Administración de procesos de reingeniería de software orientada a aspectos

Isabel Espinoza-Espinoza, Ulises Juárez-Martínez

División de Estudios de Posgrado e Investigación, Instituto Tecnológico de Orizaba
isabel.espinoza83@gmail.com, ujuarez@ito-depi.edu.mx

Resumen. Actualmente existen muchos sistemas heredados obsoletos y con un arduo trabajo de mantenimiento para los cuales es necesario reconstruirlos o desecharlos. Ante la necesidad de migrar un sistema a un nuevo enfoque, es factible aplicar un proceso de reingeniería que permita obtener un sistema que satisfaga los requerimientos y que se realice bajo los enfoques y paradigmas actuales que mejoran la calidad del producto. Recientes enfoques de programación otorgan grandes beneficios, como lo es el paradigma orientado a aspectos (OA), éste provee un mejor análisis y comprensión del sistema, mayor nivel de abstracción, mayor legibilidad en el código, alto nivel de reutilización, así como también facilita la adaptación y el mantenimiento. Es de suma importancia administrar adecuadamente los procesos de reingeniería para que el nuevo sistema cumpla con requerimientos de calidad. Bajo dichas premisas este trabajo presenta una propuesta de administración de procesos de reingeniería OA alineada a MoProSoft.

Palabras clave: Reingeniería de software, orientación a aspectos, desarrollo de software basado en componentes, MoProSoft.

1. Introducción

Ante la creciente necesidad de ser más competitivos en la industria del software, especialmente con estrategias que promuevan su reutilización y escalabilidad, es necesario no sólo transferir conocimiento de nuevos paradigmas, además es imperante que las empresas de desarrollo de software cuenten con elementos tangibles de casos que les permitan formar una base de insumos reutilizables con el objetivo de incrementar su productividad. Con base en la amplia literatura reportada, en términos generales, la capacidad de adaptación y mantenimiento de los sistemas de software, la reutilización de componentes, así como la evolución en general, se ve favorecida con el modelo OA. La problemática esencial radica en cómo aplicar actividades de mantenimiento a insumos y/o productos de software que soporten las mejores prácticas de adaptación. Dado que la OA favorece este tipo de actividades, se plantea una propuesta de administración de procesos de reingeniería de software que permita lograr principalmente tres atributos de calidad de los sistemas en general: la adaptación, mejorar

Tendencias Tecnológicas en Computación
Research in Computing Science 66, 2013, pp. 95-108
Paper received 26-04-2013 and accepted 01-08-2013

el mantenimiento y robustecer la escalabilidad; además de favorecer la adopción del paradigma OA en la industria.

El presente trabajo está alineado a la iniciativa de mejora continua en la industria de desarrollo de software, consiste en el análisis y selección de técnicas, métodos y herramientas óptimas para la aplicación de estrategias de reingeniería de software OA. Con dicha finalidad se ha tomado un caso de estudio real de la industria (sección 4), actualmente en operación, el cual requiere alta capacidad de adaptación y mantenimiento pero al mismo tiempo presenta problemas fuertes de código invasivo y disperso. Los insumos, así como el producto final, deben ser adaptables ante las diferentes necesidades, siendo regulado con lineamientos fiscales cada vez que la institución acreditada lo solicite. Ante esta situación de adaptación, se requiere una solución independiente dedicada exclusivamente al proceso de negocio en cuestión, que sea modular y de fácil mantenimiento, con poca afectación en funcionalidad en los cambios que pudieran surgir tanto en la propia solución, como en otra que lo utilice. También es deseable evitar el volver a ejecutar pruebas a toda la funcionalidad, o causar nuevos errores al realizar cambios.

Este artículo está organizado como se indica a continuación. La sección 2 presenta los antecedentes, la sección 3 muestra los trabajos relacionados así como el análisis de las estrategias de reingeniería OA, el caso de estudio se presenta en la sección 4. A continuación, en la sección 5 se presenta la metodología enfocada en la administración de procesos de reingeniería OA y la discusión de la misma en la sección 6. Finalmente se dan las conclusiones y el trabajo a futuro en las secciones 7 y 8 respectivamente.

2. Antecedentes

Este trabajo se fundamenta en cinco elementos principales que se exponen a continuación.

2.1 Reingeniería de Software

De acuerdo a [1] las tareas de reingeniería se hacen presentes cuando se requieren cambios en un sistema de software existente con la intención de cambiar su estructura o fundamento técnico, tales como dependencias del sistema operativo, al necesitarse cambios en: la plataforma, bases de datos y procesamiento de transacciones del sistema. También se necesita cuando hay documentación incompleta, baja comprensión de código fuente o dependencias complicadas entre los módulos. Al mismo tiempo se da cuando la tecnología se vuelve obsoleta o ya no es la deseada, de tal forma que resulta necesario realizar una migración como consecuencia del cambio de lenguaje de programación. El proceso de reingeniería consiste en reconstruir un sistema, creando un producto con una funcionalidad nueva, un mejor rendimiento y fiabilidad, y un mantenimiento mejorado.

Considerando que la OA aporta mayores beneficios en el desarrollo de software, un proceso de reingeniería de software OA debe permitir al mismo tiempo la detección de aspectos.

2.2 Desarrollo de Software OA

El desarrollo de software OA tiene como objetivo hacer los sistemas más fáciles de mantener y reutilizar al ofrecer mecanismos para encapsular requerimientos no funcionales [2]. Este desarrollo está centrado en la separación de asuntos (concerns); un asunto es algo de interés para un stakeholder o para un grupo de ellos (tales como asuntos funcionales, de calidad de servicios, de políticas, organizacionales, entre otros). En [2] se establecen las siguientes actividades para el desarrollo de software OA:

- Ingeniería de requerimientos orientada a asuntos, en la que se identifica el conjunto de requerimientos funcionales y no funcionales del sistema.
- En el diseño se identifican y diseñan los aspectos, especificando los lugares dónde éstos se entrelazan con las funcionalidades del sistema (figura 1).
- En la fase de implementación se codifican las funcionalidades centrales y los aspectos.
- La fase de verificación y validación busca demostrar a través de un conjunto de pruebas que el sistema reúne las especificaciones y las necesidades reales de los stakeholders.

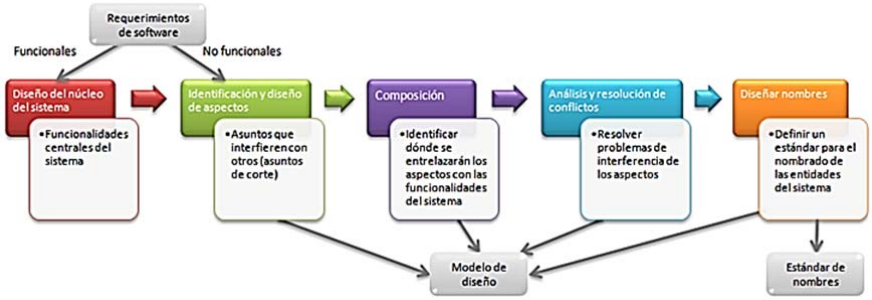

Figura 1. Proceso de diseño OA, adaptado de [2].

A diferencia de un desarrollo de software OA, en la fase de mantenimiento del desarrollo de software Orientado a Objetos (OO) cuando los requerimientos iniciales cambian, se invierte mucho tiempo analizando cuáles partes de los componentes cambian, incluso se omiten algunas secciones que tienen que cambiar, o en otro caso se introducen errores en el sistema, además existe la posibilidad de introducir código

enredado (tangling) o disperso (scattering); el primero se presenta cuando un módulo en un sistema incluye código que se implementa en diferentes requerimientos del sistema; el segundo se da cuando la implementación de un asunto se dispersa a través de muchos componentes en un programa. Al desarrollar software OA, los aspectos se comprenden, reutilizan y modifican de forma independiente, contrarrestando la situación anterior.

2.3 Programación orientada a aspectos

La programación OA es una técnica que permite la abstracción y encapsulación de asuntos de corte (aspectos) proporcionando nuevos niveles de modularidad. Trabaja sobre paradigmas existentes (estructurado y OO) gracias al entrelazado se da la interacción entre aspectos y objetos, brindando así la infraestructura necesaria para producir sistemas completos.

Los lenguajes OA trabajan con: eventos identificables en la ejecución de un programa (para clases, métodos y campos) llamados puntos de unión, conjuntos de puntos de unión conocidos como cortes y avisos que son el código que se ejecuta al identificar un punto de unión, los avisos son módulos reutilizables. Adicionalmente los lenguajes OA soportan asuntos identificables como logging, tracing, profiling, aplicación de políticas, optimización, seguridad, autenticación, autorización y administración de transacciones. La evolución de un sistema se da de forma estática al modificar la estructura de las clases y dinámica al detectar eventos en tiempo de ejecución y ejecuciones de métodos y acceso a campos.

2.4 Ingeniería de Software basada en componentes

Considerando que el caso de estudio dará como resultado un software que se integre a los sistemas de los clientes o a un ERP de la empresa, surge la necesidad de emplear un desarrollo de software basado en componentes. La ingeniería de software basada en componentes es el proceso bajo el cual: se definen, implementan, integran o componen dentro de un sistema, un conjunto de componentes independientes con bajo acoplamiento y alta cohesión [2].

Los elementos esenciales que cumplirá un sistema de software basado en componentes son: independencia, estar construidos bajo un estándar, contar con un middleware que brinde soporte a la integración de componentes y realizarse bajo un proceso de desarrollo orientado hacia la ingeniería de software basada en componentes.

Los componentes al ser independientes no interfieren con otras operaciones, ocultan los detalles de implementación, por lo que si la implementación de algún componente cambia, no se afecta el resto del sistema. La comunicación entre componentes se da por interfaces bien definidas, de tal forma que si se le da mantenimiento a una interfaz, un componente será remplazado por otro que brinde funcionalidades mejoradas o adicionales en caso de que sea necesario el remplazo. La infraestructura de los

componentes provee una plataforma de alto nivel que reduce los costos en el desarrollo de aplicaciones.

2.5 Administración de proyectos específicos

MoProSoft es un modelo de procesos para la industria de software mexicano, su objetivo es fomentar la estandarización incorporando las mejores prácticas en gestión e ingeniería de software, brindando un modelo basado en estándares internacionales que es fácil de entender y de aplicar, no es costoso en su adopción y busca ser la base para alcanzar evaluaciones exitosas con otros modelos y normas como ISO 9000, 2000 o CMM V 1.1 [3].

La estructura de procesos de MoProSoft dentro de la categoría Operación cuenta con un proceso llamado Administración de proyectos específicos, tiene como propósito establecer y llevar a cabo sistemáticamente las actividades que logren cumplir con los objetivos de un proyecto en tiempo y costo esperados [3].

Sus objetivos son:

- Lograr los objetivos del proyecto en tiempo y costo a través de la coordinación y el manejo de recursos.
- Mantener informado al cliente a través de reuniones de avance del proyecto.
- Atender las solicitudes de cambio del cliente mediante la recepción y análisis de estas.

El logro de los objetivos mencionados estará dado por la realización de las actividades mostradas en la figura 2, las cuales son incorporadas dentro del presente trabajo con el fin de administrar adecuadamente los procesos de reingeniería OA.

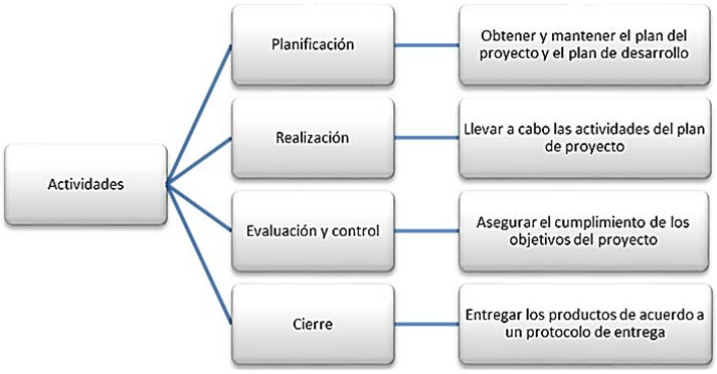

Figura 2. Administración de proyectos específicos.

3. Trabajos relacionados

En [4] se plantea que algunas veces muchos documentos y código fuente se pierde, conservando solo los archivos binarios ejecutables, los cuales no se pueden actualizar sin documentos o código fuente que sustente los cambios. Para ello se realizó un trabajo de reingeniería a sistemas heredados que contaban solo con el archivo de código binario ejecutable, utilizando para ello técnicas de reflexión y descompilación para generar diagramas de clase.

En [5] se aborda el modelo de diseño para la orientación a aspectos (DMAsp Design Model for Aspect Orientation), el cual se basa en el enfoque de aspectos y tiene soporte computacional de la ReJAsp (Reingeniería con Java usando AspectJ). Considera tres fases: entender la funcionalidad del software, abordar los asuntos de corte y comparar el software OO con el software OA.

En [6] se plantean técnicas como la identificación de interfaces candidatas, minería de aspectos, minería combinada, análisis identificado y recomposición, tales técnicas para llevar a cabo la migración hacia aspectos. El estudio indica que la migración de interfaces candidatas a aspectos tiene un impacto limitado en el tamaño de la descomposición principal, pero al mismo tiempo produce una mejora en el código modularizado. Desde el punto de vista de atributos de calidad externos, la modularización de la implementación de las interfaces de cortes en punto simplifica la compresión del código fuente.

En [7] se hace hincapié en la dificultad de identificar y comprender cuáles son las porciones de código directa o indirectamente afectadas por aspectos y cuáles se ejecutan en realidad en un punto dado en el código, para ello se propone un grafo de flujo de control de aspectos inter procedimentales, que representa las interacciones entre los aspectos y los componentes OO. También se utiliza un modelo métrico para evaluar el nivel de interferencia que los aspectos pueden introducir en un sistema OA.

En [8] se presenta un enfoque de ingeniería inversa a Interfaz Gráfica de Usuario (GUI) que resulta innovador gracias a su uso eficiente, la instrumentación no intrusiva y la técnica de análisis dinámico; se utiliza AspectJ para monitorizar y registrar la información de la GUI de una aplicación Java.

En [9] se menciona que la mayoría del software existente no fue diseñado para adaptación, por ello se propuso un enfoque basado en modelos para presentar la adaptación dinámica a los sistemas heredados no adaptativos, mientras se garantizan sus propiedades de mantenimiento.

En [10] se describe la reingeniería de una biblioteca OO llamada Framework de edición gráfica. Se encontraron mejoras en la modularidad pero se observó la necesidad de herramientas de automatización y recomposición para apoyar la reingeniería OA. Los resultados indicaron la reducción de la complejidad y del acoplamiento, pero sólo a un pequeño grado, se asume que el resultado estuvo limitado al corto tiempo de ejecución.

3.1 Análisis de estrategias de reingeniería OA

La tabla 1 muestra un análisis comparativo de las técnicas mencionadas previamente que los autores proponen utilizar para la resolución de problemas similares al del caso de estudio en cuestión.

Tabla 1. Técnicas de reingeniería OA relacionadas con el proyecto

Técnica	Objetivo	Ventaja	Desventaja
Identificación de interfaces candidatas a aspectos [6]	Identificación de aspectos a partir de: • Interfaces de paquetes externos. • Relación de nombres de las interfaces implementadas. • Agrupamiento de métodos de acuerdo a la relación de llamadas.	Probado en proyectos con más de medio millón de líneas de código. Mejora la comprensión de código, mantenimiento y modularidad.	Deben existir interfaces bien definidas e implementadas, así como seguir estándares de nomenclatura. Impacto limitado en la reducción del tamaño del código.
Análisis identificativo [6]	Identificación de asuntos de corte a través de la nomenclatura.		
Recomposición [6]	Conversión de interfaces candidatas en aspectos.		
Modelo métrico [7]	Evalúa la interferencia que los aspectos introducen en un sistema OA.	Se analiza la alteración estática o dinámica en los componentes base a partir de la interferencia de los aspectos. Considerando la magnitud de interferencia se decide si los componentes están bien diseñados, si necesitan ser reestructurados o se les debe aplicar reingeniería.	Se necesita una herramienta automatizada para evaluar el grafo de flujo de control de aspectos inter procedimentales con base a los criterios propuestos.
Modelo de conducción para solventar la adaptación dinámica de sistemas heredados no adaptativos [9]	Aseguramiento de la adaptación dinámica a través de: Análisis de requerimientos: selección de las propiedades que necesita satisfacer el	Modelo simple.	No presenta técnicas innovadoras.

Técnica	Objetivo	Ventaja	Desventaja
	software. Análisis y diseño del modelo: se generan diagramas de estado a partir del código fuente y se verifican. Se crea y verifica un modelo de adaptación. Implementación: usa clases en Java que implementen el comportamiento requerido en el modelo de adaptación, se utiliza OA para incorporar estas clases y su implementación.		
Recomposición OA de forma semi automática [10]	Recomposición de código usando renombre, composición, relocalización, descomposición y abstracción de elementos tales como identificadores, métodos y clases. Extrae características en aspectos y parte de código en avisos.	Los resultados basados en las métricas aplicadas indican que se redujo la complejidad y el acoplamiento.	Algunos autores obtuvieron resultados desfavorables en la aplicación de recomposición: Reducción de 10% de código sólo después de 7 meses de recomposición (código de 140 Kloc de Java). En sistemas donde se modifica el código de 20 a 25% esto puede ser contraproducente. Gran cantidad de tiempo invertido en el análisis y rediseño, sobre todo al identificar aspectos.
Métricas empleadas [10]: Aopmetrisc (herramienta para código OO y OA). Tamaño (LOC). Complejidad de operaciones por módulo. Acoplamiento en la llamada de métodos. Falta de cohesión en operaciones. Dependencia de paquetes. OA: verifica correspondencia de asuntos de corte con aspectos.	Obtiene indicadores de la calidad final del sistema al que se le aplicó recomposición de código. Se uso AJDT en eclipse y JUnit para pruebas de unidad.		

4. Caso de estudio

El caso de estudio se centra en el proceso de negocio de facturación electrónica, está desarrollado bajo el paradigma OO, en el sistema operativo Windows 7 utilizando el lenguaje C#.

Actualmente existen dos versiones del producto desarrollado, para ellas se requirieron actualizaciones en los requerimientos solicitados por la institución que establece lineamientos y políticas para dicho producto de software, sin embargo, existe documentación incompleta e inconsistente para fundamentar dicho trabajo. Nuestro caso de estudio corresponde a la tercera versión del producto de software, el cual considera todas las actualizaciones pertinentes.

En el análisis de información del sistema actual, se ha detectado la ausencia del diagrama de clases cuya vista estática es de suma importancia para entender en términos de objetos cómo está constituido el sistema y cómo se relacionan sus elementos, también hay inconsistencias en los esquemas de bases de datos respecto a los cambios requeridos recientemente por la institución reguladora.

En el análisis del código fuente, se identificaron casos de código invasivo y disperso. Por ejemplo, el código de la figura 3 corresponde a la incorporación de impuestos trasladados a un comprobante fiscal. El código invasivo se observa en las líneas 4 a la 8, debido a que el impuesto trasladado está fijo, ocasionando fuertes problemas de mantenimiento. Ante un cambio deben actualizarse directamente las líneas de código correspondientes en todos los módulos involucrados.

En el código fuente también se observó que los componentes que conforman el sistema actual han sido desarrollados como "paquetes" que agrupan clases relacionadas por la funcionalidad que les corresponde, sin embargo, algunos componentes son altamente acoplados y bajamente cohesivos al depender entre sí y no modularizarse adecuadamente.

```
1  List<AdesoftSerializador.CFD32.ComprobanteImpuestosTraslado> MyTraslado=
2  new List<AdesoftSerializador.CFD32.ComprobanteImpuestosTraslado>();
3  AdesoftSerializador.CFD32.ComprobanteImpuestosTraslado MyIvaT=
4  new AdesoftSerializador.CFD32.ComprobanteImpuestosTraslado(){
5          importe=decimal.Parse("16.00"),
6          impuesto=AdesoftSerializador.CFD32.ComprobanteImpuestosTraslado.IVA,
7          tasa=decimal.Parse("16.000000")
8  };
9  Mytraslado.Add(MyIvaT);
```

Figura 3. Código invasivo

Considerando lo anterior, se ha identificado el conjunto de actividades necesarias para proponer una metodología que brinde los lineamientos a seguir para un proceso de reingeniería OA, para ello se han analizado las técnicas OA (sección 3.1) incorporando aquellas que son viables a esté proyecto.

Dicha metodología considera la concepción del nuevo sistema identificando, separando y encapsulando los asuntos de corte, a fin de tener un código no disperso, no invasivo, entendible y reutilizable, con ello será más fácil y eficiente su mantenimiento pese a los cambios que se presenten, facilitando así la evolución del sistema. Así

también se considera la modularización adecuada de las funcionalidades del sistema a fin de producir componentes independientes y cohesivos.

La metodología incluye la documentación de artefactos de software OA para la reingeniería: ingeniería inversa, arquitectura, análisis, diseño e implementación.

5. Metodología propuesta

A diferencia de otras aproximaciones de reingeniería OA, la metodología propuesta utiliza el Enfoque de Temas [11] que indica lineamientos robustos para analizar y diseñar software OA. También se utiliza la separación de asuntos simétrica, que permite separar los aspectos de sus propias funcionalidades y de la funcionalidad central del sistema para una mejor modularidad, al final se integran los aspectos con las funcionalidades generando el sistema completo. Se ha integrado ASAAM (Método de Análisis Aspectual de Arquitectura de Software) [12] para evaluar los componentes resultantes.

Nuestra propuesta integra además una cuidadosa selección de técnicas (sección 3.1) viables para robustecer su aplicación práctica, efectiva y tangible. Incluso no se requiere de la adquisición de herramientas o el aprendizaje de técnicas sofisticadas que repercutan en el costo y tiempo de realización del proyecto a diferencia de las técnicas descartadas. Se excluyen además aquellas que son dependientes de algún lenguaje de programación.

A continuación se presenta la metodología que comprende la administración de procesos para cumplir con el propósito del caso de estudio.

Planificación

- Definición del plan del proyecto [3].
- Definición del plan de desarrollo [3].
- Definición del plan de verificación y validación [3].
- Definición de acciones correctivas y preventivas [3].

Análisis y diseño

- Actualización de los requerimientos funcionales, no funcionales y las restricciones del sistema, a partir de la información recabada del sistema actual OO y de los cambios requeridos.
- Aplicación de ingeniería inversa para la obtención de los modelos faltantes e incompletos (en nuestro caso se utiliza la herramienta CASE Enterprise Architect).
- Adecuación del modelado recuperado (paso anterior) de acuerdo al análisis y diseño OA del Enfoque de Temas [11].
- Identificación de aspectos [6] a partir del análisis del código perteneciente al sistema actual.
- Diseño de componentes de la aplicación.
- Registro del reporte de actividades [3].
- Revisión del plan del proyecto y plan de desarrollo [3].

Implementación

- Migración de código fuente (en nuestro caso de C# a Java).
- Recomposición de código fuente del sistema actual OO al detectarse código enredado, duplicado o disperso [6].
- Codificación del producto. Se considera el lenguaje AspectJ ya que es el más utilizado en la industria, su modelo de puntos de unión es el más expresivo y robusto y es utilizado por otros lenguajes OA, adicionalmente se cuenta con mayor documentación y soporte que respalde su uso. Para trabajar con AspectJ se utilizará el IDE Eclipse.
- Registro del reporte de actividades [3].

Pruebas

- Evaluación de los componentes construidos. Se consideran las reglas heurísticas de ASAAM para identificar si los componentes son cohesivos, compuestos, enmarañados o mal definidos. ASAAM cubre la evaluación de la interferencia que los aspectos introducen a un sistema OA sugerida por [7] al identificar la necesidad de reestructurar los componentes o aplicarles reingeniería en caso de que estos hayan sido concebidos inadecuadamente. Con ASAAM también se logran satisfacer los objetivos de las métricas empleadas por [10].

Cierre

- Actualización del reporte de actividades [3].
- Registro del seguimiento al proyecto [3].
- Evaluación del plan del proyecto [3].
- Evaluación del plan del desarrollo [3].

Las actividades correspondientes al modelo propuesto por [9] se cubren totalmente con la ejecución de la metodología propuesta.

Finalmente para comprobar la mejora en el mantenimiento, se planea la definición de cambios en el nuevo sistema OA a fin de verificar su impacto. Si surgen modificaciones por la institución reguladora del producto de software, se atenderán dichos cambios verificando la facilidad de mantenimiento, considerando para ello la cantidad de tiempo y esfuerzo invertido utilizando las métricas de COCOMO [2].

6. Discusión

La aplicación de ésta metodología guía a los interesados en la ejecución de un proceso de reingeniería OA, proporcionando una guía puntual, ordenada y sencilla.

Hasta este momento se han observado mejoras de consistencia y mantenimiento a aplicar en el sistema actual que será reconstruido; las mejoras ya son tangibles al identificar código invasivo en el sistema (sección 4) alto acoplamiento y baja cohesión en los componentes desarrollados; lo anterior indica la necesidad de completar la

ejecución de la propuesta planteada. Los resultados de este proyecto servirán para promover la adopción del paradigma OA en la industria a través de la difusión de las lecciones aprendidas durante la realización del caso de estudio.

Se observó que los problemas planteados en la literatura reportada surgieron de la necesidad de aplicar procesos de reingeniería bajo situaciones comunes como: falta de documentación, mejora de procesos, recomposición de código, entre otras; sin embargo, para la aplicación de técnicas tanto en reingeniería como en la incorporación del paradigma OA, en su mayoría los problemas se plantean con casos de estudio que fueron concebidos para comprobar la efectividad de las técnicas empleadas. Esta metodología se diferencía de las demás al contar con un caso de estudio real de la industria, que reúne las condiciones necesarias para incorporar los cuatro enfoques: las mejores prácticas de reingeniería, la promoción de la adopción del paradigma OA, el desarrollo de software basado en componentes y la administración de procesos de reingeniería alineada a MoProSoft. Todo ello generará conocimiento sobre los cuatro enfoques y permitirá la toma de decisiones principalmente para la adopción del paradigma OA en la industria de software y con ello la mejora de procesos de software y la evolución del software.

La industria de desarrollo de software se verá beneficiada por las siguientes razones:

- No es necesario invertir gran cantidad de tiempo en el aprendizaje del conjunto de técnicas y actividades planteadas.
- La curva de aprendizaje necesaria para la adopción de un nuevo enfoque dependerá de las características del equipo de trabajo, sin embargo, se asume que está no debería ser alta para personas que se dediquen al desarrollo de software.
- Este caso de estudio brinda pruebas tangibles que indican que en casos reales funciona una nueva forma de trabajo, descartando la idea de: “la teoría no siempre resulta cierta en la práctica”.
- No es necesario invertir costos elevados para ejecutar la metodología.
- La difusión de las lecciones aprendidas hará que las empresas conozcan los resultados, y por lo tanto el enfoque OA ya no resulte desconocido.

7. Conclusiones

Debido a que es deseable que las empresas de desarrollo de software generen productos escalables, que tengan una larga vida útil gracias a un mantenimiento eficaz, es deseable contar con elementos que indiquen un proceso de desarrollo de software efectivo. El principal aporte de este trabajo es la propuesta metodológica, la cual está robustecida gracias a la incorporación de estrategias viables sustentadas en el análisis de los trabajos relacionados (sección 3), contando también con el empleo de métodos, enfoques y paradigmas que resultan ser efectivos en problemas del contexto planteado. Actualmente esta metodología se ha trabajado en las fases de análisis y diseño.

Esta metodología es un elemento importante al indicar la línea a seguir para la aplicación de un proceso de reingeniería OA bajo situaciones similares, requiriendo un bajo costo y corto tiempo de aprendizaje para la ejecución de las actividades indicadas.

Finalmente este trabajo confirma la necesidad de demostrar de forma tangible los beneficios de la incorporación del paradigma OA en la industria.

8. Trabajo a futuro

Como parte de un proceso evolutivo de software, se ha trabajado en las fases de planificación, análisis y diseño planteadas en la metodología propuesta, cuyos artefactos resultantes serán refinados para proceder con la implementación. Como trabajo futuro se plantea la codificación del producto correspondiente al caso de estudio, posteriormente se utilizará ASAAM [12] para evaluar que el código final (componentes) no está enmarañado o disperso y que los aspectos existentes son los necesarios o si hubo abusos en la implementación de estos. Se realizará un comparativo que demuestre los beneficios del sistema construido contra el sistema actual, dicho comparativo será respaldado por un conjunto de pruebas como por ejemplo: el conteo de líneas de código que indicará las diferencias de tamaño de los sistemas, el análisis del registro del PSP (Proceso Personal de Software) cuya información reflejará tiempos invertidos en el proceso de desarrollo.

También se realizará la difusión de las lecciones aprendidas, las cuales servirán como guía para la adopción del paradigma OA bajo enfoques similares.

Agradecimientos

A la Dirección General de Educación Superior Tecnológica y al Consejo Nacional de Ciencia y Tecnología por su apoyo en la realización del presente trabajo de investigación.

Referencias

1. J. Borchers. Invited talk: Reengineering from a practitioner's view- a personal lesson's learned assessment. 15th European Conference on Software Maintenance and Reengineering, 2011, pp. 1-2.
2. I. Sommerville. Software Engineering. United Kingdom, Addison-Wesley, 2006.
3. H. Oktaba, C. Alquicira, A. Su, et al. Modelo de Procesos para la Industria de Software. MoProSoft. Versión 1.3, México D.F. Secretaría de Economía, 2005.

4. L. Chen, G. Wang, M. Xu and Z. Zeng. Reengineering of Java Legacy System Based on Aspect-Oriented Programming. Second International Workshop on Education Technology and Computer Science (ETCS), Wuhan, 2010, pp. 220-223.
5. H.A.X. Costa, P.A. Parreira, V. Vieira de Camargo and R.A. Dellosso. Recovering Class Models Stereotyped with Crosscutting Concerns. 16th Working Conference on Reverse Engineering, Lille, 2009, pp. 311-312.
6. M. Ceccato. Automatic Support for the Migration Towards Aspects. 12th European Conference on Software Maintenance and Reengineering, Athens, 2008, pp. 298-301.
7. M.L. Bernardi. Reverse Engineering of Aspect Oriented Systems to Support their Comprehension, Evolution, Testing and Assessment. 12th European Conference on Software Maintenance and Reengineering, Athens, 2008, pp. 290-293.
8. H. Samir and A. Kamel. Automated reverse engineering of Java graphical user interfaces for web migration. ITI 5th International Conference on Information and Communications Technology, Cairo, 2007, pp. 157-162.
9. J. Zhang and B.H.C. Cheng. Towards Re-engineering Legacy Systems for Assured Dynamic Adaptation. 29th International Conference on Software Engineering Workshops, Minneapolis, MN, 2007, pp. 10.
10. A. O'Riordan. Aspect-Oriented Reengineering of an Object-oriented Library in a Short Iteration Agile Process. Informatics (03505596); Vol. 35 Issue 4, 2011, pp. 499-511, 13.
11. S. Clarke, E. Baniassad. Aspect-Oriented Analysis and Design: The Theme Approach. Addison-Wesley, 2005.
12. B. Tekinerdogan. ASAAM: Aspectual Software Architecture Analysis Method. Working IEEE/IFIP Conference on Software Architecture, IEEE Computer Society, 2004, pp. 5-14.