Universidad de las Ciencias Informáticas

Facultad 2

"Desarrollo de una herramienta para la ayuda en la toma de decisiones de la vista arquitectónica de sistema."

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas.

Autores:

Franyudey Gonzalez Martinez

Omar Isalgué Begué

Tutores:

Ing. Larisa González Álvarez

Ing. Tahirí Rivero Alvarez

Ciudad de la Habana, Junio del 2010

Declaración de autoría

Declaro que somos los únicos autores de este trabajo y autorizamos al departamento de Producción del Centro de Informatización de la Gestión de Entidades de la Universidad de las Ciencias Informáticas; así como a dicho centro para que hagan el uso que estimen pertinente con este trabajo.

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

Franyudey Gonzalez Martinez

_____

Firma del Autor

Omar Isalgué Begué

_____

Firma del Autor

Larisa González Álvarez

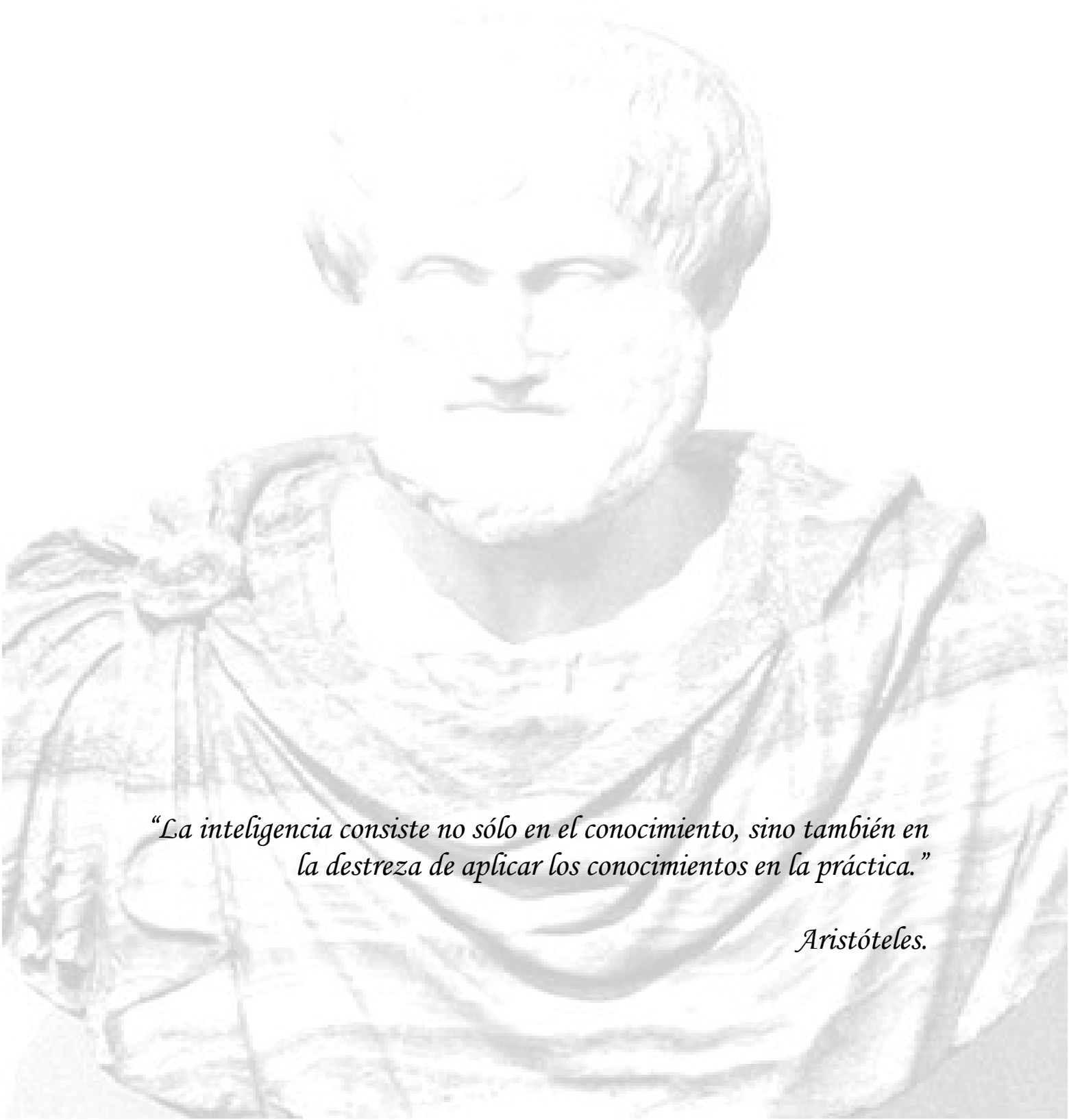
_____

Firma del Tutor

Tahirí Rivero Alvarez

_____

Firma del Tutor



"La inteligencia consiste no sólo en el conocimiento, sino también en la destreza de aplicar los conocimientos en la práctica."

Aristóteles.

Agradecimientos

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

Especialmente quiero agradecer a mi madre por ser la luz de mi vida, mi máxima inspiración, sin ella no hubiese sido posible llegar hasta aquí. A mi esposa por no haber dudado ni un segundo, haberme apoyado siempre y hacerme el hombre más feliz de este mundo. A mi niñito querido, la razón de mi ser, por regalarme esta satisfacción de ser padre y hacer que me sienta el hombre más afortunado de la tierra, eres lo que más quiero en la vida. A mi abuela por darme fuerzas con su amor y cariño. A mi tía Yuyú y a Zamara por haberse preocupado siempre por mí y no haber dudado ni un segundo en ayudarme. A mi padrastro por su apoyo incondicional. A Robertico, Yanerkis, Madelis, Daynel, Ernesto y Francito por brindarme su apoyo. A mi compañero de tesis y amigo Omar por haber confiado en mí para esta ardua tarea, por todas las horas de estudio y dedicación que pasamos juntos, por no haberse rendido ni un instante hasta no ver materializado este trabajo. A nuestras tutoras Larisa González Álvarez y Tahirí Rivero Alvarez, por toda la confianza y el apoyo transmitido, por exigirnos cumplir nuestra tarea a cabalidad y no dejarnos solos ni un instante. A todos mis compañeros de aula, de apartamento y del proyecto, por haberme apoyado en todo momento de la carrera. En fin a todos los que de una forma u otra han confiado en mí y me han animado en esta tarea tan difícil.

Franyudey

A mi familia por haberme dado todo su apoyo y haber confiado en mí, en especial a mis padres por tener fe en mí y haberme apoyado en estos largos 5 años para ver realizado el fruto de mi carrera. A todos mis amigos los buenos de verdad que viene compartiendo conmigo desde primer año, mis compañeros de aula y de cuarto. A mi colega de tesis por no darse por vencido ante las dificultades. Al equipo de desarrollo de la Línea de Auditoria. A mis tutoras por habernos guiado para realizar esta gran tarea. En fin a todas esas personas que en estos 5 años han sido de una forma u otra mi fuente de energía.

Omar

Dedicatoria

Dedico esta tesis a toda mi familia, especialmente a mi madre, mi esposa, mi niño, mi abuela Cary y a mi tía por brindarme su amor, su apoyo, sus consejos, su comprensión, su ejemplo y su confianza.
Los quiere mucho Franyudey.

A mi mamá, mi papá, mis hermanas, mi abuela y mis primos. En general a toda mi familia porque más cerca o más lejos todos estuvieron presente.
Los quiere Omar.

Resumen

Para mejorar el proceso de toma de decisiones, a lo largo de los años han surgido diversas herramientas informáticas en diferentes sectores de la sociedad que posibilitan la toma de decisiones.

En el proyecto de planificación de recursos empresariales (ERP-Cuba), la trasmisión de información se hace difícil, por lo que para gestionar la misma entre los subsistemas, módulos y componentes se necesita de una herramienta informática que permita ayudar en la toma de decisiones de la vista arquitectónica de sistema. Es por ello que se hace necesario crear una nueva herramienta, basada en software libre, ayudando en la gestión de la información y así permitir la toma de decisiones en la vista arquitectónica de sistema.

Para el desarrollo de la herramienta se utilizó como lenguaje de programación del lado del servidor PHP y del lado del cliente Java Script aplicándose ambos mediante el IDE de desarrollo Zend Studio para Eclipse. La especificación, construcción y documentación de la solución se realizó basándose en el Modelo de Desarrollo Orientado a Componentes (MDOC) haciéndose uso del Lenguaje Unificado de Modelado (UML) para el diseño, aplicándose a través de la herramienta de modelado Visual Paradigm para UML. Como resultado se obtuvo una solución capaz de gestionar la información de forma correcta para así ayudar en la toma de decisiones de la vista arquitectónica de sistema en el proyecto ERP-CUBA.

Palabras claves: Toma de decisiones, herramienta, subsistemas, vista arquitectónica de sistema.

Índice de contenido

Índice de figuras

Índice de tablas

Introducción

En la actualidad, con el desarrollo tecnológico alcanzado a nivel mundial, debido al incremento en el empleo de las tecnologías en casi todas las esferas de la sociedad y con el objetivo de elevar el nivel económico, muchas organizaciones, instituciones y empresas han puesto en práctica aplicaciones informáticas que le ayuden a tener un mejor control sobre los recursos que manejan.

Dentro de estas aplicaciones informáticas se encuentran los sistemas de planificación de recursos empresariales (ERP[1] por sus siglas en inglés). Un sistema ERP es un software de gestión que tiene como objetivo integrar y automatizar los procesos que se llevan a cabo en las empresas. Está compuesto por diferentes módulos, entre los que se encuentran Recursos Humanos, Ventas, Contabilidad y Finanzas, Compras, Producción; brindando información integrada de los procesos del negocio, por lo que deben ser altamente configurables para responder a las necesidades específicas de cada organización.

Es importante considerar que estas aplicaciones informáticas estén desarrolladas sobre una arquitectura sólida, ya que la misma no sólo dicta cómo debe construirse el sistema, sino que determina si la aplicación tiene los atributos de calidad esenciales para un proyecto exitoso.

La arquitectura de software AS[2] es una disciplina que tiene por objetivo definir la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre estos y el entorno, y los principios que orientan su diseño y evolución (Reynoso, 2004). El principal producto obtenido en esta disciplina es la especificación formal de la arquitectura a través de vistas que capturan las diferentes perspectivas del sistema que posteriormente servirán de base a cada uno de los implicados para, de manera coherente, conducir el desarrollo posterior del producto.

[1] Enterprise Resource Planning traducido al español como planificación de recursos empresariales: software que permite a las empresas controlar la información que se genera en cada departamento y cada nivel de la misma. (Badilla, 2010)

[2] Arquitectura de software

La AS constituye un punto clave en el proceso de desarrollo del software, por lo que además de buscar un diseño sólido es preciso valorar las decisiones tomadas durante el diseño. Esto brinda la posibilidad de evaluar el grado en que se han alcanzado los atributos de calidad que se persiguen y pone al descubierto tanto los puntos débiles como las potencialidades de la arquitectura propuesta.

Cuba se encuentra inmersa en el desarrollo de este tipo de aplicaciones informáticas, por lo que se le da la tarea a la Universidad de las Ciencias Informáticas (UCI), de desarrollar un sistema para la planificación de los recursos empresariales lo suficientemente flexible como para soportar la economía y sistema social vigente en el país. Para desarrollar este sistema se conformó un equipo compuesto por analistas, desarrolladores y arquitectos. Durante el desarrollo del mismo el equipo de arquitectura se enfrentó a una serie de dificultades en la comunicación entre los diferentes subsistemas, módulos y componentes, debido a la gran cantidad de información que se maneja, tales como:

- ✓ La gestión de la información referente a la integración de los subsistemas, módulos y componentes, se hace de forma manual, lo que convierte la solicitud de servicios en un proceso engorroso.
- ✓ La gestión de la información es personalizada, lo cual retarda la gestión, ya que involucra personas de diferentes áreas de trabajo.
- ✓ La información está centralizada pero no aporta los datos suficientes como para tomar decisiones.
- ✓ Las herramientas estudiadas no cuentan con todas las funcionalidades para la gestión de la información y así ayudar en la toma de decisiones.

Todas estas deficiencias traen como consecuencia baja eficiencia en la utilización del tiempo y los recursos humanos y poca integridad de la información.

La situación anteriormente expuesta conlleva al siguiente **problema a resolver**: la gestión actual de la información referente a la arquitectura de sistema que tributa a la toma de decisiones en el proyecto ERP-CUBA está afectando la integridad de los datos a analizar.

Con el objetivo de resolver dicho problema se propone como **objeto de estudio** Arquitectura de Software y como **campo de acción** Arquitectura de Sistema.

El **objetivo general** que se persigue con la realización de este trabajo es: desarrollar una herramienta que permita gestionar la información referente a la arquitectura de sistema, que tributa a la toma de decisiones en el proyecto ERP-CUBA, de manera

que se favorezca la integridad de los datos a analizar. En el mismo se han desglosado los siguientes **objetivos específicos:**

- Realizar el marco teórico de la investigación.
- Realizar el diseño de la herramienta.
- Desarrollar la solución propuesta.
- Realizar las pruebas de funcionamiento a la aplicación.

Tareas investigativas:

- Realizar un estudio de aplicaciones relacionadas a la gestión de información del flujo de arquitectura de sistemas.
- Sintetizar los diferentes elementos que contempla la arquitectura de sistema, para sentar las bases teóricas de la propuesta.
- Determinar las herramientas a emplear en el desarrollo del software.
- Confeccionar los diagramas de clases del diseño para definir las clases y sus relaciones.
- Diseñar la interfaz de usuario.
- Realizar el diseño de la base de datos.
- Implementar las funcionalidades del negocio del software centrado en los requisitos que debe cumplir el producto final.
- Implementar la capa de acceso a datos.
- Elaborar los casos de pruebas.
- Probar la aplicación.

Para guiar la investigación se planteó la siguiente **idea a defender:** si se logra desarrollar una herramienta que permita gestionar la información referente a la arquitectura de sistema en el proyecto ERP-CUBA de manera que tribute a la toma de decisiones, se logrará la integridad de los datos a analizar.

Con el propósito de desarrollar las tareas planteadas, se utilizaron los **métodos de investigación** siguientes:

Métodos teóricos:

- **Histórico-Lógico:** para conocer la esencia de la arquitectura de sistema y la metodología de desarrollo, sus trayectorias históricas y tendencias actuales.
- **Analítico-Sintético:** para desglosar la información obtenida del tema, obtener las características principales sintetizadas y arribar a conclusiones.
- **Sistémico:** para determinar los elementos arquitectónicos que componen el sistema y sus relaciones.

Métodos Empíricos:

- **Observación:** para determinar los elementos que van a definir y restringir el sistema.

El documento cuenta con la siguiente estructura:

Capítulo I. Fundamentación Teórica.

En este capítulo se analizan algunos conceptos, como arquitectura de software y arquitectura de sistema. También se estudian diferentes herramientas relacionados con el proceso de toma de decisiones, así como el modelo de desarrollo, las herramientas, tecnologías y lenguajes propuestos por el equipo de arquitectura del proyecto ERP-CUBA.

Capítulo II Desarrollo de la solución.

En este capítulo se ofrece un estudio de la solución propuesta, a partir de la obtención de los requisitos. Se detallan los diagramas de las principales clases que fueron definidas. También se muestra el modelo de datos y el diagrama de componentes para un mejor entendimiento de la solución propuesta.

Capítulo III. Evaluación de la solución.

En este capítulo se muestran los resultados obtenidos para validar la solución propuesta, mediante la aplicación de las métricas de calidad, que les fueron aplicadas a las clases y las pruebas de caja blanca, hechas a fragmentos de código.

Capítulo 1: Fundamentación Teórica.

Introducción

En este capítulo se realiza un análisis acerca de los principales aspectos de la arquitectura de software, destacando dentro de la misma la arquitectura de sistema. La toma de decisiones arquitectónicas, herramientas basadas en lenguajes de descripción arquitectónica (ADL[3] por sus siglas en inglés) para la ayuda en la toma de decisiones. Se lleva a cabo un estudio de las herramientas informáticas y tecnologías, proponiendo lenguajes y gestor de bases de datos a utilizar.

1.1 Arquitectura de software

El término de arquitectura de software, nace a partir de los sistemas de mediana y gran envergadura que proponen una solución a un problema específico. Dentro de las definiciones existentes se pueden citar algunas que son muy difundidas en todo el mundo. Una de ellas expuestas en el "Libro Ingeniería del Software, Un enfoque práctico", la cual plantea que: "la arquitectura de software de un sistema de programa o computación es la estructura de las estructuras del sistema, la cual comprende los componentes del software, las propiedades de esos componentes visibles externamente y las relaciones entre ellos." (Pressman., 2001)

La definición planteada anteriormente se refiere a la arquitectura como la estructura que tendrá un sistema de software. En lo adelante se entiende por arquitectura de software lo planteado por la IEEE Std2 1471-2000: "la arquitectura de software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución.". (Reynoso, 2004)

El objetivo primario de la arquitectura de software es el de aportar elementos que ayuden en la toma de decisiones significativas y al mismo tiempo, proporcionar conceptos y un lenguaje común que permitan la comunicación entre todos los que tienen parte en el proceso de desarrollo del software. Estas decisiones significativas se toman en base a:

3 Architectural description languages traducido al español como Lenguajes de descripción arquitectónica: es un lenguaje que provee características para modelar un sistema de software a nivel arquitectónico. (Medina, 2008)

- La organización del sistema de software.
- La selección de los elementos estructurales y sus interfaces a través de los cuales se constituye el sistema.
- Su comportamiento, según resultados de las colaboraciones entre esos elementos.
- El estilo arquitectónico que guía esta organización: los elementos estáticos y dinámicos; sus interfaces, su colaboración y su composición.

La arquitectura de software aporta una visión de los módulos, al mismo tiempo aporta elementos que ayudan a la toma de decisiones, proporcionando un lenguaje común entre los participantes de un proyecto.

1.1.1 Arquitectura de sistema

La arquitectura de sistema proporciona una visión global del sistema a construir. Describe la estructura y la organización de los componentes del software, sus propiedades y las conexiones entre ellos. Los componentes del software incluyen módulos de programas y varias representaciones de datos que son manipulados por el programa. (Pressman, 2002)

En lo adelante se entiende por arquitectura de sistema quien modela el sistema que se va a desarrollar. Además define de manera abstracta los componentes que van a llevar a cabo alguna tarea, sus interfaces y la comunicación entre ellos.

1.1.2 Estilos arquitectónicos

Los estilos arquitectónicos que se utilizaron fueron los mismos en los cuales está enmarcado el proyecto ERP-CUBA, ya sean la arquitectura en capas, la arquitectura basada en componentes y el estilo modelo Modelo-Vista- Controlador (MVC[4]).

El estilo arquitectónico constituye uno de los conceptos más importantes dentro de la arquitectura de software ya que describe y proporciona las propiedades básicas de una arquitectura. También se refiere a la organización y estructura de un sistema.

Según Pressman: “cada estilo describe una categoría del sistema que contiene: un conjunto de componentes por ejemplo, una base de datos, módulos computacionales que realizan una función requerida por el sistema; un conjunto de conectores que posibilitan la comunicación, la coordinación y la cooperación entre los componentes;

[4] Model View Controller traducido al español como modelo vista controlador

restricciones que definen como se pueden integrar los componentes que forman el sistema; y modelos semánticos que permiten al diseñador entender las propiedades globales de un sistema para analizar las propiedades conocidas de sus partes constituyentes". (Pressman., 2001)

Grupo de estilos arquitectónicos:

- **Estilo de llamada y retorno**: esta familia de estilos enfatiza la modificabilidad y la escalabilidad. Son los estilos más generalizados en sistemas de gran escala. Miembros de la familia son las arquitecturas de programa principal y subrutina, los sistemas basados en llamadas a procedimientos remotos, los sistemas orientados a objetos y los sistemas jerárquicos en capas.
    - ✓ Arquitectura en capas.

Las arquitecturas en capas constituyen uno de los estilos más conocidos y utilizados en la actualidad. Este estilo proporciona una organización jerárquica de tal manera que cada capa brinda servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior. Cada capa suele ser una entidad compleja, formada por un conjunto de paquetes o subsistemas.

Ventajas

Soporta un diseño basado en niveles de abstracción crecientes, lo cual permite la partición de un problema complejo en una secuencia de pasos incrementales. Proporciona amplia reutilización, ya que se pueden utilizar diferentes implementaciones o versiones de una misma capa en la medida que soporten las mismas interfaces de cara a las capas adyacentes. Mejora el soporte del sistema al admitir muy naturalmente optimizaciones y refinamientos.

Desventajas

Muchos problemas no admiten un buen mapeo en una estructura jerárquica. A veces es también extremadamente difícil encontrar el nivel de abstracción correcto. Además los cambios en las capas inferiores tienden a filtrarse hacia superiores. (Reynoso, 2004)

- ✓ Arquitectura basada en componentes.

La reutilización de componentes de software se ha convertido en una gran necesidad, de aquí que surja este estilo. Es uno de los más usados para la producción de software comercial. Por lo general, los componentes terminan siendo subsistemas que tienen una funcionalidad específica, generalmente aparecen en forma de librerías (DLL[5]), pero no quiere decir que esta sea la única forma de verlos. Existen muchas definiciones para el término de componentes de software. Según Krutchen, un componente es una parte no trivial, casi independiente y reemplazable de un sistema que cumple una función dentro del contexto de una arquitectura bien definida. Un componente cumple con un conjunto de interfaces y provee la realización física de ellas.

Ventajas

- Reutilización del software: se llega a alcanzar un mayor nivel de reutilización de software.
- Simplifica las pruebas: permite que las pruebas sean ejecutadas probando cada uno de los componentes antes de probar el conjunto completo de componentes ensamblados.
- Simplifica el mantenimiento del sistema: cuando existe un débil acoplamiento entre componentes, el desarrollador es libre de actualizar y/o agregar componentes según sea necesario, sin afectar otras partes del sistema. (Reynoso, 2004)
- Mayor calidad: dado que un componente puede ser construido y luego mejorado continuamente por un experto u organización, la calidad de una aplicación basada en componentes mejorará con el paso del tiempo. (Casal, 2009)

Desventajas

- Pobre comprensibilidad: Puede ser difícil prever qué pasará en respuesta a una acción.

✓ Modelo Vista Controlador

MVC cuenta con tres partes y contiene:

- Un modelo que se encarga de gestionar los datos.
- Una vista que gestiona como se muestran esos datos.
- Un controlador que determina que modificaciones hay que hacer cuando se interacciona con el elemento.

[5] Dynamic Link Library traducido al español como biblioteca de enlace dinámico

El estilo arquitectónico modelo-vista-controlador (MVC) fue utilizado en la solución de la siguiente forma:

En las clases del modelo representadas en los diagramas de clases se representa la lógica de negocio. La vista transforma el modelo en una página Web que permite al usuario interactuar con ella. La controladora se encarga de procesar las interacciones del usuario y realiza los cambios apropiados en el modelo o en la vista y arquitectura basada en componentes por las ventajas que brinda el mismo de integración y desarrollo vertical de los componentes ya definidos.

1.1.3 Patrones de diseño

El uso de patrones en las tareas del diseño es una actividad que aporta beneficios a los equipos de desarrollo, ya que les facilita el trabajo porque son propuestas de soluciones a problemas de diseño no trivial que son efectivas y reusables.

Patrones Grasp[6]

- **Experto:** Consiste en asignar una responsabilidad al experto en información, es decir, la clase que tiene la información necesaria para cumplir con la responsabilidad (Larman, 2004), en este caso la clase Controladora le asigna la responsabilidad a la clase Modelo de la lógica de la aplicación.

- **Controlador:** Sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa (Larman, 2004), la clase Controladora es la encargada de llevar el control y seguimiento de la lógica del negocio ya que recibe los datos del usuario y los envía a las distintas clases según el método llamado.

Patrones de comportamiento

Los patrones de comportamiento estudian las relaciones entre llamadas sobre los diferentes objetos, normalmente ligados con la dimensión temporal.

Debido a sus funciones se destacan para la solución los siguientes patrones de comportamiento:

[6] **General** Responsability Assignment Software Patterns traducido al español como Patrones de Software para la asignación General de Responsabilidad

Cadena de responsabilidad

Permite la separación entre objetos, pasando la tarea de un objeto al siguiente dentro de una cadena hasta que la tarea es reconocida. (Kramek, 2007)

Estrategia

Encapsula un algoritmo dentro de una clase. (Kramek, 2007)

1.2 La toma de decisiones

Una decisión es una elección consciente y racional, orientada a conseguir un objetivo, que se realiza entre diversas posibilidades de actuación (o alternativas). Antes de tomar una decisión se debe calcular cuál será el resultado de escoger una alternativa. En función de las consecuencias previsibles para cada alternativa se tomará la decisión. Así, los elementos que constituyen la estructura de la decisión son: los objetivos de quien decide y las restricciones para conseguirlos; las alternativas posibles y potenciales; las consecuencias de cada alternativa; el escenario en el que se toma la decisión y las preferencias de quien decide. (Ferreira, 2005)

En lo adelante se entiende por toma de decisiones, el proceso mediante el cual se realiza una elección entre las alternativas o formas para resolver diferentes situaciones de la vida. Consiste, básicamente, en elegir una alternativa entre las disponibles, a los efectos de resolver un problema actual o potencial.

Importancia de la toma de decisiones

La toma de decisiones es de gran importancia ya que a través de la aplicación de un buen procedimiento, o modelo de toma de decisiones, permite el ahorro de tiempo, esfuerzo y energía. Además es importante porque el empleo de una buena decisión, indica que un problema o situación es valorado y considerado profundamente para elegir el camino a seguir según las diferentes alternativas y operaciones.

En el proceso de desarrollo de software la toma de decisiones es importante, ya que las mismas deben cumplir con ciertas características como son: ser rápida, oportuna, fundamentada en información concreta, que permita tomar decisiones eficientes, efectivas y con un bajo costo para el equipo de desarrollo; pues de ello dependerá el éxito o fracaso de una organización. Es aquí donde surge la necesidad del soporte de

sistemas como una herramienta para la toma de decisiones acorde a los objetivos estratégicos de cada organización.

1.2.1 Herramientas basadas en ADL para la toma de decisiones

Las herramientas de apoyo a las decisiones permiten obtener la información requerida durante el proceso de la toma de decisiones. Esto implica que uno de los objetivos de una herramienta de apoyo a las decisiones sea proporcionar cierta cantidad de información, con el fin de decidir lo más adecuado.

A continuación se presentarán algunas herramientas informáticas basadas en ADL que ayudan en la toma de decisiones.

ADL es un lenguaje descriptivo de modelado que se focaliza en la estructura de alto nivel de la aplicación antes que en los detalles de implementación de sus módulos concretos. Este debe proporcionar un modelo explícito de componentes, conectores y sus respectivas configuraciones, además, suministrar soporte de herramientas para el desarrollo de soluciones basadas en arquitectura. (Reynoso, 2004)

Tabla 1 Significado de los indicadores seleccionados para la comparación cualitativa.

Indicadores	**Significado**
Integridad de comunicación	El ADL debe dar la posibilidad de que todos sus componentes se relacionen entre sí.
Varios tipos de componentes	El ADL debe dar la posibilidad de crear más de un tipo de componente.
Dinámica	El ADL debe dar la posibilidad de modelar arquitecturas cambiantes en el tiempo.
Tributa a la toma de decisiones.	El ADL debe dar la posibilidad de gestionar información para así poder tomar decisiones.

Tabla 2 Comparación cualitativa entre herramientas basadas en ADL.

	Herramientas(ADL)			
Indicadores	Aesop	Darwin	Wright	Acme
Integridad de comunicación	Si	Si	Si	Si
Varios tipos de componentes	Si	Si	Si	Si
Dinámica	No	Si	No	No
Tributa a la toma de decisiones	No	No	No	No

Las herramientas analizadas anteriormente sin lugar a duda son muy importantes por sus funcionalidades, pero no son factibles para la subdirección de arquitectura del proyecto ERP-Cuba. Las mismas se centran en los elementos más relevantes de la arquitectura de software tales como componentes, conectores, restricciones y sub estructuras y no permiten una gestión de información para así poder tomar decisiones. Es por ello que se necesita un producto que se adapte a las condiciones actuales de la vista arquitectónica de sistema del proyecto ERP-Cuba.

1.3 Modelo de desarrollo, tecnologías y herramientas

En la actualidad, con el avance que ha alcanzado la informática, han surgido diversidad de herramientas informáticas y tecnologías, estas ayudan en el desarrollo de aplicaciones informáticas. Dentro de estas aplicaciones se incluyen las Web, las cuales ofrecen grandes facilidades para establecer comunicación con el usuario por su capacidad para ser visualizadas desde cualquier parte del mundo haciendo uso de un navegador. Por ello a continuación se exponen el modelo de desarrollo, las tecnologías y herramientas informáticas que por sus características fueron definidas en el marco del proyecto ERP-Cuba y aplicadas al sistema Cedrux y por consiguiente al desarrollo de la solución propuesta por formar parte de dicho proyecto.

1.3.1 Modelo de desarrollo

El modelo de desarrollo orientado a componentes (MDOC) definido por el proyecto ERP-Cuba, para la realización del sistema Cedrux es un modelo de desarrollo orientado a las necesidades y artefactos generados durante el proceso de desarrollo de CEDRUX. Es una combinación de diferentes metodologías de las cuales se ha tomado lo que sería más conveniente para llevar a término el proyecto. Entre las

características que posee se encuentran que se modela el negocio mediante procesos; la ingeniería de requisitos es mucho más clara que en otras metodologías; es orientada a componentes, posibilitando la independencia de funciones del sistema a la hora de mantener o modificar el sistema funcional. Este modelo de desarrollo permitirá la generación de artefactos de vital importancia en el diseño e implementación como son: Modelo conceptual, Especificación de requisitos y los Diagramas de clases, Modelo de datos y Diagrama de componentes.

1.3.2 Herramientas CASE[7] para el modelado UML[8]

Las Herramientas CASE pueden definirse como un conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del ciclo de vida de desarrollo de un software. La misión de estas herramientas que utilizan UML como notación para elaborar los modelos, es comunicar, de la manera más eficiente posible a todos los agentes de un proyecto, todas aquellas decisiones que se toman con respecto a la arquitectura del sistema en discusión y que son determinantes para cumplir con los objetivos del proyecto. (Marzo, 2004)

Visual Paradigm

Es una herramienta CASE que utiliza "UML": como lenguaje de modelaje. La herramienta está desarrollada por Visual Paradigm Internacional una de las principales compañías de herramientas CASE donde su mayor éxito consiste en el uso libre del producto mencionado.

Características

Visual Paradigm utiliza UML como lenguaje de modelado para la construcción de los sistemas software ofreciendo soluciones de software que permiten a las organizaciones desarrollar las aplicaciones de calidad dentro de sus características están:

- Tiene la capacidad de ejecutarse sobre diferentes sistemas operativos lo que le confiere la característica de ser multiplataforma.
- Integra diferentes funcionalidades para el desarrollo de aplicaciones como el modelado de UML, el modelado de base de datos, el modelado de

[7] Computer Aided Software Engineering traducido al español como Ingeniería de Software Asistida por Ordenador
[8] Unified Modeling Language traducido al español como Lenguaje Unificado de Modelado

requerimientos, el modelado del proceso de negocio, la interoperabilidad, la generación de documentación entre otros.

- Además presenta una potente integración con el lenguaje Java, mediante su herramienta.
- Permite realizar reingeniería inversa de los datos.
- La herramienta Visual Paradigm se convierte en la más conveniente para utilizarla en el desarrollo de sistemas por ser multiplataforma, por exportar código alrededor de diez lenguajes incluyendo PHP. (Sierra, 2005)

1.3.3 Tecnología del lado del servidor

Consiste en un conjunto de símbolos, reglas sintácticas y semánticas que definen su estructura, el significado de sus elementos y expresiones. Un lenguaje de programación permite a uno o más programadores especificar de manera precisa: sobre qué datos una computadora debe operar, cómo deben ser estos almacenados y transmitidos y qué acciones debe tomar bajo una variada gama de circunstancias. (Ortiz, 2007).

La programación del lado del servidor es un elemento agregado muy importante en el diseño y construcción de los sitios Web, ya que permite de una u otra forma el manejo de datos dinámicamente. Los lenguajes del lado del servidor son aquellos lenguajes que son reconocidos, ejecutados e interpretados por el propio servidor y que se envían al cliente en un formato compresible para él.

PHP

PHP es un lenguaje de script interpretado en el lado del servidor utilizado para la generación de páginas Web dinámicas, embebido en páginas de lenguaje de marcas de hipertexto (HTML[9] por sus siglas en inglés Hyper Text Mark-up Language) y ejecutado en el servidor.

La mayor parte de su sintaxis ha sido tomada de C, Java y Perl con algunas características específicas de sí mismo. La meta del lenguaje es permitir rápidamente a los desarrolladores la generación dinámica de páginas. No es un lenguaje de marcas como podría ser HTML. Está más cercano a Java Script o a C. (Hinostroza, 2007).

Ventajas de PHP

- Muy fácil de aprender.

[9] Hyper Text Mark-up Language traducido al español como lenguaje de marcas de hipertexto.

- Se caracteriza por ser un lenguaje muy rápido.
- Es un lenguaje multiplataforma: Linux, Windows, entre otros.
- Capacidad de conexión con la mayoría de los manejadores de base de datos: MysSQL, PostgreSQL, Oracle, MS SQL Server, entre otros.
- Capacidad de expandir su potencial utilizando módulos.
- Posee documentación en su página oficial la cual incluye descripción y ejemplos de cada una de sus funciones.
- Es libre, por lo que se presenta como una alternativa de fácil acceso para todos.
- Incluye gran cantidad de funciones.
- No requiere definición de tipos de variables ni manejo detallado del bajo nivel. (Valdés, 2007)

1.3.4 Tecnologías del lado del cliente

La parte de las aplicaciones web está formada por el código HTML que forma la página, con opciones a códigos ejecutables mediantes los lenguajes de scripting de los navegadores (JavaScript) o mediante pequeños programas (applets) en Java. Los lenguajes scripts son lenguajes interpretados, no compilados; lo que quiere decir que el programa que lo interpreta, el navegador en este caso, va leyendo el código y ejecutándolo al mismo tiempo en vez de transformarlo a código de máquina, esto trae consigo errores en tiempo de ejecución.

Unas de las principales ventajas de un lenguaje interpretado es que es independiente de la máquina y del sistema operativo ya que no contiene instrucciones propias de un procesador sino llamadas a funciones que el intérprete deberá reconocer.

HTML

El HTML es un lenguaje de marcación de elementos para la creación de documentos hipertexto, muy fácil de aprender, lo que permite que cualquier persona, aunque no haya programado en la vida, pueda enfrentarse a la tarea de crear una página Web. HTML es fácil y se puede dominar el lenguaje con rapidez.

Este lenguaje se escribe en un documento de texto, por eso se necesita un editor de textos para escribir una página Web. Así pues, el archivo donde está contenido el código HTML es un archivo de texto, con una peculiaridad, que tiene extensión .html o .htm (es indiferente cuál utilizar). De modo que cuando se programe en HTML se

realizará con un editor de textos, lo más sencillo posible y se guardarán los trabajos con extensión .html. (Alvarez, 2003)

JavaScript

Se trata de un lenguaje de la programación del lado del cliente, porque es el navegador el que soporta la carga de procesamiento. Gracias a su compatibilidad con la mayoría de los navegadores modernos, es el lenguaje del lado del cliente más utilizado.

Es un lenguaje de programación interpretado por el navegador que se utiliza para controlar su apariencia y manipular los eventos que ocurran en su ventana. Mediante JavaScript se pueden conseguir interesantes efectos en las páginas web, validar formularios, abrir y cerrar ventanas, cambiar dinámicamente el aspecto y los contenidos de una página, realizar cálculos matemáticos sencillos.

Con este se puede crear efectos especiales en las páginas y definir interactividades con el usuario. El navegador del cliente es el encargado de interpretar las instrucciones JavaScript y ejecutarlas para realizar estos efectos e interactividades, de modo que el mayor recurso, y tal vez el único, con que cuenta este lenguaje es el propio navegador. (Alvarez, 2003)

Es un lenguaje de programación bastante sencillo y pensado para hacer las cosas con rapidez, a veces con ligereza. Incluso las personas que no tengan una experiencia previa en la programación podrán aprender este lenguaje con facilidad y utilizarlo en todo su potencia con sólo un poco de práctica.

Entre las acciones típicas que se pueden realizar con este lenguaje se tienen dos vertientes. Por un lado, efectos especiales sobre páginas web, para crear contenidos dinámicos y elementos de la página que tengan movimiento, cambien de color o cualquier otro dinamismo, por el otro, JavaScript permite ejecutar instrucciones como respuesta a las acciones del usuario, con lo que se puede crear páginas interactivas con programas como calculadoras, agendas o tablas de cálculo. (Alvarez, 2003)

Es un lenguaje con muchas posibilidades, permite la programación de pequeños scripts, pero también de programas más grandes, orientados a objetos, con funciones, estructura de datos complejos. Además pone a disposición del programador todos los elementos que forman la página web, para que éste pueda acceder a ellos y modificarlos dinámicamente. (Alvarez, 2003)

1.3.5 Gestor de bases de datos

Un sistema gestor de base de datos (SGBD) es un conjunto de programas que permiten crear y mantener una base de datos, asegurando su integridad, confidencialidad y seguridad. Por tanto debe permitir:

- Definir una base de datos: especificar tipos, estructuras y restricciones de datos.
- Construir la base de datos: guardar los datos en algún medio controlado por el mismo SGBD.
- Manipular la base de datos: realizar consultas, actualizarla, generar informes.

PostgreSQL

PostgreSQL es un sistema de gestión de bases de datos objeto-relacional (ORDBMS), basados en el proyecto Postgres, de la Universidad de Berkeley; es una derivación libre de este proyecto. Es uno de los primeros en muchos conceptos en el sistema de objeto-relacional actual, incluido más tarde en otros sistema de gestión comercial, el mismo incluye características de la orientación a objetos, como pueden ser la herencia, tipos de datos funciones, restricciones, disparadores, reglas e integridad relacional.

Sus principales características como gestor de bases de datos son:

- Implementación del estándar SQL92/SQL99.
- Soporta distintos tipos de datos, además del soporte para los tipos bases; también soporta datos de tipo fecha, monetarios, elementos gráficos, datos sobre redes, cadenas de bits. También permite la creación de tipos propios.
- Incorpora una estructura de datos (arreglos).
- Incorpora funciones de diversa índole: manejo de fechas, geométricas, orientadas a operaciones como redes.
- Permite la declaración de funciones propias, así como la definición de disparadores.
- Soporta el uso de índices, reglas y vistas.
- Incluye la herencia entre tablas (aunque no entre objetos, ya que no existen), por lo que este gestor de bases de datos se le incluye entre los gestores objeto-relacionales.

- Permite la gestión de diferentes usuarios, como también los permisos asignados a cada uno de ellos.
- Tiene la capacidad de comprobar la integridad referencial, así como también la de almacenar procedimientos en la propia bases de datos, equiparando con los gestores de bases de datos de alto nivel, como son Oracle.
- Posee una gran escalabilidad. Es capaz de ajustarse al número de CPUs[10] y a la cantidad de memoria que posee el sistema de forma óptima, haciéndole capaz de soportar una mayor cantidad de peticiones simultáneas de manera correcta.
- Implementa el uso del mecanismo de vuelta atrás para subconsultas y transacciones, haciendo su funcionamiento mucho más eficaz, y ofreciendo soluciones en campos en las que MySQL no podría. (Pecos, 2003)

1.3.6 Herramienta de desarrollo

Zend Studio para eclipse

Zend Studio es una aplicación utilizada por los impulsores del lenguaje PHP, es una herramienta orientada a desarrollar aplicaciones web, como no, en lenguaje PHP. El programa, además de servir de editor de texto para páginas PHP, cuenta con una ayuda que le facilita al usuario gran información para la creación de proyectos.

El programa está escrito en Java, esto ha permitido lanzar con relativa facilidad y rapidez versiones del producto para Windows, Linux y MacOS, aunque el desarrollo de las versiones de este último sistema se retrase un poco más.

Entre las facilidades más novedosas que brinda se encuentra el editor, la gestión de proyecto y sin dejar de mencionar su herramienta de depuración.

Lo más destacable del editor de Zend Studio es que contiene una ayuda contextual con todas las librerías de funciones del lenguaje que asiste en todo momento ofreciendo nombres de las funciones y parámetros que deben recibir. Aunque esta ayuda contextual no sólo se queda en las funciones definidas en el lenguaje, sino que también reporta ayudas con las funciones que se vallan creando, incluso en páginas que se tengan incluidas con la función include(). (Alvarez, 2003)

[10] Central Processing Unit traducido al español como unidad de proceso central

La gestión de proyecto mejora la productividad en la programación. Los proyectos permiten guardar mucha más información al programa sobre los archivos, discos, servidores que se gestionan en las aplicaciones PHP.

Herramienta de depuración, se puede ejecutar páginas y conocer en todo momento el contenido de las variables de la aplicación, colocar puntos de parada de los scripts y realizar las acciones típicas de depuración. (Alvarez, 2003)

Características:

- No requiere la instalación previa de PHP ni del entorno de ejecución de Java.
- Soporte para PHP 4 y PHP 5.
- Resaltado de sintaxis, autocompletado de código, ayuda de código y lista de parámetros de funciones y métodos de clase.
- Plegado de código (comentarios, bloques de phpDoc, cuerpo de funciones y métodos e implementación de clases).
- Inserción automática de paréntesis y corchetes de cierre.
- Sangrado automático y otras ayudas de formato de código.
- Emparejamiento (matching) de paréntesis y corchetes (si se sitúa el cursor sobre un paréntesis (corchete) de apertura (cierre), Zend Studio localiza el correspondiente paréntesis (corchete) de cierre (apertura)).
- Detección de errores de sintaxis en tiempo real.
- Funciones de depuración: botón de ejecución y traza, marcadores, puntos de parada (breakpoints), seguimiento de variables y mensajes de error del intérprete de PHP. Permite también la depuración en servidores remotos (requiere Zend Platform).
- Instalación de barras de herramientas para Internet Explorer y Mozilla Firefox (opcional).
- Soporte para gestión de grandes proyectos de desarrollo.
- Manual de PHP integrado.
- Soporte para control de versiones usando Subversion (a elección del desarrollador).
- Cliente FTP integrado.
- Soporte para navegación en bases de datos y ejecución de consultas SQL. (Almada, 2008)

1.4 Conclusiones

En el presente capítulo se trataron diferentes conceptos que son de vital importancia para dar solución al problema planteado como son arquitectura de software y arquitectura de sistema. Además se realizó un estudio detallado de los principales estilos arquitectónicos y patrones que permitan lograr un diseño robusto y escalable, pero a la vez que se ajuste a las características del sistema. Además se estudiaron otros aspectos como el modelo de desarrollo, lenguajes y otras herramientas que complementan el desarrollo del sistema.

Capítulo 2: Desarrollo de la Solución.

Introducción

En este capítulo se lleva a cabo el análisis de la solución propuesta. Se realiza un levantamiento de los requisitos funcionales. Se muestran los tipos de relaciones que existen entre las clases, mediante el diseño de clases. Se presentan además los artefactos de implementación que se obtienen a través de la construcción de los componentes que integran el sistema.

2.1 Levantamiento de requisitos

Para el desarrollo de la aplicación se definieron con los clientes las actividades fundamentales con las que debe contar el software para satisfacer las necesidades de los usuarios finales. Uno de los aspectos más importantes para el éxito de un proyecto de software lo constituye el levantamiento correcto de los requisitos funcionales, lo que constituye el entendimiento y comprensión de los problemas que se necesitan solucionar y cómo resolverlos.

Este entendimiento viene forzado por la cantidad y la calidad de información suministrada por la persona que tenga ese problema. Información en la que debería especificarse de forma clara e inequívoca la realidad: un producto nuevo, una actualización de un producto antiguo, o la agregación de una nueva funcionalidad a un producto ya en funcionamiento. Todo ello expuesto en unos requisitos que en el propio lenguaje del propietario del sistema, actual o futuro, expresa su idea o visión de lo que necesita; permite lograr una comunicación efectiva entre los usuarios y el equipo de proyecto con el objetivo de llegar a un entendimiento de lo que hay que realizar, siendo así la clave del éxito en la producción de un software.

Requisitos funcionales: según RUP son capacidades o condiciones que el sistema debe cumplir.

2.1.1 Requisitos funcionales

RF[11] 1 Cargar XML

ARF[12] 1 Gestionar subsistema.

RF 2 Adicionar subsistema.

RF 3 Modificar subsistema.

RF 4 Eliminar subsistema.

RF 5 Buscar subsistema.

RF 6 Listar subsistema.

ARF 2 Gestionar servicio a un subsistema

RF 7 Adicionar servicio a un subsistema.

RF 8 Modificar servicio a un subsistema.

RF 9 Eliminar servicio a un subsistema.

RF 10 Mostrar descripción de servicio de un subsistema.

RF 11 Adicionar parámetro a un servicio de un subsistema.

RF 12 Modificar parámetro a un servicio de un subsistema.

RF 13 Eliminar parámetro a un servicio de un subsistema.

RF 14 Mostrar servicio de un subsistema.

Rf 15 Buscar servicio de un subsistema.

Rf 16 Listar servicio de un subsistema.

RF 17 Calcular criticidad en un subsistema.

RF 18 Calcular complejidad en un subsistema.

RF 19 Obtener estadística de mayor criticidad de un subsistema.

RF 20 Obtener estadística de menor criticidad de un subsistema.

RF 21 Obtener estadística de mayor complejidad de un subsistema.

RF 22 Obtener estadística de menor complejidad de un subsistema.

RF 23 Ordenar por estadística de criticidad de subsistema.

RF 24 Ordenar por estadística de complejidad de subsistema.

RF 25 Obtener reporte de estadística de criticidad de un subsistema.

RF 26 Obtener reporte de estadística de complejidad de un subsistema.

ARF 3 Gestionar componente.

RF 27 Adicionar componente.

RF 28 Modificar componente.

RF 29 Eliminar componente.

[11] Requisito funcional

[12] Agrupamiento de requisitos funcionales

RF 30 Buscar componente.

RF 31 Listar componente.

ARF 4 Gestionar servicio de un componente

RF 32 Adicionar servicio a un componente.

RF 33 Modificar servicio a un componente.

RF 34 Eliminar servicio a un componente.

RF 35 Mostrar servicio de un componente.

RF 36 Adicionar parámetro a un servicio de un componente.

RF 37 Modificar parámetro a un servicio de un componente.

RF 38 Eliminar parámetro a un servicio de un componente.

RF 39 Mostrar descripción de servicio de un componente.

RF 40 Buscar servicio de un componente.

RF 41 Listar servicio de un componente.

RF 42 Calcular criticidad de un componente.

RF 43 Calcular complejidad de un componente.

RF 44 Obtener estadística de mayor criticidad de un componente.

RF 45 Obtener estadística de menor criticidad de un componente.

RF 46 Obtener estadística de mayor complejidad de un componente.

RF 47 Obtener estadística de menor complejidad de un componente.

RF 48 Ordenar por estadística de criticidad de un componente.

RF 49 Ordenar por estadística de complejidad de un componente.

RF 50 Obtener reporte de estadística de criticidad de un componente.

RF 51 Obtener reporte de estadística de complejidad de un componente.

RF 52 Imprimir información.

RF 53 Exportar como XML.

2.2 Modelo conceptual

El modelo conceptual es una representación de conceptos en un dominio del problema. Además este modelo muestra asociaciones entre conceptos y atributos de conceptos. Se puede ver como un modelo que comunica los términos importantes y cómo se relacionan entre sí.

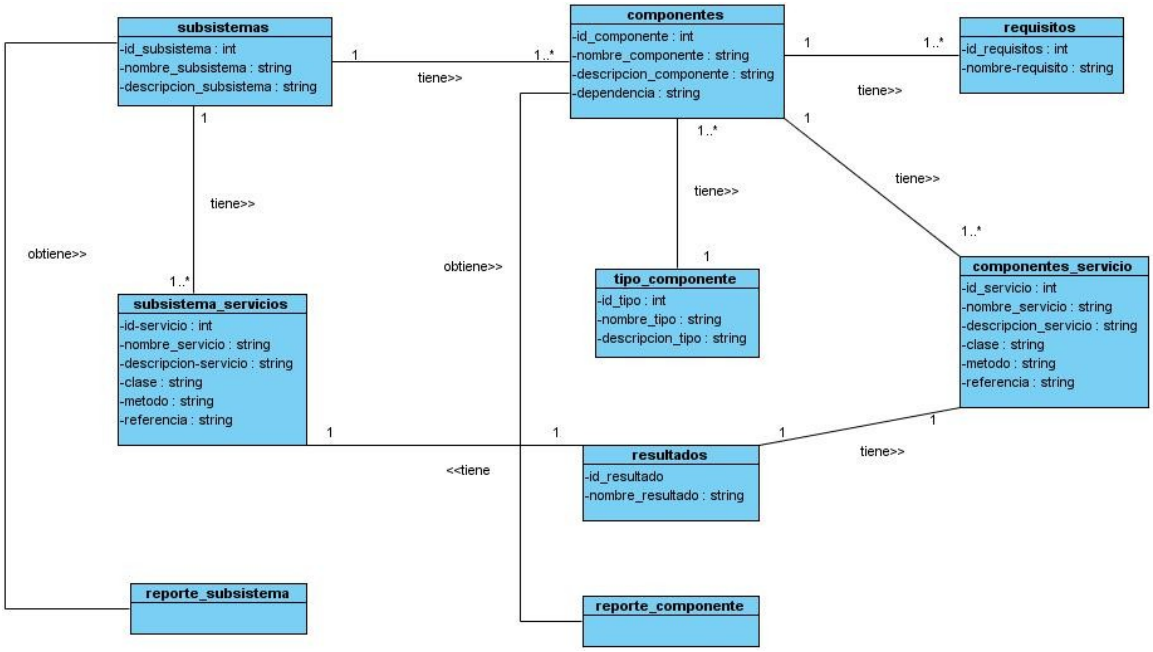


Figura 1 Modelo conceptual

2.3 Diseño

El modelo de diseño es una abstracción del modelo de implementación y su código fuente, el cual fundamentalmente se emplea para representar y documentar su diseño. Es usado como entrada esencial en las actividades relacionadas a implementación.

El modelo de diseño puede contener: los diagramas, las clases, paquetes, subsistemas, cápsulas, protocolos, interfaces, relaciones, colaboraciones, atributos, las realizaciones de los casos de uso, entre otros que se puedan considerar para el sistema en desarrollo.

Los estereotipos que se usan son:

<<Form>>

Figura 2 Formulario.

Grupo de elementos de entrada que son parte de una página cliente. Se relaciona directamente con la etiqueta de igual nombre del HTML.

<<Client page>>

Figura 3 Página cliente.

Una instancia de página cliente es una página Web, con formato HTML; mezcla de datos, presentación y lógica. Son interpretadas por el browser. Cada página cliente solo puede ser construida por una página servidor.

2.31Aplicación de los patrones del diseño

El patrón de comportamiento cadena de responsabilidad se reflejó en el desarrollo de la aplicación mediante el tratamiento de errores puesto que a través del lanzamiento de excepciones de las clases y las capturas de las mismas por el aspecto gestor global de excepciones (ZendExtexception) junto al identificador de las mismas (exception), se manifiesta que el manejador específico que satisface la petición, no se conoce “a priori” y se quiere trasladar dicha petición a un objeto de entre unos cuantos, sin especificar explícitamente el receptor final.

El uso del patrón de comportamiento estrategia en el desarrollo de la herramienta se evidencia en que las clases definen numerosos comportamientos, los cuales se manifiestan como definiciones condicionales múltiples de sus operaciones, además de que los algoritmos usan datos que los clientes no tienen por qué conocer.

2.3.2 Diseño de clases

Aquí se realizaron los diagramas de clases del diseño, ya que son los que describen gráficamente las especificaciones de las clases de software y de las interfaces en una aplicación, contienen información como: clases, asociaciones y atributos, interfaces con sus operaciones y constantes, métodos, navegabilidad, dependencias.

En los diagramas de clases a continuación se pueden observar cómo quedan definidas las clases con todas sus funcionalidades y cómo se relacionan entre ellas. Las clases cuyos nombres terminan en Controller se refieren a las clases controladoras que son las que contienen los atributos y los métodos necesarios y se encargan de seleccionar qué clase modelo (las que el nombre termina en Model) es la adecuada para realizar la acción encomendada por el usuario, o sea insertar, actualizar (modificar) o eliminar.

En la Figura 4 se muestra el diagrama de clases de diseño para el requisito Gestionar servicio subsistema. En él se representa el controlador ServicioSubsistema, el cuál es el encargado de construir la página cliente: serviciosubsistema.phtml mediante la relación <<build>>. Esta página cliente contiene al formulario FR Servicio Subsistema, el cual toma los datos que el usuario ingrese en la página y los envía al controlador ServicioSubsistema para su posterior procesamiento.

El controlador ServicioSubsistema tiene una relación <<instantiate>> con la clase del modelo ServicioSubsistemaModel, pues esta última tiene la responsabilidad de realizar las operaciones con cierta complejidad sobre la información de las tablas de la base de datos y enviar el resultado a la controladora.

DC[13] Servicio subsistema

[13] Class diagram traducido al español como diagrama de clases.

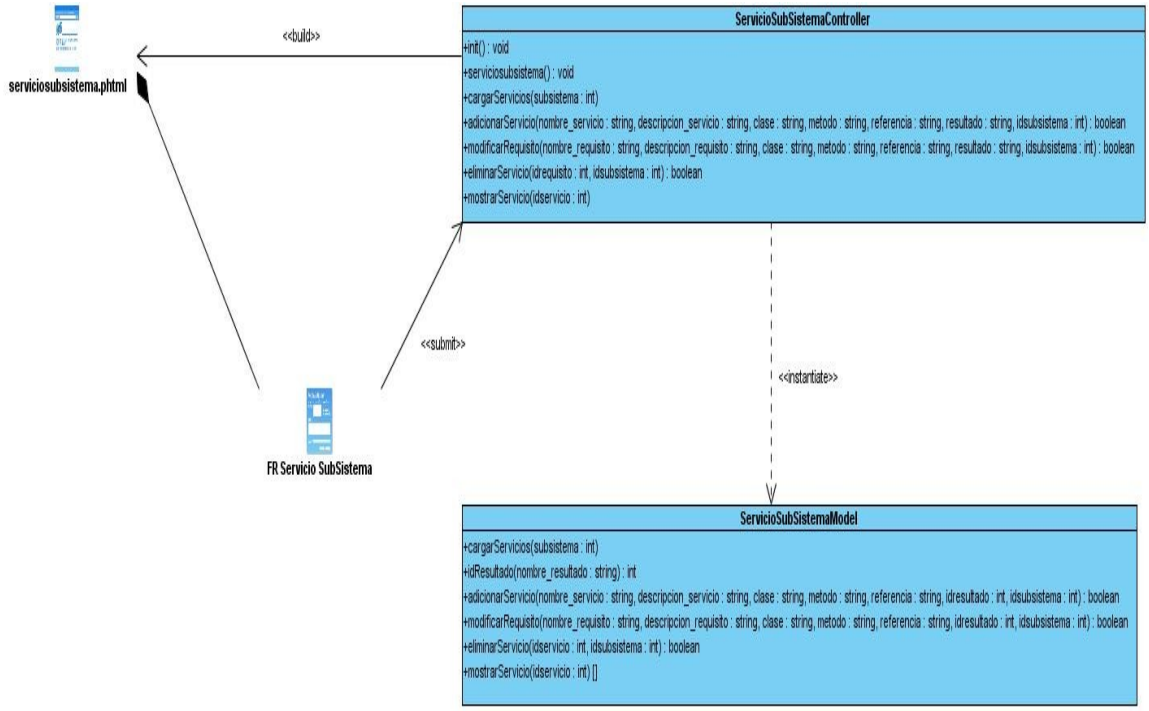


Figura 4 Diagrama de clases Servicio subsistema.

DC Gestionar requisito

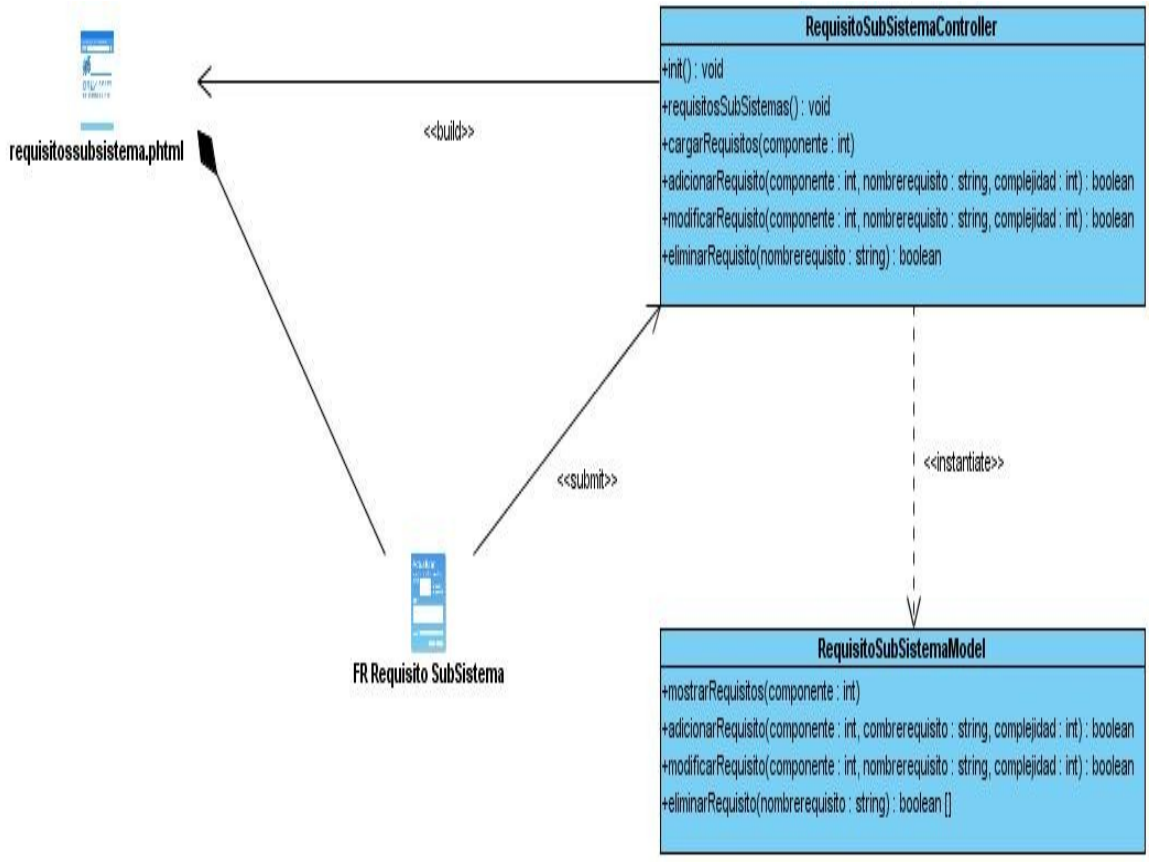


Figura 5 Diagrama de clases Gestionar requisito.

DC Servicio componente

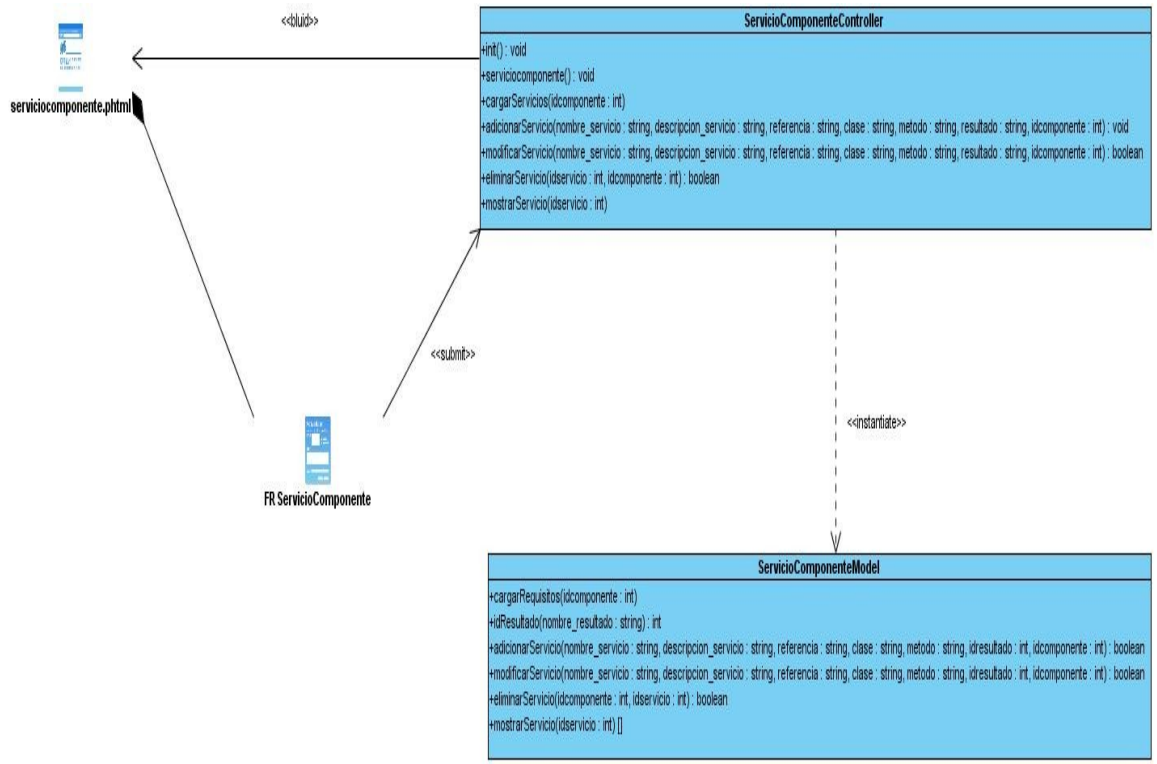


Figura 6 Diagrama de clases Servicio componente.

DC Subsistema

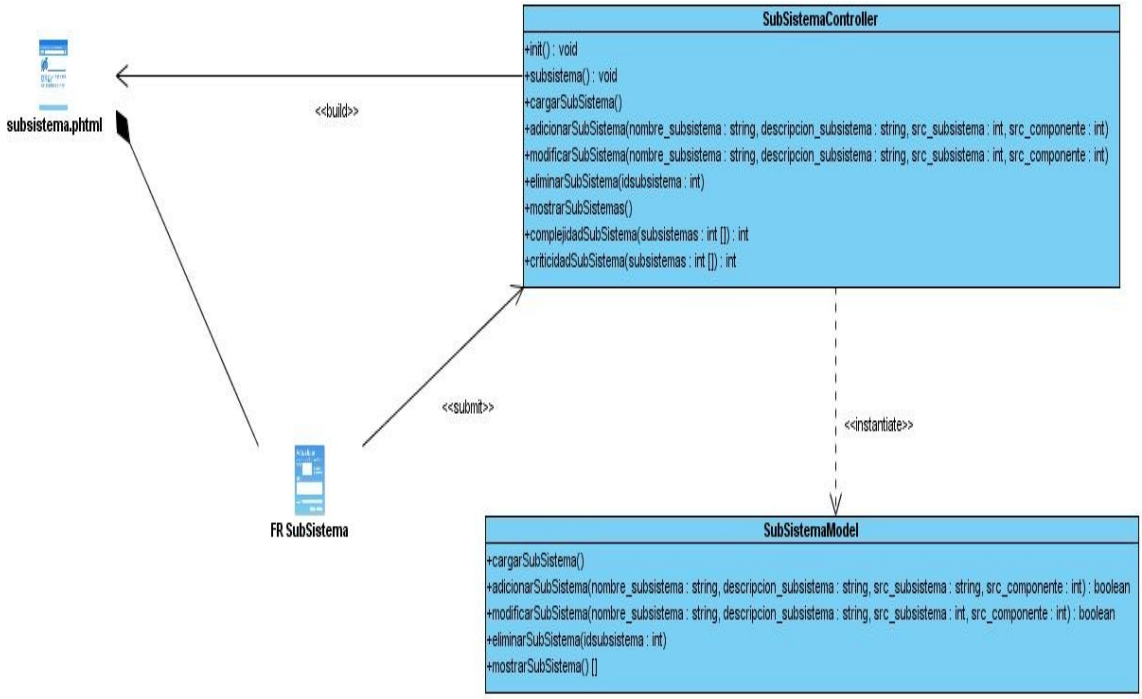


Figura 7 Diagrama de clases Subsistema.

DC Tipo componente

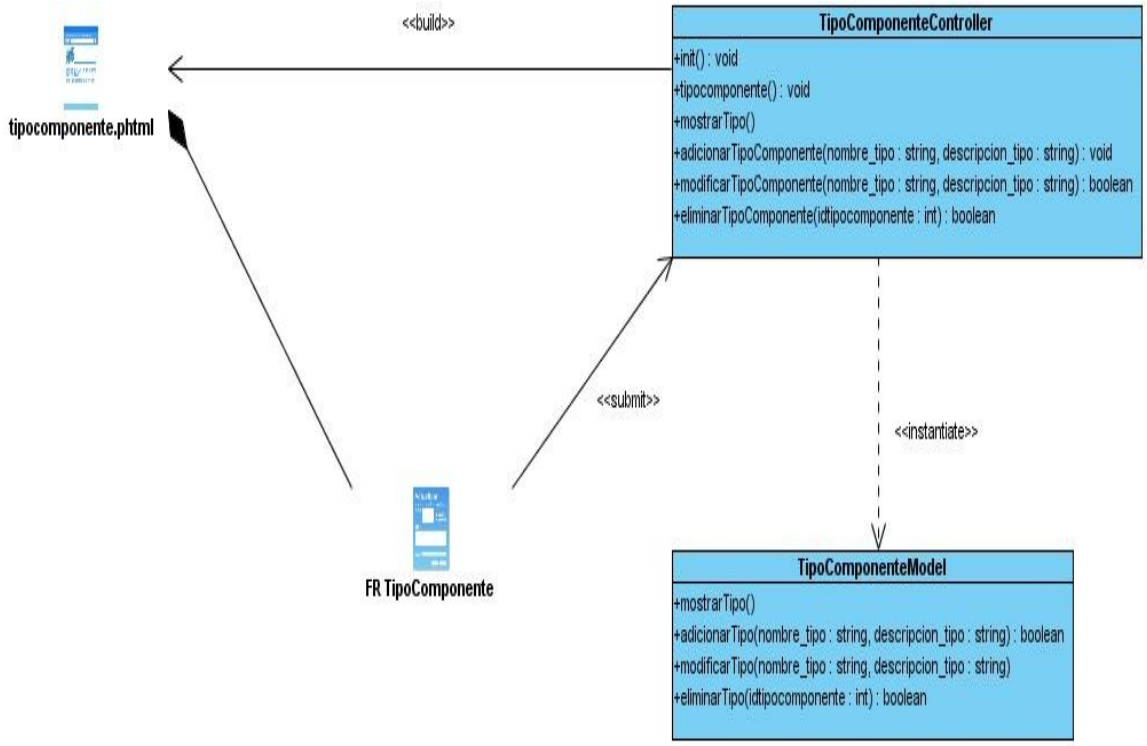


Figura 8 Diagrama de clases Tipo componente.

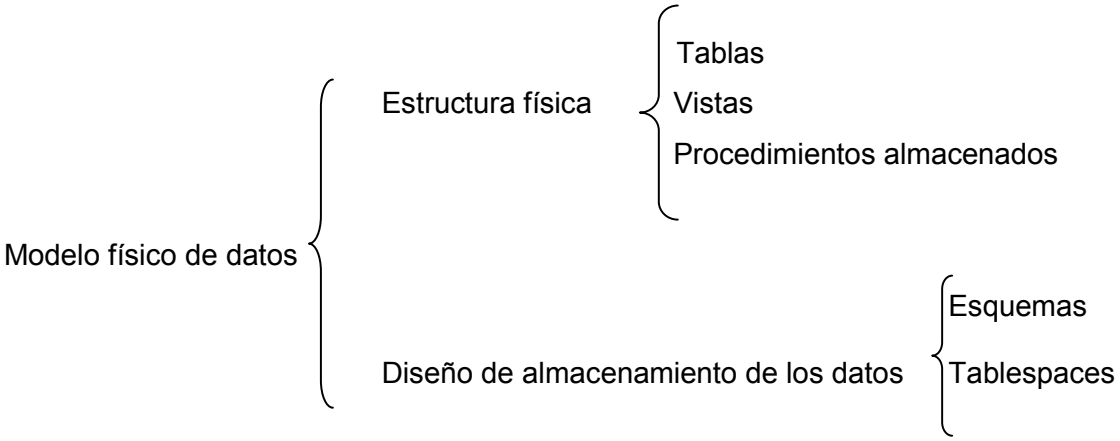
2.4 Artefactos de implementación

- Modelo de datos.
- Diagrama de componentes.

2.4.1 Modelo de datos

El modelo de datos ha sido diseñado manteniendo la estructura y el estandarizado trazado por parte del proyecto ERP-Cuba. Partiendo desde el nombre del esquema, de las tablas, las funciones y las secuencias hasta los tipo de datos de cada campo dentro de las tablas.

Este representa los datos en el más bajo nivel y permite identificar algunos detalles de implantación para el manejo del hardware de almacenamiento, proporciona referencias de cómo se almacenan los datos en la computadora, el formato de los registros, la estructura de los ficheros (ordenados, desordenados) y los métodos de acceso utilizados. Los conceptos de este modelo están dirigidos fundamentalmente al personal informático, no a los usuarios finales.



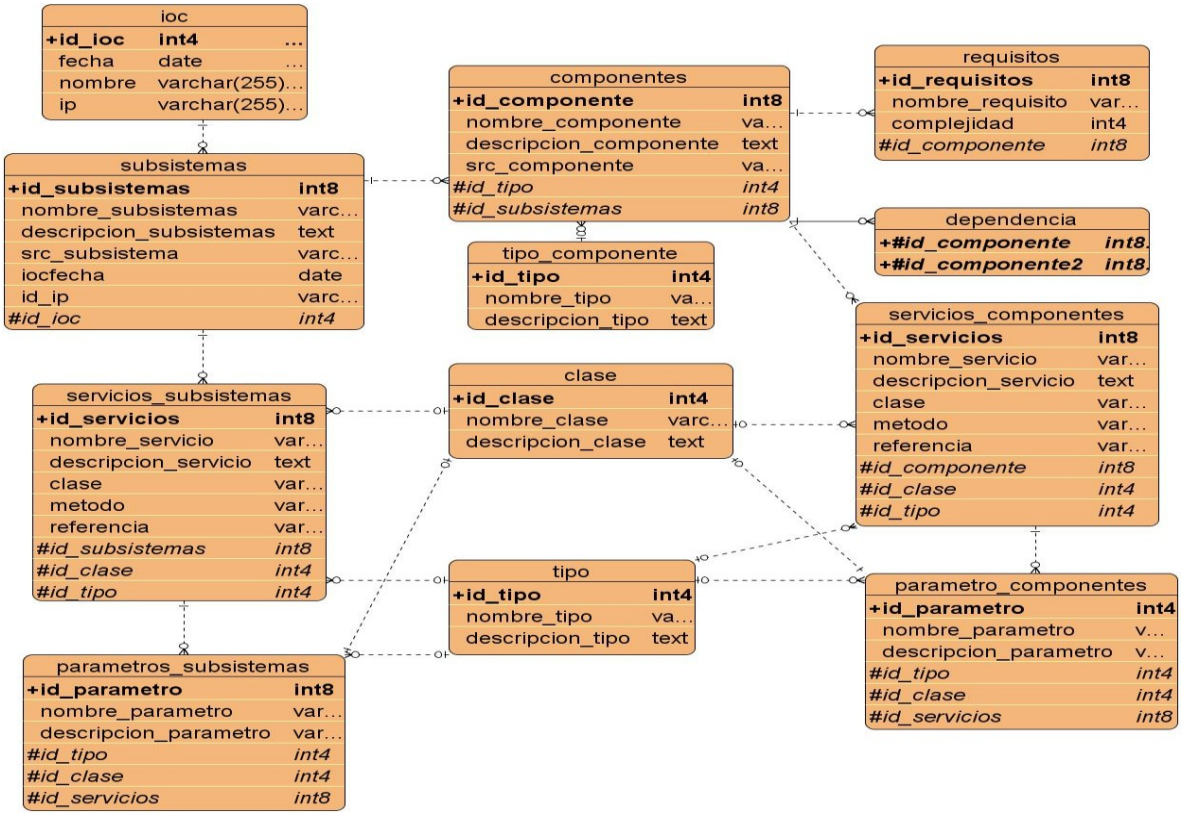


Figura 9 Modelo de datos.

Descripción de las tablas

Nombre: tipo.		
Descripción: tabla que guarda todos los datos de cada uno de los tipos de datos de los parámetros.		
Atributos	**Tipo**	**Descripción**
id_tipo	int4	Identificador de la tabla.
nombre_tipo	varchar	Nombre del tipo de dato.
descripcion_tipo	text	Descripción del tipo de dato.

Nombre: clase.		
Descripción: tabla que guarda todos los datos de cada uno de los tipos de datos de los parámetros.		
Atributos	**Tipo**	**Descripción**
id_clase	int4	Identificador de la tabla.
nombre_clase	varchar	Nombre de la clase.
descripcion_clase	text	Descripción de la clase.

Nombre: tipo_componente.		
Descripción: tabla que guarda todos los datos de cada uno de los tipos de componentes.		
Atributos	**Tipo**	**Descripción**
id_tipo	int4	Identificador de la tabla.
nombre_tipo	varchar	Nombre del tipo de componente.
descripcion_tipo	text	Descripción del tipo de componente.

Nombre: ioc.		
Descripción: tabla que guarda todos los datos de las fechas y nombre de los archivos importados.		
Atributos	**Tipo**	**Descripción**
id_ioc	int4	Identificador de la tabla.
nombre	varchar	Nombre del archivo para mostrar.
fecha	date	Fecha actual en la que se importo el archivo.

Nombre: subsistemas.		
Descripción: tabla que guarda todos los datos de los subsistemas.		
Atributos	**Tipo**	**Descripción**
id_subsistemas	int8	Identificador de la tabla.
nombre_subsistemas	varchar	Nombre del subsistema.
descripcion_subsistemas	text	Descripción del subsistema.
src_subsistema	varchar	Dirección real del subsistema.
id_ioc	int4	Identificador de la tabla de los archivos importados.
id_clase	int4	Identificador de la tabla de clases.
id_tipo	int4	Identificador de la tabla de tipos de datos.

Nombre: servicios_subsistemas.		
Descripción: tabla que guarda todos los datos de los servicios referentes a un subsistema.		
Atributos	**Tipo**	**Descripción**
id_servicios	int8	Identificador de la tabla.
nombre_servicio	varchar	Nombre del servicio.
descripcion_servicio	text	Descripción del servicio.
clase	varchar	Clase a la cual pertenece ese servicio.

metodo	varchar	Nombre del método de este servicio.
referencia	varchar	Dirección real del servicio.
id_subsistemas	int8	Identificador de la tabla de subsistema.
id_clase	int4	Identificador de la tabla clase.
id_tipo	int4	Identificador de la tabla tipo de datos.

Nombre: parametros_subsistemas.		
Descripción: tabla que guarda todos los datos de los parámetros de los servicios de un subsistema.		
Atributos	**Tipo**	**Descripción**
id_parametro	int8	Identificador de la tabla.
nombre_parametro	varchar	Nombre del parámetro.
descripcion_parametro	text	Descripción del parámetro.
id_tipo	int4	Identificador de la tabla de tipo de datos.
id_clase	int4	Identificador de la clases.
id_servicios	int8	Identificador del servicio subsistema.

Nombre: componentes.		
Descripción: tabla que guarda todos los datos de los componentes de un subsistema.		
Atributos	**Tipo**	**Descripción**
id_componente	int8	Identificador de la tabla.
nombre_componente	varchar	Nombre del componente.
descripcion_componente	text	Descripción del componente.
src_componente	varchar	Dirección real del componente.
id_tipo	int4	Identificador de la tipo de componente.
id_subsistemas	int8	Identificador de la tabla de subsistema.

Nombre: requisitos.		
Descripción: tabla que guarda todos los datos de los requisitos de los componentes.		
Atributos	**Tipo**	**Descripción**
id_requisitos	int8	Identificador de la tabla.
nombre_requisito	varchar	Nombre del requisito.
complejidad	int4	Complejidad del requisito.
id_componente	int4	Identificado de la tabla componente.
id_tipo	int4	Identificador de la tipo de componente.
id_subsistemas	int8	Identificador de la tabla de subsistema.

Nombre: dependencia.
Descripción: tabla que guarda todos los datos de la relación de dependencia de los

componentes.		
Atributos	**Tipo**	**Descripción**
id_componente	int8	Identificador de la tabla componente.
id_componente2	int8	Identificador de la tabla componente.

Nombre: servicios_componentes.		
Descripción: tabla que guarda todos los datos de los servicios de los componentes.		
Atributos	**Tipo**	**Descripción**
id_servicios	int8	Identificador de la tabla componente.
nombre_servicio	varchar	Nombre del servicio.
descripcion_servicio	text	Descripción del servicio.
clase	varchar	Clase a la cual pertenece ese servicio.
metodo	varchar	Nombre del método de este servicio.
referencia	varchar	Dirección real del servicio
id_componente	int8	Identificador de la tabla de componente.
id_clase	int4	Identificador de la tabla clase.
id_tipo	int4	Identificador de la tabla tipo de datos.

Nombre: parametros_componente.		
Descripción: tabla que guarda todos los datos de los parámetros de los servicios de un componente.		
Atributos	**Tipo**	**Descripción**
id_parametro	int8	Identificador de la tabla.
nombre_parametro	varchar	Nombre del parámetro.
descripcion_parametro	text	Descripción del parámetro.
id_tipo	int4	Identificador de la tabla de tipo de datos.
id_clase	int4	Identificador de la clases.
id_servicios	int8	Identificador del servicio del componente.

2.4.2 Diagrama de componentes

Partiendo de la estructuración del diagrama de componentes se puede destacar la gestión de software, la reutilización, los requisitos relacionados con la facilidad de desarrollo y las restricciones según los lenguajes de programación y las herramientas utilizadas durante el proceso de desarrollo.

Los diagramas de componentes son utilizados para modelar la vista estática del sistema y representan las dependencias y organización entre los componentes que lo componen, estos diagramas muestran cómo quedan organizadas las clases del diseño

en términos de componentes. No es necesario que un diagrama incluya todos los componentes del sistema, normalmente se realizan por partes. Uno de los usos principales es que puede servir para ver qué componentes pueden compartirse entre sistemas o entre diferentes partes de un sistema. Este diagrama está compuesto por seis componentes, dentro de los cuales se encuentra el componente ioc que brinda los servicios de cargar y exportar XML, el componente subsistema consume el servicio cargar subsistema que se encuentra incluido dentro del servicio cargar XML que brinda el componente ioc. El componente subsistema es el encargado de gestionar la información con respecto a los mismos, este está compuesto por el componente servicio subsistema el cuál se encarga de gestionar los servicios de dicho subsistema y por el componente componente cuyas funcionalidades son las de gestionar los componentes de dicho subsistema, el mismo está compuesto por el componente servicio componente el cuál se encarga de gestionar los servicios de dicho componente y por el componente requisito cuyo objetivo es gestionar los requisitos de dicho componente.

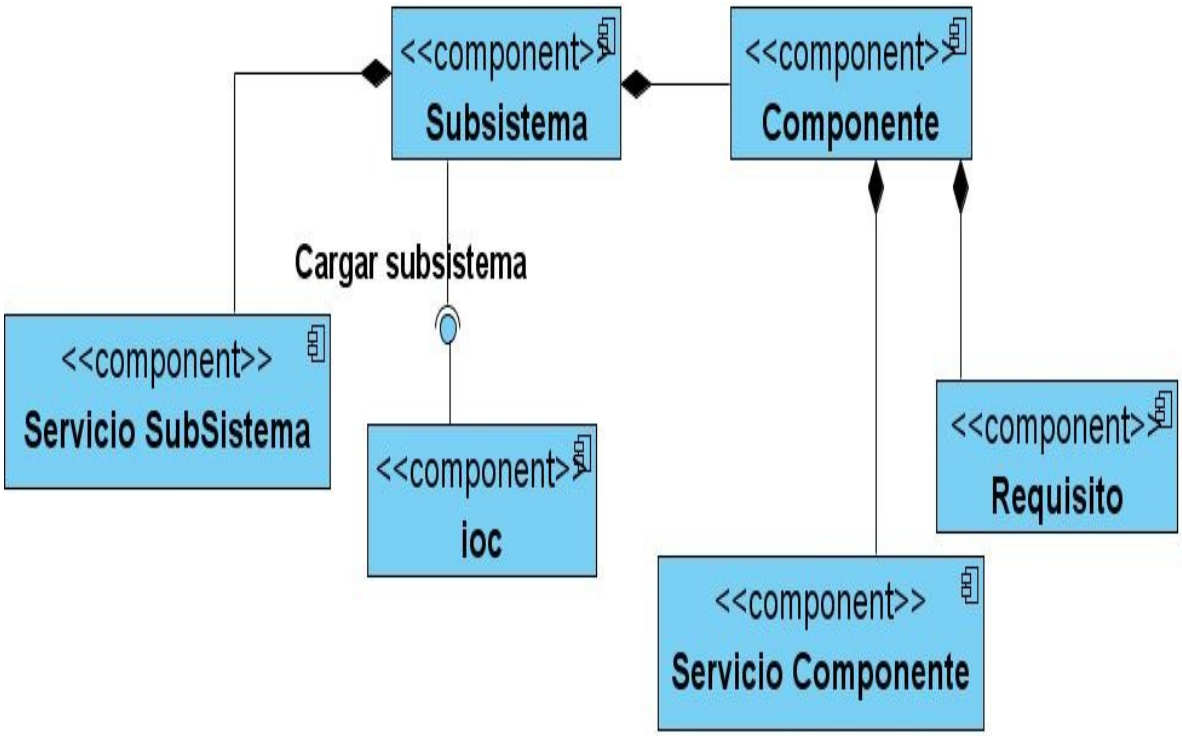


Figura 10 Diagrama de componentes.

2.5 Conclusiones

Durante el transcurso del capítulo se definieron los requisitos funcionales que contiene la aplicación mediante el levantamiento de los mismos. También se plasmó el diseño

de clases para una mejor comprensión de cómo están estructuradas las clases que conforman toda la capa del negocio de los componentes. Se expusieron además los distintos artefactos de implementación generados: diagrama de componentes y el modelo de datos. Quedando constituida de esta forma una guía práctica para la implementación de la solución propuesta.

Capítulo 3: Evaluación de la solución.

Introducción

En la implementación de diferentes sistemas informáticos juega un papel esencial el uso de las técnicas de evaluación dinámica o pruebas. Para lograr esto se deben llevar a cabo estrategias a la hora de evaluar dinámicamente el software, pues para la evaluación de este se debe comenzar desde los componentes más simples y pequeños e ir avanzando progresivamente hasta probar todo el software en su conjunto. Es por esta razón que las técnicas utilizadas son las pruebas unitarias. Es una forma de probar el correcto funcionamiento de un módulo de código, sirve para asegurar que cada uno de los módulos funcione correctamente por separado.

La prueba y validación de los resultados no es un proceso que se realiza una vez desarrollado el software, debe efectuarse en cada una de las etapas de desarrollo. Es fundamental medir la cobertura de las pruebas, es decir, la determinación de cuándo se han realizado las suficientes pruebas para arribar a una determinación. Si se siguen encontrando errores cada vez que se procesa el programa, las pruebas deben continuar. En este capítulo se realiza la validación a la solución propuesta en el capítulo anterior, de manera que se puede comprobar la eficiencia de las clases u operaciones utilizadas para dar respuesta a los distintos requisitos planteados por el cliente.

3.1 Métricas

Métrica es el término que describe varios casos de medición. Siendo una métrica una medida estadística (no cuantitativa como en otras disciplinas ejemplo física) que se aplica a todos los aspectos de calidad de software, los cuales deben ser medidos desde diferentes puntos de vista como el análisis, construcción, funcional, documentación, métodos, proceso, usuario, entre otros.

Métricas técnicas. Estas se presentan en el libro de Ingeniería del software de Pressman página 58. Las mismas se derivan de una relación empírica según las medidas contables del dominio de información del software y de evaluaciones de complejidad. Ejemplo,

Número de entradas usuario – cada una de las entradas de datos.
Número de salidas usuario – cada una de las salidas de datos.

Número de peticiones usuario – cada generación de un evento.

Número de archivos – cada tabla, archivo,…

Número de interfaces externas – son interfaces, discos, copias de seguridad, transmisiones de datos. (Pressman., 2001)

Tamaño operacional de clase (TOC):

Está dado por el número de métodos asignados a una clase.

Tabla 3 Descripción de la métrica Tamaño operacional de clase

Atributo que afecta	**Modo en que lo afecta**
Responsabilidad	Un aumento del TOC implica un aumento de la responsabilidad asignada a la clase.
Complejidad de implementación	Un aumento del TOC implica un aumento de la complejidad de implementación de la clase.
Reutilización	Un aumento del TOC implica una disminución en el grado de reutilización de la clase.

Relaciones entre clases (RC):

Está dado por el número de relaciones de uso de una clase con otras.

Tabla 4 Descripción de la métrica Relaciones entre clases

Atributo que afecta	**Modo en que lo afecta**
Acoplamiento	Un aumento del RC implica un aumento del acoplamiento de la clase.
Complejidad del mantenimiento	Un aumento del RC implica un aumento de la complejidad de mantenimiento de la clase.
Reutilización	Un aumento del RC implica una disminución en el grado de reutilización de la clase.
Cantidad de pruebas	Un aumento del RC implica un aumento de la cantidad de pruebas de unidad necesarias para probar una clase.

Resultados del instrumento de evaluación de la métrica Tamaño operacional de clase (TOC).

Ver instrumentos y tabla de resultados en (Anexo 1 Instrumento de medición de la métrica Tamaño operacional de clase (TOC)).

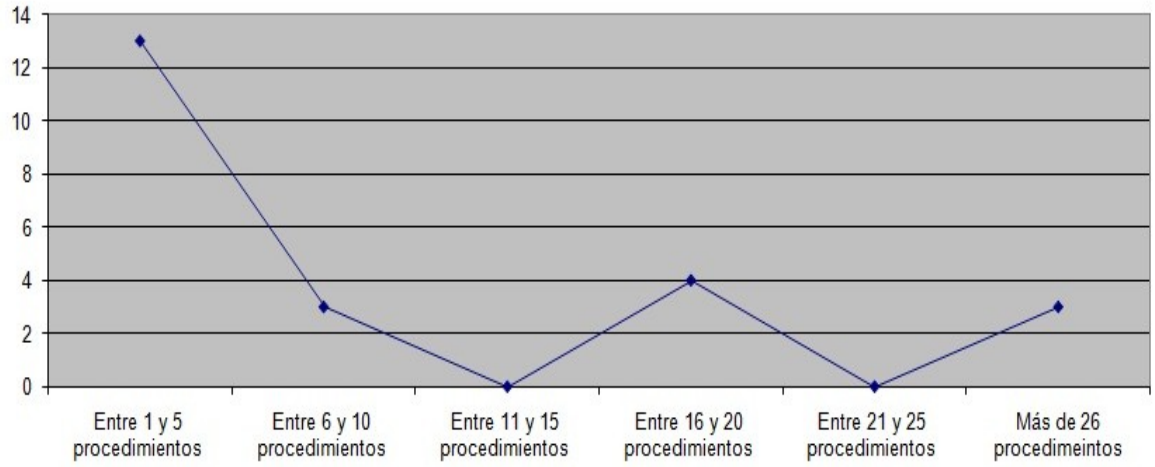


Figura 11 Representación de los resultados obtenidos en el instrumento agrupado en los intervalos definidos.

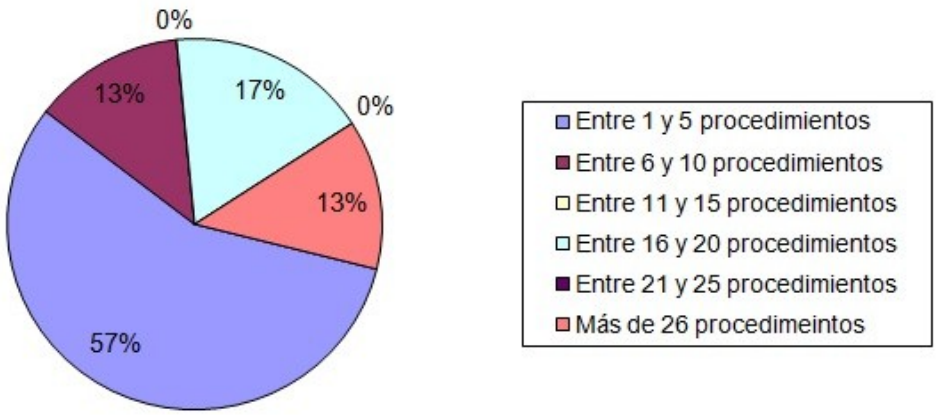


Figura 12 Representación en % de los resultados obtenidos en el instrumento agrupado en los intervalos definidos.

Figura 13 Representación de la incidencia de los resultados de la evaluación de la métrica Tamaño operacional de clase en el atributo Responsabilidad.

Figura 14 Representación de la incidencia de los resultados de la evaluación de la métrica Tamaño operacional de clase en el atributo Complejidad de implementación.

Figura 15 Representación de la incidencia de los resultados de la evaluación de la métrica Tamaño operacional de clase en el atributo Reutilización.

Haciendo un análisis de los resultados obtenidos en la evaluación del instrumento de medición de la métrica TOC, se puede concluir que el diseño de los componentes File Up Load, Principal, Requisito, Subsistema y Componente tienen una calidad favorable

debido a que el 87% de las clases incluidas en estos componentes poseen menos cantidad de operaciones que la mitad del valor máximo registrado en las mediciones. Además el 87% de las clases poseen evaluaciones positivas en los atributos de calidad (Responsabilidad, Complejidad de implementación y Reutilización).

Resultados del instrumento de evaluación de la métrica Relaciones entre clases (RC)

Ver instrumentos y tabla de resultados en (Anexo 2 Instrumento de medición de la métrica Relaciones entre clases (RC)).

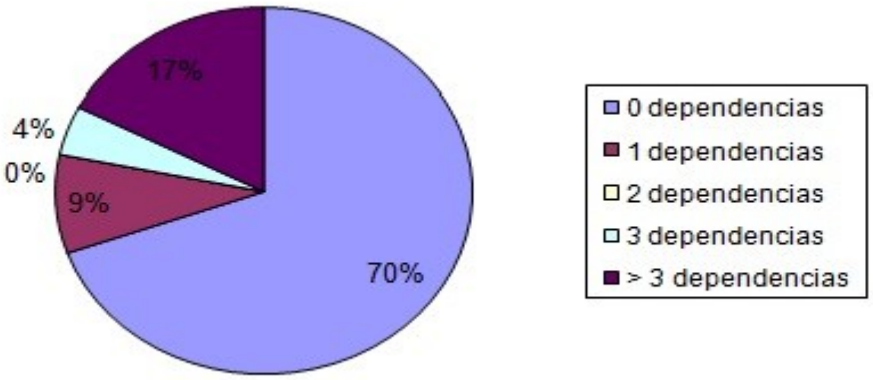


Figura 16 Representación en % de los resultados obtenidos en el instrumento agrupado en los intervalos definidos.

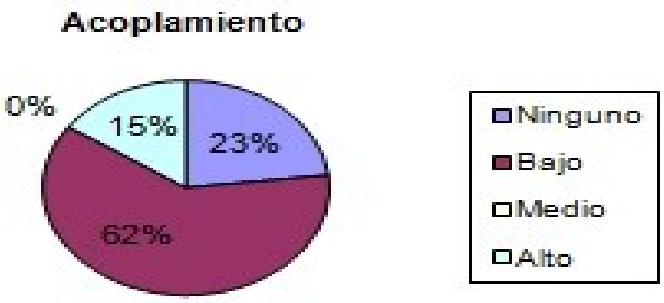


Figura 17 Representación de la incidencia de los resultados de la evaluación de la métrica Relaciones entre clases en el atributo Acoplamiento.

Figura 18 Representación de la incidencia de los resultados de la evaluación de la métrica Relaciones entre clases en el atributo Complejidad de mantenimiento.

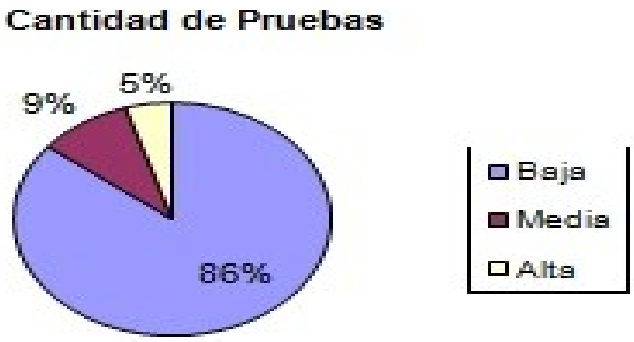


Figura 19 Representación de la incidencia de los resultados de la evaluación de la métrica Relaciones entre clases en el atributo Cantidad de pruebas.

Figura 20 Representación de la incidencia de los resultados de la evaluación de la métrica Relaciones entre clases en el atributo Reutilización.

Haciendo un análisis de los resultados obtenidos en la evaluación del instrumento de medición de la métrica RC, se puede concluir que el diseño de los componentes File Up Load, Principal, Requisito, Subsistema y Componente tienen una calidad aceptable teniendo en cuenta que el 70% de las clases incluidas en estos componentes poseen menos de 3 dependencias de otras clases. Además el 85% de las clases no poseen acoplamiento con otras, esto trae consigo que posean índices aceptables en cuanto a Acoplamiento en un 85%. Los atributos de calidad Complejidad de mantenimiento y Cantidad de pruebas se comportan satisfactoriamente en un 86% y Reutilización tiene índices aceptables en un 80% de las clases.

3.2 Pruebas de software

La IEEE define las pruebas como una actividad en la cual un sistema o componente es ejecutado bajo condiciones específicas, se observan o almacenan los resultados realizándose una evaluación de algún aspecto del sistema o componente. Las pruebas son sets de uno o más casos de pruebas.

Un proceso de pruebas son conceptos de pruebas, estrategias, técnicas y medidas que deben integrarse en un determinado proceso controlado y dirigido por personas, apoyando a las actividades de prueba y proporcionando una guía para los equipos de pruebas, desde la planificación de prueba, para probar la salida de evaluación, de tal forma que proporcione garantía de que los objetivos de las pruebas se cumplan de manera rentable. (Brown, 1998)

Tomando como base los conceptos anteriormente citados se puede llegar a la conclusión de que las pruebas son una actividad en la cual un sistema o componente es ejecutado bajo condiciones específicas, donde se valida la calidad y funcionalidad del producto mediante el análisis y evaluación de los resultados de dicha ejecución con el fin de detectar errores, que impidan que dicho producto responda con las necesidades especificadas previamente.

Un proceso de prueba es el establecimiento de los objetivos de las pruebas, el diseño de los casos de pruebas a aplicar, la codificación de los casos de pruebas, la ejecución de los mismos y el análisis de los resultados de dicha ejecución.

3.2.1 Objetivos

El objetivo esencial en la etapa de pruebas es garantizar la calidad del producto desarrollado. (Collado, 2003)

Esto implica:

- ✓ Verificar la interacción de componentes.
- ✓ Verificar la integración adecuada de los componentes.
- ✓ Verificar que todos los requisitos se han implementado correctamente.
- ✓ Identificar y asegurar que los defectos encontrados se han corregido antes de entregar el software al cliente.
- ✓ Diseñar pruebas que sistemáticamente saquen a la luz diferentes clases de errores, haciéndolo con la menor cantidad de tiempo y esfuerzo."

La prueba no es una actividad sencilla, no es una etapa del proyecto en la cual se asegura la calidad, sino que debe ocurrir durante todo el ciclo de vida: probar la funcionalidad de los primeros prototipos; probar la estabilidad, cobertura y rendimiento de la arquitectura; probar el producto final, entre otras. Lo que conduce al principal beneficio de la prueba: proporcionar realimentación mientras hay todavía tiempo y recursos para hacer algo.

3.2.2 Alcance

Se lleva a cabo la prueba y se evalúan los resultados obtenidos frente a los resultados esperados. Si se descubren datos erróneos da lugar a que existe un error y hay que corregirlo, aquí comienza el proceso de depuración de errores. Basado en las estructuras de control del diseño procedimental para generar los casos de prueba que:

- ✓ Garanticen que se recorran por lo menos una vez todos los caminos independientes de cada módulo.
- ✓ Se ejecuten todas las decisiones lógicas en su parte verdadera y en su parte falsa.
- ✓ Se recorran todos los bucles.
- ✓ Se utilizan las estructuras de datos internas para garantizar su validez.
- ✓ Se invierte tiempo y esfuerzo en los detalles de control debido a que:
- ✓ Los errores suelen estar en situaciones fuera de las normales.
- ✓ A menudo, caminos que se piensa que tienen pocas posibilidades de recorrerse, son recorridos regularmente.
- ✓ Los errores tipográficos son aleatorios. Puede que no sean detectados por los procesadores de la sintaxis del lenguaje particular y estar presentes en cualquier camino lógico. (León, 2005)

3.3 Descripción de las pruebas

Las pruebas de unidad son la manera de comprobar el funcionamiento correcto de determinado módulo de código, ayuda a independizar el módulo, significa esto que se pueden probar los módulos independientemente uno de otros.

Antes de iniciar cualquier otra prueba es necesario probar el flujo de datos de la interfaz del módulo. Si los datos no fluyen correctamente, todas las demás pruebas no tienen sentido.

Los “test de unidades” o pruebas de unidad son orientados en la mayoría de los casos a las pruebas de “caja blanca” aunque para realizar uno de estos test es necesario probar el flujo de datos desde la interfaz del componente. Si los datos no se comportan correctamente al ser introducidos en la aplicación, todas las demás pruebas no tienen sentido. (Bartolomé, 2007)

Estas pruebas son las denominadas pruebas de caja blanca y son aplicadas a componentes representados en el modelo de implementación para verificar que los flujos de control y de datos están cubiertos, y que estos funcionen como se espera.

Otra prueba realizada a la solución obtenida fue la de caja negra, esta prueba se aplica sobre las interfaces del sistema, para ello las mismas deben de estar bien definidas en términos de entradas y salidas. Esta prueba se aplicó a la solución, en el laboratorio de la subdirección de arquitectura del proyecto ERP-Cuba.

3.3.1 Prueba de caja blanca o estructural.

La prueba del camino básico es una técnica de prueba de caja blanca propuesta por Tom McCabe en 1976. Esta permite al diseñador de casos de prueba obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución.

Los casos de prueba obtenidos del conjunto básico garantizan que durante la prueba se ejecute por lo menos una vez cada sentencia del programa. (Barrios, 2007)

Para aplicar la técnica del camino básico se debe introducir la notación para la representación del flujo de control, este puede representarse por un grafo de flujo en el cual:

- ✓ Cada nodo del grafo corresponde a una o más sentencias de código fuente.

- ✓ Todo segmento de código de cualquier programa se puede traducir a un grafo de flujo.
- ✓ Se calcula la complejidad ciclomática del grafo.

Para construir el grafo se debe tener en cuenta la notación para las instrucciones.

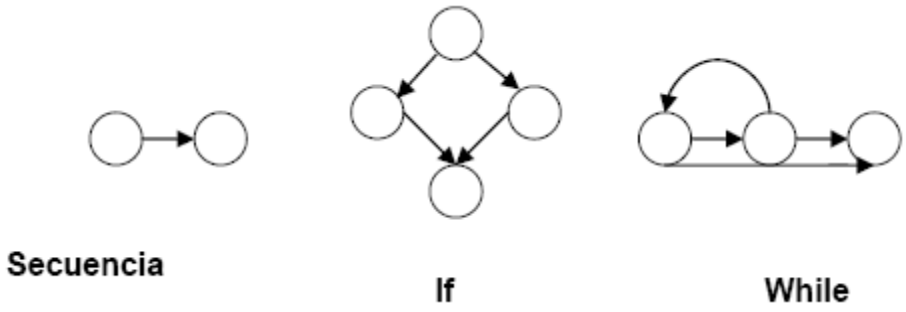


Figura 21 Notación de grafos de flujo para las instrucciones: Secuenciales, If, While.

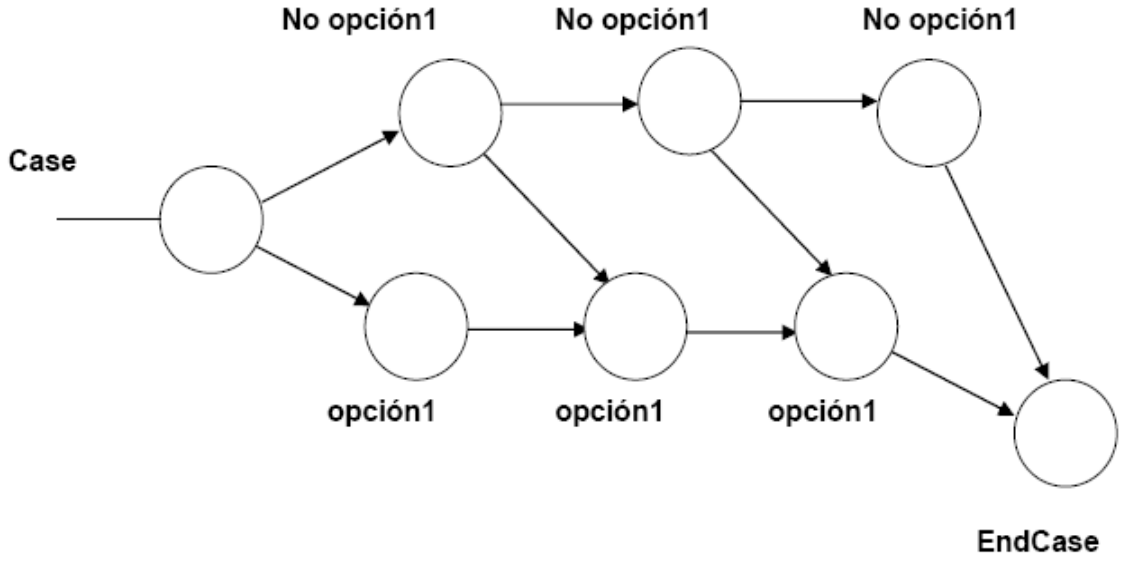


Figura 22 Notación de grafos de flujo para la instrucción Case.

Un grafo de flujo está formado por 3 componentes fundamentales que ayudan a su elaboración y comprensión, además brindan información para confirmar que el trabajo se está haciendo adecuadamente.

Componentes del grafo de flujo

Nodo: Cada círculo representado se denomina nodo del grafo de flujo, el cual representa una o más secuencias procedimentales. Un solo nodo puede corresponder

a una secuencia de procesos o a una sentencia de decisión. Puede ser también que hayan nodos que no se asocien, se utilizan principalmente al inicio y final del grafo.

Aristas: Las flechas del grafo se denominan aristas y representan el flujo de control, son análogas a las representadas en un diagrama de flujo. Una arista debe terminar en un nodo, incluso aunque el nodo no represente ninguna sentencia procedimental.

Regiones: Las regiones son las áreas delimitadas por las aristas y nodos. También se incluye el área exterior del grafo, contando como una región más. Las regiones se enumeran y la cantidad de regiones es equivalente a la cantidad de caminos independientes del conjunto básico de un programa.

En la figura se observa la representación un grafo de flujo, en el cual aparecen sus componentes.

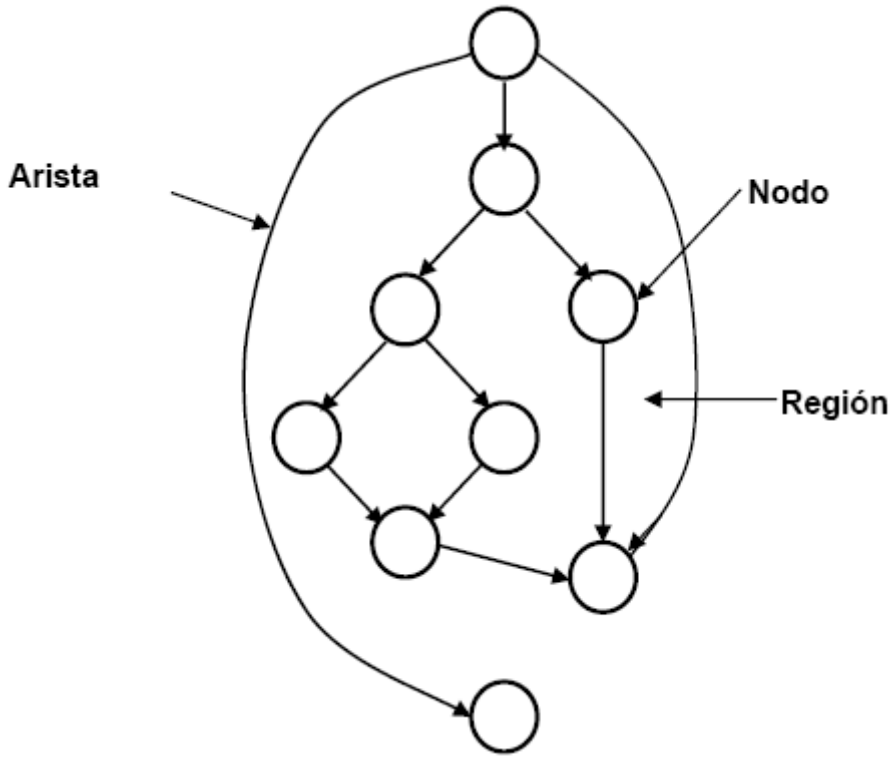


Figura 23 Componentes de los grafos de flujo.

Cálculo de la complejidad ciclomática a partir de un segmento de código.

La complejidad ciclomática es una métrica de software extremadamente útil, proporciona una medición cuantitativa de la complejidad lógica de un programa. El valor calculado como complejidad ciclomática define el número de caminos

independientes del conjunto básico de un programa y nos da un límite superior para el número de pruebas que se deben realizar para asegurar que se ejecute cada sentencia al menos una vez. Un camino independiente es cualquier camino del programa que introduce por lo menos un nuevo conjunto de sentencias de procesamiento o una nueva condición. El camino independiente se debe mover por lo menos por una arista que no haya sido recorrida anteriormente. Si se diseñan pruebas que fuesen el recorrido de esos caminos, se garantiza que se ejecute al menos una vez cada sentencia del programa y que cada condición se ejecute en sus variantes verdadera y falsa. Se debe tener en cuenta que de un mismo diseño procedimental se pueden derivar varios conjuntos básicos. Formas fundamentales de calcular la complejidad: Atendiendo al número de regiones del grafo de flujo: El número de regiones del grafo de flujo coincide con la complejidad ciclomática. (Gómez, 2008)

1. La complejidad ciclomática, V (G), se define como:
V (G) = A - N + 2 Donde: A es el número de aristas del grafo y N es el número de nodos.
2. La complejidad ciclomática, V (G), también se define como:
V (G) = P + 1 Donde: P es el número de nodos predicados contenidos en el grafo G

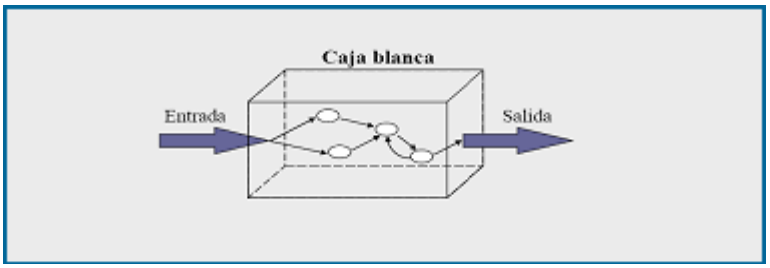


Figura 24 Representación de pruebas de caja blanca.

Mediante las pruebas de la caja blanca se puede obtener casos de prueba que:

1. Garanticen que se ejerciten por lo menos una vez todos los caminos independientes de cada módulo, programa o método.
2. Ejerciten todas las decisiones lógicas en las vertientes verdadera y falsa.
3. Ejecuten todos los bucles en sus límites operacionales.
4. Ejerciten las estructuras internas de datos para asegurar su validez.

Algunas de las técnicas de prueba de caja blanca son:

1. Prueba de condición: método de diseño de casos de prueba que ejercita las condiciones lógicas contenidas en el módulo de un programa.

2. Prueba de flujo de datos: se seleccionan caminos de prueba de un programa de acuerdo con la ubicación de las definiciones y los usos de las variables del programa.

3. Prueba de bucles: es una técnica de prueba de caja blanca que se centra exclusivamente en la validez de las construcciones de bucles.

4. Prueba del camino básico: esta técnica permite obtener una medida de la complejidad lógica de un diseño y usar esta medida como guía para la definición de un conjunto básico.

La idea es derivar casos de prueba a partir de un conjunto dado de caminos independientes por los cuales puede circular el flujo de control. Para obtener dicho conjunto de caminos independientes se construye el grafo de flujo asociado y se calcula su complejidad ciclomática.

Aplicando el diseño de casos de pruebas al software en cuestión se puede conseguir una prueba más completa, descubriendo y corrigiendo el número mayor de errores antes de que comiencen las "pruebas del cliente". Para probar se necesita una organización y planificación de las pruebas que se van a realizar.

La forma más práctica de aplicar los tipos de pruebas y la que se propone es la siguiente:

- ✓ Medir la cobertura de caja blanca que se ha logrado con las fases previas y añadir más pruebas de caja blanca hasta lograr la cobertura deseada.

- ✓ Diseñar casos de prueba para examinar la lógica del programa para garantizar que se ejerciten todos los caminos independientes de cada módulo, todas las decisiones lógicas, y que se ejecutan todos los bucles y las estructuras de datos internas.

3.4 Aplicación de pruebas de caja blanca

Para realizar la prueba de caja blanca específicamente la prueba del camino básico es necesario calcular antes la complejidad ciclomática del algoritmo o fragmento de código a analizar. A continuación se enumeran las sentencias de código del procedimiento realizado sobre el método adicionarServicioAction () el cual se encarga de adicionar un servicio en un subsistema.

```php
<?php
    public function adicionarServicioAction()
    {
        try
        {
            $idResultado = '';(1)
            $subSistemaObjeto = new SubSistemaModel();(1)
            $idsubsistema = $this->_request->getPost('subsistema');(1)
            $nombre = $this->_request->getPost('tfNombre');(1)
            $descripcion = $this->_request->getPost('taDescripcion');(1)
            $clase = $this->_request->getPost('tfClase');(1)
            $metodo = $this->_request->getPost('tfMetodo');(1)
            $referencia = $this->_request->getPost('tfReferencia');(1)
            $tipo = $this->_request->getPost('tipo');(1)
            if ($tipo == 1)(2)
            {
                $resultado =  $this->_request->getPost('cbTipoDatos');(3)
                $idResultado = $subSistemaObjeto->idTipo($resultado);(4)
            }
            else
            {
                $resultado =  $this->_request->getPost('cbClases');(5)
                $idResultado = $subSistemaObjeto->idClase($resultado);(6)
            }
            $subSistemaObjeto->insertarServicio($idsubsistema, $nombre, $descripcion,
                                                $clase, $metodo, $referencia, $tipo, $idResultado);(7)
            echo ("{'codMsg':1}");(8)
        }
        catch (Exception $error)
        {
            throw $error;
        }
    }
?>
```

Figura 25 Representación del método adicionarServicio ().

Seguidamente se construye el grafo de flujo asociado al código anterior.

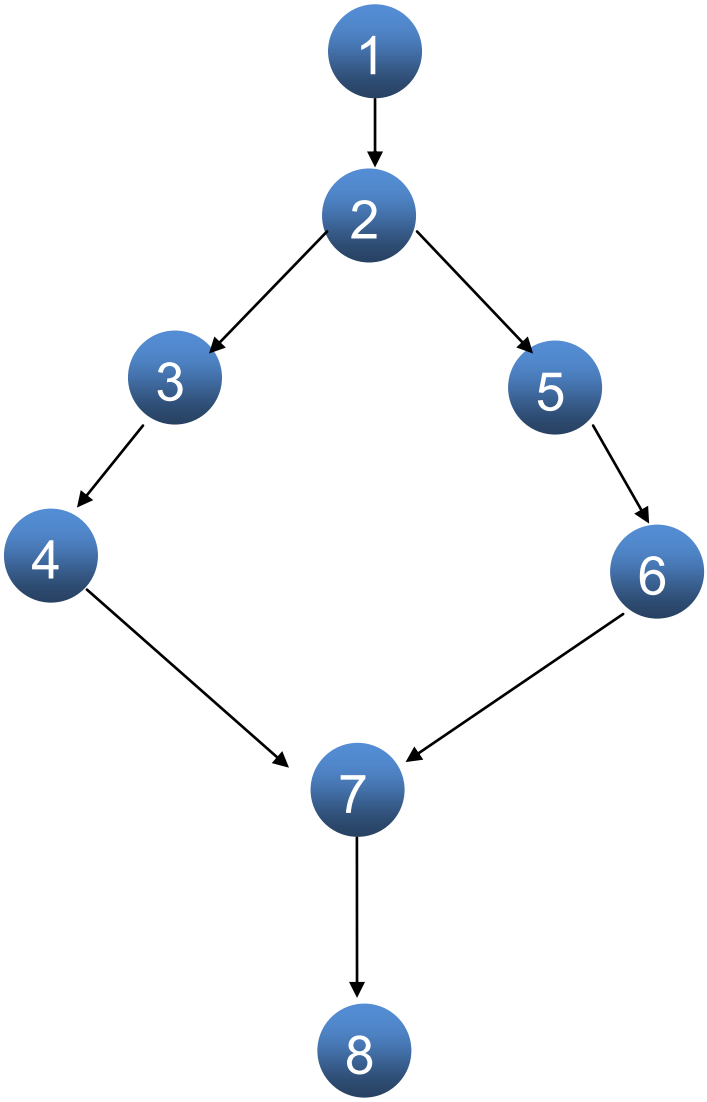


Figura 26 Representación del flujo asociado al método adicionarServicio ().

Cálculo de la complejidad ciclomática a partir de un segmento de código.

Para efectuar el cálculo de la complejidad ciclomática del código es necesario tener varios parámetros como son la cantidad total de aristas del grafo, cantidad total de nodos para la siguiente fórmula:

V (G) = (A – N) + 2
V (G) = (8 – 8) + 2
V (G) = 2

Siendo: A: la cantidad total de aristas N: la cantidad total de nodos. Se puede calcular también de esta otra manera:

V (G) = P + 1
V (G) = 1 + 1
V (G) = 2

Para: P: Cantidad total de nodos predicados (son los nodos de los cuales parten dos o más aristas).

V (G) = R
V (G) = 2

R: Cantidad total de regiones, para cada fórmula V (G) representa el valor del cálculo de la complejidad ciclomática.

Se ha obtenido el mismo resultado luego de haber calculado la complejidad ciclomática mediante las tres fórmulas por lo que se puede concluir que la complejidad ciclomática del código que implementa el método adicionarServicio (). Es igual a 2 unidades, esto da lugar a que existan 2 caminos básicos por los que el flujo puede transitar, la complejidad ciclomática va a representar el límite del mínimo número total de casos de pruebas para el procedimiento tratado.

A continuación se representan los caminos básicos por los que puede recorrer el flujo. Se subrayan las representaciones y los elementos de cada camino que los hacen independientes a los demás.

Tabla 5 Caminos básicos del flujo

Número	**Camino básico**
1	1-2-3-4-7-8
2	1-2-5-6-7-8

Ahora se procede a ejecutar los casos de pruebas para este procedimiento, se debe realizar al menos un caso de prueba por cada camino básico. Para la exitosa realización de los mismos es necesario cumplir con las siguientes exigencias:

- ✓ Descripción: se hace la entrada de datos necesaria, validando que ningún parámetro obligatorio pase nulo al procedimiento, o se entre algún dato erróneo.

- ✓ Condición de ejecución: se especifica cada parámetro para que cumpla una condición deseada para ver el funcionamiento del procedimiento.
- ✓ Entrada: se muestran los parámetros que entran al procedimiento.
- ✓ Resultados esperados: se expone resultado que se espera que devuelva el procedimiento.

Caso de prueba para el camino básico 1:

Camino 1: [1 - 2 - 3 - 4 - 7 - 8]
Condición de ejecución:
Tipo tendrá valor de 1.
Entrada:
$resultado = cbTipoDatos.
Resultados esperados:
Se espera que se lance un mensaje con el siguiente texto: “El servicio fue insertado correctamente”.

Caso de prueba para el camino básico 2:

Camino 2: [1-2-5-6-7-8]
Condición de ejecución:
Tipo tendrá valor diferente de 1.
Entrada:
$resultado = cbClases.
Resultados esperados:
Se espera que se lance un mensaje con el siguiente texto: “El servicio fue insertado correctamente”.

3.5 Conclusiones

Con el desarrollo del presente capítulo para la comprensión del mismo se analizaron diferentes conceptos como el de las pruebas de software, sus objetivos, diferentes alcances y aplicaciones. Se realizó también una breve descripción de los test de unidad y las pruebas de caja blanca, haciendo énfasis en las pruebas del camino básico y el cálculo de la complejidad ciclomática.

Al aplicar estas pruebas de unidad a un fragmento de código de la implementación, del cual se obtuvo una complejidad ciclomática de 2 unidades, (dando lugar a 2 caminos básicos, comprobándose que eran los únicos por los que siempre transitaba el flujo),

se diseñaron 2 casos de pruebas de los cuales se obtuvieron los resultados esperados.

Por último, se elaboraron instrumentos inspirados en las métricas para la calidad del diseño como Tamaño operacional de clase (TOC) y las Relaciones entre clases (RC), evaluándose varios factores claves como la reutilización, complejidad de implementación, entre otros.

Los resultados de las pruebas realizadas al software y la aplicación de las métricas en ambos casos descritos anteriormente dan lugar a que la implementación realizada presenta una calidad aceptable, siendo positiva la validación de la solución propuesta.

Conclusiones generales

Al culminar la investigación se le dio cumplimiento a los objetivos planteados, alcanzando el resultado propuesto, una herramienta para la ayuda en la toma de decisiones en la vista arquitectónica de sistema. Con diferentes funcionalidades para ayudar en la gestión de la información en el proyecto ERP CUBA y así ayudar a la hora de tomar decisiones. Reafirmando así la utilidad y la validez del empleo de las tecnologías informáticas para apoyar las labores en cualquier esfera de la vida. Lo anterior se ve demostrado a través de lo siguiente:

- ✓ Se analizaron los fundamentos teóricos, demostrando la necesidad del nuevo sistema.
- ✓ Se realizó un estudio acerca del modelo de desarrollo, tecnologías y herramientas definidas para obtener la nueva solución, reafirmando que eran las idóneas para ser utilizadas.
- ✓ Se logró obtener el diseño e implementación de una herramienta que permite gestionar la información referente a la arquitectura de sistema, que tributa a la toma de decisiones en el proyecto, favoreciendo la integridad de los datos a analizar.
- ✓ Se aplicaron las pruebas al sistema obteniéndose resultados positivos.

Recomendaciones

Teniendo como base los resultados de este trabajo y la experiencia adquirida durante el desarrollo del mismo, se recomienda:

- ✓ Continuar con la investigación para garantizar nuevas mejoras en futuras versiones del sistema.
- ✓ Agregar funcionalidades, como graficar toda la información que se gestiona.

Bibliografía

Bibliografía referenciada

1) **Almada, Federico. 2008.** Zend Studio for Eclipse, desarrollo profesional en PHP (http://www.techtear.com/2008/01/22/zend-studio-for-eclipse-desarrollo-profesional-en-php/). 2008.
2) **Alvarez, Miguel Angel. 2003.** *Editor web orientado a la programación de páginas PHP, con ayudas en la gestión de proyectos y depuración de código. (http://www.desarrolloweb.com/articulos/1178.php).* 2003.
3) **—.** *Qué es HTML. (http://www.desarrolloweb.com/articulos/que-es-html.html).*
4) **—.** *Qué es Javascript (http: //www.desarrolloweb.com/articulos/25.php.).*
5) **Badilla, Ricardo Montaño. 2010.** *Sistema ERP. Definición, funcionamiento, ventajas y desventajas(http://www.gestiopolis.com/administracion-estrategia/erp-definicion-funcionamiento-ventajas-desventajas.htm).* 2010.
6) **Barrios, Grupo ARQUISOFT - Johanna Rojas - Emilio. 2007.** *Métodos de prueba de caja blanca (http://www.udistrital.edu.co/comunidad/grupos/arquisoft/fileadmin/Estudiantes/Pruebas/HTML%20-%20Pruebas%20de%20software/node26.html).* 2007.
7) **Bartolomé, Jordi. 2007.** *Dossier sobre el test de software (http://www.tolaemon.com/otros/testcast3.htm).* 2007.
8) **Brown, A. W. and C.Wallnau. 1998.** *IEEE Software: s.n., 1998. 37-46...* 1998.
9) **Casal, Julio. 2009.** *Desarrollo de Software basado en Componentes. http://msdn.microsoft.com/es-es/library/bb972268.aspx.* 2009.
10) **Collado, Manuel. 2003.** *Pruebas de software(http://www.google.com.cu/url?sa=t&source=web&ct=res&cd=1&ved=0CAcQFjAA&url=http%3A%2F%2Flml.ls.fi.upm.es%2Fftp%2Fed2%2F0203%2FApuntes%2Fpruebas.ppt&rct=j&q=objetivo+de+las+pruebas+de+software&ei=F_DRS9SJNMHflgfCqoHvDA&usg=AFQjCNFkFpxs).* 2003.
11) **Ferreira, Matías Martínez. 2005.** *La toma de decisiones(http://www.gestiopolis.com/recursos4/docs/ger/todecisiones.htm).* 2005.
12) **Gómez, Diego. 2008.** *Cómo entender la Complejidad Ciclomática (http://www.dosideas.com/noticias/metodologias/320-como-entender-la-complejidad-ciclomatica.html).* 2008.

13) **Hinostroza, Raul Rodas. linuxcentro.net. 2007.** *http://www.linuxcentro.net/linux/staticpages/index.php?page=CaracteristicasPHP...* 2007.

14) **Ivar Jacobson, Grady Booch, James Rumbaugh. 2000.** *El proceso Unificado del desarrollo de software. Madrid: Addison Wesley, 2000. 84-7829-036-2.* 2000.

15) **Kramek, Andy. 2007.** *Patrones de diseño - La cadena de responsabilidad (http://www.portalfox.com/index.php?name=News&file=article&sid=2423).* 2007.

16) **—. 2007.** *Patrones de diseño - La estrategia (http://www.portalfox.com/index.php?name=News&file=article&sid=2384).* 2007.

17) **Larman, Craig. 2004.** *UML y Patrones. La Habana: Félix Varela, 2004.* 2004.

18) **Marzo, Josep Vilalta. 2004.** *Criterios de selección de una herramienta CASE - UML(http://www.vico.org/aRecursosPrivats/UML_TRAD/talleres/mapas/UMLTRAD_101A/LinkedDocuments/SeleccionCASE_vvc.pdf).* 2004.

19) **Medina, Cesar Julio Bustacara. 2008.** *Lenguajes de Descripción de Arquitectura (ADL)[http://sophia.javeriana.edu.co/~cbustaca/Arquitectura%20Software/Clases/Conceptos/Presentaciones/arquitecturas_software_ADLs.pdf].* 2008.

20) **Ortiz, Karen. Programación. 2007.** *http://karen_ortiz.nireblog.com/post/2008/05/07/lenguaje-de-programacion.* 2007.

21) **Pecos, Daniel. netpecos.org. 2003.** *http://www.netpecos.org/docs/mysql_postgres/index.html...* 2003.

22) **Pressman, Roger S. 2002.** *Ingeniería de Software, un enfoque práctico. Quinta edición. s.l.: McGraw-Hill Companies, pág. 640. ISBN: 8448132149.* 2002.

23) **Pressman. 2001.** *Ingeniería del software Un enfoque práctico. s.l.: Mcgraw-hill, 2001. 8448132149.* 2001.

24) **Reynoso, Carlos Billy. 2004.** *Introducción a la Arquitectura de Software (http://www.willydev.net/descargas/prev/IntroArq.pdf).* 2004.

25) **Reynoso, Carlos y Kicillof, Nicolás. 2004.** *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft. UNIVERSIDAD DE BUENOS AIRES.* 2004.

26) **Reynoso, Nicolás Kicillof y Carlos. 2004.** *De Lenguajes de descripción arquitectónica de Software (ADL). (http://www.willydev.net/descargas/prev/ADL.pdf).* 2004.

27) **Sierra, María. 2005.** *Trabajando con Visual Paradigm for UML(http://personales.unican.es/ruizfr/is1/doc/lab/01/is1-p01-trans.pdf).* 2005.

28) **Valdés, Damián Pérez. Los diferentes lenguajes de programación para la web. Maestros del Web. 2007.** *http://www.maestrosdelweb.com/principiantes/los-diferentes-lenguajes-de-programacion-para-la-web/.* 2007.

Bibliografía consultada

1) **ARREGUI, J. J. (Septiembre 2005.).** Revisión Sistemática de Métricas de Diseño Orientado a Objetos. Madrid, España.: Universidad Politécnica de Madrid, Facultad de Informática.
2) **Daedalus. (2006).** Diseño de sistemas. http://www.daedalus.es/AreasISDiseno-E.php.
3) **Equipo Arquitectura del ERP Cedrux. (2008).** Especificación Técnica para el marco de la arquitectura.
4) **Garlan, D. S. (1994).** An Introduction to Software Architecture.
5) **Informáticas, Instituto Nacional de tecnologías. Metodologías Informáticas. 1997.** *http://www.ongei.gob.pe/publica/metodologias/Lib5083/INDEX.HTM.* 1997.
6) **Microsoft. (2007).** Diseño del software.
7) **Moreno, G. A. (2006, abril 25).** L.S.I. Departamento de lenguajes y sistemas informáticos. Universidad de Granada. Departamento de lenguajes y sistemas informáticos. Universidad de Grana*da.: http://lsi.ugr.es*
8) **Paradigm, International Visual. 2005.** *http://www.visual-paradigm.com/product/vpuml/communityedition.jsp.*
9) **Pressman, R. (2000). In R. Pressman, Ingeniería del software: Un enfoque práctico. McGraw Hill.**
10) **Luis Giraldo, Yuliana Zapata. 2005.** HERRAMIENTAS DE DESARROLLO DE INGENIERIA DE SW PARA LINUX.
11) **tufunción. 2007.** *Zend Studio (http://www.tufuncion.com/zend-studio).*
12) **Zend.** Evaluando Zend studio (http://www.maestrosdelweb.com/editorial/zendstudio/).

Anexos

Anexo 1 Instrumento de medición de la métrica Tamaño operacional de clase (TOC)

Tabla 6 Rango de valores de para la evaluación técnica de los atributos de calidad (Responsabilidad, Complejidad de Implementación y Reutilización) relacionados con la métrica Tamaño operacional de clase.

	Categoria	Criterio
Responsabilidad	Baja	< =Prom.
	Media	Entre Prom. y 2* Pom.
	Alta	> 2* Prom.
Complejidad implementación	Baja	< =Prom.
	Media	Entre Prom. y 2* Pom.
	Alta	> 2* Prom.
Reutilización	Baja	> 2*Prom.
	Media	Entre Prom. y 2* Pom.
	Alta	<= Prom.

Tabla 7 Resultados de la evaluación de la métrica Tamaño operacional de clase y su influencia en los atributos de calidad (Responsabilidad, Complejidad de implementación y Reutilización).

No	Componente	Clase	Cantidad de Proce	Responsabilidad	Complejidad	Reutilización
1	Componente	ComponenteController	16	Media	Media	Media
2	Componente	CrearComponenteController	4	Baja	Baja	Alta
3	Componente	DependenciaController	4	Baja	Baja	Alta
4	Componente	ExportarController	5	Baja	Baja	Alta
5	Componente	ModificarComponenteController	5	Baja	Baja	Alta
6	Componente	TipoComponenteController	6	Baja	Baja	Alta
7	Componente	ComponenteModel	42	Alta	Alta	Baja
8	Componente	ExportarModel	2	Baja	Baja	Alta
9	Componente	GeckoXML	10	Baja	Baja	Alta
10	File Up Load	FileUpLoadController	5	Baja	Baja	Alta
11	File Up Load	FileUpLoadInternoController	5	Baja	Baja	Alta
12	File Up Load	FileUpModel	20	Media	Media	Media
13	File Up Load	xmlParseModel	4	Baja	Baja	Alta
14	Principal	PrincipalController	26	Alta	Alta	Baja
15	Principal	PrincipalModel	18	Media	Media	Media
16	Principal	xmlParseModel	4	Baja	Baja	Alta
17	Requisito	RequisitosController	5	Baja	Baja	Alta
18	Requisito	RequisitosModel	6	Baja	Baja	Alta
19	Requisito	CrearSubSistemaController	3	Baja	Baja	Alta
20	SubSistema	ExportarController	5	Baja	Baja	Alta
21	SubSistema	ModificarSubSistemaController	4	Baja	Baja	Alta
22	SubSistema	SubSistemaController	18	Media	Media	Media
23	SubSistema	SubSistemaModel	34	Alta	Alta	Baja

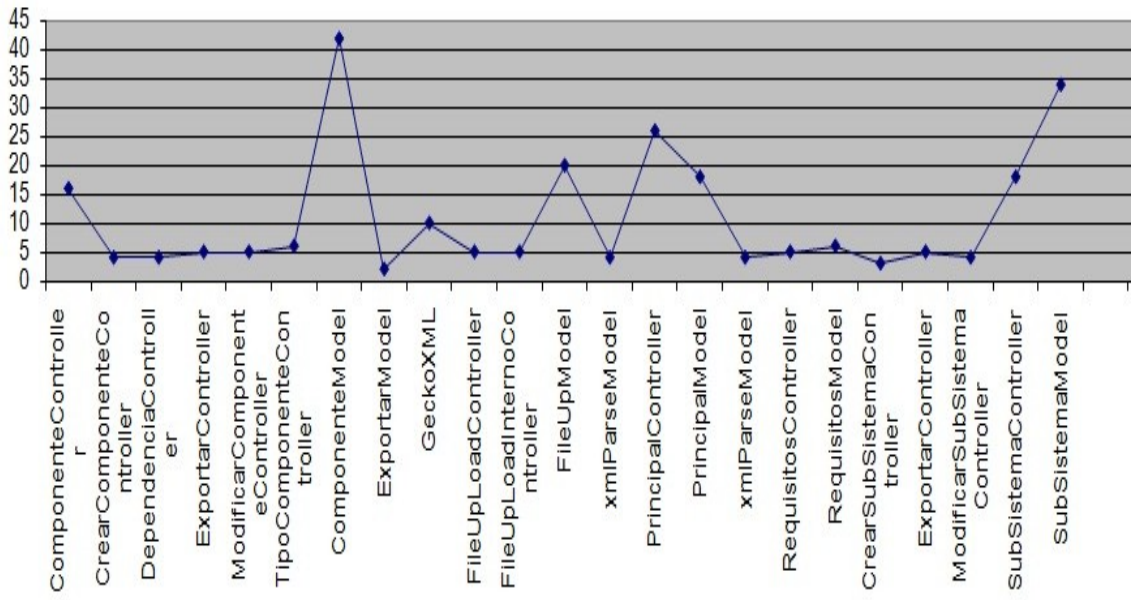


Figura 27 Gráfica de los resultados de la evaluación de la métrica Tamaño operacional de clase y su influencia en los atributos de calidad (Responsabilidad, Complejidad de implementación y Reutilización).

Anexo 2 Instrumento de medición de la métrica Relaciones entre clases (RC)

Tabla 8 Rango de valores de para la evaluación técnica de los atributos de calidad (Acoplamiento, Complejidad de mantenimiento, Reutilización y Cantidad de pruebas) relacionados con la métrica Relaciones entre clases.

	Categoria	Criterio
Acoplamiento	Ninguno	0
	Bajo	1
	Medio	2
	Alto	>2

	Categoria	Criterio
Complejidad Mant.	Baja	<= Prom.
	Media	Entre Prom. y 2*Prom.
	Alta	> 2*Prom.

	Categoria	Criterio
Reutilización	Baja	>2* Prom.
	Media	Entre Prom. y 2*Prom.
	Alta	<= Prom.

	Categoria	Criterio
Cantidad de Pruebas	Baja	<= Prom.
	Media	Entre Prom. y 2*Prom.
	Alta	> 2*Prom.

Tabla 9 Resultados de la evaluación de la métrica Relaciones entre clases y su influencia en los atributos de calidad (Acoplamiento, Complejidad de mantenimiento, Reutilización y Cantidad de pruebas).

No	Subsistema	Clase	Cantidad de Relaciones d	Acoplamiento	Complejidad Mant.	Reutilización	Cantidad de Pruebas
1	Componente	ComponenteController	0	Ninguna	Baja	Alta	Baja
2	Componente	CrearComponenteController	0	Ninguna	Baja	Alta	Baja
3	Componente	DependenciaController	0	Ninguna	Baja	Alta	Baja
4	Componente	ExportarController	0	Ninguna	Baja	Alta	Baja
5	Componente	ModificarComponenteController	0	Ninguna	Baja	Alta	Baja
6	Componente	TipoComponenteController	0	Ninguna	Baja	Alta	Baja
7	Componente	ComponenteModel	3	Alta	Alta	Baja	Alta
8	Componente	ExportarModel	5	Alta	Alta	Baja	Alta
9	Componente	GeckoXML	1	Baja	Baja	Alta	Baja
10	File Up Load	FileUpLoadController	0	Ninguna	Baja	Alta	Baja
11	File Up Load	FileUpLoadInternoController	0	Ninguna	Baja	Alta	Baja
12	File Up Load	FileUpModel	6	Alta	Alta	Baja	Alta
13	File Up Load	xmlParseModel	0	Ninguna	Baja	Alta	Baja
14	Principal	PrincipalController	0	Ninguna	Baja	Alta	Baja
	Principal	PrincipalModel	5	Alta	Alta	Baja	Alta
	Principal	xmlParseModel	0	Ninguna	Baja	Alta	Baja
	Requisito	RequisitosController	0	Ninguna	Baja	Alta	Baja
	Requisito	RequisitosModel	0	Ninguna	Baja	Alta	Baja
	Requisito	CrearSubSistemaController	1	Baja	Baja	Alta	Baja
	SubSistema	ExportarController	0	Ninguna	Baja	Alta	Baja
	SubSistema	ModificarSubSistemaController	0	Ninguna	Baja	Alta	Baja
	SubSistema	SubSistemaController	0	Ninguna	Baja	Alta	Baja
	SubSistema	SubSistemaModel	6	Alta	Alta	Baja	Alta

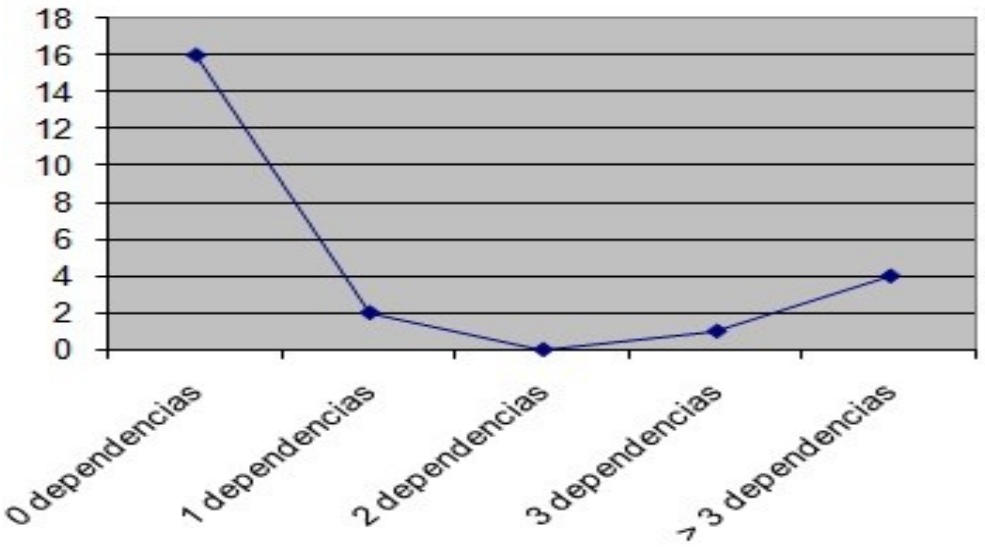
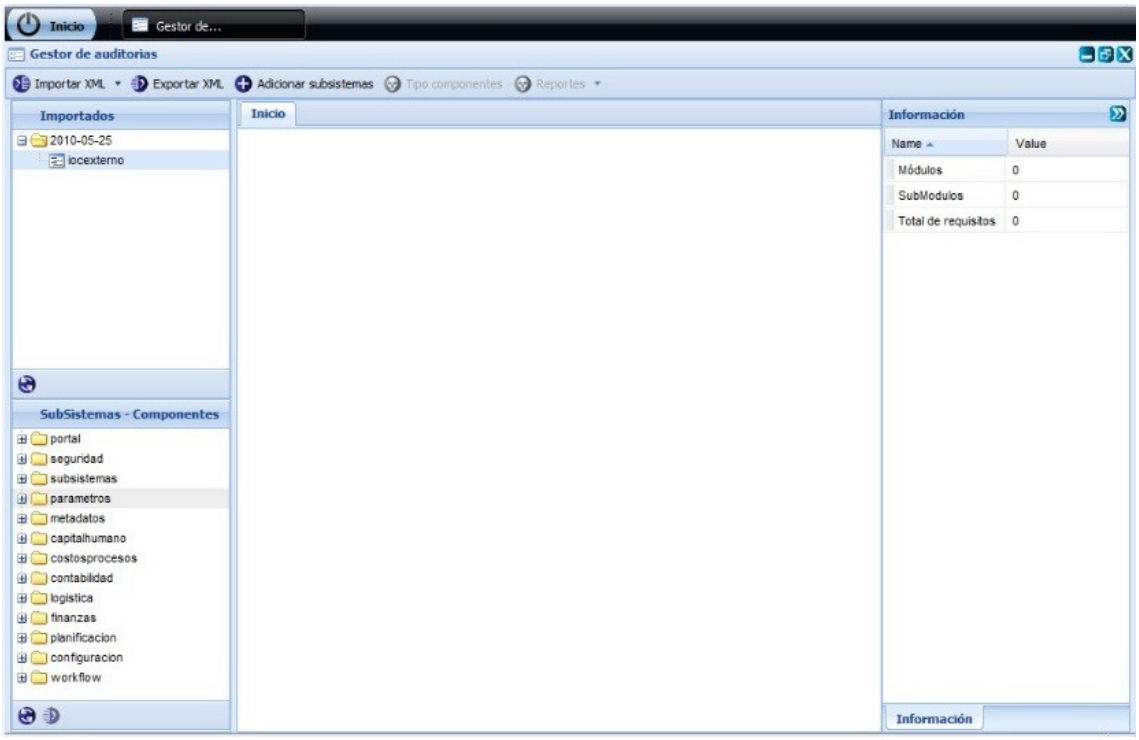
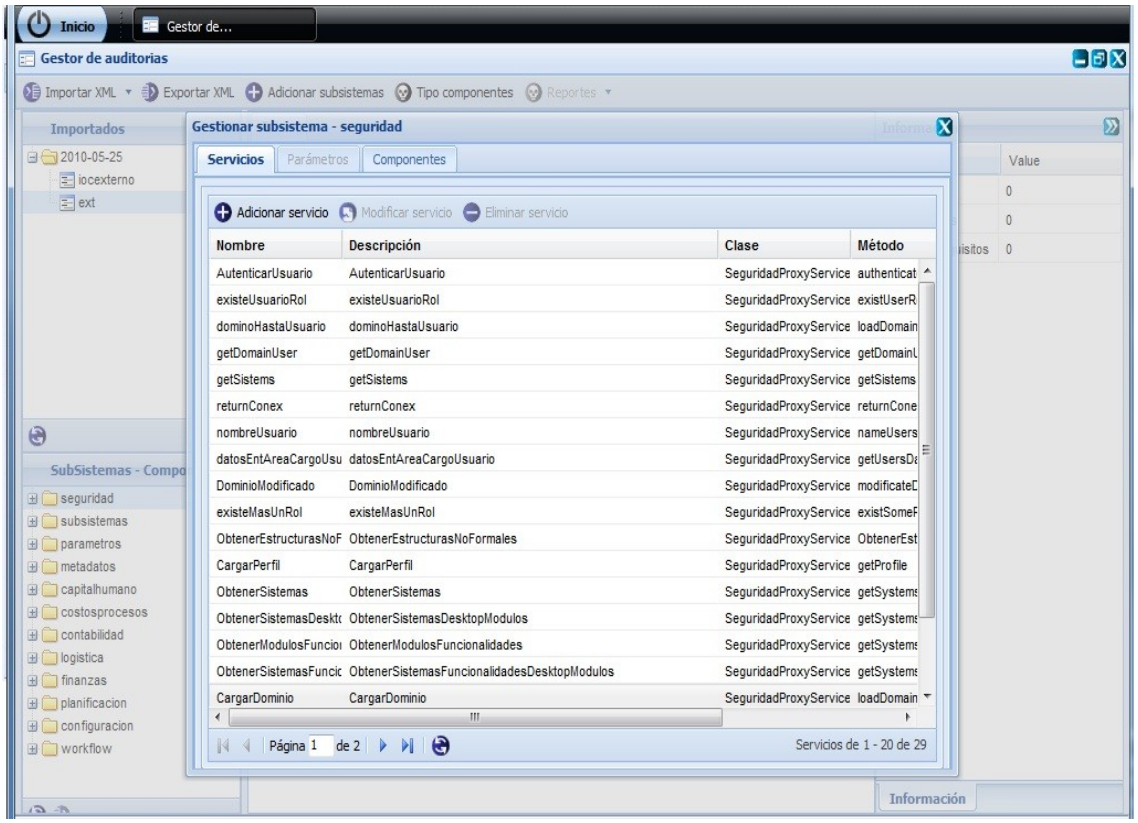


Figura 28 Gráfica de los resultados de la evaluación de la métrica Relaciones entre clases agrupados por la tendencia de los valores.

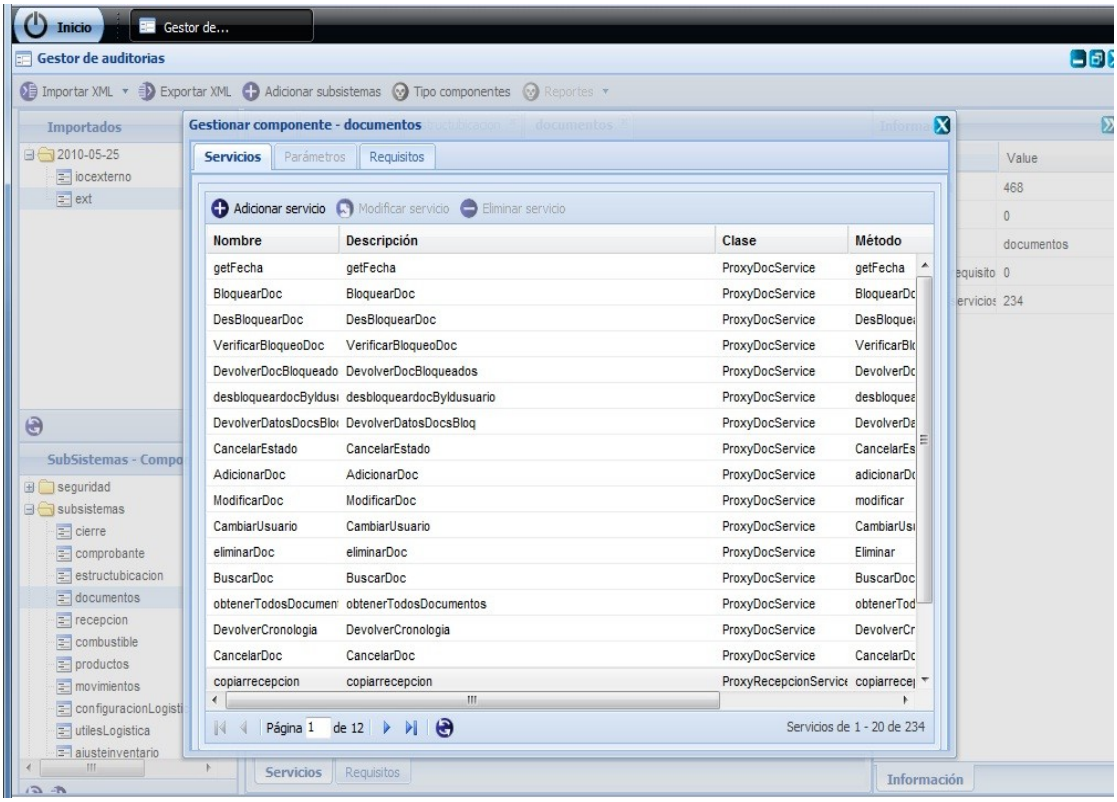
Anexo 3 Interfaz principal



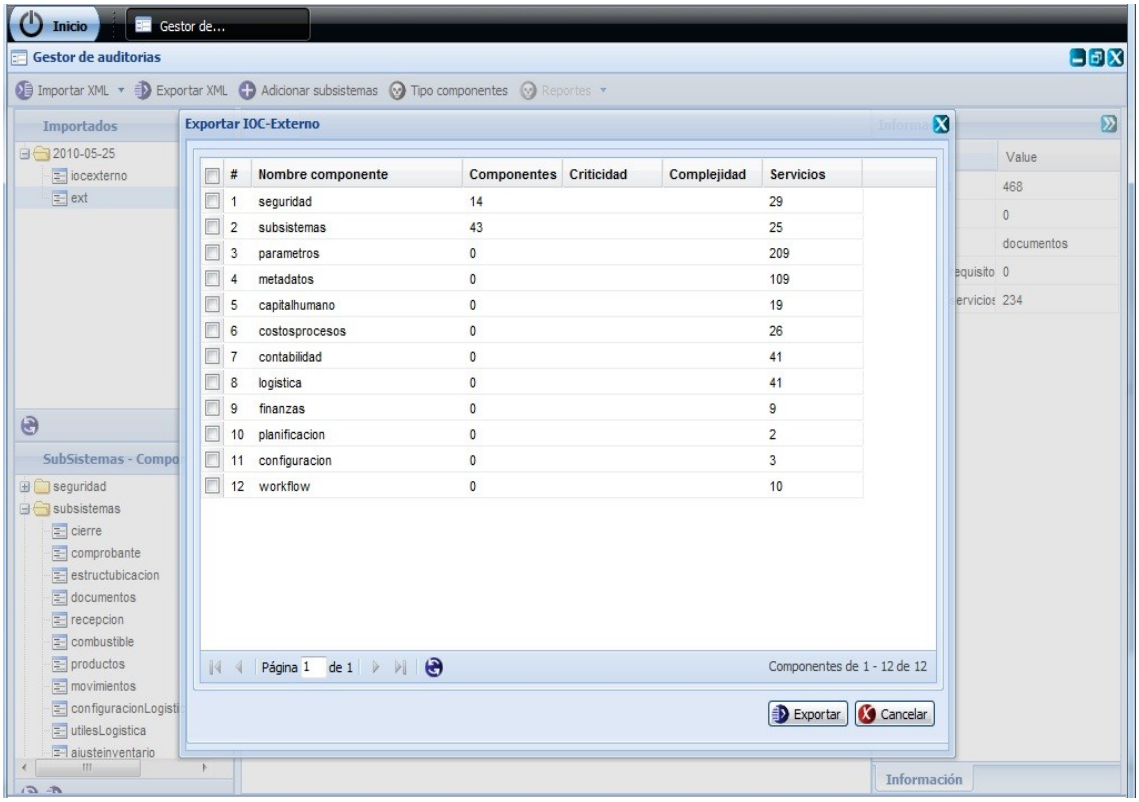
Anexo 4 Interfaz gestionar subsistema



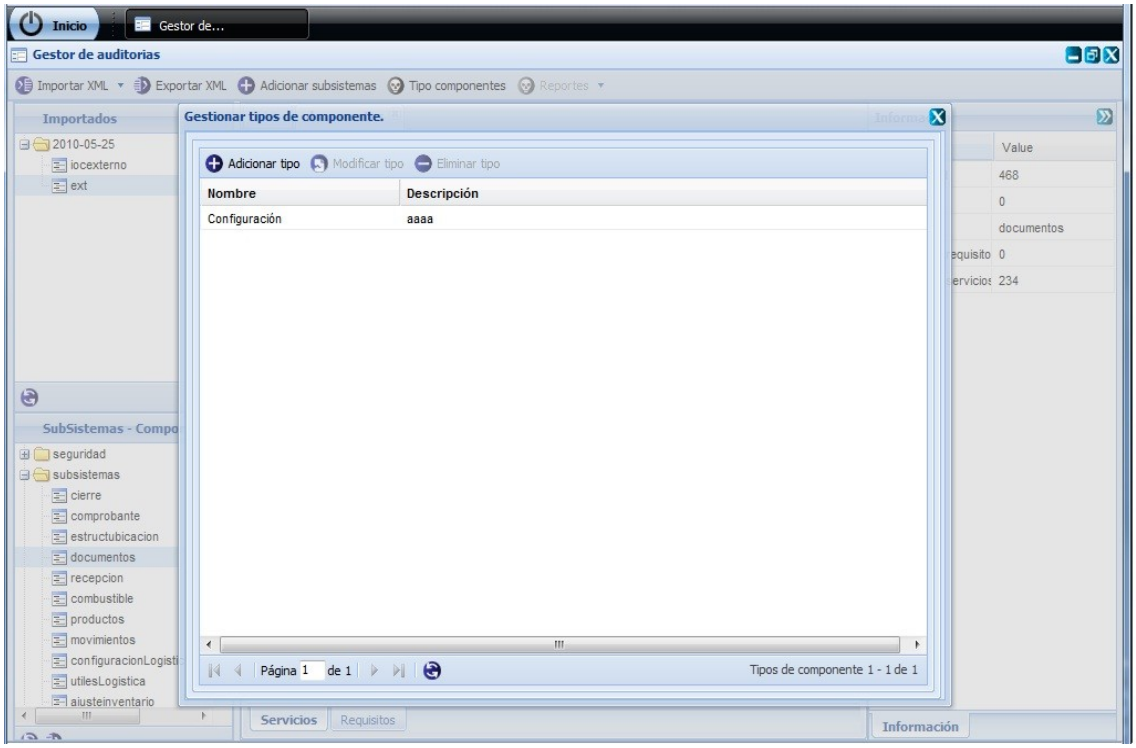
Anexo 5 Interfaz gestionar componente



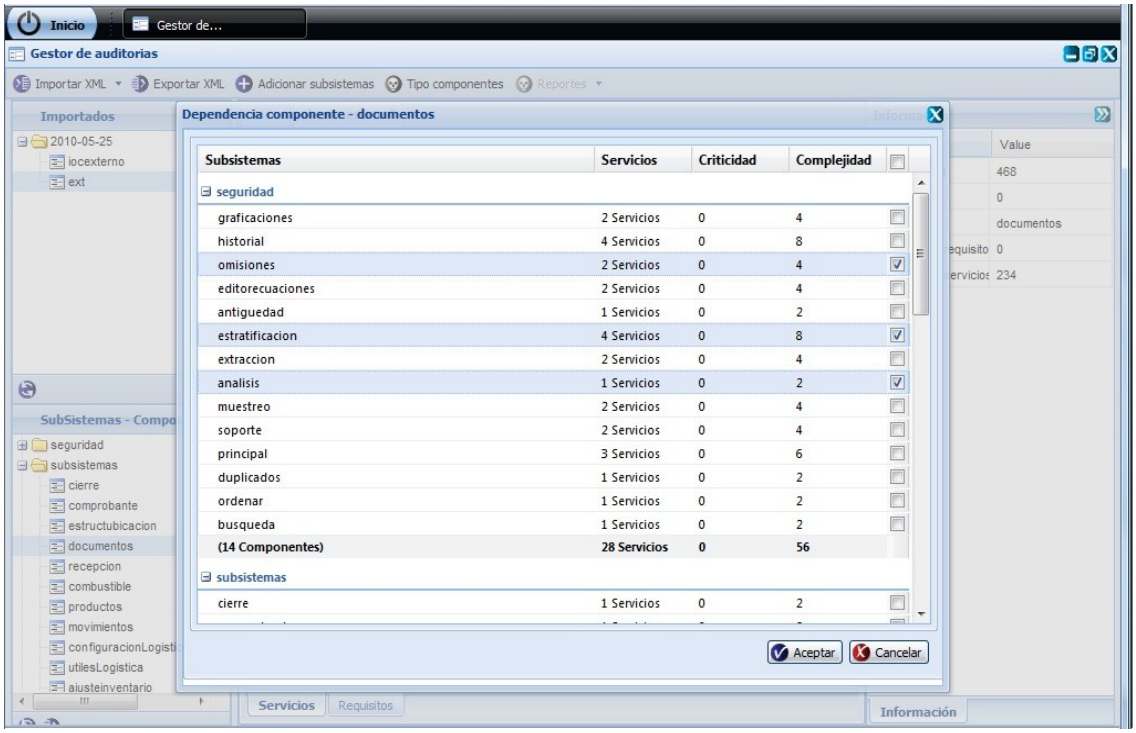
Anexo 6 Interfaz exportar ioc externo



Anexo 7 Interfaz gestionar tipo de componente



Anexo 8 Interfaz dependencia componente



Anexo 9 Interfaz importar xml ioc