Universidad de las Ciencias Informáticas

Facultad 3

Título: "Refactorización del componente Configuración del subsistema Contabilidad de Xedro-ERP."

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autor(es): Eloy Manuel Méndez Rivera

René Acosta Ramírez

Tutor(es): MSc. Joisel Pérez Pérez

Co-tutor: Ing. Didier Roque Ginebra

Junio 2014

Declaración de autoría

Declaramos que somos los únicos autores de este trabajo y autorizamos a la Facultad3 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmamos la presente a los _____ días del mes de _____del año_____.

_____
René Acosta Ramírez

_____
Eloy Manuel Méndez Rivera

_____
MsC.Joisel Pérez Pérez

_____
Ing.Didier Roque Ginebra

Datos de contacto

Autores:

René Acosta Ramírez
Universidad de las Ciencias Informáticas
Teléfono:8372026

Eloy Manuel Méndez Rivera
Universidad de las Ciencias Informáticas
Teléfono:8358979

Tutores:
Msc Joisel Pérez Pérez
Teléfono:047383383

Ing Didier Roque Ginebra
Universidad de las Ciencias Informáticas

Agradecimientos

René Acosta Ramírez

A mis padres, porque gracias a sus constantes sacrificios puedo lograr este sueño tanto mío como suyo. Sin su apoyo incondicional no hubiese podido pensar siquiera en este día. A Yadirka y Reinier por enseñarme que la amistad es algo que llega para quedarse y demostrarme que cuando uno se propone algo lo puede alcanzar. Si no fuera por ustedes nunca hubiera aprendido a estudiar. A José Javier por ayudarme a permanecer en la carrera, cuando pensé que tendría que abandonarla. A Indra con la cual estaré eternamente en deuda y ella sabe por qué. A Jennifer por ser una amiga incondicional y brindarme su amistad sincera. A Joisel, el cual más que un tutor es un amigo, sus consejos me sirvieron de mucho y aunque él no lo crea los milagros existen, me gradué. A Rosalina por ser la mejor de las profesoras, y sobre todo por la amistad brindada sin interés ninguno, además de soportar los pisotones en el tango me enseño el valor de la amistad. A todos mis compañeros de aula por apoyarme cuando necesité de su ayuda. A Leticia por ser mi compañera de escritura, y por darme esos consejos para enfrentarme a la vida. A Claudia por ayudarme con el documento, sé que si no lo decía me enfrentaba a una muerte dolorosa. A Jessica por soportarme en estos últimos meses, que sé que no he sido fácil. Y bueno porque es obligado a mi compañero de tesis Eloy, por ser como un hermano y demostrarme que nunca es tarde para encontrar una buena amistad. A Esteban, Yancy, David, Duniesky y Dariel por animarme a seguir adelante en todo momento. A Yucona por ser un amigo incondicional. Si se queda alguien que me disculpen pero son muchas personas y poco espacio.

Eloy Manuel Méndez Rivera

A mi familia por todo el apoyo y la confianza depositada en mí, en especial a mi mama por ser mi mayor apoyo y por todos los sacrificios hechos para que pudiera llegar aquí hoy, muchas gracias mami. A mi papa por su constante preocupación. A mis hermanas por su cariño. A mi tío Eddy por su dedicación. A todos mis amigos aquí en la escuela quienes son mucho más que eso, por todos los momentos que vivimos juntos tanto en las buenas como en las malas muchas gracias Robin, Freddy, Yadirka, Reinier, Franke, Luis Manuel. A Anet y familia por aceptarme como uno más de ellos por tanto tiempo. A Rosita porque además de ser la mejor profesora y ser quien único me mantenía despierto en los turnos de clase por ser la amiga que tanto necesite y que siempre estuvo ahí. A mis compañeros de grupo por brindarme su amistad. A Doris por confiar en mi todo este tiempo y hacérmelo saber, a su familia por su apoyo y preocupación. Sin que me quede otro remedio tengo que agradecerle a mi compañero de tesis Rene por soportarme todo este tiempo por escuchar mis pleitos y dejar que me ganara su amistad muchas gracias hermano. Espero no se me quede nadie, pero si es así muchas gracias de verdad.

Dedicatoria

René Acosta Ramírez

Este triunfo se lo dedico a mis abuelos no los tengo físicamente a mi lado, pero siempre están junto a mí. Les digo lleno de orgullo que ya Bururú es un ingeniero.

A mi tía para que vea que no le estaba robando el dinero al estado. A Dariel para que vea que si se puede lograr lo que uno se propone.

A mi hermana por ser la fuente de inspiración para seguir superándome y aunque ella no lo sabe porque nunca se lo he dicho, ella es mi ejemplo de persona a seguir.

A mis padres porque sin todos los sacrificios que han hecho no pudiera estar diciéndoles en este momento que son lo más importante en mi vida y que los quiero más que a nada.

Eloy Manuel Méndez Rivera

Quiero dedicarle este resultado a mi mama por ser mi ejemplo, por darme su amor y cariño, por hacer de mi la persona que soy, por apoyarme en todo momento sin esperar nada a cambio, por sus buenos consejos siempre oportunos, por sacrificarse y dar todo para que en un futuro fuese un profesional,
por sentirse orgullosa de verme cumplir mis metas. Te quiero mucho.

A mi papa por esperar siempre lo mejor de mí.

A mis abuelos por todos los mimos, regaños y concejos.

A mis hermanas y mi sobrino por ser mi fuente de inspiración y para que se sientan orgullosos de mí.

Resumen

El rendimiento es un factor sumamente importante a tener en cuenta en el desarrollo de software. Es primordial que un sistema además de cumplir con los requisitos funcionales y las necesidades básicas del cliente pueda a su vez tener una respuesta rápida a las peticiones realizadas. El objeto de la presente investigación es diseñar una solución para aumentar el rendimiento de uno de los componentes del subsistema Contabilidad del producto Xedro-ERP a partir de la aplicación de una refactorización que incluye la capa de acceso a datos y la lógica del negocio empleada en el desarrollo de dicho componente. Se propone la utilización de técnicas de refactorización como extraer clases, introducir parámetros como objetos, disminuir complejidad de los métodos y la actualización de tecnología de terceros como técnica de refactorización. Se realiza una evaluación de resultados obtenidos después de aplicar la solución, demostrándose la efectividad y eficiencia de la misma a través de las pruebas de estrés, de carga, de estabilidad y el tiempo de respuesta del sistema.

Palabras clave: Refactorización, rendimiento, técnicas de refactorización.

Contenido

Introducción

En la actualidad el aumento de la competitividad entre las empresas, la necesidad de ampliar su mercado para obtener mejores resultados, entre otros factores, ha impulsado el auge en la informatización de la gestión empresarial. Este hecho ha traído como consecuencia la utilización de sistemas hechos a la medida para las empresas.

Estos sistemas después de un periodo de tiempo en explotación generan errores. Ya que finalmente todo programa o herramienta informatizada que es utilizada, en algún momento produce errores o es necesario hacerle mejoras. Estos errores en algunas ocasiones pueden ir desde un simple cambio en la interfaz, hasta hacer otras correcciones sin afectar el funcionamiento del sistema. En ocasiones por el tiempo de uso, el atraso tecnológico y otros factores, un simple mantenimiento no es la solución al problema y es necesario realizar cambios más profundos, cambios que implican mejorar el diseño del código después de que este ha sido escrito.

En este sentido la depuración del código fuente en ocasiones va más allá de aplicar estándares de codificación. Si en verdad lo que se desea es cambiar su estructura interna para hacerlo más comprensible (en cuanto a legibilidad y claridad) y mejorar su rendimiento, sin cambiar su comportamiento visible, debe entonces realizarse un proceso un poco más complejo. Algunos de los problemas más comunes por los que se depura el código fuente son los siguientes:

- Código duplicado.
- Que un método tenga un número de líneas de código que se pueda disminuir.
- Identificadores de variables demasiado largos o cortos.
- Clases intermediarias cuyo único trabajo es la delegación de funcionalidades.
- Cambios en cadena lo que trae consigo que un cambio en una clase implica cambiar otras muchas.

En cualquiera de estas circunstancias es muy difícil afrontar un proceso de cambio sin que se afecte el rendimiento. El proceso mediante el cual se cambia un sistema de software de tal manera que no altere el comportamiento externo del código aun fortaleciendo su estructura interna es conocido como refactorización, y se considera unos de los procesos de la reingeniería informática. Es una manera disciplinada de limpiar el código, que minimiza las posibilidades de introducir errores. En esencia cuando se refactoriza se está mejorando el diseño del código después de este ha sido escrito. (Fowler, 2002)

En la Universidad de las Ciencias Informáticas (UCI en lo adelante) se está desarrollando en el Centro de Informatización de Entidades (CEIGE en lo adelante), un Sistema Integral de Gestión de Entidades denominado Xedro-ERP. El desarrollo de dicho sistema comenzó en el 2008 por lo que varias generaciones de desarrolladores han aportado sus ideas en la construcción del mismo. A pesar de que existen estándares definidos para el trabajo en cada rol, cada persona aporta su propio estilo, lo que trae como consecuencia código sin optimizar y esto impacta en el rendimiento del sistema. En el proceso de desarrollo de software en ocasiones se le da mayor atención a la entrega según el cronograma de actividades, priorizando el desarrollo y entrega de la solución terminada en el tiempo pactado. Lo que trae como consecuencia que se dedique menos atención a la optimización de la solución brindada conllevando nuevamente a un impacto negativo en el rendimiento del sistema.

En el componente Configuración del subsistema Contabilidad de Xedro-ERP algunas de las razones por las que es necesario depurar el código están dadas porque cada cierto tiempo se han ido incorporando desarrolladores, los cuales no conocen todas las decisiones previamente tomadas ni las ideas claves detrás de las clases implementadas. Por lo que inevitablemente, la transición ha traído consigo una pérdida del conocimiento del proyecto. Nuevas y más personas en el equipo de desarrollo, trae consigo menos comunicación entre ellas, y como resultado, más decisiones personales, las cuales están basadas en resolver el problema con una funcionalidad operativa, pero sin tener en cuenta la optimización del código obtenido. Esto a la par que aumentan las funcionalidades y el código fuente. A raíz de ello nuevamente se vuelve a afectar el rendimiento del sistema.

Aparejado a ello se tiene el hecho de que las tecnologías de terceros empleadas para el desarrollo del componente no han sido actualizadas. En el mercado existen versiones más

recientes, que presentan cambios que inciden directamente sobre el rendimiento del sistema, por ejemplo Doctrine en su versión 1.2, empleado como ORM para el marco de trabajo Sauxe (MT en lo adelante) hace llamadas innecesarias a clases para abrir conexiones a las base de datos.

De ahí que a partir de la problemática antes descrita se identifique el siguiente **problema a resolver:**

¿Cómo mejorar el rendimiento del componente Configuración del subsistema Contabilidad de Xedro-ERP?

Planteándose como **objeto de estudio:** La refactorización en el desarrollo de software y como **campo de acción** la migración del componente Configuración del subsistema Contabilidad de Xedro-ERP.

Para solucionar el problema identificado se propone como **objetivo general:** Refactorizar el componente Configuración del subsistema Contabilidad de Xedro-ERP a partir de la versión 2.0 de Doctrine que posibilite mejorar el rendimiento del mismo. Y para darle cumplimiento a este objetivo se definen los siguientes **objetivos específicos:**

- Fundamentar la investigación mediante la elaboración del Marco Teórico para sustentar los conceptos y la propuesta de desarrollo del componente.
- Realizar el diseño e implementación del componente Configuración del subsistema Contabilidad de Xedro-ERP para mejorar el rendimiento del mismo.
- Realizar pruebas de rendimiento para validar el componente implementado.

Como idea a defender se define: La refactorización del componente Configuración del subsistema Contabilidad de Xedro-ERP haciendo uso de la versión 2.0 de Doctrine mejoraría el rendimiento del mismo.

En el desarrollo de la investigación se han utilizado los siguientes **métodos científicos:**

- **Histórico lógico**: Este método se emplea mediante la realización de un estudio del estado del arte en la investigación para conocer la evolución y desarrollo de los

diferentes temas asociados a la problemática, revelando las etapas principales de su desenvolvimiento y las conexiones fundamentales.

- **Análisis y Síntesis:** El análisis es un procedimiento mental mediante el cual un todo complejo se descompone en sus diversas partes y cualidades. Permite estudiar el funcionamiento del componente y la relación existente entre las clases que lo componen.

- **Inducción – Deducción:** Al aplicar este método se estudian algunas de las técnicas de refactorización más utilizados actualmente, con el fin de definir sus características particulares y beneficios que brinda su uso.

Los **métodos empíricos** empleados son:

- **Observación:** Con su utilización se centralizó toda la atención en la situación actual existente en el componente Configuración y en los inconvenientes de la utilización de la versión 1.2.2 de Doctrine.

- **La Medición:** Con su utilización se obtiene información acerca del rendimiento del componente Configuración para determinar la eficacia de la solución.

El contenido del trabajo se divide en tres capítulos:

CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA.

En este capítulo se lleva a cabo el estudio del estado del arte, realizándose una investigación las técnicas que se aplican durante la refactorización y herramientas que ayuden a ello. Se describen también las tecnologías y metodologías que se utilizarán en el desarrollo de la solución.

CAPÍTULO 2. DISEÑO E IMPLEMENTACIÓN DE LA REFACTORIZACIÓN.

Este capítulo detalla las características bajo las cuales operará el componente para aplicar la refactorización, en correspondencia con la metodología propuesta en el capítulo 1. Se describe cómo es realizado hoy en día el negocio que se desea informatizar además de la propuesta de solución. En consecuencia, se obtendrá un conjunto de artefactos establecidos por la

metodología de desarrollo escogida. Igualmente se especificará la utilización de un conjunto de patrones dentro del diseño del componente.

CAPÍTULO 3. VALIDACIÓN DE LA SOLUCIÓN PROPUESTA.

El punto de partida de este capítulo está basado en los artefactos obtenidos del diseño y la implementación de la refactorización en el capítulo 2 para comenzar las diferentes validaciones y pruebas del sistema. La validación de la implementación se realiza aplicando pruebas de caja negra a partir de los diseños de casos de pruebas como parte de los artefactos que se deben generar.

Capítulo 1 FUNDAMENTACIÓN TEÓRICA

Introducción

En este capítulo se reflejan los temas fundamentales que sustentan la investigación. En él se abordan varios conceptos asociados a la refactorización durante el proceso de desarrollo de software y la importancia de poseer un código claro y funcional. Además se hace un estudio del arte sobre las principales herramientas y técnicas empleadas para llevar a cabo este proceso. Conjuntamente se realiza un análisis sobre el empleo de la migración a una versión superior del ORM empleado en el MT como elemento de apoyo a la refactorización. Se definen los lenguajes, las tecnologías, además de las herramientas y la metodología que se utilizan en el desarrollo del componente.

1.1 Refactorización en el desarrollo de Software

La refactorización es el proceso de cambiar un sistema de software de tal manera que no altere el comportamiento externo del código fortaleciendo su estructura interna. Es una manera disciplinada de limpiar el código, minimizando las posibilidades de introducir errores. En esencia cuando se refactoriza se está mejorando el diseño del código después de que este ha sido escrito. (Fowler, 2003)

La refactorización va más allá de una simple depuración de código, ya que sabiendo qué técnica de refactorización usar y cómo usarla, proporciona una vía más eficiente y controlada al programar. El principal propósito de la refactorización es hacer el código más fácil de entender y modificar para obtener el rendimiento que se necesita del sistema. Para lograr esto se pueden hacer muchos cambios internos mientras el software todavía lleve a cabo la misma función que hacía antes.

1.2 Importancia de la refactorización

Sin refactorización, el diseño del sistema decaerá. Si se realizan cambios para obtener algún resultado a corto plazo, como por ejemplo darle solución a algún error existente sin tener en cuenta el diseño, el código pierde su estructura. Por lo que se hace más difícil ver el diseño mediante la lectura del código. La pérdida de la estructura de código tiene un efecto acumulativo, cuanto más difícil sea ver el diseño del código, más difícil es para conservarlo y se

desintegrará con mayor rapidez. Con la refactorización se pone el código en orden, pues el trabajo se realiza para eliminar los bits que no están realmente en el lugar correcto, para lograr que el código diga todo una vez y solo una vez, que es la esencia de un buen diseño.

1.2.1 Problemas que resuelve la Refactorización

Entre los principales problemas a los que les da solución la refactorización se encuentran los llamados malos olores en el código o bad smells, ejemplos de estos son:

Código duplicado: Si aparece la misma estructura de código en más de un lugar. La misma expresión en dos métodos en la misma clase o la misma expresión en dos subclases hermanas, de seguro que el programa va a ser mejor si se encuentra la manera de unificar estas. Para esto es necesario eliminar las líneas de código que son exactamente iguales en varios sitios, o bien eliminar líneas de código muy parecidas o con estructura similar en varios sitios.

Métodos muy largos: Cuanto más largo es un procedimiento más difícil es de entender. Los lenguajes antiguos realizaban una sobrecarga de llamadas a subrutinas, lo que disuadía a la gente de usar métodos pequeños. En la actualidad los lenguajes orientados a objetos han eliminado prácticamente los gastos generales para las llamadas en proceso. Pero aún hay una sobrecarga para el lector del código, ya que hay que cambiar el contexto para ver lo que hace el subprocedimiento. En este caso hay que particionar el código fuente en partes y extraerlos para crear métodos más pequeños asignándole un buen nombre para facilitar su entendimiento, para que sean más fáciles de mantener y de reusar.

Clases muy grandes: Cuando una clase está tratando de hacer demasiado, suelen aparecer con mucha frecuencia demasiadas variables de instancia. Cuando una clase tiene demasiadas variables de instancia, la duplicación de código es muy frecuente. En estos casos se debería tratar de identificar qué cosas hace esa clase, ver si realmente todas esas cosas tienen algo que ver la una con la otra y si no es así, hacer clases más pequeñas, de forma que cada una trate una de esas cosas.

Métodos que necesitan muchos parámetros: Desde las primeras programaciones es muy común pasar por parámetro todo lo necesario. Sin tener en cuenta que esto hace que los métodos sean más difíciles de entender, porque se convierten en inconsistentes y difíciles de

usar, ya que ante la necesidad de realizar algún cambio se deberán modificar un mayor número de parámetros uno por uno.

Como se puede observar todos estos malos olores inciden de forma negativa en el diseño de un programa, haciendo que este sea lento y su código difícil de entender. Para hacer frente a estas situaciones la refactorización plantea una serie de técnicas que serán tratadas a continuación. (Fowler, 2003)

1.2.2 Técnicas de Refactorización

Como se mencionaba anteriormente la Refactorización brinda un grupo de técnicas para resolver los problemas por los cuales se ve afectado un buen diseño de código. Estas se encuentran agrupadas de acuerdo con su objetivo, definiéndose por grupos, entre ellas se encuentran:

1. **Componiendo Métodos:** su objetivo es hacer los métodos más cortos y la lógica de los métodos menos compleja.
    - **Extraer Método:** Consiste en extraer una porción de código y crear un método propio para dicho código, esto permite evitar los métodos largos y eliminar comentarios del código, es decir, hacer que éste sea autodocumentado, que revele su intención. Para hacer que funcione, el nombre de los métodos tiene que ser lo suficientemente claro para que pueda expresar el propósito del método.

2. **Mover Funcionalidad entre Objetos:** su objetivo es saber el lugar correcto donde poner las responsabilidades.
    - **Mover Método:** Cuando un método está usando más funcionalidades de otra clase que de aquella en la que está definido. Se debe crear un nuevo método con un cuerpo similar en la otra clase. Luego convertir el método anterior en una simple delegación o eliminarlo por completo.

    - **Extraer Clase:** Hay una clase haciendo el trabajo que deberían hacer entre dos. Para solucionar esto se puede crear una nueva clase y mover los atributos y métodos relevantes de la clase antigua a la nueva.

3. **Organizar los datos:** su objetivo es darle una mejor organización a los datos ya sean estructuras de clases o relaciones etc.
    - **Cambiar asociación unidireccional a bidireccional:** Cuando se tienen dos clases que necesitan usar funciones de cada una y solo existe un vínculo de un solo sentido, se debe añadir de nuevo los punteros, y cambiarlos modificadores para actualizar ambos conjuntos.

    - **Cambiar asociación bidireccional a unidireccional:** Cuando se tiene una asociación de dos vías pero una de las clases ya no necesita las funciones de la otra, se debe borrar los fines innecesarios de la asociación.

4. **Simplificar expresiones condicionales:** su objetivo es simplificar la lógica condicional puesto que esta suele ser muy engorrosa y complicar demasiado el código.
    - **Eliminar Puntos de Control:** Una variable está actuando como un indicador de control para una serie de expresiones booleanas. En vez de eso, usar un “break” o un “return” suele ser una mejor solución. Los lenguajes de programación cuentan con las sentencias break y continúe precisamente para evitar esta complejidad.

5. **Hacer más simples las llamadas a métodos:** su objetivo es hacer que las interfaces de los objetos sean más claras.
    - **Introducir los parámetros como un objeto:** Cuando se tiene un grupo de parámetros que comúnmente van juntos lo ideal es crear un objeto que los contenga a todos y pasar este último como parámetro.

6. **Manejar la generalización:** su objetivo es organizar la jerarquía de herencia adecuadamente.
    - **Ascender Método:** Si se tienen métodos con resultados idénticos en clases hijas, sería una mejor solución si se movieran estos hacia la clase padre.

    - **Extraer Subclase:** Cuando una clase tiene características que solo son usadas en algunas instancias, se debe crear un clase hija donde ubicar estas características.

1.2.3 La migración de la capa de acceso a datos como técnica de Refactorización

En la rama informática un mayor valor del versionado del software generalmente indica mayor evolución y mejor resultado al emplearlo en este contexto, al proceso de elevar un producto tipo software a otro de mayor nivel se conoce como migración de software (Mastermagazine, 2013). En ocasiones al migrar una herramienta a una versión más actualizada se deben realizar ciertos cambios que dan solución a algunos de los problemas de no refactorizar un sistema informático, pues de manera inconsciente se engloban algunas de las técnicas de refactorización mencionadas anteriormente.
La migración de la capa de acceso a datos es vista como técnica de refactorización debido a que con su implementación se le da solución a varios problemas:

- En el MT existen varios componentes, por su lado la base de datos trabaja con esquemas independientes relacionados con cada uno de estos, al realizar llamadas a funciones entre varios componentes el marco de persistencia de datos de Doctrine v1.2 tiene que abrir una nueva conexión, realizar la consulta y cerrar la conexión por cada llamada que se realice, siendo este un proceso repetitivo.

Por su parte Doctrine 2.0 no posee código que lo acople a una tecnología de persistencia, ya que no es su función manejarla. Los objetos de negocio solo deben preocuparse de cumplir con el modelo del dominio del diagrama de clases con las asociaciones pertinentes, eliminándose el innecesario instanciado de la conexión.

En estos elementos se resumen las técnicas **Extraer Clase e Introducir los parámetros como un objeto** descritas en el epígrafe Técnicas de Refactorización.

- Otro elemento que evidencia la limpieza del código y la eliminación de clases innecesarias se puede observar en la forma de realizar el mapeo de las tablas de la base de datos entre estas dos versiones.

En la versión 1.2 de Doctrine se generan 2 ficheros de Mapeo, el fichero Base que extiende de Doctrine_Record y otro fichero que extiende del fichero Base. Las relaciones entre tablas

se mapean en el fichero que extiende de la clase base donde también se encuentras los métodos del acceso a datos. Esto responsabiliza a la clase encargada del acceso a datos de tener que ocuparse de las relaciones.
Mientras que en Doctrine 2.0 la particularidad del mapeo está dada porque no se genera un fichero por la tabla en base de datos que guarda las llaves primarias de las tablas a unir, en este caso se hace referencia a la entidad de mapeo con que se posee la relación y mediante la anotación @JoinTable se le dice cuál es la tabla a través de la cual se relacionan. Otro aspecto que se debe tener en cuenta es que en estos casos se genera el constructor y dentro se inicializa la variable de la relación ManyToMany como un ArrayCollection definido por Doctrine. (Doctrine-Project.org, 2013)

Lo antes expuesto engloba las técnicas correspondientes a los grupos **Organizar los datos** y **Hacer más simples las llamadas a métodos** descritas en Técnicas de Refactorización.

Debido a lo antes expuesto se puede decir que la migración a Doctrine 2 funciona como una técnica de Refactorización, puesto que vincula un conjunto de sus técnicas. Menos cantidad de clases intermedias y una optimización de los algoritmos de acceso a datos posibilitarán un aumento en el rendimiento del componente Configuración del Subsistema Contabilidad de Xedro-ERP.

1.2.4. Estado del Componente Configuración antes de comenzar la refactorización

El componente Configuraciones del subsistema Contabilidad de Xedro-ERP presenta las dificultades expuestas en la problemática descrita en la introducción de la presente investigación, como resultado del constante ir y venir de desarrolladores, existen en el componente ficheros de código que presentan problemas en cuanto a la legibilidad de los métodos implementados. El código repetido es un problema frecuente dentro de dicho componente e incluso la incoherencia en cuanto al patrón arquitectónico empleado en su implementación. A su vez existen funcionalidades con un exceso de líneas de código innecesarias, que de reformularse, se mantendría la lógica del negocio y se ganaría en organización y legibilidad del código, lo que ayudaría a futuros programadores a entender lo que está implementado. Estas dificultades se repiten en la mayoría de las clases que conforman el componente, aparejado a ello la falta de

documentación hace más difícil entender el significado de cada línea de código. Analizándose estos aspectos puede notarse que los ¨malos olores¨ están presentes en todo el componente lo que puede eliminarse aplicando un proceso de refactorización empleando algunas de las técnicas existentes.

1.3 Estrategias para Refactorizar

Dado el grado de importancia de la refactorización se han desarrollado herramientas automatizadas para un gran número de lenguajes de programación, así como también para distintos tipos de artefactos que se generan a lo largo del proceso de desarrollo de software.

Estas herramientas automatizadas son aplicadas básicamente en tres variantes. La primera como plug-inaplicado a un entorno de desarrollo. La segunda formando parte de un entorno integrado de desarrollo (IDE), y la tercera, como una aplicación stand-alone. Las dos primeras variantes son las más encontradas. En la actualidad, la mayoría de los sistemas integrados de desarrollo, poseen un número de opciones automatizadas para realizar refactorizaciones. Estas utilidades están disponibles para los programadores como opciones de menú. Estas herramientas asisten al programador en el proceso de refactorización y favorecen la escritura código fuente, brindando la posibilidad de refactorizar en el mismo momento en el cual el código fuente es escrito. Ejemplos de estos productos se encuentran en entornos como **Eclipse**, **Visual Estudio**, **NetBeans**, etc. Todos ellos proporcionan opciones para aplicar técnicas de refactorización.
A continuación se detallan algunas de las herramientas existentes para refactorizar.

- **Smalltalk Refactoring Browser (**SRB en lo adelante**):** Esta herramienta se desarrolló para VisualWorks, VisualWorks/ENVY, e IBM Smalltalk[1]. Incluye las características del entorno de ventanas estándar de Smalltalk.

- **JFactor:** Es en realidad una familia de productos que provee herramientas para aplicar técnicas de refactorización. Este es capaz de integrarse al entorno de desarrollo

[1] Entornos de desarrollo que utilizan el lenguaje de programación Smalltalk.

VisualAge[2]. Este producto permite aplicar las siguientes refactorizaciones: extraer método, renombrar método, encapsular métodos.

- **XRefactory:** Es una herramienta de desarrollo de software para C y Java que proporciona la posibilidad de aplicar técnicas de refactorización. Una de sus características es que este producto es un plug-in para Emacs.[3]Otra característica de este producto es que sus creadores afirman que ha sido desarrollado para trabajar en proyectos de gran envergadura, proyectos con millones de líneas de código fuente. Este producto permite aplicar: extracción y renombramiento de paquetes, clases, parámetros, variables, inserción, borrado y movimiento de parámetros, atributos y métodos; encapsulamiento de atributos, entre otros.

IDEs soportados por la herramienta

- jEdit.
- Netbeans - parcialmente soportado.
- JBuilder - parcialmente soportado.
- También funciona como herramienta stand-alone.

Características:

JRefactory permite aplicar las siguientes refactorizaciones: extracción y renombramiento de paquetes, clases, parámetros, variables, inserción, borrado y movimiento de parámetros, atributos y métodos; encapsulamiento de atributos, entre otros.

- **SlickEdit:** Editor multi-lenguaje y multi-plataforma, dentro de sus características proporciona la posibilidad de la aplicación de refactorizaciones al código fuente escrito con esta herramienta.

- **Ref++:** Plug-in para Visual Estudio.Net que proporciona características de refactorización para C++.

- **Refactor!:** para Visual Basic es una herramienta en formato de plug-in, que permite a los programadores aplicar refactorizaciones a código fuente escrito para este lenguaje.

[2] Familia de entornos de desarrollo creada por la compañía IBM que incluye soporte para múltiples lenguajes de programación.
[3] Emacs es un editor de texto con gran variedad de funciones que forma parte del proyecto GNU.

Según sus creadores es capaz de aplicar 30 refactorizaciones. El producto es para Visual Basic .Net no aplicable a Visual Basic 6.

- **Aivosto Project Analyzer:** Esta aplicación, según sus creadores, es un revisor de código y una herramienta para analizar la calidad del código fuente. Unas de las características de la misma es que realiza un análisis de impacto antes de realizar algún cambio al código fuente de la aplicación.

- **ModelMaker:** Esta herramienta permite aplicar técnicas de refactorización a programas pascal o Delphi. Permite realizar refactorizaciones como por ejemplo: Extraer Clases, Extraer Método / Interfaz, Renombrar Parámetro etc.

Cabe destacar que si bien la cantidad de herramientas de refactorización es numerosa, esta práctica es muy reciente, por lo que es de esperar que las herramientas evolucionen en un número aún mayor.

La estrategia de refactorización a seguir puede ser de dos tipos: automatizada o manual. En el caso de emplear la estrategia manual se puede incurrir en la introducción de errores, puesto que es el programador quien debe tomar las decisiones y cambiar la estructura de la codificación. Sin embargo permite a su vez mantener intacta la lógica del negocio, aunque con el uso de la refactorización automatizada se obtienen buenos resultados, un menor número de errores en cuanto a código se refiere y la estandarización del código fuente, puede ser peligrosa, pues los cambios realizados por las herramientas pudieran afectar la lógica del negocio lo cual atenta directamente contra el principio fundamental de la refactorización.

Al estudiar las herramientas para refactorización automática se llegó a considerar un grupo de parámetros para determinar la utilización de las mismas. Los resultados se muestran en la siguiente tabla:

Tabla 1.Aplicabilidad de las herramientas automatizadas en la refactorización del componente.

Herramienta	Compatible con lenguajes de programación de MT	Compatible con ORM Doctrine	Compatible con IDE utilizado	Utilidad
SRB	No es compatible con PHP, ni JavaScript	No Compatible	No Compatible	No

XRefactory	No es compatible con PHP	No Compatible	Compatible	No
JRefactory	No es compatible con PHP	No Compatible	Parcial	No
Ref++	No Compatible	No Compatible	No Compatible	No
Refactor!	No Compatible	No Compatible	No Compatible	No

Teniendo en cuenta los resultados mostrados en la tabla ninguna de las herramientas estudiadas puede aplicarse en la refactorización del componente, por lo que se decide realizar la refactorización de forma manual.

1.4 Modelo de Desarrollo de Software

Para la refactorización del componente, el modelo de desarrollo de software empleado es una adaptación del propuesto por el centro. Esto es posible debido a que este modelo es:

- Centrado en la arquitectura.
- Basado en componentes.
- Ágil y adaptable al cambio.
- Iterativo e incremental.

Según (Obregón, 2012) por cada disciplina deben ser generados una serie de artefactos los cuales ayudan a una mejor comprensión del proceso de desarrollo que se lleva a cabo. A continuación se presentan dichos artefactos, además de otros aspectos que corresponden a cada disciplina y que son utilizados el desarrollo del presente trabajo de diploma.

Disciplina de Diseño

Durante esta disciplina es modelado el sistema para que soporte todos los requisitos. Esto tributa a una arquitectura sólida y estable que se convierte en un plano para la próxima fase, ya que los artefactos que se generan en esta etapa son más formales y específicos para una implementación. (Obregón, 2012)

Artefactos

- El Modelo datos.

- Diagramas de clases.
- Guía para utilizar los componentes en el nuevo Marco de Trabajo.
- Diagramas de secuencias.

1.4.1 Arquitectura

Según la definición brindada por la (IEEE, 2000)“la arquitectura del software es la organización fundamental de un sistema, formada por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán, y los principios que orientan su diseño y evolución”. Se aplica el tipo de arquitectura la cual está definida por modelo de desarrollo del centro, y además está basada en el estilo arquitectónico Modelo-Vista-Controlador (MVC).

Este es un estilo arquitectónico usado principalmente en aplicaciones que manejan gran cantidad de datos y transacciones complejas, donde se requiere una mejor separación de los conceptos para que el desarrollo esté estructurado de una mejor manera, coincidiendo con (Baryolo, y otros, 2008).

- **Modelo:** Esta capa es la representación específica de la información con la cual el sistema opera. Se limita a lo relativo de la vista y su controlador facilitando las presentaciones visuales complejas.

- **Vista:** Esta capa presenta el modelo en un formato adecuado para interactuar, usualmente la interfaz de usuario.

- **Controlador:** Este responde a eventos, usualmente acciones del usuario, e invoca peticiones al modelo y notifica a la vista.

En la figura 1 se muestra la estructura del estilo arquitectónico Modelo-Vista-Controlador que utiliza Xedro-ERP, la cual presenta la particularidad de que la vista no realiza solicitudes al modelo (Andux, 2010).

Figura 1. Estructura del estilo arquitectónico Modelo-Vista-Controlador

Patrones de diseño

De manera general existen para el diseño de software los denominados patrones de diseño, porque constituyen soluciones simples y elegantes a problemas específicos y comunes del diseño orientado a objetos. Dentro de ellos, para esta investigación se propone emplear algunos de los patrones generales de software para la asignación de responsabilidades (GRASP, por su acrónimo en inglés), ya que describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones, según lo indicado por (Larman, 1999). Dichos patrones son:

Patrón Experto: Su función es asignar una responsabilidad a la clase que tiene la información necesaria para cumplirla.

Patrón Creador: El propósito fundamental de este patrón es encontrar un creador que debemos conectar con el objeto producido en cualquier evento. Al escogerlo como creador, se da soporte al bajo acoplamiento.

Patrón Controlador: El propósito de este patrón es encontrar la clase idónea para asignarle la responsabilidad de manejar los eventos del sistema y brindarle al usuario el resultado del evento ejecutado.

Disciplina Implementación

Luego de los resultados del diseño se implementa el sistema en términos de componentes A través de ficheros de código fuente, scripts y ejecutables.

Artefactos

- Ficheros de código.
- Base de Datos.
- Guía para la integración del componente Configuraciones del subsistema Contabilidad de Xedro-ERP.

Disciplina Pruebas internas

Durante esta disciplina se desarrollan las pruebas verificando el resultado de la implementación. En ella se detectan posibles errores y el software, es decir requisitos que el producto debería cumplir y que aún no los cumple (Obregón, 2012).

Los métodos de pruebas independientemente del nivel en que se enmarque la prueba, ayudan a encontrar buenos conjuntos de casos de prueba para detectar diferentes tipos de errores. Existen dos enfoques alternativos según indica (Ramírez, 2012), descritos como: pruebas de Caja Blanca y Caja Negra. En la primera mencionada se cuenta con el código, pues se conoce el mismo y se ejecutan cada uno de sus elementos. Por su parte en las pruebas de caja negra se cuenta con la interfaz y se prueba cada uno de los elementos que la componen para llevar a cabo la validación. Para la validación del componente Configuración se propone realizar pruebas de rendimiento con el objetivo de lograr la satisfacción de las necesidades del cliente.

1.5 Herramientas

Visual Paradigm para UML 8.0

Es una herramienta que emplea UML 2.1 y permite representar el ciclo de vida completo del desarrollo de software. Posibilita la representación de varios tipos de diagramas, entre otras

opciones. Dentro de sus características fundamentales se encuentran que soporta la opción de usar BPMN[4].

NetBeans 7.2

Es un entorno de desarrollo integrado, disponible para varios Sistemas Operativos como, Windows, Mac, Linux y Solaris. Es de código abierto, escrito completamente en Java. Es una plataforma de aplicaciones que permite a los desarrolladores crear aplicaciones del tipo web, de escritorio y móviles, utilizando la plataforma Java, así como un número importante de módulos para extenderlo a otros lenguajes. (Oracle Corporation)

PostgreSQL

PostgreSQL 8.4 es un sistema de base de datos relacional de código abierto, que se destaca por su robustez, escalabilidad y cumplimiento de los estándares SQL. Este cuenta con diversas versiones para sistemas operativos tales como: Linux, Windows, Unix, Mac OS X, Solaris, BSD, Tru64 y otros.

La versión 8.4 cuenta con una gran cantidad de funcionalidades y mejoras que se le incorporaron para la administración, consultas y programación de PostgreSQL en aras de incrementar su rendimiento. La mayoría de los nuevos cambios con los que cuenta esta versión son herramientas incorporadas, órdenes de administración y monitoreo. Entre las mejoras más significativas que se le añadieron se encuentra la restauración de bases de datos en procesos paralelos, que acelera la recuperación de un respaldo hasta 8 veces mayor. ()

Apache 2.4.4

Servidor web de software libre desarrollado por la Apache Software Foundation cuyo objetivo es servir o suministrar páginas web a los clientes web o navegadores que las solicitan. Es una tecnología gratuita y de código abierto, lo cual le brinda transparencia al software de manera que se pueda conocer lo que se está instalando como servidor. Permite personalizar la respuesta ante los posibles errores que se puedan dar en el servidor. Es posible configurar Apache para que ejecute un determinado script cuando ocurra un error en concreto. ()

Mozilla Firefox 26.0

[4] **Business Process Modeling Notation** es una notación gráfica que permite el modelado de procesos de negocio, en un formato de flujo de trabajo.

Es un navegador libre y de código abierto. Es usado para visualizar páginas web. Incluye corrector ortográfico y marcadores dinámicos. Además se pueden añadir funciones a través de complementos desarrollados por terceros. Es multiplataforma, permite la integración con antivirus, realiza la navegación por pestañas. Presenta compatibilidad para múltiples extensiones y utiliza el sistema SSL[5] para proteger la comunicación con los servidores web, utilizando además fuerte criptografía cuando se utiliza el protocolo HTTPS[6]. (Mozilla Firefox, 2009)

Sauxe

El marco de trabajo Sauxe posee un conjunto de componentes reutilizables que provee una estructura genérica y el comportamiento para una familia de abstracciones, logrando una mayor estandarización, flexibilidad, integración y agilidad en el proceso de desarrollo (Baryolo, y otros, 2008). Sauxe utiliza **ExtJS 2.0** para implementar la capa de presentación, se apoya en Zend-Ext, una extensión de **Zend Framework** para el desarrollo de la lógica del negocio, y para la gestión de los datos que maneja, **Doctrine 1.2**. Utiliza como estilo arquitectónico el Modelo-Vista-Controlador. También tiene como propósito insertar la inversión de controles para la integración de servicios (Baryolo, y otros, 2008). Además como herramienta de seguridad emplea el sistema **Acaxia** (Sistema de Gestión Integral de Seguridad), el cual está desarrollado sobre software libre e incorpora procesos como la administración de conexiones y de perfiles. La estructura de estos Frameworks se encuentra formada por diferentes lenguajes, tecnologías y notaciones tales como:

- **HTML 5 (Lenguaje de Marcas de Hipertexto)** es un conjunto de etiquetas, complementadas por extensiones que permiten dar formato a un archivo para que pueda ser visualizado en forma de página web (HTML.net, 2012).

- **XML 1.0 (Lenguaje de Etiquetado Extensible)** es un formato que permite la lectura de datos a través de diferentes aplicaciones y además es utilizado para estructurar, almacenar e intercambiar información (Colectivo editorial, 2008).

- **JSON (Notación de Objetos de JavaScript)** es un formato de intercambio de datos que está constituido por una colección de pares de nombre/valor. Actualmente se ha convertido en un estándar en el desarrollo de aplicaciones web donde en ocasiones

5 **Secure Sockets Layer:** protocolo criptográfico que proporciona comunicación segura en Internet.
6 **HTTPS: (**Hypertext Transfer Protocol Secure) Protocolo de transferencia de hipertexto seguro.

sustituye a XML para permitir la integración de servicios en el navegador del usuario como indica (Colectivo de JSON, 2011).

- **JavaScript 1.6** es un lenguaje de programación interpretado y orientado a objetos. Permite a los desarrolladores añadir y crear interactividad en el desarrollo y diseño de sitios web así como la validación de datos (Mozilla Developer Network, 2011)

- **AJAX (JavaScript y XML asíncronos)** es una técnica de desarrollo web para crear aplicaciones interactivas, la cual se ejecuta en el navegador del usuario. Mantiene comunicación asíncrona con el servidor, para de esta forma poder realizar cambios sobre la misma página sin necesidad de recargarla (ProgramacionWeb.net, 2009).

- **PHP 5.4 o posterior (Hipertext Preprocesor)** es un lenguaje de programación que se ejecuta en el servidor. Es gratuito e independiente de la plataforma de desarrollo. Puede ser utilizado en cualquier marco de trabajo o IDE de desarrollo que lo soporte (The PHP Group, 2011).

- **PL/pgSQL** es un lenguaje de procedimientos almacenados para Postgres que está provisto por el gestor de base de datos PostgreSQL, y permite además ejecutar comandos SQL mediante un lenguaje de sentencias imperativas y uso de funciones (Martinez).

- **DQL** es el lenguaje de consulta de *Doctrine* que proporciona capacidades de consulta sobre los objetos del modelo de datos.

1.6 Conclusiones Parciales

Una vez finalizado el presente capítulo se pudo arribar a las siguientes conclusiones:

- El análisis realizado permitió detectar que los sistemas estudiados no cumplen con los requerimientos establecidos para la refactorización de los componentes con Doctrine 2.0, por lo que se debe realizar la refactorización de forma manual.

- El ambiente de desarrollo integrado por el modelo utilizado y arquitectura seleccionadas, además de los lenguajes, tecnologías y herramientas definidas, brindarán la posibilidad de obtener una solución refactorizada basada en el cumplimiento de los principios de soberanía e independencia tecnológica.

Capítulo 2 DISEÑO E IMPLEMENTACIÓN DE LA REFACTORIZACIÓN

Introducción

En el presente capítulo se especifican puntos importantes para llevar a cabo la refactorización del componente, comenzando un diseño donde se realizan los diagramas de clases del diseño, diagramas de interacción y el modelo de datos. Además se especifica la utilización de un conjunto de patrones dentro del diseño del componente para cumplir con las buenas prácticas de la programación y los estándares de calidad y rendimiento.

2.1 Diseño de la solución

El modelo de desarrollo del sistema no está totalmente acorde al propuesto por el centro, debido a que tiene características propias dentro de las cuales se encuentra que tiene como punto de partida la integración del componente con un marco de trabajo que emplea Doctrine 2 para aplicar las técnicas de la refactorización de software. Se parte del diseño pues el análisis y levantamiento de requisitos junto a la parte del diseño correspondiente a las Interfaces de Usuario fue realizado por otro equipo de desarrollo siendo aprobado por el centro y no sufre cambios en la actualidad.

Para realizar las transformaciones necesarias para trabajar los Componentes desarrollados en Sauxe 1.x en una versión superior del MT deben seguir los siguientes pasos para tener el entorno de trabajo correctamente configurado.

Lo primero es montar el marco de trabajo. Esta guía está creada para preparar el sistema operativo Ubuntu 12.04, para ello es necesario tener instalado Apache2, PHP 5.4 y como gestor de Base de Datos Postgresql. En el caso de Postgres se trabajó con la versión 8.4 para evitar problemas de compatibilidad.

Luego de instalado Apache se procede a configurarlo para levantar el marco de trabajo.

Paso 1-Editar el archivo **/etc/apache2/sites-enabled/000-default.**

```
1  <VirtualHost *:5901>
2  ServerAdmin webmaster@localhost
3  DocumentRoot /home/reink/sauxe2/web
4      <Directory />
5  Options FollowSymLinks
6  AllowOverride None
7  </Directory>
8
9  <Directory /home/reink/sauxe2/web/>
10 Options Indexes FollowSymLinks MultiViews
11         AllowOverride None
12         Order allow,deny
13         allow from all
14 </Directory>
15
```

Figura 2.Modificaciones al archivo

Se agregan esas líneas de códigos teniendo en cuenta que en la línea 1 va el puerto por donde se va a levantar el Marco de Trabajo en el ejemplo es por el 5901.

En la línea 3 se pone la dirección donde se encuentra el Marco de Trabajo y dentro de este la carpeta web para publicar el sitio.

En la línea 9 va la misma dirección que se estableció anteriormente terminándola en /.

```
16 ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/
17 <Directory "/usr/lib/cgi-bin">
18     AllowOverride None
19     Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch
20         Order allow,deny
21         Allow from all
22 </Directory>
23
24     ErrorLog ${APACHE_LOG_DIR}/error.log
25
26 # Possible values include: debug, info, notice, warn, error, crit,
27     # alert, emerg.
28     LogLevel warn
29     CustomLog ${APACHE_LOG_DIR}/access.log combined
30
```

Figura 3.Modificaciones al archivo (Continuación)

En el mismo archivo debajo de las líneas de la imagen 1 se ponen las de la imagen 2 textualmente pues esto define donde se van a registrar los reportes de errores de Apache.

El paso 3 es el último que se lleva a cabo en este archivo, pero ojo es uno de los más importantes pues en él se configura el Alias, que sirve para no tener una URL tan extensa en el navegador.

```
30
31  Alias /acaxia/ "/home/reink/sauxe2/web/saml/www/"
32     <Directory "/home/reink/sauxe2/web/saml/www/">
33     Options Indexes MultiViews FollowSymLinks
34           AllowOverride None
35     Order deny,allow
36    Deny from all
37    Allow from 127.0.0.0/255.0.0.0 ::1/128
38      </Directory>
39  </VirtualHost>
40
```

Figura 4. Configuración del Alias.

En la línea 31 se establece primeramente el nombre del alias en este caso acaxia, se debe mantener este pues es el alias definido en el Marco de Trabajo para manejar la seguridad. Luego entre comillas la dirección donde está el marco de trabajo, y se llega hasta la carpeta www de saml que se encuentra dentro de web. En la línea 32 se utiliza la misma dirección que se definió en el Alias. El resto de los parámetros se ponen textuales.

Después se reinicia el servicio Apache poniendo en consola la línea

/etc/init.d/apache2 restart.

Ahora se agrega el puerto que se asignó al virtualhost al archivo port.conf.

sudo gedit /etc/apache2/ports.conf (esto abre el archivo con permisos administrativos).

Se agrega las siguientes líneas al archivo.

NameVirtualHost *:5901

Listen 5901

Se procede a reiniciar el servicio Apache.

Luego de configurado Apache se debe preparar la Base de Datos pues el esquema seguridad sufrió modificaciones con respecto al esquema de la Base de Datos bdincidencia1.1 que es la empleada en Contabilidad.

Para esta parte se recomienda tener un Backup de las bases de datos Cedrux utilizada en el Departamento de Tecnología y uno de bdincidencia1.1

Para restaurar los backups existen dos vías: una empleando la interfaz del pgAdminIII y la otra utilizando la consola.

Antes de restaurar las bases de datos es necesario correr el script de los roles para que luego no den errores al restaurar dichas bases de datos. Este script puede encontrarse en los documentos adjuntos del trabajo de Diploma antes mencionado.

De ser necesario crear algún rol que no se encuentre en dicho script se puede hacer mediante una simple consulta SQL:

CREATE ROLE nombre del rol.

Esta línea crea el rol

ALTER ROLE nombre del rol SUPERUSER INHERIT CREATEROLE CREATEDB LOGIN.

Esta línea establece el nivel de privilegios del rol sobre las bases de datos.

Hecho esto de los roles se restauran las bases de datos **cedrux** e **incidencia1.1**.

Si se va a restaurar a través del **pgAdmin** se realizan los siguientes pasos.

1-Crear una nueva Base de Datos.

2-Dar Clic derecho encima de ella y seleccionar la opción **RESTAURAR**.

3-Se busca el lugar donde está el backup que se desea restaurar y se acepta.

El proceso comienza y para saber si concluyó satisfactoriamente al final debe retornar0

Con estos pasos se restaura desde la interfaz gráfica. Pero es más efectivo hacerlo mediante la consola.

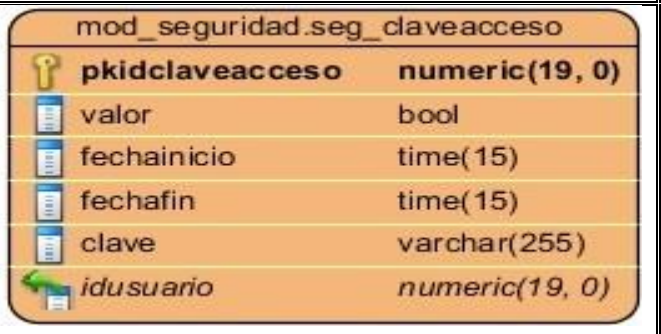
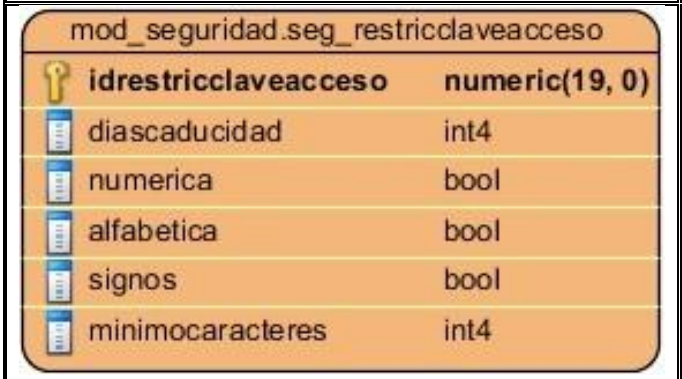
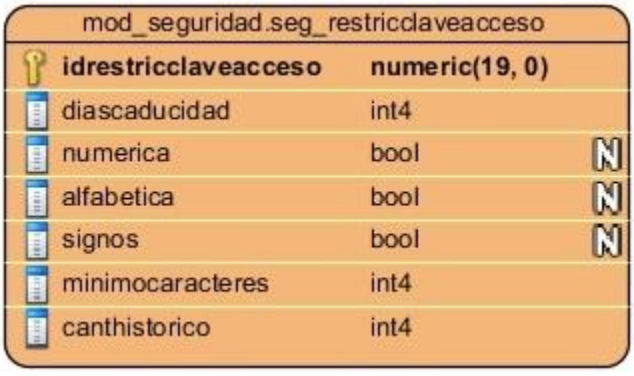
Para ello se ponen las siguientes líneas de código.

psql -d nombre base datos -f "ruta del archivo. sql" -h servidor de bd -p puerto -U postgres –W.

Después que se restauren las bases de datos debe hacerse una mezcla de ambas para obtener una que tenga los datos de incidencia1.1 que se necesitan y la estructura necesaria del

esquema seguridad para levantar la nueva versión del Marco de Trabajo. Para ello se debe comparar tabla por tabla del esquema seguridad de ambas tablas y modificar las de bdincidencia1.1 agregando las tablas que estén en la base de datos denominada sauxe y falten en la otra base de datos. También en las tablas que se mantienen se deben revisar los campos debido a que se realizaron transformaciones en el modelo de datos.

La siguiente tabla ilustra los cambios a los que fueron sometidas algunas de las tablas existentes para el correcto funcionamiento de la aplicación.

BD incidencia1.1 sin transformaciones	BD incidencia1.1 funcional para nuevo marco trabajo
	
	
	

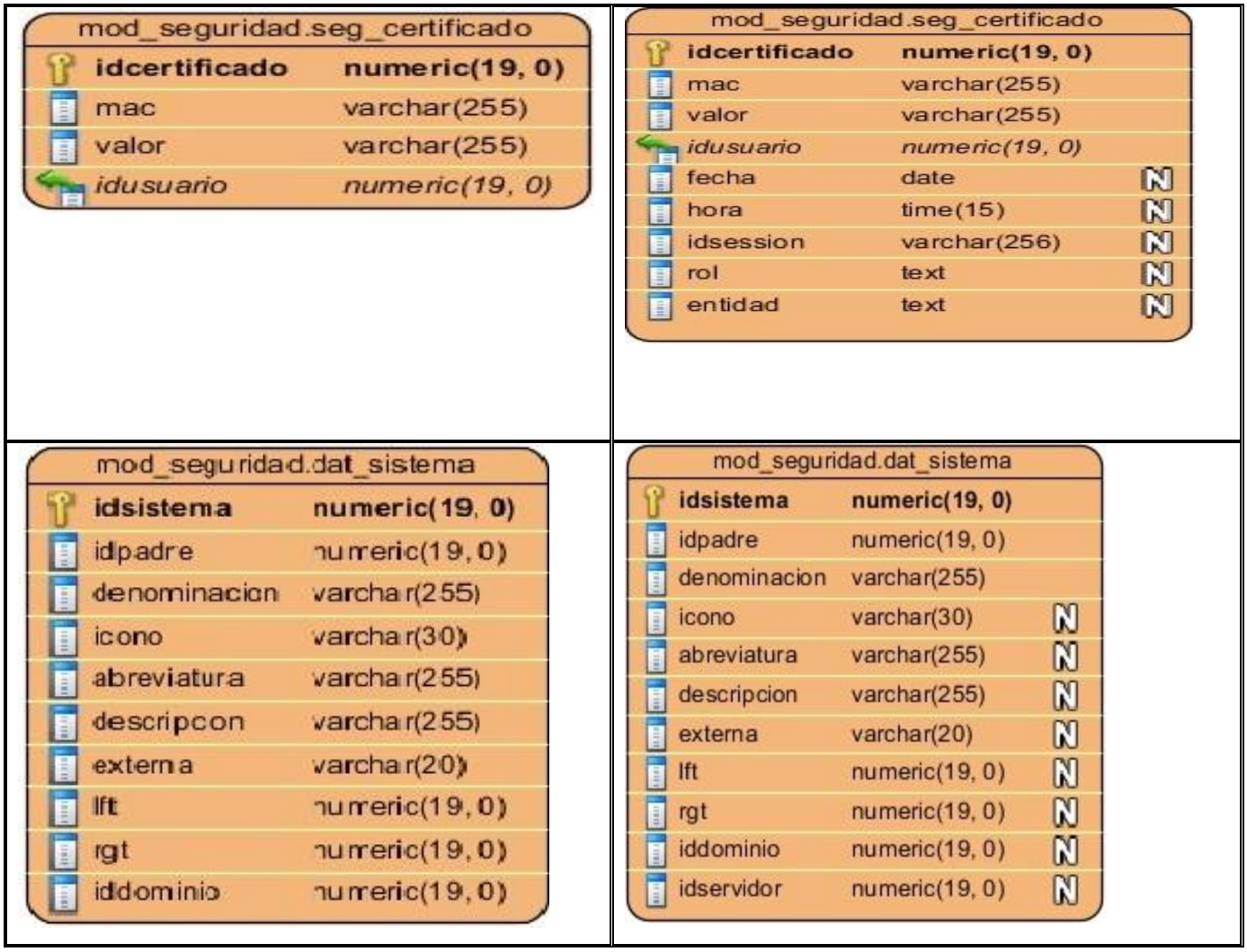


Figura 5. Cambios necesarios en los campos de las tablas

Con las bases de datos cargadas y modificadas se debe dar permisos al rol sauxe sobre las tablas de los esquemas con los que se trabajan y además debe darse permiso a los esquemas **mod_seguridad**,**mod_estructuracomp**,**mod_datosmaestros**. Cuando se dice esquemas se refiere a todos los elementos dentro del esquema (Tablas, Secuencias, Funciones).

Para cumplir esto se abre un editor de consultas SQL y se pone la siguiente sentencia

GRANT ALL ON mod_seguridad.seg_usuario TO sauxe;

En este ejemplo se da permiso al rol sauxe sobre la tabla seg_usuario del esquema mod_seguridad.

Para las secuencias es prácticamente la misma consulta

GRANT ALL ON mod_seguridad.sec_seg_usuario_sec TO sauxe.

Luego de haber realizado los pasos anteriores de proceder a insertar de forma manual los datos que a continuación se presentarán (Figura 6 y Figura 7) en las tablas **mod_seguridad.seg_claveacceso** y en la tabla **mod_seguridad.seg_certificado**

	pkidclaveacc [PK] numeric	valor boolean	clave character varying(255)	idusuario numeric(19,0	fechainicio date	fechafin date
1	9000000	TRUE	79e487994d6fa7782bd318514555a5ce	90000000000	2014-01-01	2050-01-01

Figura 6. Datos a insertar en la tabla seg_claveacceso.

	idcertificado [PK] numeric	mac character var	valor character var	idusuario numeric(19,0	fecha date	idsession character var	rol text	entidad text	hora time without
1	9000000	1	190000000000	90000000000	2013-02-15		calidad	UCI	09:15:13

Figura 7. Datos a insertar en la tabla seg_certificado

Modelo de Datos

Por el tamaño del modelo de datos se decidió para una mejor comprensión dividirlo en secciones. En la sección 1 (Ver Anexo 1) se muestran las tablas que poseen relación con la tabla usuario: señalizadas en verde las tablas que sufrieron modificaciones para llevar a cabo la integración del componente al nuevo MT y en malva las tablas agregadas. En el Anexo 2 se muestran las tablas relacionadas con la tabla sistema, la cuales se encargan de las funcionalidades, las acciones sobre cada funcionalidad. A su vez estas tablas están relacionadas con las de la sección 1 para dar al usuario logueado los permisos sobre las funcionalidades. En la sección 3 (Ver Anexo 3) se muestra un grupo de tablas que no están relacionadas entre sí, pero son necesarias para la integración del componente.

Diagrama de clases

Como plantea (Arizaca, 2009) el Diagrama de Clases es uno de los diagramas principales para el análisis y diseño de un software, ya que sirve para visualizar las distintas relaciones estructurales y de herencia. Incluye además definiciones para atributos y operaciones. En el presente trabajo se mostraran los diagramas de clases correspondiente a las funcionalidades seleccionadas para refactorizar pertenecientes al componente Configuraciones.

El archivo estructuraeconomica.phtml será el encargado de dibujar las interfaces del sistema y a su vez contendrá las clases pertenecientes a la librería ExtJS la cual es la encargada de construir dichas interfaces. Además tendrá contenido un formulario que será el encargado de enviar la información entrada por el usuario hacia el servidor, en donde mediante una instancia única con el controlador frontal de Zend, los datos serán enviados a la controladora correspondiente para ser gestionados. En este diagrama se puede observar también las clases necesarias para acceder a la información almacenada a nivel de datos que cambiaron durante el proceso de refactorización (Ver Anexo 4).

Diagrama de Secuencia

Los diagramas de interacción modelan los aspectos dinámicos de un sistema y muestran un perspectiva general del flujo de control dentro del sistema o proceso de negocio (Ávila, 2009).
Por tanto se realizaron diagramas de secuencia, que son un tipo de diagrama de interacción que modela la secuencia lógica de una acción ejecutada por un usuario y el orden en que suceden los mensajes. A continuación se muestra el diagrama de secuencia para adicionar un Contenido Económico al sistema (Ver Anexo 5).

2.2 Desarrollo de la Refactorización

Al migrar el componente se debe tener en cuenta que la estructura de carpetas cambia casi por completo en cuando a la capa de acceso a datos pues desaparece la carpeta Domain y se agregan tres nuevas carpetas llamadas Entity, Proxy y Repository, mientras que la carpeta bussines se mantiene como indica la figura 8.

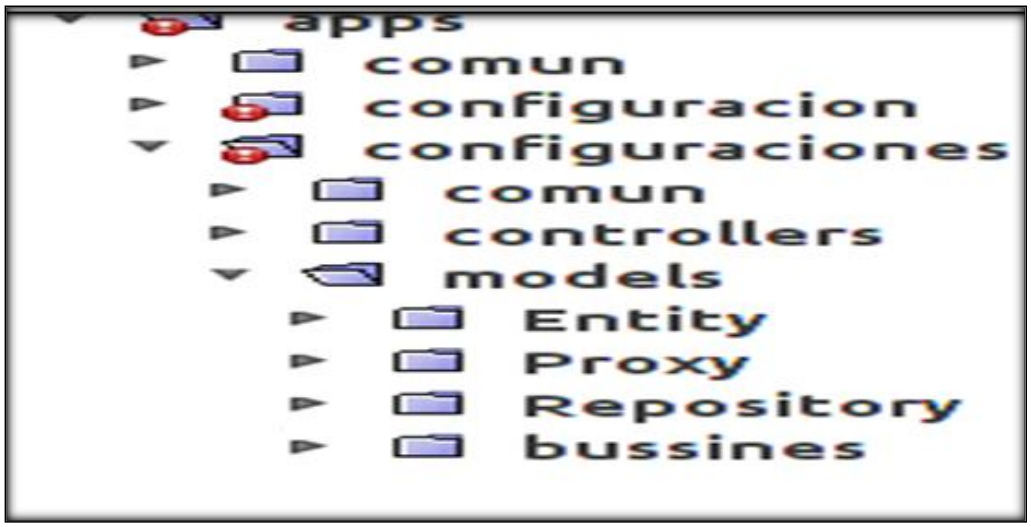


Figura 8. Estructura de carpetas

Tabla 2. Estructura de carpetas

Carpeta	Función
Entity/	En esta carpeta se almacenan las entidades mapeadas, donde se definen los atributos de las tablas de la Base de datos y las relaciones entre clases además de los métodos **Get** y **Set** para cada uno de ellos. Las clases de esta carpeta poseen una clase repositorio ubicada en la carpeta **Repository**
Proxy/	En la carpeta Proxy se ubican los ficheros generados como resultado del mapeo que contienen las clases Proxy usadas por Doctrine 2. Estas clases no son más que objetos que se ponen en el lugar del objeto real, y se utilizan para realizar varias funciones, pero principalmente, para la transparencia de carga diferida. Los objetos proxy con sus instalaciones de carga diferida ayudan a mantener el

	subconjunto de objetos que ya están en la memoria conectados con el resto de los objetos
Repository/	En esta carpeta se encuentran las clases repositorios de las clases **Entity**. Estas clases son las encargadas de implementar los métodos de acceso a datos. Los métodos aquí implementados son los que se encontraban en los ficheros de la carpeta Domain, pero con la nueva estructura de crear las consultas.
bussines/	En la carpeta bussines se mantienen las clases Models las cuales van a mediar entre la controladora y las Entity para el acceso a datos.

Para llevar a cabo la refactorización del componente se hizo uso de las técnicas descritas en el capítulo anterior. A continuación se pondrán ejemplos de la aplicación de estas técnicas para una mejor comprensión.

```
//---------- Retorno toda la informacion de las cuentas que esta en la tabla de Datos conf_cuentaredondeo ----------
$ctagastoRedondeo = ConfCuentaredondeo::cargarCtaRedondeo();
```

Figura 9. Parte de un método sin aplicar ninguna técnica refactorización.

La figura 9 muestra parte de un método que forma parte de la clase ClasificadorcuentasconfController donde se hace la llamada a un método de la clase Confcuentaredondeo, la cual se encarga de la conexión a la base de datos para obtener la información de las cuentas desarrollada con Doctrine 1.

Al utilizar Doctrine 2 como apoyo a la refactorización la llamada no es tan simple como solicitar el método directamente a la clase donde está implementado sino que se debe almacenar una instancia de la clase TransactionManager de Doctrine y solicitar la conexión activa para acceder al método como indica la figura 10.

```
// Retorno toda la informacion de las cuentas que esta en la tabla de Datos conf_cuentaredondeo
$mg=ZendExt_Aspect_TransactionManager::getInstance();

    $_em=$mg->openConections('configuraciones');
    $ctagastoRedondeo=$_em->getRepository("configuraciones\models\Entity\Confcuentaredondeo")
            ->CargarCtaRedondeo($_em);
```

Figura 10. Método empleando Doctrine2.

Siguiendo las buenas prácticas de refactorización y empleando la técnica Extraer Método explicada en el capítulo anterior se decide implementar estos métodos en las clases Models, las cuales son las encargadas de manejar el acceso a datos. Las clases Models quedan encargadas de llamar a los métodos de las clases Repository para la obtención de los datos lo que no influye en el comportamiento del diseño arquitectónico Modelo-Vista-Controlador y no se saturan los métodos.

```
//---------- Retorno toda la informacion de las cuentas que esta en la tabla de Datos conf_cuentaredondeo ---------
$ctagastoRedondeo = ConfCuentaredondeoModel::cargarCtaRedondeo();
```

Figura 11. Método Refactorizado

Otra ventaja de utilizar la migración a Doctrine 2.0 para refactorizar el componente Configuraciones es en cuanto a la relación con la base de datos y el mapeo de las tablas de la misma, junto a los métodos donde se hacen las consultas a la fuente de datos. Nótese que en los métodos implementados en la primera imagen (Doctrine 1.2) se hacen llamadas a subrutinas para establecer cada una de las partes que componen las sentencias SQL a ejecutar lo que significa que se guarde un espacio en memoria para cada una de ellas y luego un método para ensamblar cada una de las partes.

```
static public function BuscarSubsistema($idsubsistema) {
    $q = new Doctrine_Query();
    $result = $q->from("NomSub s")
            ->where("s.idsubsistema= " . $idsubsistema)
            ->execute()
            ->toArray();
    return $result;
}
```

Figura 12. Acceso a datos en Doctrine 1.2.

Por su parte luego del proceso de refactorización empleando la migración se llama a una subrutina donde va toda la sentencia, lo que equivale una sola llamada para comunicarse con la base de datos, dotando al sistema de una mayor rapidez a la hora de realizar las consultas.

```
public function BuscarSubsistema($idsubsistema, $_em) {

    $result = $_em
            ->createQuery("from configuracion\subsistemas\models\Entity\NomSub s "
                    . " where s.idsubsistema= $idsubsistema ")
            ->getArrayResult();
    return $result;
}
```

Figura 13. Acceso a datos en Doctrine 2.0

2.3 Conclusiones parciales

Entre los principales resultados alcanzados durante el desarrollo del capítulo se resaltan los siguientes:

- El diseño de la solución posibilitó el uso del componente Configuraciones del subsistema Contabilidad de Xedro-ERP, empleando Doctrine 2, en el marco de trabajo Sauxe.
- El diagrama de clases del diseño permitió conocer la estructura y las relaciones entre las clases que se manejan en el componente al aplicar la estrategia de refactorización.

Capítulo 3. VALIDACIÓN DE LA SOLUCIÓN PROPUESTA.

3.1 Introducción

En este capítulo se abordaran los supuestos teóricos para la realización de las pruebas de software para la validación de la solución desarrollada, haciendo énfasis en las pruebas de rendimiento para dar cumplimiento a uno de los objetivos específicos de la investigación.

3.2 Pruebas de software

Simultáneamente al proceso de desarrollo deben realizarse actividades que garanticen la calidad del producto que se está construyendo. Una de estas la constituye las Pruebas de Software, mediante las cuales un sistema o componente es ejecutado bajo condiciones específicas, observándose o almacenándose los resultados y realizando una evaluación de algún aspecto del sistema o componente.

Según Pressman estas pruebas de software son un elemento crítico para la garantía de la calidad del software y presenta una revisión final de las especificaciones, del diseño, y de la codificación. (Pressman, 2005)

Dichas pruebas van dirigidas a componentes del software o al sistema de software en su totalidad, con el objetivo de medir el grado en que el software cumple con los requerimientos especificados. Una forma de ejecutar las pruebas es a través de métodos los que a su vez poseen distintas técnicas que garantizan la correcta ejecución de esta actividad.

3.2.1. Métodos de Pruebas

Para realizar las pruebas se deben definir previamente los diferentes métodos de pruebas a emplear, de forma que permitan al probador comprender cómo puede realizarlas de la manera más adecuada, o sea, con una alta probabilidad de encontrar el mayor número de errores con el mínimo de esfuerzo y en el menor tiempo posible. Los métodos más utilizados son las pruebas de caja negra y las pruebas de caja blanca.

Pruebas de caja negra

Estas pruebas tienen como objetivo demostrar que las funciones del software son operativas, que las entradas se aceptan de forma adecuada y se produce un resultado correcto. y que la integridad de la información externa se mantiene. Estas pruebas comprueban si el sistema cumple con los requisitos funcionales, ignorando la estructura lógica interna del software.

Se centran principalmente en los requisitos funcionales del software que permiten encontrar: Funciones incorrectas o ausentes, errores de interfaz, errores en estructuras de datos o en accesos a las bases de datos externas, errores de rendimiento, errores de inicialización y terminación. (Pressman, 2005)

Las pruebas de caja negra tienen un conjunto de técnicas que permiten diseñar los casos de pruebas que satisfacen el siguiente criterio: reduce el número de casos de pruebas y dicen algo sobre la presencia o ausencia de errores. Estas técnicas son pruebas basadas en grafo, análisis de valores límite, partición equivalente.

Partición equivalente

Este es el método más utilizado, se enfoca en la realización de los casos de pruebas mediante la partición equivalencia, se basa en la evaluación de las clases de equivalencia para una condición de entrada, o sea, busca los distintos datos de entrada tanto válidos como no válidos que se puedan entrar a un programa para descubrir cualquier fallo que éste tenga con respecto a los valores entrados. Una condición de entrada es un valor numérico específico, un rango de valores, un conjunto de valores relacionados o una condición lógica.

El objetivo de partición equivalente es reducir el posible conjunto de casos de prueba en uno más pequeño, un conjunto manejable que evalúe bien el software.

Reglas que ayudan a definir clases de equivalencia

- Si una condición de entrada especifica un rango, se identifica una clase de equivalencia válida y dos inválidas.

-Si una condición de entrada requiere un valor específico, se identifica una clase de equivalencia válida y dos inválidas.

-Si una condición de entrada especifica un miembro de un conjunto, se identifica una clase de equivalencia válida y una inválida.

- Si una condición de entrada es lógica, se identifica una clase de equivalencia válida y una inválida.

Una vez definido la clase de equivalencia se procede a diseñar casos de pruebas para la partición de equivalencia, especificar que existen dos tipos de clases de equivalencia, la válida (entradas de valores válidos al programa) y la inválida (entradas de valores inválidos al programa). Después de asignarle un identificador a la clase de equivalencia, se procede a definir casos de pruebas con cada clase equivalencia.

-Primero se escriben caso de pruebas que cubran tantas clases equivalencias válidas sean posibles y así sucesivamente hasta que todas sean cubiertas por casos de pruebas.
- Segundo se escriben caso de pruebas que cubran tantas clase equivalencia inválidas sean posibles y así sucesivamente hasta que todas sean cubiertas por casos de pruebas.

Pruebas basadas en grafo

Definir cuáles son los objetos (objetos de datos, objetos de programa tales como módulos o colecciones de sentencias del lenguaje de programación) que se modelan en el software y las relaciones de conexión entre ellos. Se modela un grafos de estos objetos y sus relaciones y después se define una serie de pruebas que cubran todo el grafo y verifiquen que las relaciones entre estos objetos son las esperadas. (Pressman, 2005)

Análisis de valores límite

La mayoría de las veces los errores tienden a darse más en los extremos de los valores de entrada que en el centro, es por eso que con la técnica análisis de valores límite (AVL) se crean casos de pruebas que analicen los valores límite. Esta técnica complementa la de partición equivalente.

En resumen, las pruebas de caja negra se centran en lo que se espera de un módulo, es decir, intentan encontrar casos en que el módulo no se corresponda con su especificación. Por ello se denominan pruebas funcionales, y el probador se limita a suministrarle datos como entrada y estudiar la salida, sin preocuparse de lo que pueda estar haciendo el módulo por dentro. Las pruebas de caja negra están especialmente indicadas en aquellos módulos que van a ser interfaz con el usuario. O también la mayoría de las veces se diseñan los casos de pruebas para cuando el sistema esté completamente construido, o sea, cuando se hagan las pruebas de integración que es donde se involucran una serie de módulos y se termina probando el sistema en conjunto. Dentro de estas pruebas utilizan distintas técnicas que reducen el número de casos de pruebas y ayudan a detectar mayor cantidad de errores.

Pruebas de caja blanca

Se comprueban los caminos lógicos del software. Se puede examinar el estado del programa en varios puntos para determinar si el estado real coincide con el esperado (sobre el código).

Requieren del conocimiento de la estructura interna del programa, estas pruebas deben garantizar como mínimo que se ejercite por lo menos una vez todos los caminos independientes para cada módulo, que se ejerciten todas las decisiones lógicas en sus vertientes verdaderas y

falsa, que se ejerciten las estructuras internas de datos para asegurar su validez, además que se ejecuten todos los bucles en sus límites y con sus límites operacionales. (Pressman, 2005)
Las pruebas de Caja Blanca tienen un conjunto de técnicas que permiten diseñar los casos de pruebas que buscan errores en la estructura lógica de la aplicación. Entre estas técnicas se encuentra la prueba del camino básico.

Pruebas del camino básico

La prueba del camino básico es una técnica de prueba (propuesta por Tom McCabe) y la más usada.
Permite al diseñador de caso de prueba obtener una medida de la complejidad lógica de un diseño procedimental, y utilizar esta medida como guía para la definición de una serie de caminos básicos de ejecución, los casos de pruebas obtenidos del conjunto básico garantizan que cada camino se ejecuta al menos una vez o sea cada sentencia del programa.
Como apoyo para llevar a cabo la prueba del camino básico se utiliza los grafos de flujo que son para representar el flujo de control lógico de un programa. Y a este grafo se le calculará la complejidad ciclomática que no es más, que una métrica del software que proporciona una medición cuantitativa de la complejidad lógica de un programa. El valor calculado como complejidad ciclomática, define el número de caminos independiente del conjunto básico de un programa y brinda el límite superior para el número de pruebas que se deben realizar para asegurar que se ejecuta cada sentencia al menos una vez.
En el grafo de flujo el camino independiente está constituido por al menos una arista que no haya sido recorrida anteriormente. (Pressman, 2005)
Con estas pruebas se está siempre observando el código, es por eso que también se les llaman pruebas estructurales o caja de cristal. Este tipo de pruebas permitirá diseñar casos de pruebas que encuentren errores, cuando se esté implementando funcionalidades en el desarrollo del software, cuando hallan condiciones o controles anormales en el código. Estas pruebas realizan un examen minucioso de los detalles procedimentales, comprobando los caminos lógicos y bucles del programa y examinado el estado del programa en varios puntos.

Pruebas de Rendimiento

Según la IEEE: “Las pruebas de rendimientos es el grado en que un sistema o componente realiza sus funciones designadas dentro de las limitaciones dadas, tales como la velocidad, precisión o el uso de la memoria”.

Las pruebas de rendimiento tienen como propósito validar o determinar la velocidad, escalabilidad y las características de estabilidad de una aplicación que esté a prueba.

Tipos de pruebas de rendimiento

Existen diferentes tipos de pruebas de rendimiento que ayudarán a mejorar las capacidades de una aplicación observando y evaluando las respuestas del sistema ante todas las posibles situaciones que se puedan dar. Cada una de estas pruebas tiene sus objetivos y características.

Para ajustar los resultados obtenidos a los objetivos y necesidades que persigue esta investigación se centrará más en las pruebas de carga y estrés, aclarar que estas pruebas aunque estén en el grupo de las pruebas no funcionales también se pueden incluir en las funcionales, ya que además de ver los requisitos de hardware, software y del servidor, se especifican en la interfaz de la aplicación que se está probando, ejemplo: si con determinada cantidad de usuarios conectados concurrentemente, no se muestra alguna imagen, texto o gráfica.

Pruebas de Estrés

Estas pruebas son utilizadas normalmente para someter a la aplicación al límite de su funcionamiento mediante la ejecución de un número de usuarios muy superior al esperado. Esta prueba tiene como finalidad el determinar la robustez de una aplicación cuando la carga es extrema y ayuda a administradores a determinar si la aplicación se comportará debidamente ante diferentes situaciones. (2010)

Esta prueba consiste en sobrecargar la aplicación hasta que falle y determinar el punto donde empieza a tener problema en su funcionamiento. Ayuda a los administradores determinar si la aplicación rendirá lo suficiente en caso de que la carga real supere a la carga esperada. Su principal objetivo es analizar cómo se comporta el sistema cuando la carga sobrepasa su capacidad, saber si su integridad no se afecta en ningún momento durante la prueba.

Pruebas Carga

Una prueba de carga se realiza generalmente para observar el comportamiento de una aplicación bajo una cantidad de peticiones esperada. Esta carga puede ser el número esperado de usuarios concurrentes utilizando la aplicación y que realizan un número específico de

transacciones durante el tiempo que dura la carga. El resultado de esta prueba nos dará el tiempo de respuesta de todas las transacciones críticas.

Esta prueba puede mostrar los tiempos de respuesta de todas las transacciones importantes de la aplicación. Si la base de datos, el servidor de aplicaciones, y otros componentes sufren afectaciones durante la carga, la carga se diseña lo más real posible y mide anticipadamente el rendimiento que proporcionará la aplicación. Identifica los puntos de ruptura de la aplicación por debajo de la demanda máxima.

Prueba de Estabilidad

Esta prueba se realiza para determinar si la aplicación puede aguantar varias pruebas de cargas seguidas o sea que se prueba continuamente según el total de carga que soporte el sistema. Estas pruebas buscan posibles deterioros o degradaciones en el rendimiento del sistema.

Las pruebas de rendimiento pueden tener distintos objetivos. Por ejemplo, pueden demostrar que el sistema cumple los criterios de rendimiento preestablecidos o pueden medir que partes del sistema o que carga hacen que el sistema rinda de forma incorrecta, permite saber qué tan bien se comportará una aplicación y cuántos usuarios concurrentes soportará en condiciones extremas. Pueden servir para validar y verificar requisitos en la calidad del sistema como escalabilidad, fiabilidad, estabilidad, velocidad y uso de los recursos.

3.3 Estrategia de validación de la solución

Para llevar a cabo la validación de la solución se realizarán pruebas de rendimiento descritas en el epígrafe anterior, además de las prueba de carga, estrés y estabilidad se tendrá en cuenta el flujo de datos con el que puede tratar el sistema y el tiempo total que demorá el sistema en realizar las operaciones solicitadas por el usuario.

Para obtener estos indicadores se hará uso de la herramienta JMeter 2.3.1.

JMeter una herramienta Java dentro del proyecto de Jakarta, que permite realizar pruebas de rendimiento y pruebas funcionales sobre aplicaciones web y bases de datos. JMeter se destaca por su versatilidad, estabilidad, y por ser de uso gratuito. JMeter permite realizar pruebas web clásicas, pero también permite realizar test de FTP, JDBC, JNDI, LDAP, SOAP/XML-RPC y Web Service (en Beta). También permite la ejecución de pruebas distribuidas entre distintos ordenadores para realizar pruebas de rendimiento.

La Figura 15 muestra como se ve el informe de resultados que proporciona la herramienta al realizar las diferentes pruebas de rendimiento.

# Muestras	Media	Mediana	Linea de 90%	Mín	Máx	% Error	Rendimiento	Kb/sec
100	8306	9369	12167	493	12629	0,00%	7,2/sec	129,0
300	4242	4858	5861	323	12262	0,00%	10,1/sec	181,0
100	4208	4500	5886	480	6272	0,00%	3,7/sec	65,7
500	5048	5004	9382	323	12629	0,00%	16,2/sec	289,6

Figura 15. Informe resultante de las pruebas con Jmeter.

Muestras: El número de muestras para cada URL.

Media: El tiempo medio transcurrido para un conjunto de resultados.

Mín: El mínimo tiempo transcurrido para las muestras de la URL dada.

Máx: El máximo tiempo transcurrido para las muestras de la URL dada.

Error %: Porcentaje de las peticiones con errores.

Rendimiento: Rendimiento medido en base a peticiones por segundo/minuto/hora.

Kb/sec: Rendimiento medido en Kilobytes por segundo.

Al realizar las pruebas de estrés se ven reflejada en la columna %error, mientras menor sea el valor de esta columna mejor es el comportamiento del sistema ante la cantidad de usuarios conectados simultáneamente. En cuanto a las pruebas de carga se analiza el valor de la columna Kb/sec, mientras mayor sea el valor mejor cantidad de datos soporta el sistema. Para evaluar la estabilidad se debe tener en cuenta de que esta es inversamente proporcional al valor resultante en la columna %Error, mientras menor sea este valor mayor será la estabilidad del sistema analizado.

Además de estas pruebas básicas se tendrá en cuenta para el análisis del rendimiento con el flujo de datos por petición, el cual se calcula dividiendo el valor de la columna Kb/sec entre la columna rendimiento. Este flujo representa el máximo volumen que puede manejar el sistema por cada petición.

Otro modo de evaluar el rendimiento es el tiempo total que el sistema tarda en realizar las operaciones solicitadas por el usuario, para ello se multiplicará el valor de la columna # Muestras por el valor de la columna Media.

Las pruebas se realizaron para la cantidad de 100, 250 y 500 usuarios obteniéndose los siguientes resultados:

	# Muestras	Media	Mediana	Linea de 9...	Mín	Máx	% Error	Rendimi...	Kb/sec
	100	14252	12654	23674	1692	31447	0,00%	3,1/sec	5,5
	300	7477	7042	14198	186	18373	0,00%	6,0/sec	10,9
	100	7309	6877	12573	588	16323	0,00%	2,1/sec	4,2
TOTAL	500	8798	7650	16323	186	31447	0,00%	9,6/sec	17,8

Figura 16. Pruebas para 100 usuarios antes de refactorizar.

	# Muestras	Media	Mediana	Linea de 9...	Mín	Máx	% Error	Rendimi...	Kb/sec
	250	64928	98813	99788	1556	100531	0,00%	2,5/sec	3,8
	750	37618	18220	82950	846	180193	0,00%	3,1/sec	4,5
	250	34782	25006	96941	4250	119091	0,00%	1,1/sec	1,6
TOTAL	1250	42513	25143	99200	846	180193	0,00%	5,2/sec	7,5

Figura 17. Pruebas para 250 usuarios antes de refactorizar.

	# Muestras	Media	Mediana	Linea de 9...	Mín	Máx	% Error	Rendimi...	Kb/sec
	500	95310	90180	178629	3697	219541	0,00%	2,3/sec	3,2
	1500	70154	73596	130374	220	343410	3,47%	3,1/sec	4,6
	500	66626	56834	119544	3566	237007	4,20%	1,1/sec	1,6
TOTAL	2500	74480	77534	144215	220	343410	2,92%	5,2/sec	7,6

Figura 18. Pruebas para 500 usuarios antes de refactorizar.

Al realizar las pruebas al sistema refactorizado los resultados obtenidos fueron los siguientes:

	# Muestras	Media	Mediana	Linea de 9...	Mín	Máx	% Error	Rendimi...	Kb/sec
	100	14252	12654	23674	1692	31447	0,00%	3,1/sec	5,5
	300	7477	7042	14198	186	18373	0,00%	6,0/sec	10,9
	100	7309	6877	12573	588	16323	0,00%	2,1/sec	4,2
TOTAL	500	8798	7650	16323	186	31447	0,00%	9,6/sec	17,8

Figura 19. Pruebas para 100 usuarios luego de refactorizar.

	# Muestras	Media	Mediana	Linea de 90%	Mín	Máx	% Error	Rendimiento	Kb/sec
	250	17550	17731	30302	584	33582	0,00%	6,9/sec	123,4
	750	10562	11733	13904	743	33524	0,00%	10,5/sec	187,9
	250	10793	11240	13258	787	15228	0,00%	3,7/sec	67,0
TOTAL	1250	12006	11858	18185	584	33582	0,00%	17,3/sec	309,6

Figura 20. Pruebas para 250 usuarios luego de refactorizar.

	# Muestras	Media	Mediana	Linea de 90%	Mín	Máx	% Error	Rendimiento	Kb/sec
	500	51749	32920	95405	659	127335	6,20%	3,8/sec	64,8
	1500	11656	12452	14979	795	90011	0,00%	10,7/sec	191,2
	500	11535	12858	14916	1012	16296	0,00%	3,7/sec	65,4
TOTAL	2500	19650	13476	33677	659	127335	1,24%	17,7/sec	313,6

Figura 21. Pruebas para 500 usuarios luego de refactorizar.

3.4 Interpretación de los resultados

Para realizar un análisis detallado de los resultados arrojados con las pruebas se hace una interpretación de los mismos a través de la Figura 22.

		Antes			Después		
Estrés	Cantidad de usuarios	100	250	500	100	250	500
	% de error	0.0	0.0	2.92	0.0	0.0	0.0
Carga	Datos/seg	17.8	7.5	7.6	289	389	313
Estabilidad	Resp N Cargas	Si	Si	No	Si	Si	Si
Flujo de datos X petición	Carga/peticiones	1.85	1.44	1.46	17.9	17.8	17.9
Tiempo total utilizado	muestras * media	74	885	3103	42	250	818

Figura 22. Comparativa de las pruebas realizadas.

El estrés es medido por dos factores importantes: la cantidad de usuarios conectados simultáneamente a la aplicación y el porciento de error en el que incurre la aplicación por la cantidad de peticiones. Para 100 y 200 usuarios el estrés se comporta de la misma forma para el sistema antes de refactorizar y luego de aplicar la refactorización, pero para los 500 usuarios el sistema sin refactorizar tiene fallos mientras que después de aplicar la solución no da error, lo que significa que después de refactorizar el sistema es capaz de soportar mayor números de conexiones simultáneas sin presentar errores.

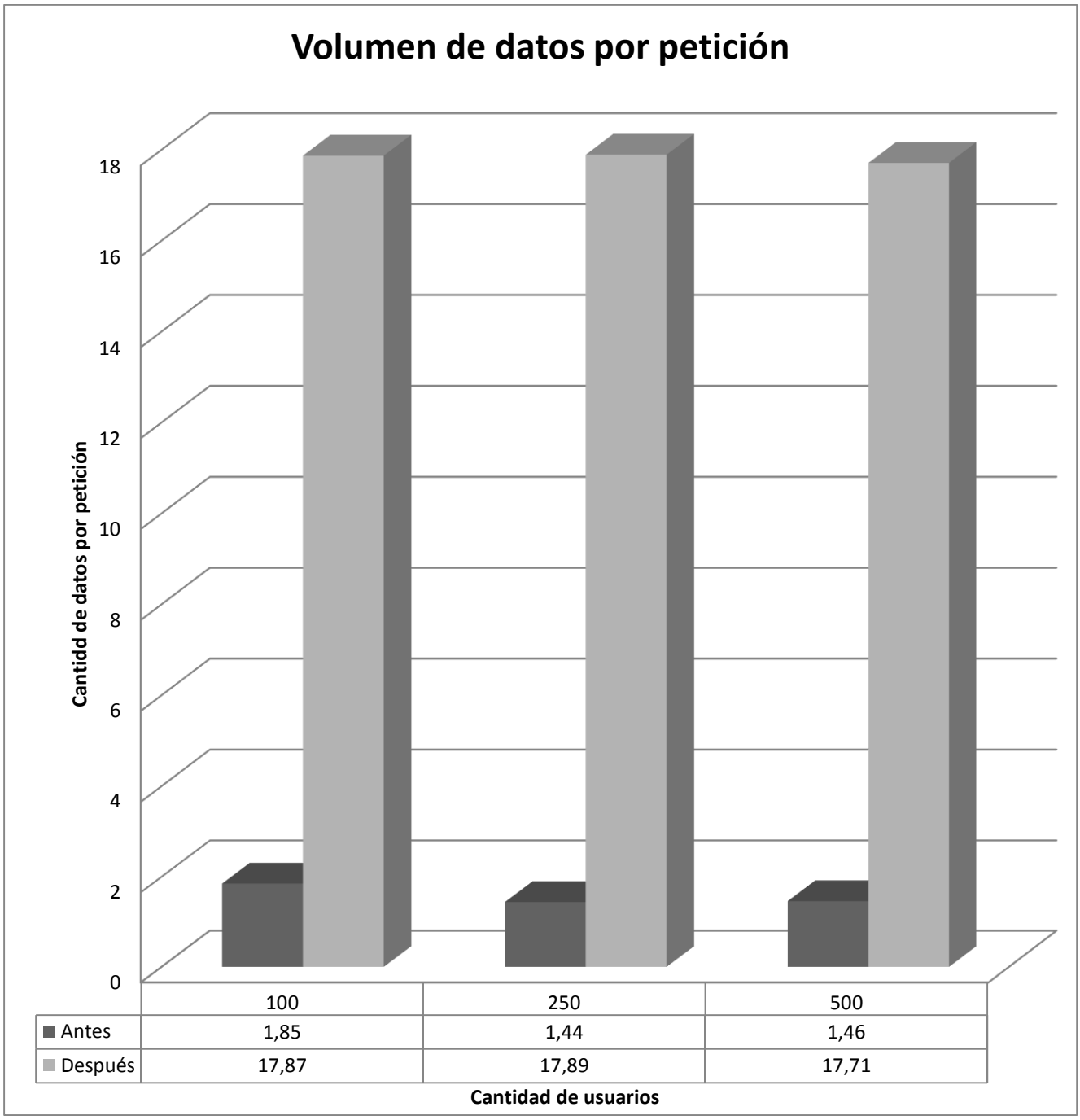
En cuanto a la carga la evaluación se realizó teniendo en cuenta la cantidad de datos por segundos que es capaz de procesar el sistema, en la tabla se refleja que antes de refactorizar la cantidad de datos por segundos que procesa el sistema es aproximadamente 17 veces menor que la que puede soportar la aplicación luego de refactorizarlo.

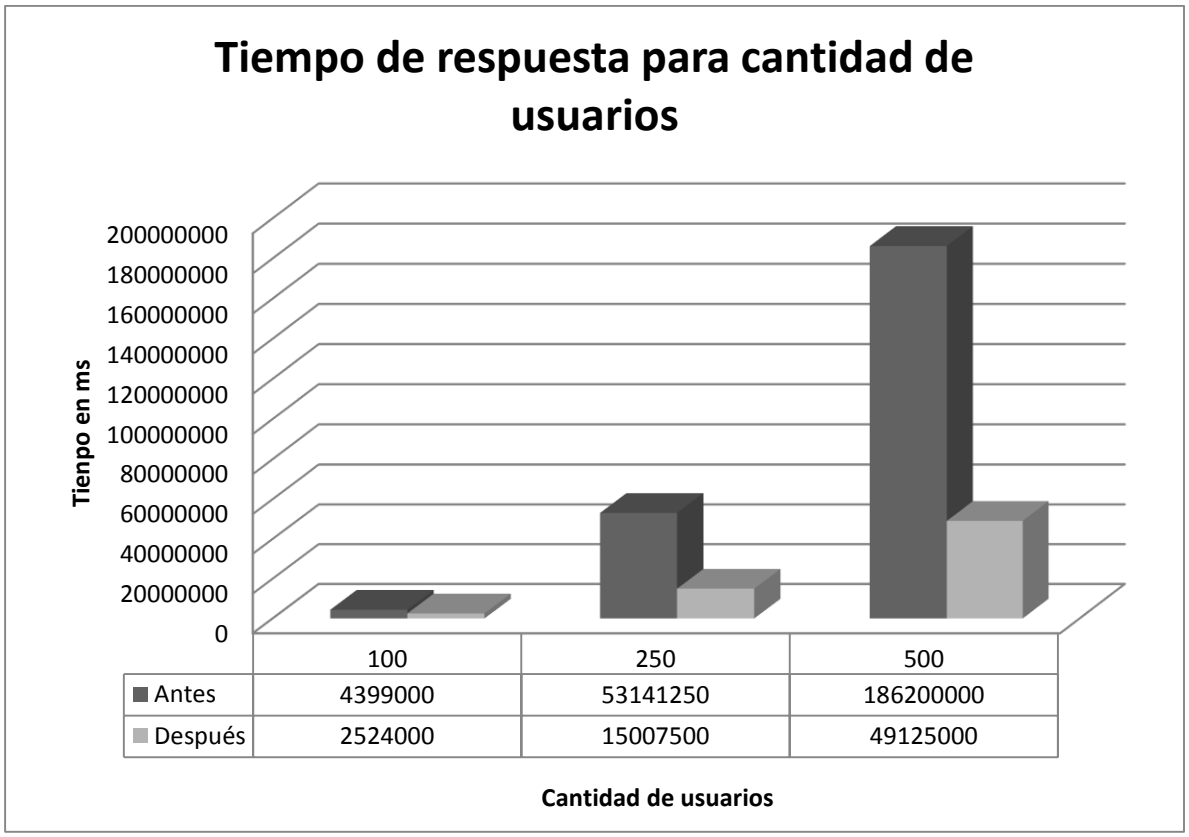
En cuanto a la estabilidad el comportamiento fue similar para 100 y 250 usuarios, pero la diferencia se nota al llegar a los 500 usuarios pues antes de refactorizar el sistema no fue estable, presentando un 2.92% de error, mientras que el sistema luego de refactorizado si se mantiene estable al tener 500 usuarios realizando peticiones de forma simultánea.

Otro criterio tenido en cuenta es el flujo de datos por petición que puede soportar el sistema, o sea cual es el volumen de datos máximo por petición que puede soportar el sistema, lo que significa que mientras mayor sea este volumen mayor cantidad de datos puede procesar el sistema para las peticiones, en el caso del componente en cuestión el flujo de datos por petición era menor antes de refactorizar, luego de refactorizar el volumen de datos que se puede procesar es mayor.

En cuanto al tiempo de respuesta se evidenció una notable mejoría pues antes de refactorizar para 100 usuarios el sistema tardó 74 minutos aproximadamente en cumplir con las peticiones de los usuarios, después de refactorizar solo tardó aproximadamente 42 minutos. A medida que aumentó la cantidad de usuarios también lo hizo la diferencia en los tiempos de respuesta. Para 250 usuarios el tiempo de respuesta fue de 885 minutos mientras que luego de refactorizar solo demoró 250 minutos. La mayor diferencia se aprecia cuando el número de usuarios aumenta a 500 el tiempo de respuesta en minutos es de 3103, mientras que luego de refactorizar fue de 818 minutos evidenciando una mejoría considerable y lo más significativo es que el resultado para 500 usuarios en el sistema refactorizado es inferior al sistema antes de refactorizar con 250 usuarios.

Para una mejor comprensión visual de estos resultados puede verse las gráficas que a continuación se muestran.


Volumen de datos por petición
Cantidd de datos por petición
0
2
4
6
8
10
12
14
16
18
100
250
500
Antes
1,85
1,44
1,46
Después
17,87
17,89
17,71
Cantidad de usuarios



3.5 Conclusiones parciales

La validación de la propuesta de solución arrojó como resultado

- Luego de refactorizar el número de usuarios soportado por la aplicación antes de presentar algún fallo es mayor que el que soportaba antes de refactorizar.
- El volumen de datos procesados por el sistema antes de refactorizar es menor, lo que indica que la refactorización influyó de forma positiva en el rendimiento del mismo.
- El sistema es más estable luego de refactorizado debido a que soporta mayor cantidad de actividad por parte de los usuarios.
- El tiempo de respuesta disminuyó de forma considerable al llevarse a cabo el proceso de refactorización lo que evidencia la mejoría aportada por la solución propuesta.

Conclusiones Generales

Una vez terminado el presente Trabajo de Diploma se puede concluir que se desarrollaron todas las tareas a fin de cumplir los objetivos propuestos, resaltando que:

- El Marco Teórico de la investigación posibilitó sustentar los conceptos y la propuesta de desarrollo evidenciando la necesidad de la refactorización del componente Configuraciones del subsistema Contabilidad de Xedro-ERP.
- El diseño e implementación de la solución posibilitó establecer las bases para mejorar el diseño del código mediante su refactorización.
- La validación de la solución permitió evaluar su rendimiento arrojando resultados favorables.

Recomendaciones

- Potenciar la refactorización como estrategia para disminuir el costo del mantenimiento del sistema.
- Extender la actualización de las tecnologías de terceros como técnica de refactorización a otras capas del sistema.

Glosario

Ant: Es una herramienta Open-Source utilizada en la compilación y creación de programas Java, Está escrito en XML y Java.

BSD: La licencia BSD (Berkeley Software Distribution) fue creada inicialmente para los sistemas operativos de la Universidad de Berkeley. Se califica como una licencia mucho más libre que la GPL y más rápida.

Log: Registro oficial de eventos durante un rango de tiempo en particular. Para los profesionales en seguridad informática es usado para registrar datos o información sobre: quien, que, cuando, donde y por qué: un evento ocurre. Por lo tanto se puede decir que un log es una evidencia digital.

Secure Sockets Layer: protocolo criptográfico que proporciona comunicación segura en Internet.

ExtJS: Es una librería construida con JavaScript para la construcción de componentes para el diseño de interfaces de usuario del lado del cliente haciendo uso extensivo de Ajax. Emplea una arquitectura flexible que permite construir aplicaciones complejas utilizando componentes predefinidos.

Doctrine: Es un mapeador de objetos relacional de para PHP. Permite trabajar con los datos persistidos como si fueran parte de una base de datos orientada a objetos posibilitando escribir consultas a la base de datos en un dialecto orientado a objetos de propiedad SQL.

Bibliografía

Martínez Rivera Daniela. 2012. Herramienta Case Visual Paradigm. [En línea] 17 de 4 de 2012. [Citado el: 16 de 1 de 2013.] http://dianbeel.blogspot.com/2012/06/segundo-trabajo-herramienta-case-visual.html.

Andux, Yadira Piñera. 2010. Formalización y estandarización de la documentación técnica de la arquitectura tecnológica del Marco de Trabajo Sauxe versión 2.0. La Habana, Cuba : UCI, 2010.

Apache Software Foundation. [En línea] [Citado el: 15 de 12 de 2013.] http://www.apache.org.

Aquino, A.y.L., Yasser. 2009. Implementación del módulo de Contabilidad General del Sistema Integral de Gestión Cedrux. 2009.

Arizaca, Lic. Eliza. 2009. Análisis y diseño de sistemas II. La Paz, Bolivia : s.n., 2009.

Ávila, Emilio Aviles: Diagramas de iteracción (Secuencia y Comunicación/Colaboración). 2009. Slideshare. [En línea] 07 de 06 de 2009. [Citado el: 20 de 2 de 2013.] http://www.slideshare.net/techmi/curso-uml-23-diagramas-de-interaccin.

Barnes, D.J. y M. Kölling. 2007. Programación orientada a objetos con Java. [En línea] 2007. [Citado el: 02 de 05 de 2013.] http://predesys.googlecode.com/svn-history/r27/trunk/documentation/chapters/systemtests/unittests.tex.

Baryolo, Oiner Gómez, Cabrera, Marianela Tenrero y Silega, Nemuris Martínez. 2008. Plantilla Registro de la Propiedad intelectual(Sauxe). La Habana : UCI, 2008.

CENDITEL. 2011. Plataforma de Desarrollo de Software Libre (PDSL). Manual del Usuario del Sistema de Control de Versiones (SVN). [En línea] CENDITEL, 2011. [Citado el: 20 de 11 de 2012.] http://plataforma.cenditel.gob.ve/wiki/ManualUsuarioSvn.

—. 2011. Plataforma de Desarrollo de Software Libre (PDSL). Manual del Usuario del Sistema de Control de Versiones (SVN). [En línea] CENDITEL, 2011. [Citado el: 30 de Noviembre de 2011.] http://plataforma.cenditel.gob.ve/wiki/ManualUsuarioSvn.

Colectivo de JSON. 2011. JSON. [En línea] JSON, 2011. [Citado el: 30 de Noviembre de 2011.] http://www.json.org/json-es.html.

Colectivo editorial. 2008. W3C Consortium. [En línea] 9 de 1 de 2008. [Citado el: 19 de 11 de 2012.] http://www.w3c.es/divulgacion/guiasbreves/tecnologiasxml.

Cristian Salazar. 2012. Modelo de datos. [En línea] 28 de 09 de 2012. http://www.slideshare.net/csalazarc/modelo-de-datos-14506949.

Doctrine-Project.org. 2013. Doctrine. [En línea] 2013. [Citado el: 16 de Febrero de 2014.] http://www.doctrine-project.org.

EcuRed. Patrones de diseño y arquitectura. [En línea] [Citado el: 8 de 12 de 2012.] Patrones_de_diseño_y_arquitectura.htm.

Fowler, Martin. 2003. Refactoring - Improving the Design of Existing. 2003.

—. 2003. Patterns of enterprise application architecture. s.l. : Addison-Wesley, 2003. ISBN 978-0-321-12742-6.

Grupo de autores. 2012. HTML.net. [En línea] 2012. [Citado el: 16 de Enero de 2012.] http://es.html.net/tutorials/html/lesson2.php.

HTML.net. 2012. HTML.net. [En línea] 2012. [Citado el: 19 de 11 de 2012.] http://es.html.net/tutorials/html/lesson2.php.

IEEE. 2000. 2000. Std 1471-2000.

Jatun S.R.L. 2005. Java Enterprise Edition. [En línea] 2005. [Citado el: 8 de 12 de 2012.] http://www.jatun.com/web/company/training/javaee5.

Josep Casanovas. 2009. Usabilidad y arquitectura del software. [En línea] 9 de 9 de 2009. [Citado el: 8 de 12 de 2012.] 1622.php.htm.

Larman, Craig. 1999. UML y Patrones Introducción al análisis y diseño orientado a objetos. México : Prentice Hall, 1999. ISBN 0-13-748880-7.

Martinez, Rafael. PostgreSQL-es. [En línea] [Citado el: 26 de Noviembre de 2011.] http://www.postgresql.org.es/node/297.

Mastermagazine. 2013. Mastermagazine. [En línea] 14 de 11 de 2013. [Citado el: 19 de 1 de 2014.] www.mastermagazine.info.

Microsoft Dynamics AX. 2011. Gestión Financiera. 2011.

Mozilla Developer Network. 2011. Mozilla Developer Network. [En línea] 09 de 11 de 2011. [Citado el: 5 de 2 de 2014.] https://developer.mozilla.org/es/docs/Gu%C3%ADa_JavaScript_1.5/Concepto_de_JavaScript.

—. 2011. Mozilla Developer Network. [En línea] 09 de 11 de 2011. [Citado el: 5 de 22 de 2013.] https://developer.mozilla.org/es/docs/Gu%C3%ADa_JavaScript_1.5/Concepto_de_JavaScript.

Mozilla Firefox. 2009. GetFirefox. [En línea] 2009. [Citado el: 30 de Noviembre de 2011.] http://www.getfirefox.es/firefox-features..

Obregón, Ing. William González. 2012. Modelo de desarrollo de software. La Habana, Cuba : UCI, 2012.

Oracle Corporation. NetBeans. [En línea] [Citado el: 30 de Noviembre de 2013.] http://netbeans.org/community/releases/70/.

Parametric Technology Corporation. 2006. Verificación y validación. 2006. 2089-VV-TS-EN1206-ES.

Patrones Arquitectónicos . [En línea] [Citado el: 16 de 1 de 2013.] http://isg3.pbworks.com/w/page/7624479/Patrones%20Arquitect%C3%B3nicos.

Postgre-SQL. [En línea] [Citado el: 3 de 3 de 2014.] http://www.postgresql.org.es.

Pressman, Roger S. 2005. Ingeniería del Software. Un enfoque práctico. 2005.

2009. Proceso de Desarrollo y Gestión de Proyectos de Software. 2009.

ProgramacionWeb.net. 2009. ProgramacionWeb.net. [En línea] ProgramacionWeb.net, 2009. [Citado el: 19 de 11 de 2012.] http://www.programacionweb.net/cursos/curso.php?num=2.

Proyecto SAUXE. 2012. Arquitectura de Software. Vista de integración. 2012.

Ramírez, Jaime. 2012. Métodos de Prueba del Software. Unidad de Programación. [En línea] 2012. [Citado el: 28 de mayo de 2012.] http://lml.ls.fi.upm.es/.

2010. Testhouse.Pruebas de Rendimiento. [En línea] 2010. [Citado el: 15 de 1 de 2014.] http://www.es.testhouse.net/pruebas-de-rendimiento.

The FreeBSD Foundation. 2012. The FreeBSD Project. [En línea] 2012. [Citado el: 23 de 11 de 2012.] http://www.freebsd.org/doc/es/articles/explaining-bsd/article.html.

The PHP Group. 2011. PHP: Hipertext Preprocesor. [En línea] php.net, 4 de Febrero de 2011. [Citado el: 25 de Noviembre de 2011.] http://php.net/manual/es/intro-whatis.php.

—. 2011. PHP: Hipertext Preprocesor. [En línea] php.net, 4 de 2 de 2011. [Citado el: 19 de 11 de 2012.] http://php.net/manual/es/intro-whatis.php.

2012. THEFREEDICTIONARY. [En línea] 2012. http://es.thefreedictionary.com/vale.

Universidad Politécnica de Madrid. Patrones del "Gang of Four". Facultad de informática. Madrid : s.n.

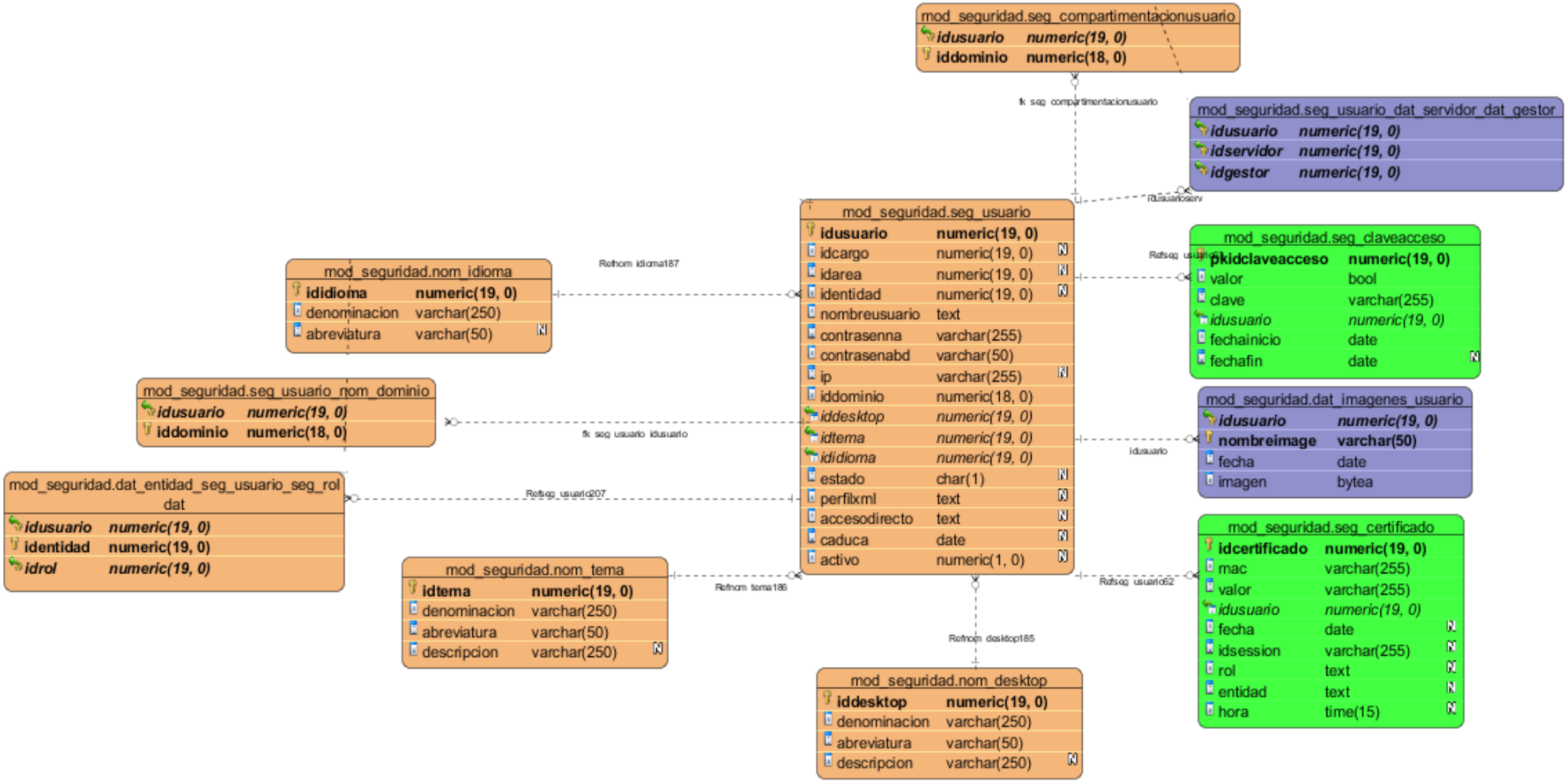
—. 2010. Patrones del "Gang of Four". Facultad de informática. Madrid : s.n., 2010.

Velázquez, Inoelkis. 2013. Migración de la capa de Acceso a Datos del Marco de Trabajo Sauxe. La Habana : s.n., 2013.

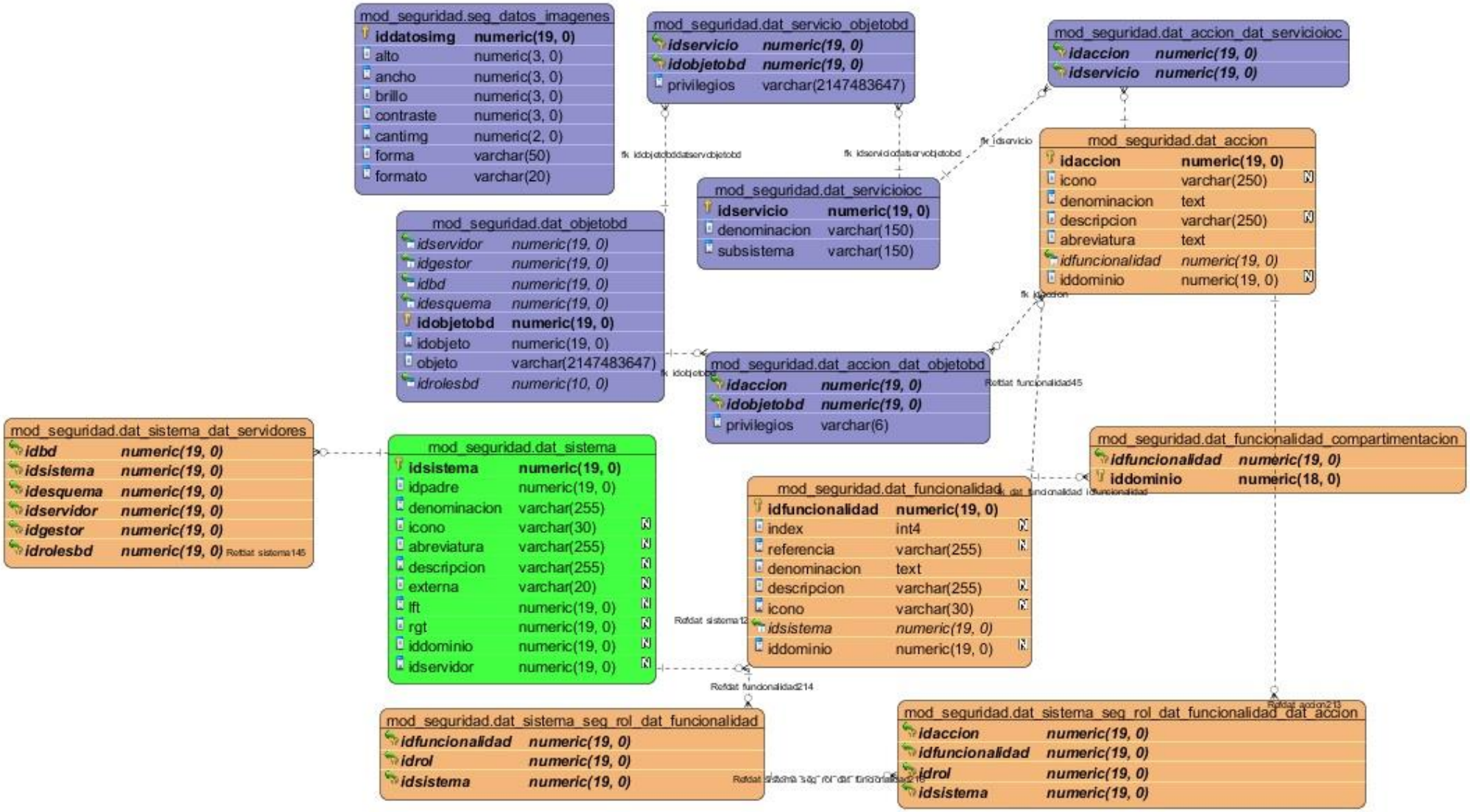
Visual Paradigm. 2007. freedownloadmanager.org. [En línea] 5 de 3 de 2007. [Citado el: 20 de 11 de 2012.] http://www.freedownloadmanager.org/es/downloads/Paradigma_Visual_para_UML_%28M%C3%8D%29_14720_p/.

W3Schools. 2012. W3Schools Online Web Tutorials. [En línea] 2012. [Citado el: 19 de 11 de 2012.] http://www.w3schools.com/js/js_intro.asp.

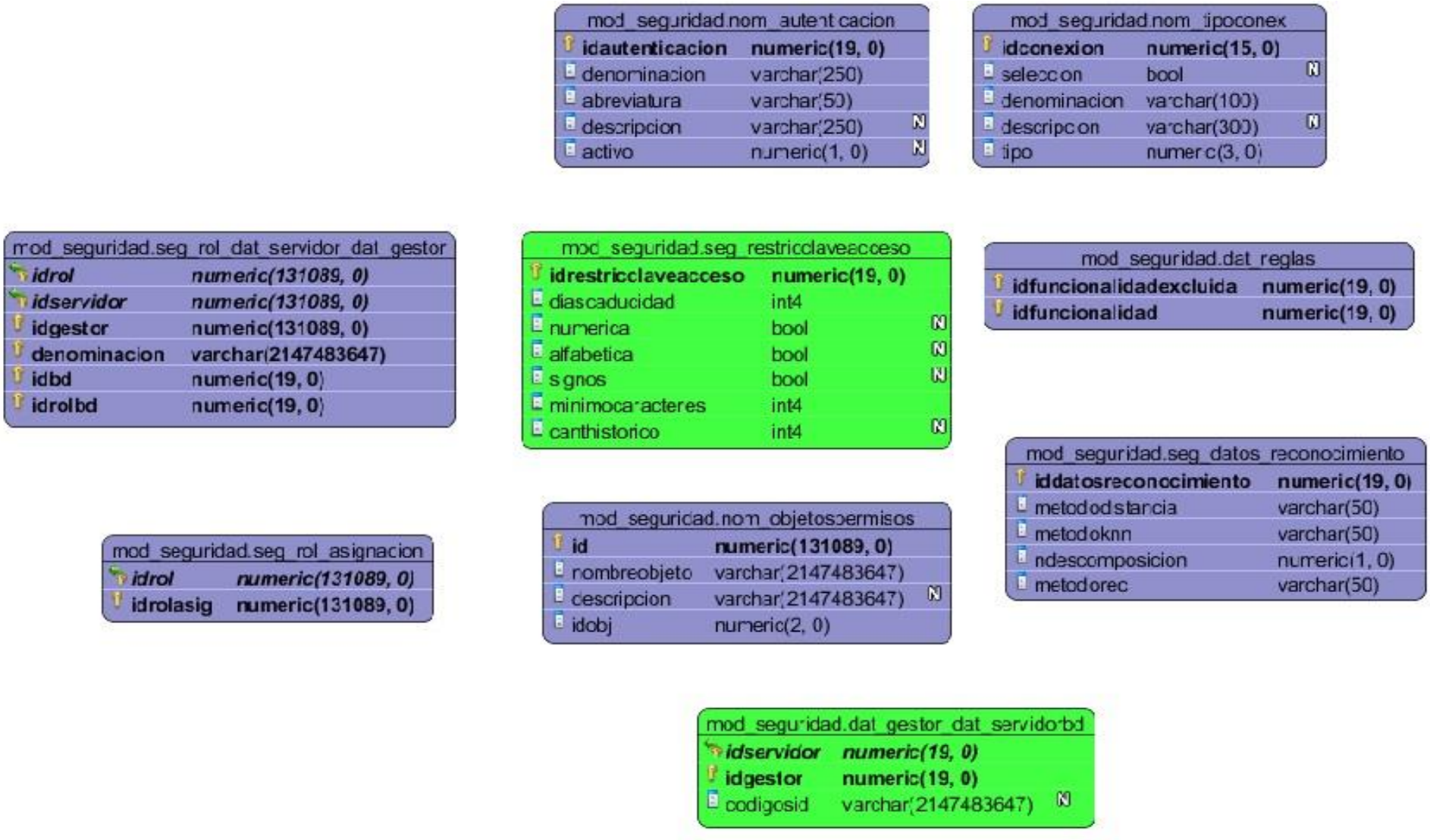
ww. [En línea]



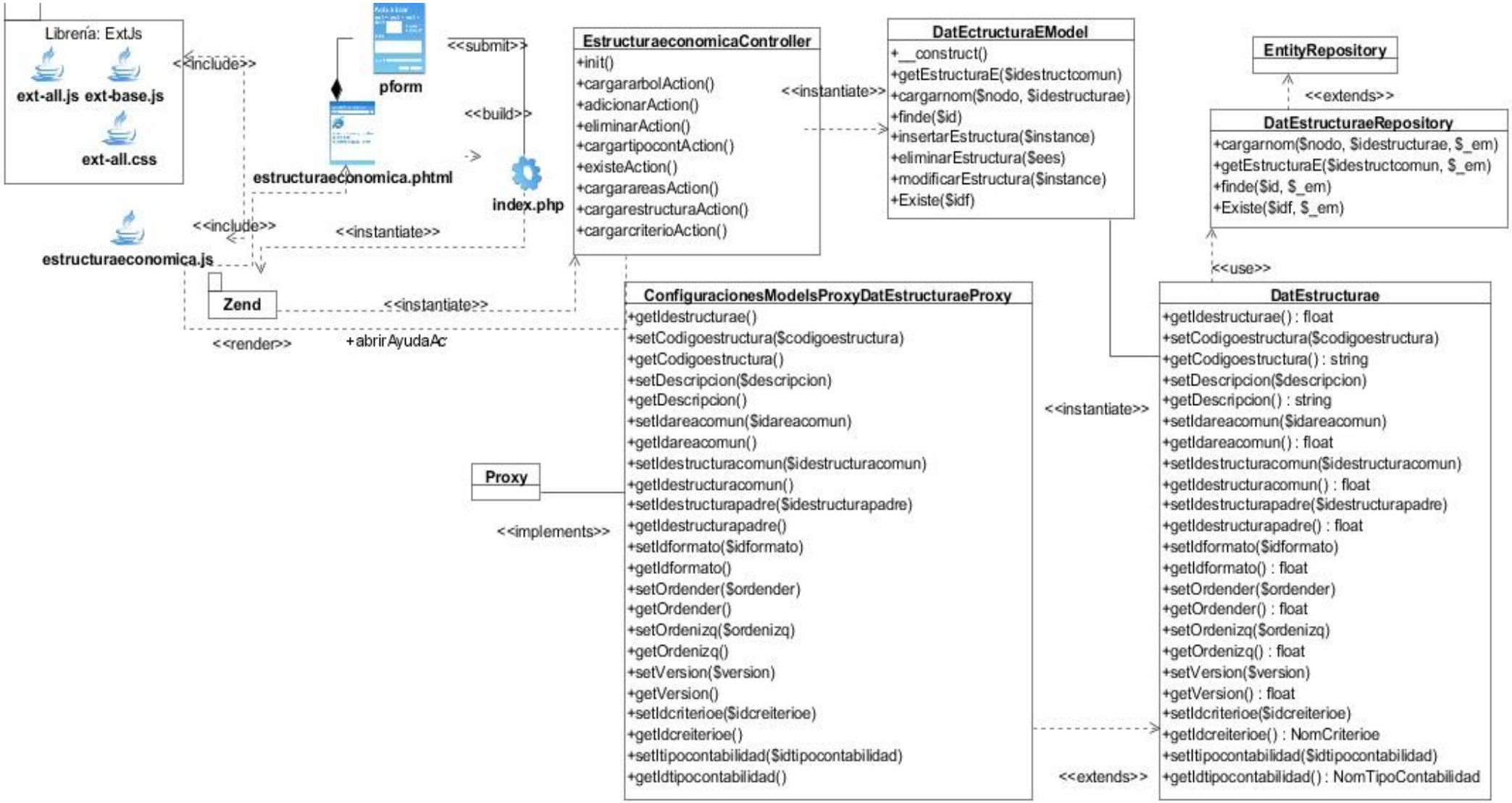
Anexo 1: Sección 1 del modelo de datos relacionada con la tabla usuario.



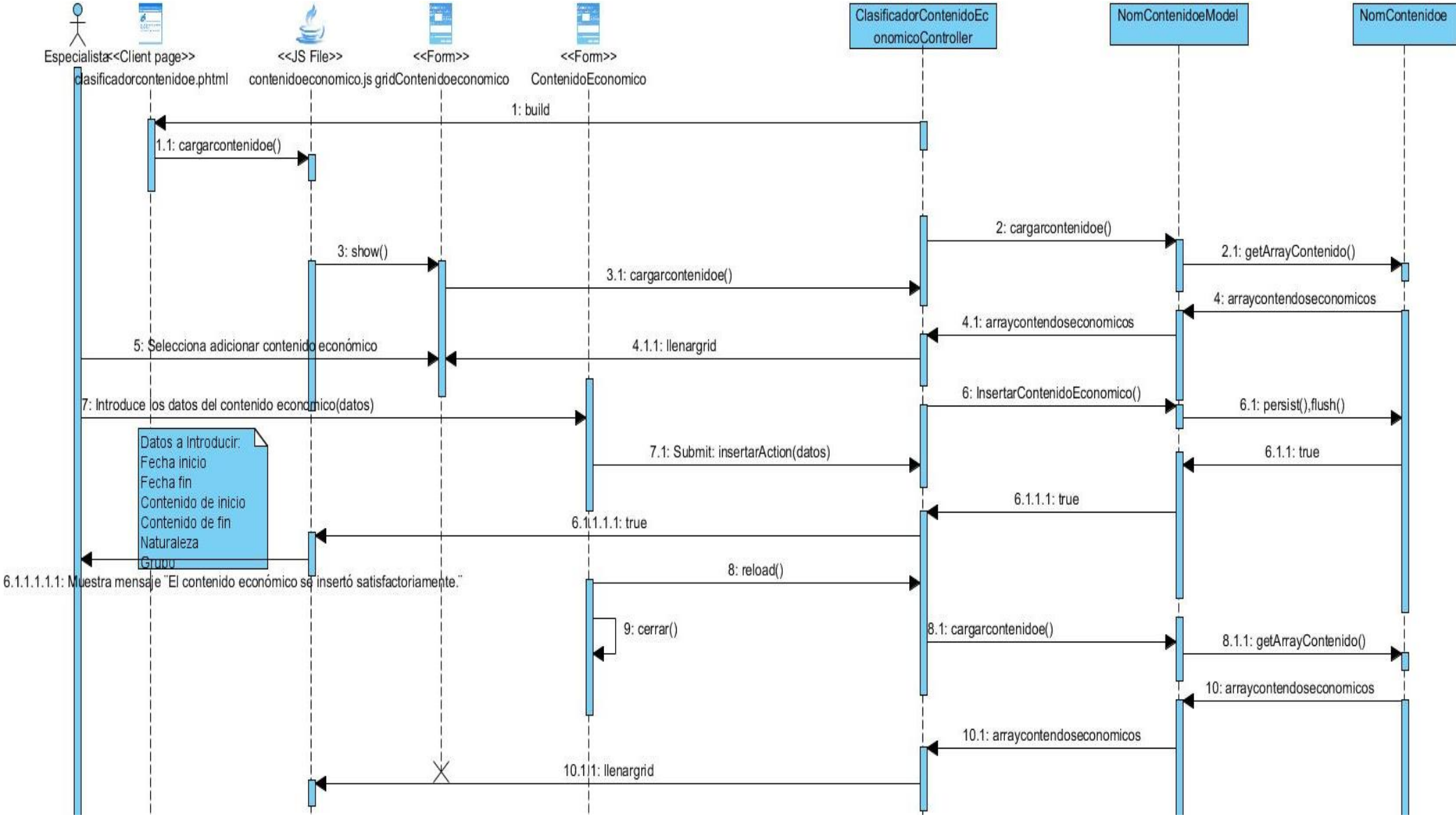
Anexo 2: Sección 2 del modelo de datos relacionadas con la tabla dat_sistema



Anexo 3: Sección 3 del modelo de datos relacionada con otras tablas requeridas por el sistema



Anexo 4.Diagrama de clases del componente Estructura Económica.



Anexo5.Diagrama de secuencia del componente Gestionar Contenido Económico.