Ingeniería Inversa y Reingeniería Aplicadas a Proyectos de Software Desarrollados por Alumnos de Nivel Licenciatura

Reyes JUÁREZ-RAMÍREZ, Guillermo LICEA, Alfredo CRISTÓBAL-SALAS
Facultad de Ciencias Químicas e Ingeniería, Ingeniería en Computación,
Universidad Autónoma de Baja California.
Calzada Tecnológico 14418, Mesa de Otay, Tijuana, Baja California, 22390, México.
{reyesjua, glicea, cristobal}@uabc.mx

RESUMEN

La ingeniería de software basada en la documentación no es una práctica común entre los desarrolladores de software principiantes; tal es el caso de los estudiantes de nivel licenciatura. Algunos motivos de este problema están relacionados con la formación académica y la falta de cultura en la práctica de un buen nivel de ingeniería. Para mitigar este problema, recomendamos que los estudiantes conozcan y practiquen otros procesos que les permitan valorar la importancia de aplicar un nivel adecuado de ingeniería en el desarrollo de software. Dos procesos representativos son la ingeniería inversa y la reingeniería. En este artículo se expone una estrategia para aplicar cada uno de estos procesos en la mejora de sistemas de software desarrollados por estudiantes. Dos conceptos que soportan a esta estrategia son: la vista de casos de uso y un grafo de relaciones.

Palabras clave: ingeniería inversa, reingeniería, documentación, casos de uso, grafo de relaciones.

1. INTRODUCCIÓN

La mayoría de los proyectos de software desarrollados por estudiantes de nivel licenciatura son construidos mediante una implementación directa, con poco nivel de análisis y diseño. Esta forma de desarrollo no demuestra un enfoque de ingeniería, y principalmente no incluye una documentación útil que facilite los tratamientos futuros. Bajo este enfoque de desarrollo, se corre el riesgo de generar costumbres inadecuadas en el estudiante sobre la organización y la calidad en el desarrollo de sistemas grandes [17].

El desarrollo de software con poco análisis y diseño, no es un mal hábito del estudiante sino una consecuencia de la estructura curricular. La mayoría de los programas educativos, para los primeros semestres del plan de estudios, tienen una estructura curricular orientada a la enseñanza de la programación [1]. Durante estos primeros semestres, la atención está orientada en generar la lógica de programación [9], y en conocer las características de un lenguaje. En los cursos de programación, aún cuando para aprobar el curso se les pide desarrollar aplicaciones de considerable funcionalidad, la atención no está centrada en la planeación del proyecto o sistema a desarrollar.

El primer contacto que los estudiantes tienen con un enfoque de ingeniería es cuando cursan una materia de ingeniería de software o alguna otra relacionada. En un curso de esta naturaleza, los estudiantes aprenden el proceso tradicional de "cascada" para el desarrollo de software, el cual incluye las fases de requerimientos, análisis, diseño e implementación [16]. Bajo la perspectiva del modelo en "cascada", en primera instancia, los estudiantes consideran que es un proceso laborioso, principalmente porque se requiere tiempo para la planeación, así como para la documentación del diseño y la implementación. Después de un curso como éste, aún cuando los estudiantes tienen una nueva visión del desarrollo de software, no son partidarios de aplicar una metodología rigurosa, todavía los proyectos desarrollados reflejan un mayor énfasis en la programación [9].

Para que los estudiantes valoren la importancia del nivel de ingeniería que se debe aplicar a un proyecto de software, y principalmente para que valoren la importancia de la documentación, es conveniente que tengan la oportunidad de practicar procesos opuestos o alternos al proceso tradicional de "cascada". Dos procesos a considerar son la reingeniería y la ingeniería inversa, los cuales pueden aplicarse sobre sistemas de software previamente desarrollados.

En este artículo se describe una estrategia para realizar la reingeniería y la ingeniería inversa de aplicaciones de software desarrolladas por estudiantes. En ambos casos se propone usar dos herramientas específicas: la vista de casos de uso y un grafo de relaciones entre artefactos. La estrategia propuesta incluye dos características básicas consideradas por Evans [8] para la enseñanza de la ingeniería de software a estudiantes de nivel licenciatura: (1) un análisis sencillo, (2) aplicabilidad en sistemas de gran escala.

Este artículo está estructurado como a continuación se indica. En la sección 2 se describen las definiciones básicas de reingeniería e ingeniería inversa. En la sección 3 se describen los mecanismos propuestos para estructurar los sistemas analizados: los casos de uso y el grafo de relaciones. En la sección 4 se describen los procedimientos propuestos para implementar la reingeniería y la ingeniería inversa. En la sección 5 se exponen dos casos de estudio y los resultados

preliminares de la aplicación de la estrategia propuesta. Finalmente, en la sección 6 se exponen las conclusiones.

2. DEFINICIONES BÁSICAS DE REINGENIERÍA E INGENIERÍA INVERSA

Para contextualizar los conceptos a utilizar de reingeniería e ingeniería inversa, primeramente describiremos los significados más amplios que se les han dado en otras disciplinas.

La ingeniería inversa ha sido ampliamente utilizada en el ámbito de la manufactura [6], y es considerada como el proceso de descubrir los principios tecnológicos de un dispositivo, un objeto o un sistema, mediante el análisis de su estructura, funcionamiento u operación [23]. Este proceso consiste en tomar una entidad por separado y analizar a detalle su funcionamiento, usualmente con el fin de construir un dispositivo o programa nuevo que hará lo mismo, pero sin copiar todos los aspectos del original.

Por su parte, la reingeniería es considerada como un rediseño radical de procesos en una organización [11], especialmente de procesos de negocios. En [11], Hammer y Champy argumentan que normalmente una empresa debe hacer reingeniería de los procesos existentes definiendo una serie de subprocesos.

Uso de la ingeniería inversa en el desarrollo de software

En el contexto del software, Chikofsky y Cross [6] establecen que la ingeniería inversa es el proceso de analizar un sistema para crear una representación del mismo, pero a un nivel más elevado de abstracción. Por otro lado, Hall [10] establece que la ingeniería inversa es un proceso que recorre hacia atrás el ciclo de desarrollo de software. Bajo el enfoque de Hall, es posible iniciar el proceso de abstracción a partir del código fuente y llegar hasta la fase de análisis, lo cual representa un flujo inverso al tradicional en el modelo de "cascada".

En la práctica, dos tipos de ingeniería inversa han sido considerados [19]: (1) basado en el código fuente, y (2) basado en el programa ejecutable. En el primer tipo, el código fuente está disponible; sin embargo, aspectos de más alto nivel no son conocidos, existe una documentación pobre o existe documentación pero no corresponde, por ejemplo, en términos de actualización. En el segundo tipo, no existe código fuente disponible, así que los esfuerzos se concentran en descubrir el correspondiente código fuente.

Uso de la reingeniería en el desarrollo de software

El principal enfoque que se le ha dado a la reingeniería es hacia las actividades de mantenimiento [19, 21]. En el contexto de mantenimiento, los objetivos de la reingeniería son: entendimiento (predecir), reparación (corregir), mejoramiento (perfeccionar), y evolución (adaptar). Otros usos que se la han dado a la reingeniería son: logro de los atributos de calidad, logro de objetivos de los requerimientos, optimización de la arquitectura, diseño de patrones, y optimización del desempeño bajo múltiples criterios. La reingeniería ha sido utilizada en los diferentes paradigmas de desarrollo [19], tales como: orientado a objetos, orientado a objetivos, orientado a agentes, y orientado a aspectos.

Implementación de mejoras en los sistemas

En el contexto de software, la evaluación de un sistema existente debe considerar cuatro elementos esenciales: (1) Qué transformar, (2) Por qué transformar, (3) Cómo transformar, (4) Sobre qué transformar. El primer elemento se refiere a definir el proceso a realizar. Los procesos más comunes son la ingeniería inversa, la redocumentación, la reestructuración y la modernización. El segundo elemento se refiere al objetivo de hacer la mejora. Tres de los objetivos más comunes son el mantenimiento, la reutilización y la integración. El tercer elemento se refiere a las técnicas utilizadas para hacer la transformación. Algunas técnicas útiles para hacer reingeniería son, entre otras, el análisis de conceptos, la visualización, la separación en unidades, y la derivación de grafos [7]. El cuarto elemento se refiere al artefacto o entidad de software sobre el cual se va a trabajar. Ejemplos de artefactos susceptibles de trabajar son los requerimientos o especificaciones de diseño, el diseño (modelado) y el código (implementación).

En el contexto de reingeniería, los grafos han sido utilizados como [7]: grafo de sintaxis, grafo de llamadas, grafo de estructuras, grafo de flujo de control, grafo de dependencia de datos y grafo de dependencia de programas. En este artículo se propone el uso de un grafo para representar los artefactos de software y las relaciones entre los mismos.

3. MECANISMOS PROPUESTOS PARA LA ESTRUCTURACIÓN DE SISTEMAS

En esta sección se describen dos conceptos que han sido tomados como base para hacer la ingeniería inversa y la reingeniería: vista de casos de uso y grafo de relaciones.

Los casos de uso

Los casos de uso expresan la funcionalidad del sistema vista desde la perspectiva del usuario [3]. Técnicamente los casos de uso son un mecanismo para representar detalles más específicos sobre los requerimientos funcionales que el sistema debe satisfacer. Para dimensionar la capacidad de un sistema, es conveniente identificar los casos de uso que contiene [5]. La robustez de un sistema también puede ser dimensionada mediante un análisis profundo de la descripción de casos de uso, especialmente a nivel de los flujos básicos y alternativos, así como el manejo de excepciones [4].

La importancia de los casos de uso es significativa no solo porque proporcionan una

descripción detallada de la funcionalidad de un sistema, sino también para porque permiten expresar la arquitectura del mismo. En el modelo 4+1 [15], los casos de uso representan la quinta vista de una arquitectura de software. La vista de casos de uso puede ser considerada como el centro en el que giran las otras cuatro vistas, debido a que es la primera que se crea dentro del ciclo de vida del desarrollo de software. Esta vista representa los escenarios donde se integran las otras vistas y define la razón de su existencia. Las primeras cuatro vistas son: (1) Vista lógica, (2) Vista de procesos, (3) Vista de implementación, y (4) Vista de desplegado. En la vista de casos de uso se consideran solamente los casos de uso que deben ser modelados, así que arquitectónicamente hablando, esta vista representa los requerimientos significativos en forma de escenarios.

A nivel de artefactos UML, las vistas se integran como sigue. La vista 1 se integra con las clases, objetos, paquetes y máquinas de estado. La vista 2 se integra con los diagramas de secuencia, diagramas de colaboración y diagramas de actividad. La vista 3 se integra con los componentes desarrollados. La vista 4 se integra con el desplegado de los componentes. Todos estos artefactos son considerados para la integración del grafo de relaciones propuesto [13, 14].

La vista lógica y la vista de implementación están estrechamente relacionadas con la funcionalidad del sistema. Estas vistas expresan cómo la funcionalidad es modelada e implementada [15]. En este artículo se pone especial atención en las vistas 1 y 3, así como en la vista de casos de uso. El proceso de ingeniería inversa tiene como objetivo identificar la vista de casos de uso a partir de la vista 3, mientras que la reingeniería se inicia a partir de la vista de casos de uso y tiene como objetivo aumentar las vistas 1 y 3.

El grafo de relaciones

Jürgen *et al.* [7] establecen que la reingeniería puede ser manejada definiendo técnicas aplicables sobre los artefactos de software. También platean que los artefactos pueden ser modelados mediante grafos, y que los servicios aplicados sobre los artefactos pueden ser vistos como manipulaciones de grafos.

Olsem **[19]** argumenta que, al analizar un sistema, si se parte del código fuente para generar el diseño e incluso los requerimientos, los artefactos generados en cada una de las fases con seguridad poseen relaciones entre ellos. Para documentar estas relaciones se requieren mecanismos que permitan expresar dicha información. En este artículo proponemos el grafo de relaciones descrito en [14] como mecanismo para documentar las relaciones entre los artefactos.

El grafo de relaciones es un mecanismo que permite conectar los artefactos de las distintas fases de desarrollo en base a las relaciones que guardan entre ellos. Los nodos del grafo representan artefactos y las aristas representan las relaciones entre dichos artefactos. Formalmente el grafo se define como $G = (V, E)$, donde V es el conjunto de vértices o nodos, y E es el conjunto de aristas. En un nodo del grafo se puede representar todo artefacto de software producido en cualquier fase del proceso de desarrollo. En este caso consideramos que el grafo de relaciones representa un medio útil para implementar la trazabilidad entre artefactos de software.

Al conectar los artefactos mediante aristas, es conveniente definir el tipo de relaciones que guardan entre ellos. Ramesh y Jarkes [20] han definido un conjunto de relaciones entre artefactos. Algunos de los tipos definidos son: satisface, justifica, describe, depende, y valida. En base a este conjunto de relaciones, al establecer una conexión entre dos artefactos, por ejemplo, *artefacto_1* y *artefacto_2*, es posible decir que el *artefacto_2* [satisface | justifica | describe | depende | valida] al *artefacto_1*. Por otro lado, Aizenbud-Reshef *et al.* [2] han definido un conjunto de relaciones más genéricas. Ejemplos de estos tipos de relaciones son: impuesta, inferida, manual, y computada. Una relación computada puede ser de dos tipos: derivación, análisis.

En el contexto del grafo de relaciones, una arista puede representar cualquier tipo de relación [14].

4. ESTRATEGIAS PARA IMPLEMENTAR LA INGENIERÍA INVERSA Y LA REINGNIERÍA

El proceso de ingeniería inversa

En el caso de los proyectos de estudiantes, el objetivo de hacer ingeniería inversa es generar una documentación del sistema que permita tratamientos futuros, tales como un incremento de funcionalidad, lo cual representa extender el tiempo de vida del mismo [12]. Debido a que el objetivo es aumentar la funcionalidad, entonces lo más conveniente es construir un modelo de casos de uso a partir del código fuente. En un sistema no documentado, los casos de uso pueden ser identificados en el código fuente, o a través de la interfaz gráfica de usuario (GUI, por sus siglas en inglés), tal como se propone en [18].

En [18], el modelo construido aparentemente se queda en una sola vista de la arquitectura. La estrategia que proponemos está basada en la identificación de casos de uso y la construcción de un grafo de relaciones que conecta los casos de uso con el código fuente. De esta forma se tendrá una combinación de la vista conceptual y la vista de implementación, teniendo la posibilidad de ir de la vista de casos de uso hacia atrás a los requerimientos, lo mismo hacia delante a las vistas lógica y de proceso; esto con el fin de completar todas las vistas de la arquitectura.

El procedimiento recomendado para hacer la ingeniería inversa a través de la GUI y en forma manual, es el siguiente:

Paso 1: Identificación de casos de uso en la GUI.

Paso 2: Identificación de las relaciones entre casos de uso.

Paso 3: Construir un grafo con los casos de uso y sus relaciones.
Paso 4: Localización de los archivos de código fuente correspondientes a los casos de uso y las entidades de código fuente derivadas. Agregar estos artefactos al grafo.
Paso 5: Identificación de las relaciones entre las entidades de código fuente. Conectar los artefactos de código fuente entre ellos y conectarlos con el nivel anterior del grafo.
Paso 6: Generación de la versión final del grafo de relaciones.

Si no se dispone de código ejecutable, la ingeniería inversa se realiza a partir del código fuente, para lo cual se utiliza un procedimiento parecido al anterior.

El proceso de reingeniería

El objetivo de hacer reingeniería es entender un sistema de software existente, a nivel de especificaciones, diseño e implementación [21]; esto con el fin de hacer una re-implementación para aumentar y/o mejorar la funcionalidad del sistema, el desempeño o la misma implementación.

El proceso de reingeniería que proponemos está orientado a sistemas que cuentan con cierto nivel de documentación. Un ejemplo del nivel de documentación es el que se logra bajo el método "MEDSOCAMP" para la enseñanza del diseño de software descrito en [13]. Bajo este enfoque, la documentación del sistema está integrada por el siguiente conjunto de artefactos: documento de requerimientos, descripción de casos de uso, prototipos de la interfaz de usuario por caso de uso, diagramas de casos de uso, diagramas de secuencia, diagramas de colaboración, clases, diagramas de clases, archivos de código fuente, archivos de código ejecutable y grafo de relaciones entre artefactos.

El procedimiento recomendado para realizar la reingeniería es el siguiente:
Paso 1: Evaluar la cantidad de funcionalidad expresada en los casos de uso.
Paso 2: Identificar la funcionalidad nueva.
Paso 3: Definir los nuevos casos de uso.
Paso 4: Hacer ingeniería hacia delante a partir de los casos de uso, completando el grafo de relaciones.
Paso 5: Generación de la versión final del grafo de relaciones.

Bajo este enfoque, incluso se pueden realizar incrementos o ajustes al código fuente, o se puede generar código fuente nuevo tratando de optimizar la funcionalidad. La arquitectura completa del sistema se expresa mediante el grafo de relaciones que conecta los casos de uso con el código fuente, pasando por los demás artefactos del modelado, teniendo también la posibilidad de llegar hacia atrás a los requerimientos. Cabe mencionar que el grafo se va completando a partir del paso 3.

5. CASOS DE ESTUDIO

En esta sección se describen dos casos de estudio, uno de ingeniería inversa y otro de reingeniería. Ambos casos fueron realizados por estudiantes de la carrera de Ingeniería en Computación. Los estudiantes que participaron en estos casos de estudio ya habían cursado la materia de Diseño de Sistemas de Información, donde aprendieron el modelado de sistemas bajo el método "MEDSOCAMP" [13].

Caso 1: Ingeniería Inversa

El proceso de ingeniería inversa, realizado en el semestre 2007-1, se aplicó a un sistema desarrollado en Visual Basic .Net. Este sistema denominado "D&EM" es un software que permite llevar el control diario de los consumos y gastos calóricos de una persona o una familia. Se eligió este sistema debido a la estructura del código fuente, ya que el entorno .Net opera con la filosofía de arquitectura de tres capas; por lo tanto se contaba con tres tipos de archivos: archivos de interfaz de usuario, archivos de lógica de negocios y archivos de manejo de datos.

Para dimensionar el sistema se tomó en cuenta la cantidad de archivos de código fuente y su contenido a nivel de clases y métodos. Los archivos, clases y métodos son tomados como *entidades*. La Tabla 1 muestra la cantidad de entidades que conforman el sistema.

En este proceso participaron siete estudiantes, formando dos equipos, uno de 3 (equipo 1) y otro de 4 personas (equipo 2). Tres de los estudiantes sabían bien la programación .Net, mientras que los otros cuatro estaban cursando una materia de Programación Visual.

Tabla 1: Artefactos del sistema "D&EM" -caso de ingeniería inversa

Entidad	**Cantidad**
Archivo de Interfaz de Usuario (GUI)	30
Archivo de procesamiento (lógica de negocio)	35
Archivo para manejo de datos	30
Clases	35
Métodos	304
Casos de uso	24
Pantallas de interfaz de usuario	27

El equipo 1 identificó los casos de uso a partir de la GUI, utilizando el procedimiento para hacer ingeniería inversa que se describe en la sección 4; mientras que el equipo 2 lo hizo en el código fuente. En la Tabla 1 se muestra la cantidad de casos de uso identificados para este sistema.

Caso 2: Reingeniería

El proceso de reingeniería, realizado en el semestre 2006-2, se aplicó a un sistema desarrollado en PHP. Este es un sistema de administración de proyectos de posgrado e

investigación (SAPPI), en la Universidad Autónoma de Baja California, Campus Tijuana.

La primera versión de este sistema fue implementada durante el semestre 2006-1 por un equipo de estudiantes en la materia Diseño de Sistemas de Información. En esta versión, el sistema SAPPI contaba con las siguientes funcionalidades: crear proyectos, autorizar proyectos, reportar avances de proyectos, consultas y modificaciones. En esta versión se generó la documentación completa en base al método MEDSOCAMP. La cantidad de los artefactos identificados para la versión 1 se muestran en la Tabla 2.

El proceso de reingeniería se aplicó al sistema SAPPI durante el semestre 2006-2 en la materia de Reingeniería de Procesos, por un equipo de estudiantes distinto a los creadores de la versión 1. En este caso se utilizó el procedimiento propuesto para hacer reingeniería descrito en la sección 4, tomando como base la documentación de la versión 1. El proceso administrativo de proyectos de investigación fue estudiado en forma completa y documentado con la técnica de diagramación RAD. En esta reestructuración del proceso se agregaron las siguientes funcionalidades: manejo de prosupuesto de proyecto, control de becarios y asignación de evaluadores. Este incremento de la funcionalidad se refleja en la cantidad de casos de uso mostrada en la Tabla 2.

Tabla 2: Artefactos del sistema "SAPPI" -caso de reingeniería

Entidad	**Cantidad** (versión 1)	**Cantidad** (versión 2)
Casos de uso	19	29
PHP	26	92
JavaScript	2	4
CSS	2	8
HTML	2	6

Resultados preliminares

Al realizar los procesos de ingeniería inversa y reingeniería, los estudiantes pueden visualizar aspectos positivos y negativos, tal como se indica en [22]. En el caso de la ingeniería inversa, una vez realizadas las revisiones de los códigos, se aplicó una encuesta entre los revisores participantes. Esta encuesta consistió en un cuestionario con preguntas de opción múltiple en su mayoría. Cabe mencionar que los estudiantes expresaron que ya tenían experiencia en revisiones informales de código. En esas experiencias previas, los objetivos de la revisión habían sido: (1) Entender el código para modificarlo, (2) Entender un problema a través del código y (3) Aprender la sintaxis de un lenguaje. También expresaron que el método de revisión del código estaba orientado a la localización por nombre de funciones/procedimientos específicos y la localización de algoritmos implementados.

Las preguntas formuladas cubren varios aspectos reflejados en los puntos que a continuación se describen:

Punto 1: Al revisar los códigos en base a la técnica de casos de uso, éstos fueron localizados más fácilmente en el nombre de las clases y el nombre de los métodos/procedimientos/funciones.

Punto 2: Las principales deficiencias encontradas en los códigos revisados, bajo esta técnica, son: inconsistencia al nombrar funciones, no hay comentarios/encabezados explicativos en las entidades de código, existe código redundante, y los nombres de las entidades se repiten.

Punto 3: No se sigue un estándar o reglas para nombrar las entidades de código conforme a los nombres de los casos de uso.

Los estudiantes opinaron que la técnica de revisión de código basada en casos de uso es fácil de entender y fácil de aplicar, siempre y cuando se tengan los conceptos básicos de casos de uso. También opinaron que es una técnica eficaz. Por otro lado, hicieron las siguientes observaciones sobre las posibles dificultades para aplicar esta técnica:

Observación 1: Hay que conocer los conceptos sobre casos de uso, a nivel de los tipos de casos de uso y sus relaciones.

Observación 2: Si el código no está implementado en base a casos de uso, resultaría difícil localizarlos, ya que el código de proyectos desarrollados por estudiantes no tiene una estructura bien definida.

Observación 3: Si no existe otra documentación a parte del código, sería necesario establecer comunicación con el desarrollador del código para aclarar dudas.

Aunque los puntos expresados por los estudiantes son bien conocidos en el ámbito académico, consideramos que el hecho de que percibieran estos aspectos resulta de bastante utilidad para que valoren la importancia de las buenas prácticas de documentación, así como el uso de estándares en el desarrollo de sistemas.

En el proceso de reingeniería se notó un gran entusiasmo en los estudiantes, ya que este proceso fue menos complicado debido a que se disponía de una documentación del sistema. Los principales aspectos positivos observados en este proceso fueron lo siguientes: facilidad para incrementar la funcionalidad del sistema, facilidad en la documentación de los incrementos en la funcionalidad y mayor calidad en el producto final.

6. CONCLUSIONES

En base a los resultados obtenidos en ambos casos de estudio, es posible concluir que la práctica de los procesos de ingeniería inversa y reingeniería resultó de mucha utilidad para los estudiantes que participaron. Bajo los escenarios de esta práctica, los estudiantes pudieron hacer una comparación entre las prácticas comunes de desarrollo de software contra un proceso bien estructurado, el cual genera sistemas con un nivel adecuado de documentación. El resultado de la comparación proporcionó un conocimiento muy

provechoso a los estudiantes, ya que les permitió valorar las bondades del proceso de ingeniería, por ejemplo, en términos de orden, nivel de documentación y calidad del producto final.

Los procedimientos propuestos para realizar la ingeniería inversa y la reingeniería resultaron ser útiles y fáciles de aplicar. Un factor importante que facilitó esta práctica fue el conocimiento previo que los estudiantes tenían sobre el proceso de modelado de sistemas, apoyado principalmente en el método MEDSOCAMP.

7. REFERENCIAS

[1] H. Abelson y P. Greenspun, "Teaching Software Engineering –Lessons from MIT" [En línea], 2001, disponible en: http://philip.greenspun.com/teaching/teaching-software-engineering, consultado en: Marzo 2007.

[2] N. Aizenbud-Reshef, B. T. Nolan, J. Rubin e Y. Shaham-Gafni, "Model Traceability" [en línea], **IBM Systems Journal**, Available on line on July 11, 2006, disponible en: http://www.research.ibm.com/journal/sj/453/aizenbud.html, consultado en: Marzo 2007.

[3] I.F. Alexander y N. Maiden, **Scenarios, Stories, Use Cases**, England: John Wiley & Sons, Ltd., 2004.

[4] B. Bernárdez, A. Durán y M. Genero, "Empirical Evaluation and Review of Metrics-Based Approach for Use Case Verification", **Journal of Research and Practice in Information Technology**, Vol. 36, No. 4, November 2004, pp. 247-258.

[5] K. Bittner e I. Spence, **Use Case Modeling**, Boston, MA : Addison-Wesley, 2003.

[6] E.J. Chikofsky y J.H. Cross, "Reverse Engineering and Design Recovery: A Taxonomy", **IEEE Software**, Vol. 7, No. 1, 1990, pp. 13-17.

[7] J. Ebert, F. Lehner, V. Riediger y A. Winter, "Software Reegineering" – Editorial, **Informatik Forschung und Entwicklung**, Vol. 19, No. 3, April 2005, pp. 125-126.

[8] D. Evans, "Teaching Software Engineering Using Lightweight Analysis" [En línea], 2001, disponible en: www.cs.virginia.edu/~evans/pubs/ccli01.pdf, citeseer.ist.psu.edu/evans01teaching.html, consultado en: Marzo 2007.

[9] D. Evans y M. Peck, Simulating Critical Software Engineering [En línea], 2004, disponible en: citeseer.ist.psu.edu/evans04simulating.html, virginia.edu/~techrep/CS200403.pdf, consultado en: Marzo 2007.

[10] P. Hall, **Software Reuse and Reverse Engineering in Practice**, London, England: Chapman & Hall, Ltd., 1990.

[11] **M**. Hammer y J. Champy, **Reengineering the Corporation**, 3rd Edition, Nicholas Brealey Publishing Ltd., 2001.

[12] I. Ivkovic y M.W. Godfrey, "Architecture Recovery of Dynamically Linked Applications: A Case Study", Proceedings of the **10th International Workshop on Program Comprehension** (IWPC'02), Paris, France, Junio 2002, pp. 178-186.

[13] R. Juárez, G. Licea, A. Cristóbal y J. Valencia, "La Enseñanza del Diseño de Software con un Enfoque Orientado a la Calidad y Mantenimiento del Producto", **4ta. Conferencia Iberoamericana en Sistemas, Cibernética e Informática** (CISCI 2005), Vol. III, Orlando, Florida, 14-17 de Julio 2005, pp. 161-166.

[14] R. Juárez-Ramírez, G. Licea, A. Cristóbal-Salas, "A Graph-Based Technique to manage Quality through the Product Lifecycle", por aparecer en las memorias de la **11th World Multiconference on Systemics, Cybernetics and Informatics** (WMCSI 2007).

[15] P. Kruchten, "The 4+1 View Model of Architecture", **IEEE Software**, Vol. 12, No. 6, November 1995, pp. 42-50.

[16] Y. Liu y E. Stroulia, "Engineering the Process of Small Novice Software Teams", Proceedings of the **10th Working Conference on Reverse Engineering 2003** (WCRE'03), pp. 102-112.

[17] L.A. Maciaszek y B.L. Liong, **Practical Software Engineering: A Case Study Approach**, Addison-Wesley, 2005.

[18] A. Memon, I. Banerjee y A. Nagarajan, "GUI Ripping: Reverse Engineering of Graphical User Interfaces for Testing", Proceedings of the **10th Working Conference on Reverse Engineering 2003** (WCRE'03), pp. 260-269.

[19] M.R. Olsem, C. Sittenauer, M. Dawood y K.J. Rasmussen, "Reengineering Technology Report" - Technical Report, Vol. 2, Software Technology Support Center, Hill AFB, UT., April 1995.

[20] B. Ramesh y M. Jarke, "Toward Reference Models for Requirements Traceability," **IEEE Transactions on Software Engineering**, Vol. 27, No. 1, (January 2001), pp. 58–93.

[21] L.H. Rosemberg y L.E. Hyatt, "Hybrid Re-Engineering"[On line], presented at the **Software Technology Conference** (STC'07), Salt Lake City, Utah, April 27- May 2, 1997, disponible en: http://satc.gsfc.nasa.gov/support/STC_APR97/hybrid/hybride1.html, consultado en: December 2006.

[22] A.I. Wang y T. Stalhane, "Using Post Mortem Analysis to Evaluate Software Architecture Student Projects", Proceedings of the **18th Conference on Software Engineering Education & Training**, CSEE&T 2005, pp. 43-50.

[23] R. Warden, **Software Reuse and Reverse Engineering in Practice**, London, England: Chapman & Hall, 1992.