ciertamente la POA favorece la capacidad adaptativa de los diseños y por tanto, de los sistemas, haciendo posible reducir el "*efecto dominó"* al realizar modificaciones a partir de nuevas consideraciones y necesidades de los clientes, sin traer grandes traumas replicados en diferentes componentes.

3.5 VALORACIONES CONSENSUADAS

Con el fin de respaldar la propuesta realizada, se implementó un pequeño sistema para la realización de pruebas a partir de la generación aleatoria de grandes cantidades de juegos de datos, con el objetivo de evaluar la efectividad de aplicación de los principios enfocados a la separación de incumbencias para el favorecimiento de la productividad. El sistema implementa el modelo matemático para el cálculo porcentual de reducción de esfuerzo (Ver 3.3) y realiza una representación tabular de los resultados alcanzados, facilitando el análisis comparativo y brindando elementos para el establecimiento de conclusiones enfocadas en el porciento de reducción del esfuerzo de realización de requerimientos mediante la separación de incumbencias usando técnicas de POA.

Se desarrollaron pruebas donde a partir de la generación de grandes cantidades de juegos de datos equivalentes a la representación de miles proyectos de desarrollo de sistemas informáticos, se alcanzaron resultados tan prometedores como los siguientes:

1. A medida que los sistemas son más grandes, abarcando gran cantidad de requerimientos del cliente, la aplicación de técnicas de POA para realizar separación de incumbencias favorece en mayor medida la reducción del esfuerzo de realización del sistema.
2. El esfuerzo se disminuye en todos los casos en más de un 50% e incluso llega a alcanzar cifras de 60% a más del 70% de reducción del esfuerzo en algunos casos específicos.

Por otra parte, se realizó una encuesta a miembros del 38.1% de los proyectos muestras para la realización del estudio de campo inicial (Ver 1.4.2), los que manifestaron aplicar técnicas de POA en el desarrollo de sus sistemas, con el objetivo de recoger la apreciación sobre la utilidad de los

principios propuestos para la toma de decisiones en el proceso de aplicación de la POA, arrojando los siguientes resultados: A partir del análisis de la propuesta de principios de buenas prácticas y del modelo matemático para la evaluación del porciento de esfuerzo disminuido a través de la aplicación de técnicas de POA en la separación de incumbencias, los encuestados clasificaron la solución propuesta a partir de las categorías de Mal, Regular, Bien y Muy Bien, de la siguiente manera:

- Mala: 0%
- Regular: 0%
- Buena: 45.45%
- Muy Buena: 54.55%

De manera que la totalidad encuestada coincide en que la guía de principios de buenas prácticas propuesta constituye un componente esencial para el apoyo al proceso de toma de decisiones en la adecuada aplicación del paradigma de POA.

CONCLUSIONES GENERALES

La propuesta de una guía de principios de buenas prácticas orientada a apoyar el proceso de introducción del paradigma de POA en los proyectos de desarrollo de aplicaciones informáticas de la UCI, permitió brindar un marco de análisis técnico para la toma de decisiones en la adecuada implementación del paradigma de orientación a aspectos, reflejado en el favorecimiento de la productividad de los equipos de desarrollo en función de la disminución del esfuerzo de realización de requerimientos de los clientes. En tanto la definición de un modelo matemático para cuantificar la factibilidad de aplicar apropiadamente los principios de separación de incumbencias propuestos, facilitó demostrar desde un enfoque comparativo, las ventajas que se alcanzan al aplicar las técnicas de la POA respecto al desarrollo de sistemas en el que no se implementan correctamente estos principios.

RECOMENDACIONES

Se recomienda validar los principios propuestos mediante algún método de experimentación en un proyecto real, que permita la cuantificación del impacto económico reflejado en el comportamiento de los índices de productividad, los costos de producción y los tiempos de entrega.

TRABAJOS FUTUROS

Implementar una solución informática para el apoyo a la toma de decisiones en el proceso de desarrollo de sistemas, específicamente en el área de la aplicación de técnicas de POA, que permita la gestión y clasificación de los requerimientos de clientes, así como el análisis de comportamiento de indicadores de factibilidad a partir de los resultados derivados de la evaluación del modelo matemático para la cuantificación porcentual de la reducción del esfuerzo de desarrollo mediante la separación de incumbencias.

BIBLIOGRAFÍA

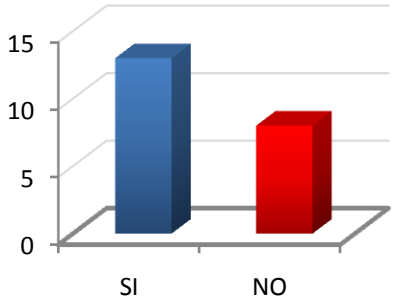
1. **ASTEASUAIN, Fernando y EZEQUIEL CONTRERAS, Bernardo. 2002.** *Programación Orientada a Aspectos. Análisis del paradigma.* Buenos Aires. Argentina : Universidad Nacional del Sur, 2002.
2. **DEITERS, Matthew. 2005.** Aspect-Oriented Programming. *MSDN.* [En línea] Microsoft Corporation, 09 de 2005. [Citado el: 08 de 11 de 2009.] http://msdn.microsoft.com/en-us/library/aa288717(VS.71).aspx.
3. **FOWLER, Martin. 2004.** MartinFowler.com. *Agile Project Managment tool from ThoughtWorks.* [En línea] 23 de Enero de 2004. [Citado el: 15 de Marzo de 2010.] ThoughtWorks es una consultora internacional sobre Tecnología de la Información. http://www.martinfowler.com/articles/injection.html.
4. **GÓMEZ ALBARÁN, Mercedez. 2005.** *Metodología basada en descomposición funcional y orientación a objetos en la introducción a la programación.* Dpto. de Sistemas Informáticos y Programación, Universidad Complutense de Madrid. Madrid : s.n., 2005. pág. 8.
5. **GREGOR, Kiczales, y otros. 1997.** *Aspect-Oriented Programming.* Palo Alto : s.n., 1997.
6. **HERNÁNDEZ LEÓN, Rolando Alfredo y COELLO GONZÁLEZ, Sayda. 2002.** *EL PARADIGMA CUANTITATIVO DE LA INVESTIGACIÓN CIENTIFICA.* Ciudad de La Habana : EDUNIV Editorial Universitaria, 2002. 959-16-0343-6.
7. **HEXACTA. 2000.** [En línea] 19 de 04 de 2000. [Citado el: 13 de 03 de 2010.] http://www.hexacta.com.
8. **IBM. 2000.** [En línea] 2000. [Citado el: 16 de 03 de 2010.] http://www.ibm.com.
9. **IEEE. 2004.** *Guide to the Software Engineering Body of Knowledge.* s.l. : IEEE Computer Society,, 2004. 0-7695-2330-7.
10. **JACOBSON, Ivar y Ng, Pan-Wei. 2004.** *Aspect-Oriented Software Development with Use Cases.* s.l. : Addison Wesley Professional, 2004. 0-321-26888-1.
11. **JACOBSON, Ivar, BOOCH, Grady y RUMBAUGH, James. 2000.** *El Proceso Unificado de Desarrollo de Software.* s.l. : Pearson Educacion, 2000.
12. **LADDAD, Ramnivas. 2002.** *I want my AOP.* s.l. : JavaWorld, 2002.

13. **LARMAN, Craig. 2003.** *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* Segunda Edición. s.l. : PEARSON EDUCACIÓN, 2003. pág. 624. Vol. 1, Edición en Español. 8420534382.
14. **MEHNER, K. y WAGNE, A. 1999.** *"An Assessment Of Aspect Language Design".* Universidad de Paderborn : Position Paper in Young Researchers Workshop, GCSE ´99, 1999.
15. **MORENO NIETO, Juan Manuel.** *AspectJ en la Programación Orientada a Aspectos.* Escuela Técnica Superior de Ingeniería Informática. Universidad de Sevilla, España : s.n.
16. **PLEIAD. 2004.** [En línea] 2004. [Citado el: 13 de 03 de 2010.] http://pleiad.dcc.uchile.cl/.
17. **REINA QUINTERO, Antonia M. 2000.** *Visión General de la Programación Orientada a Aspectos.* Sevilla, España : Universidad de Sevilla, 2000.
18. **SAGE.** Sage. [En línea] [Citado el: 20 de 03 de 2010.] http://www.sage.es.
19. **TOP 100 RESEARCH FUNDATION.** [En línea] [Citado el: 16 de 03 de 2010.] www.truffle100.com.
20. **VALDÉS GONZÁLEZ, Jorge Luis. 2008.** *Propuesta de una guía metodológica de buenas prácticas orientada a la calidad del proceso de diseño de software.* Ciudad de la Habana : s.n., 2008.
21. **VINUESA, LUIS. 2007.** *Separación Dinámica de Aspectos independiente del Lenguaje y Plataforma mediante el uso de Reflexión Computacional.* 2007.
22. **WINBLAD, Ann L., EDWARDS, Samuel D. y KING, David R. 1990.** *Software Orientado a Objetos.* Argentina : ADDISON-WESLEY/DIAZ DE SANTOS, 1990. 0-201-60117-6.
23. **XEROX Corporation. 2003.** *The AspectJ(TM) Programming Guide.* s.l. : Palo Alto Research Center, 2003.

ANEXO 1: RESULTADOS DEL CUESTIONARIO

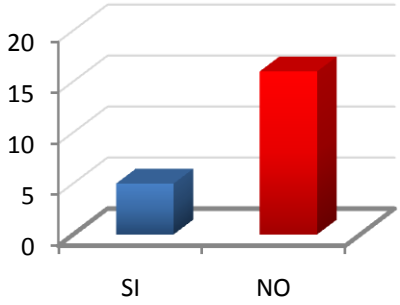
Conocimiento o no de lo que es la Programación Orientada a Aspectos:

PREGUNTA 1				
¿Conoce qué es la Programación Orientada a Aspectos (POA)?	SI	%	NO	%
	13	61.9	8	38.1



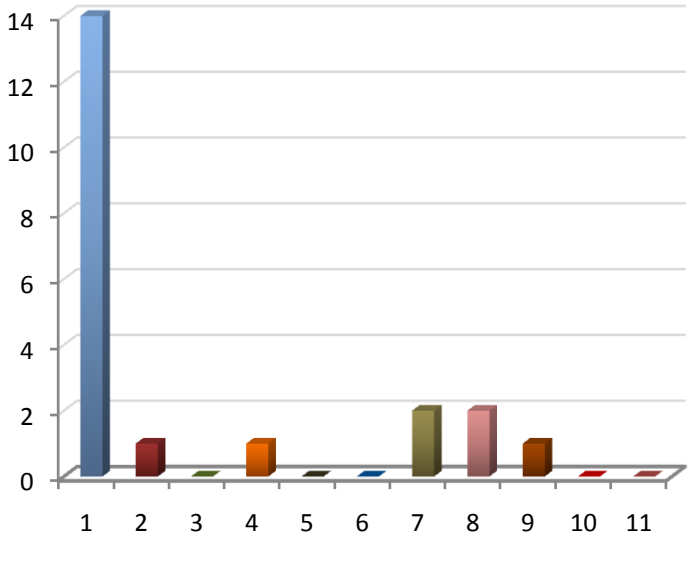
Análisis sobre experiencias de trabajo con la POA en proyectos productivos de la UCI:

PREGUNTA 3				
¿Ha tenido experiencia de trabajo con la POA?	SI	%	NO	%
	5	23.8	16	76.19



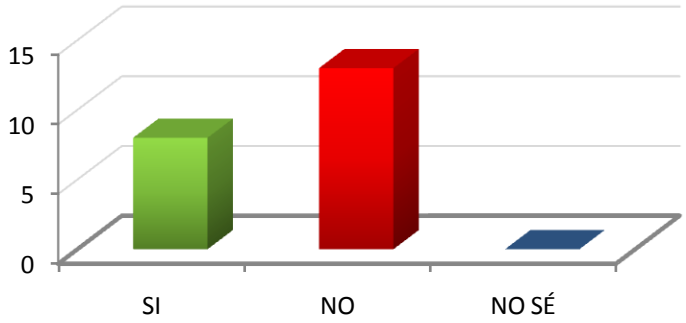
Análisis sobre el nivel de experiencia adquirido por los encuestados

Niveles de experiencias	Cant.	%
0	14	66.7
1	1	4.76
2	0	0
3	1	4.76
4	0	0
5	0	0
6	2	9.52
7	2	9.52
8	1	4.76
9	0	0
10	0	0



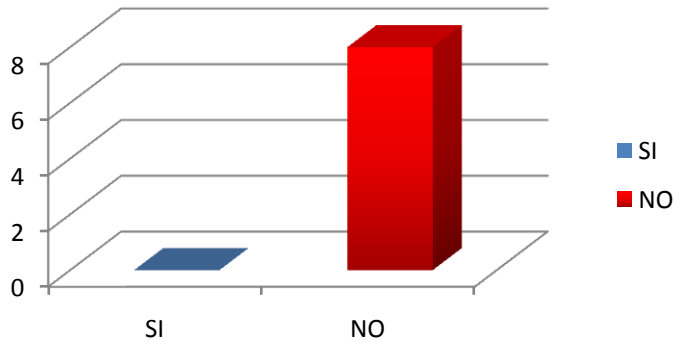
Empleo de la POA en los proyectos encuestados:

PREGUNTA 4						
¿En su proyecto se emplea la POA?	**SI**	**%**	**NO**	**%**	**NO SÉ**	**%**
	8	38.1	13	61.9	0	0



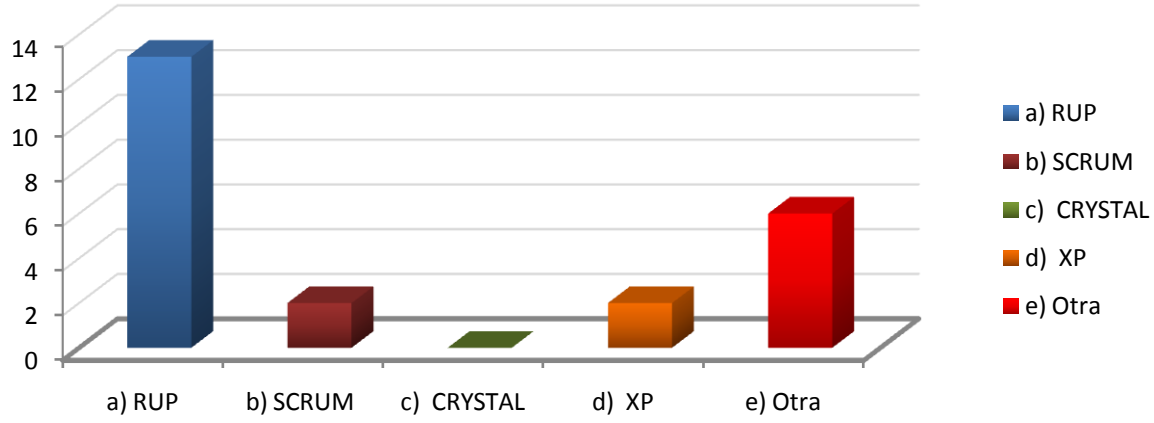
Análisis sobre el empleo de alguna guía o metodología para la aplicación de la POA en los proyectos donde se usa este paradigma:

PREGUNTA 4.1						
De emplearse la POA en su proyecto diga si para ello emplea alguna guía o metodología que le garantice su adecuada aplicación.	**SI**	**%**	**NO**	**%**	**NO SÉ**	**%**
	0	0	8	100	0	0



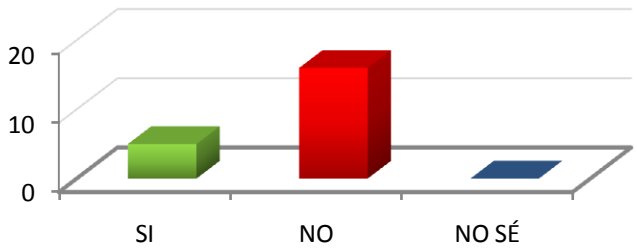
Análisis sobre el empleo de metodologías de desarrollo de software en los proyectos encuestados:

PREGUNTA 5		
¿Qué metodología de desarrollo se utiliza en su proyecto?	**Cant.**	**%**
a) RUP	13	61.9
b) SCRUM	2	9.52
c) CRYSTAL	0	0
d) XP	2	9.52
e) Otra	6	28.6



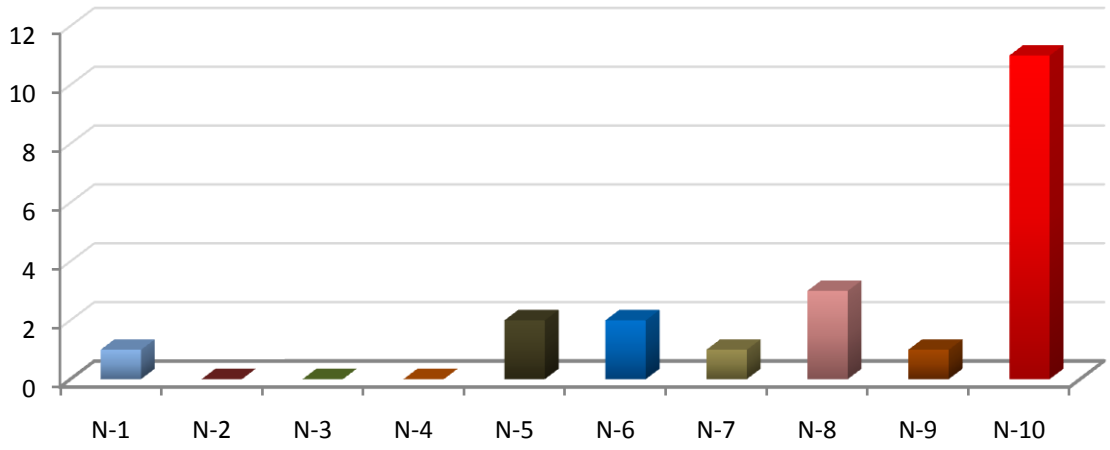
Análisis sobre el criterio de los encuestados en relación a la legibilidad o no del código de negocio cuando este ha sido mezclado con conceptos que entrecruzan el sistema, como logging, validación, manejo de excepciones, entre otros:

PREGUNTA 6						
¿Considera que estos elementos hacen menos legible y más difícil de mantener el código?	**SI**	**%**	**NO**	**%**	**NO SÉ**	**%**
	5	23.8	16	76.19	0	0



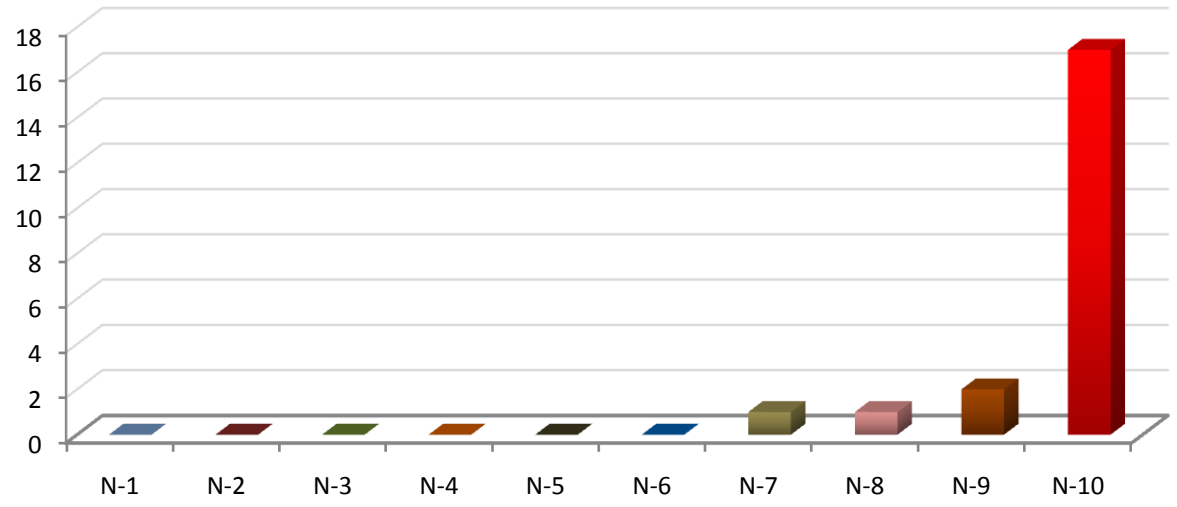
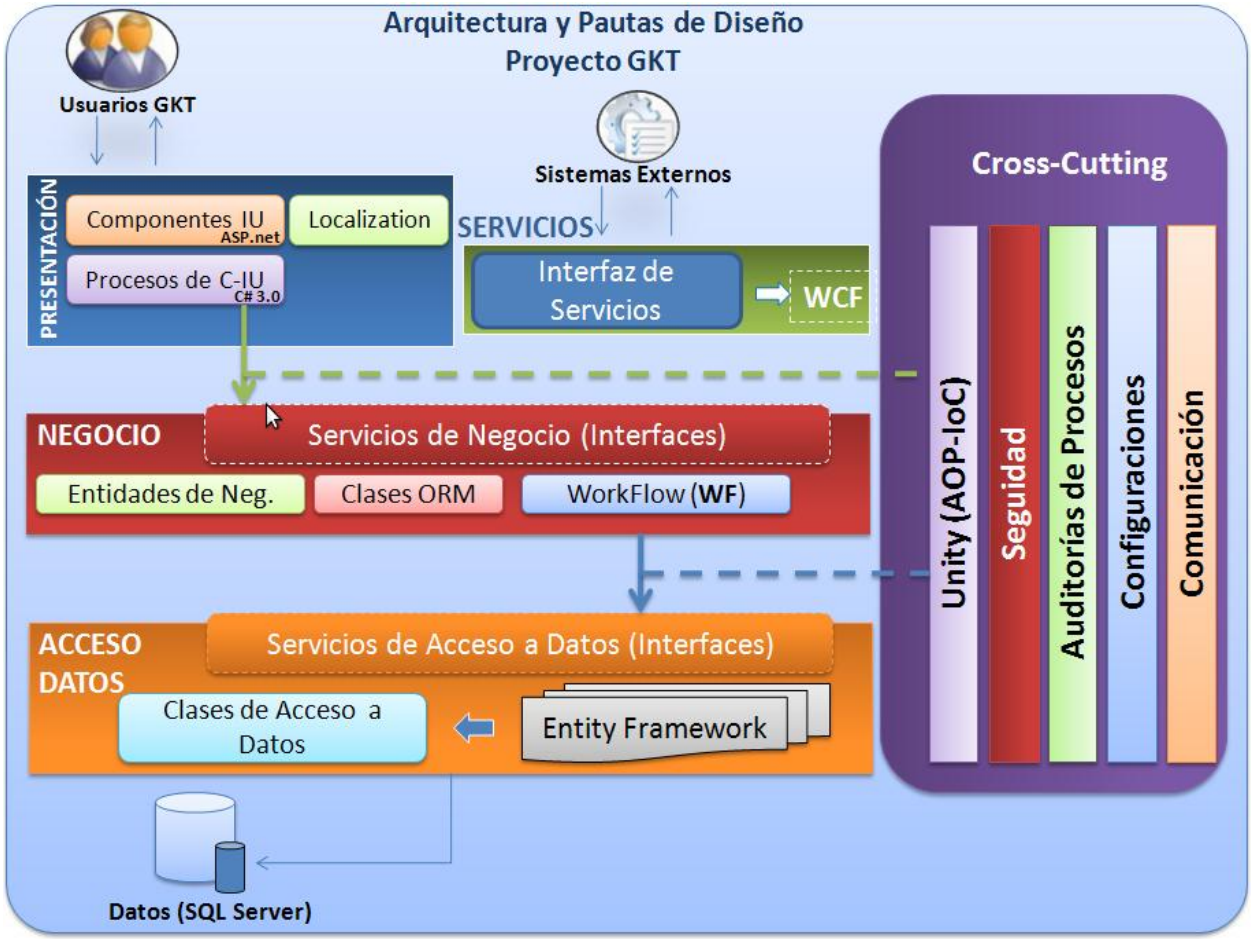
Análisis sobre el nivel de importancia que le otorgan los encuestados a poder que aislar los conceptos de entrecruzamiento del sistema y la lógica de negocio:

PREGUNTA 7			
	Nivel	**Cant.**	**%**
¿Qué nivel de importancia le otorga en una escala del 0 al 10, el tener que implementar de manera aislada los conceptos anteriormente expuestos en el punto 6 y que mediante algún mecanismos de entrelazado se logre componer tanto la lógica funcional del sistema como los conceptos aislados para alcanzar una mejor legibilidad y facilidad de mantenimiento del código?	N-1	1	4.762
	N-2	0	0
	N-3	0	0
	N-4	0	0
	N-5	2	9.524
	N-6	2	9.524
	N-7	1	4.762
	N-8	3	14.29
	N-9	1	4.762
	N-10	11	52.38



Análisis sobre el nivel de importancia que le otorgan los encuestados a contar con una guía de principios de buenas prácticas que apoye el proceso de introducción de la POA en el desarrollo de sistemas:

PREGUNTA 7			
	Nivel	**Cant.**	**%**
¿Qué nivel de importancia en una escala del 0 al 10, le otorga a contar con una Guía de Principios de Buenas Prácticas que garantice la adecuada introducción de la POA en el desarrollo de su proyecto si el mismo empleara este paradigma?	N-1	0	0
	N-2	0	0
	N-3	0	0
	N-4	0	0
	N-5	0	0
	N-6	0	0
	N-7	1	4.762
	N-8	1	4.762
	N-9	2	9.524
	N-10	17	80.95



ANEXO 2: EJEMPLO DE ARQUITECTURA SOBRE PLATAFORMA .NET QUE IMPLEMENTA LOS PATRONES IOC E IOD.