Universidad de las Ciencias Informáticas
Facultad 3

Título: Diseño e implementación del proceso Ordinario de la Materia Civil de los Tribunales Populares Cubanos.

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

Autor:
Reinier García Espino

Gabriel Ángel Pérez Carballo

Tutor:
Ing. Elsydania López Guerra

Ciudad de la Habana, Cuba

Curso: 2010-2011

Declaración de Autoría

Declaramos que somos los únicos autores de este trabajo y autorizamos a la Facultad 3 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

Autores:

_____ _____

Reinier García Espino Gabriel Ángel Pérez Carballo

Tutor:

_____

Ing. Elsydania López Guerra

Agradecimientos

Quiero agradecer:

A mis padres y a mi hermano por su apoyo incondicional, su amor, su comprensión, su dedicación hacia mi persona y su orgullo de mí.

A mi prima Ailín por ser mi ejemplo a seguir.

A mis tíos María y Andrés por estar siempre pendientes de mi trayectoria.

A mi novia Lisbeth por su apoyo, su amor y su presencia incondicional en toda esta etapa de mi vida.

A mis amigos y amigas tanto de la universidad como de Cienfuegos, por su ayuda y su sincera amistad, a Yiliam, Leinis, Yoandra, Yuleisy, Yaidel, Tony, Buty, Adrian, Doannis, Rances, Yasmany, Roberto, Hector Luis, Andris, Reinier, a los del apartamento 9109 y a todos mis compañeros de grupo.

A todos aquellos que han aportado un granito de arena en mi formación.

A mi tutora Elsydania por la ayuda y la dedicación brindada.

A todos los que siempre han confiado en mí y en lo que puedo hacer.

A todos aquellos con los que he vivido excelentes momentos.

A todos, gracias.

Reinier

Quiero agradecer:

A mis abuelos, más que eso, mis segundos padres.

A mis padres por ser personas muy especiales para mí, cada uno a su forma.

A mi novia por brindarme su apoyo a cada paso de mi vida, por formar parte de esta locura a la que llamamos amor.

A mis tíos y primos que aunque distantes siempre me han brindado su apoyo incondicional.

A la decana Yvonne Caridad y a Maydelín Rodríguez Gómez por darme varias oportunidades cuando más las necesite.

A mis compañeros de cuarto y compañeros de aula muchas gracias por siempre estar ahí.

A todos los que han creído en mí y me han apoyado cuando los baches de la vida me han hecho tropezar y levantarme con más fuerza. Muchas gracias por ser tan especiales.

Gabriel

Dedicatoria

A mis padres por ser mi luz y mi guía en momentos de inseguridad, por su confianza en mí, por su excelente educación y por su amor incondicional.

A mi hermano por demostrarme que todo está en la voluntad con que hagas las cosas.

A toda mi familia por la educación, la paciencia, la confianza y el ejemplo que siempre me han dado.

A nuestro Comandante en Jefe Fidel Castro Ruz, por ser nuestro guía y nuestra luz e iniciador de la Universidad de las Ciencias Informáticas.

Reinier

A esos ángeles guardianes que me han cuidado durante toda mi vida, mis abuelos maternos, que si hoy he podido llegar hasta aquí es gracias a ellos. Para ellos con todo mi cariño.

Gabriel

Resumen

Los Tribunales Populares Cubanos (TPC) están estructurados en 3 instancias: Municipal, Provincial y Suprema. En cada una de ellas se contemplan las materias: Civil, Penal, Laboral, Económica y Administrativa, y a su vez en cada una de ellas se trabajan varios procesos. Actualmente en los tribunales todas las actividades se realizan de forma manual, lo que provoca ralentización en la tramitación de los procesos, y por consiguiente el incumplimiento de los términos procesales. Debido a esta situación se decidió iniciar la construcción de un sistema, el cual estará integrado por varios subsistemas entre los que se encuentra el Civil. Para iniciar el desarrollo de este se identificaron y aprobaron un conjunto de requisitos para automatizar la gestión del proceso Ordinario en la instancia municipal.

En el presente trabajo se realizó un estudio de la arquitectura definida para la construcción del software, así como de las tecnologías y la metodología a utilizar. Además, se realizó el diseño a partir de los casos de uso identificados para informatizar la tramitación del proceso Ordinario de la Materia Civil. Finalmente se procedió a la implementación de las funcionalidades especificadas, y se aplicaron pruebas de caja blanca y de caja negra para verificar que el sistema construido realmente satisfacía los requisitos establecidos.

Con esta solución, los Tribunales Municipales Cubanos contarán con una herramienta que permitirá agilizar y modernizar la tramitación del proceso Civil.

Índice de contenido

Índice de figuras

Índice de tablas

Introducción

En el mundo actual la sociedad está caracterizada por ser principalmente una sociedad donde las tecnologías de la información han llegado a ser la figura representativa de la cultura, hasta el punto que para designar su marco de convivencia se alude reiteradamente a la expresión sociedad de la Información, ya que las Tecnologías de la Información y de la Comunicaciones (TIC) están presentes en los ámbitos político, social y económico. En los últimos años se ha visto una incesante y creciente aplicación de las nuevas tecnologías en todos los ámbitos posibles de la cotidianidad en la sociedad. Los tribunales aunque son caracterizados por mantener su tradición conservadora no pretenden quedar excluidos de este gran universo de desarrollo tecnológico.

En Cuba también se están dando pasos para facilitar, agilizar y organizar los procesos judiciales a través de una justicia tecnológicamente avanzada. La informática es hoy, una vía de desarrollo social y económico, en la cual la industria de software desempeña un papel primordial.
Los tribunales en Cuba están estructurados en 3 instancias: Municipal, Provincial y Supremo. En cada una de ellas se contemplan diferentes materias: Civil, Penal, Laboral, Económica y Administrativa, y a su vez cada una de estas prevé varios tipos de procesos. La materia Civil en los Tribunales Municipales Populares (TMP) tramita:

- Las demandas de contenido económico cuya cuantía, o el valor de los bienes sobre los que se litigue, no exceda de diez mil pesos.
- Los procesos sobre el estado civil de las personas y los que se susciten por la aplicación del Código de Familia, salvo los procesos de nulidad de matrimonio y los de privación o suspensión del ejercicio de la patria potestad.
- Las reclamaciones sobre alimentos.
- Los actos de jurisdicción voluntaria que no sean en negocios de comercio.
- Los procesos sucesorios.
- Los procesos de amparo fuera de actuaciones judiciales contra actos provenientes de particulares o de autoridades administrativas y los de suspensión de obra nueva.

Actualmente entre los principales problemas presentes en los tribunales de Cuba se encuentra la ralentización de la tramitación de los procesos jurídicos. Esto se debe mayormente a que todas las actividades se realizan de forma manual. Los jueces deben elaborar manualmente las resoluciones de todos los expedientes que tienen a su cargo, lo que afecta el cumplimiento de los términos

establecidos y su control. De la misma forma, los secretarios tienen que confeccionar documentos manualmente informando sobre el recibimiento de escritos, despachos, cédulas de emplazamiento, citaciones, entre otros, por lo que muchos documentos son generados varias veces. Toda esta situación provoca que no sea posible obtener información estadística actualizada de forma rápida, y totalmente fiel a la realidad. Los expedientes se guardan en los archivos físicos de cada tribunal, lo que ocasiona el deterioro de estos a causa de la humedad y por el paso del tiempo. Las búsquedas en los Libros de Sentencias, Libros de presentación de escritos y Libros de Radicación se torna complicada por el volumen de información de estos, y porque tienden a estropearse por la considerable manipulación.

En pos del proceso de informatización que ha estado realizando el país y en aras de lograr mejorar la tramitación de los procesos judiciales, se concibió en el año 2009 el Proyecto de Informatización de Tribunales en colaboración con la Universidad de las Ciencias Informáticas (UCI), con el objetivo de crear un sistema informático que permita estandarizar los procesos y documentos jurídicos, que ejecute y supervise los procesos de manera auténtica conforme a las disposiciones legales que los regulan, que genere reportes estadísticos con mayor rapidez y facilidad y que permita la identificación única de los expedientes judiciales.

El sistema en desarrollo se nombra Sistema de Informatización de Tribunales (SIT) y está integrado por un conjunto de subsistemas, entre los que se encuentra el subsistema Civil. Para lograr desarrollar el sistema se identificaron y aprobaron un conjunto de requisitos en coordinación con el cliente, los cuales representan las características y condiciones que debe cumplir el sistema. Durante la implementación se generan artefactos necesarios que deben cumplir con los requisitos definidos, algunos de estos artefactos son el diagrama de clases, diagrama de secuencia, de componentes y diagramas de despliegue.

Teniendo en cuenta lo expuesto anteriormente se plantea como **problema de investigación:**
¿Cuál es la especificación de los artefactos necesarios para la informatización de los requisitos identificados para el proceso Ordinario de la materia Civil?

En consecuencia, el **Objeto de estudio es:** Proceso de desarrollo de software para sistemas jurídicos.

El **Objetivo general** que se plantea es**:** Realizar el diseño y la implementación de los requisitos identificados para el proceso Ordinario de la materia Civil de los Tribunales Municipales Cubanos para obtener un producto funcional.

El **Campo de acción** en el cual se enmarca esta investigación es**:** Diseño e implementación de sistemas para la gestión de procesos judiciales.

A partir de lo expuesto se plantea la siguiente **hipótesis:**

Si se realiza el diseño y la implementación de los requisitos identificados para el proceso Ordinario de la materia Civil de los Tribunales Populares Cubanos, entonces se obtendrá la especificación de los artefactos necesarios para la informatización.

Para darle cumplimiento al objetivo general se han derivado un conjunto de **Objetivos específicos**, siendo estos los siguientes:

- Elaborar el marco teórico de la investigación para obtener un estudio de las principales herramientas y mecanismos para desarrollar sistemas web.
- Realizar el Modelo de diseño para transformar los requisitos identificados en una representación técnica del software que se va a desarrollar.
- Realizar el Modelo de Implementación para implementar las clases de diseño y obtener un producto funcional.
- Validar la solución propuesta para evaluar la calidad del trabajo realizado, corregir errores y asegurar que el producto obtenido satisface los requisitos acordados.

Con el propósito de darle respuesta a los objetivos específicos se definieron las siguientes **tareas de investigación:**

1. Conceptualización del proceso Ordinario de la Materia Civil.
2. Fundamentación de la arquitectura definida para la implementación de la solución informática.
3. Fundamentación de los Patrones de Diseño seleccionados para la solución propuesta.
4. Fundamentación de las tecnologías y herramientas seleccionadas para el desarrollo de la solución propuesta.
5. Elaboración de diagramas de Diseño, Secuencia, Despliegue y Componentes.
6. Implementación de los casos de uso definidos para darle solución a la problemática.
7. Diseño de casos de prueba.
8. Validación de las funcionalidades implementadas.

Para dar cumplimiento a las tareas propuestas, se emplearon métodos científicos de investigación (Teóricos y Empíricos). Para una buena elaboración de este trabajo se utilizaron los siguientes métodos:

Métodos Teóricos: Permitieron estudiar las características del objeto de estudio que no fueron observables directamente.

Histórico-lógico: En la primera parte de la investigación se desarrolló un estudio del estado del arte de la problemática y se analizó las ventajas y desventajas de cada una de las herramientas a utilizar en la aplicación.

Analítico-Sintético: Permitió la realización de un estudio teórico de la investigación y un análisis previo sobre el funcionamiento del proceso Ordinario de la Materia Civil y así extraer los elementos más importantes relacionados con el objeto de estudio.

Modelación: Para la creación del modelo de diseño e implementación del Proceso Ordinario de la Materia Civil que se lleva a cabo en los Tribunales Municipales Cubanos.

El presente trabajo de investigación está estructurado en tres capítulos, además contiene varios anexos con los artefactos generados durante el desarrollo. A continuación se describe el objetivo principal de cada uno de los capítulos:

Capítulo 1: Fundamentación Teórica. En este capítulo se abarca toda la fundamentación teórica que sustenta el progreso de la investigación para el desarrollo del sistema propuesto. Se realiza el estudio del estado del arte de la metodología, herramientas y patrones que se utilizaron para el desarrollo de la aplicación ya definidos por la arquitectura del proyecto Tribunales Populares Cubanos.

Capítulo 2: Solución Propuesta. En este capítulo se propone la solución técnica de la investigación. Se presenta el modelo de diseño conformado por los diagramas de clases y los Diagramas de Secuencia (Diagramas de Interacción) y además, el Modelo de Implementación con los Diagramas de Componentes y el Diagrama de Despliegue. Se exponen los patrones de diseño utilizados en la solución propuesta.

Capítulo 3: Análisis de Resultados. En este capítulo se evalúa el grado de calidad y fiabilidad de los resultados obtenidos con el desarrollo de este trabajo, mediante la aplicación de algunas de las métricas empleadas internacionalmente para tal fin, específicamente las asociadas a la medición de la calidad del diseño y la implementación de software.

Capítulo 1: Fundamentación Teórica

Introducción

En el presente capítulo serán abordados una serie de definiciones y conceptos que servirán de base para un mayor entendimiento del Proceso Ordinario de la Materia Civil de los Tribunales Populares Cubanos. Además, se describen las principales herramientas, plataformas y metodología propuestas para el desarrollo, los patrones que se utilizaron, así como el Marco de Trabajo (framework) sobre el que se estará desarrollando el sistema. Es de vital importancia aclarar que para el desarrollo del sistema se siguieron los lineamientos arquitectónicos establecidos por la dirección del proyecto, que plantean el uso de un modelo estandarizado, y la definición clara y precisa de las responsabilidades de cada uno de los roles que se ven involucrados en el desarrollo de la solución.

1.1 Proceso Ordinario de la Materia Civil.

El Proceso Ordinario de la Materia Civil comienza cuando un abogado en representación de una persona natural, o una persona natural dirigida por un abogado, presenta una demanda ante el Tribunal Municipal. En el tribunal la secretaria recibe y registra dicha demanda y los documentos correspondientes que la acompañan y justifican. Seguidamente, la secretaria informa al juez sobre la demanda presentada y le hace entrega de esta con los documentos presentados. El juez ponente revisa la demanda, y se pronuncia sobre esta, disponiendo su admisión, reparo o denegación. Posteriormente, en caso de que la demanda sea admitida, se informa al demandado para que conteste la demanda y se presente al proceso. A continuación comenzará un período de alegaciones, conocido en el ámbito jurídico como réplica y dúplica. Luego el proceso es abierto a la fase de Pruebas, donde las partes tendrán la posibilidad de presentar las pruebas que estimen conveniente para la aclaración del caso, las cuales serán practicadas en el período de 30 días hábiles. Una vez vencido el período para practicar pruebas, se les concede un término a las partes para solicitar Vista. Finalmente, el tribunal pone fin al proceso dictando sentencia o un auto definitivo. Si alguna de las partes no está de acuerdo, podrá solicitar recurso de Apelación para que el caso sea atendido en una instancia superior, en este caso en los Tribunales Provinciales.

1.2 Metodología de Desarrollo de Software

Las Metodologías de Desarrollo de Software tienen como objetivo presentar un conjunto de técnicas tradicionales y modernas de modelado de sistemas que permitan desarrollar software de calidad, incluyendo heurísticas de construcción y criterios de comparación de modelos de sistemas.

Se habla de detalles organizativos, de un "estilo" de hacer las cosas. Pero siguiendo un poco más allá de un simple estilo, formalizando ese "estilo" añadiendo algo de rigurosidad y normas se obtiene una metodología (RUP, 2010).

La metodología escogida por parte de la dirección del proyecto es el Proceso Unificado Racional (Rational Unified Process - RUP). Se considera la más apropiada porque es un proceso bien definido y gestionado, ideal para construir sistemas complejos y posibilita generar documentación exhaustiva de todo el proceso. Además, posibilita mitigar los riesgos en etapas tempranas y constituye una de las metodologías más utilizadas en la construcción de sistemas orientados a objetos. Se considera además, que resulta más adaptable para los proyectos a largo plazo que necesiten un desarrollo iterativo capaz de mantener un buen control sobre los cambios y que facilite sobre todo el trabajo a distancia con los clientes, aplicando los criterios de flexibilidad que ofrece, robustez en la gestión del proyecto y los cambios, la cantidad de artefactos que se generan en forma de entregables al cliente y la adaptabilidad ante un sistema de gran tamaño. (Montané Izaguirre, 2010)
Además, consta de referencias con buenos resultados en su aplicación y está incorporada a los programas de estudio de la UCI, por lo que resulta familiar a todos los miembros del proyecto.

En la Figura 1 se muestran las distintas disciplinas (flujos de trabajos) y fases por las que está compuesta la metodología RUP.

Para el desarrollo del presente trabajo se utilizaron 3 de los flujos de trabajos con los que cuenta RUP: Diseño, Implementación y Prueba, los cuales tienen su máximo desarrollo en las fases de Elaboración y Construcción.

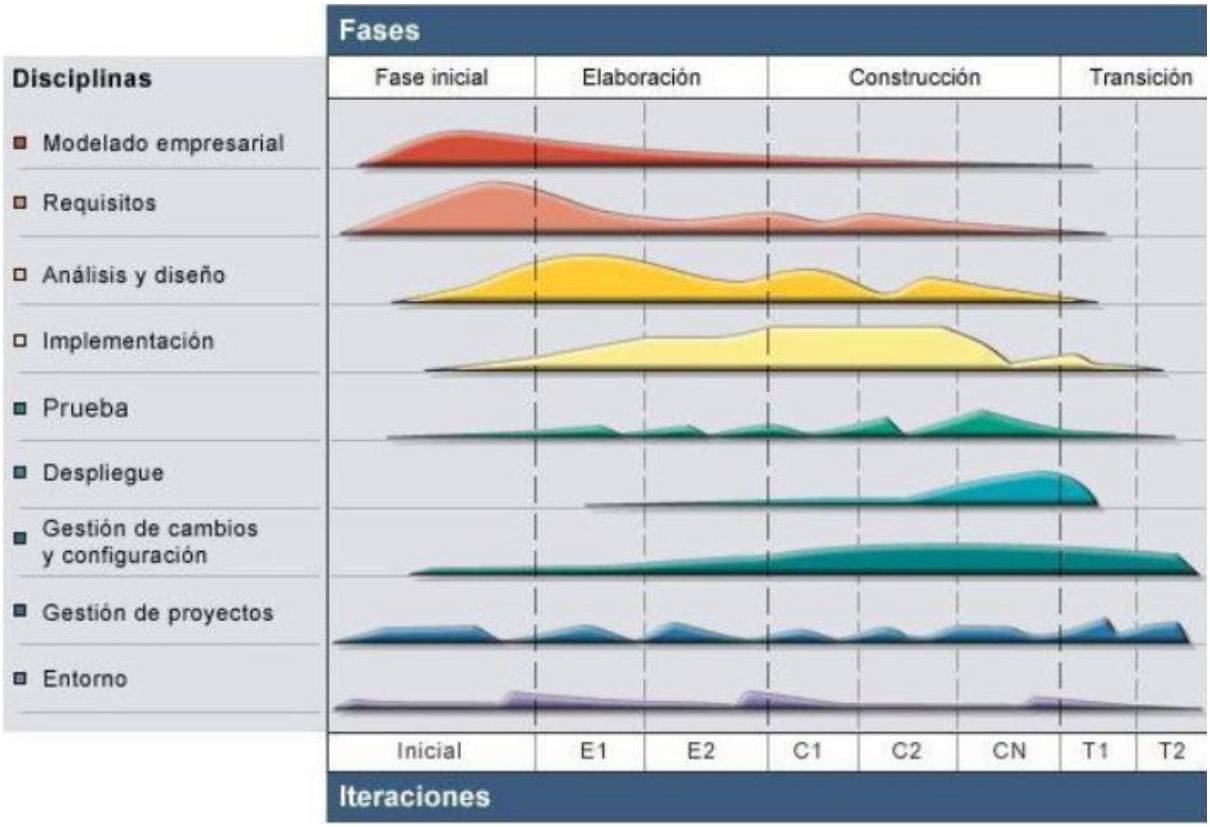

Figura 1: Fases de RUP (RUP, 2010)

1.2.1 Lenguaje de modelado UML

Un lenguaje de modelado es un conjunto estandarizado de símbolos y de modos de disponerlos para modelar un software. En la mayoría de los casos son utilizados en combinación con una metodología de desarrollo de software para realizar la especificación del desarrollo de un software y de este modo hacerlo extensivo a todo el equipo de desarrollo. El uso de un lenguaje de modelado es más sencillo que la auténtica programación. Por su robustez y fiabilidad la dirección del proyecto decidió que se utilizaría el lenguaje UML[1] (Unified Modeling Language, lenguaje unificado de modelado) para especificar los artefactos que se deben generar.

"El Lenguaje Unificado de Modelado (UML) es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software." (Rumbaugh, 2000)

UML es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software. Se utiliza en una gran variedad de formas para dar soporte a una metodología de desarrollo de software, pero no especifica en sí mismo qué metodología o proceso usar, además es

[1] Unified Modelling Language por sus siglas en inglés. Versión original en 1997. Se ha convertido en el estándar para notaciones de modelado por idea de tres expertos en modelado: Booch, Rumbaugh y Jacobson.

un lenguaje que permite la modelación de sistemas con tecnología orientada a objetos. Aunque puede utilizarse con varios procesos de desarrollo fue diseñado para utilizarse en un proceso iterativo, incremental, guiado por casos de uso y centrado en la arquitectura, que es el tipo de proceso que se considera más apropiado para el desarrollo de sistemas complejos modernos.

1.3 Arquitectura de Software

La arquitectura del software es la estructura u organización de los componentes del programa (módulos), la manera en qué estos componentes interactúan, y la estructura de datos que utilizan los componentes. (Presman, 2007) Así, los programadores, diseñadores, ingenieros y analistas pueden trabajar bajo una línea común que les posibilite la compatibilidad necesaria para lograr el objetivo deseado.

1.3.1 Estilos arquitectónicos

Se identifican los estilos arquitectónicos como un “conjunto de reglas de diseño que identifican las clases de componentes y conectores que se pueden utilizar para componer un sistema o subsistema, junto con las restricciones locales o globales de la forma en que se lleva a cabo la composición. Es en gran medida la interacción entre los componentes, mediados por conectores, lo que confiere a los distintos estilos sus características distintivas”. (Clements, Abril de 1996)

Para el desarrollo de la solución fue elegida por la dirección del proyecto la arquitectura en Capas la cual se encuentra dentro del estilo arquitectónico de Llamada y Retorno. Primero, el estilo soporta un diseño basado en niveles de abstracción crecientes, lo cual a su vez permite a los implementadores la partición de un problema complejo en una secuencia de pasos incrementales. En segundo lugar, el estilo admite muy naturalmente optimizaciones y refinamientos. En tercer lugar, proporciona amplia reutilización. Al igual que los tipos de datos abstractos, 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. Esto conduce a la posibilidad de definir interfaces de capa estándar, a partir de las cuales se pueden construir extensiones o prestaciones específicas. Además la aplicación se proveerá de un alto nivel de modificabilidad y escabilidad permitiéndole al sistema cambios ya sea de requisitos funcionales, agregaciones o mejoras en las funcionalidades.

Entre otros de los beneficios que se pueden encontrar en esta arquitectura se tienen que la arquitectura en capas:

- Puede comprender una sola capa como un todo coherente sin saber mucho sobre las otras capas.

- Puede sustituir las capas con implementaciones alternativas de los mismos servicios básicos.
- Minimizar las dependencias entre las capas.
- Una vez que tenga una capa construida, se puede utilizar para muchos servicios de nivel superiores.
- Mantenibilidad: provee una organización lógica de aplicación y desarrollo.
- Escalabilidad y rendimiento: permite distribuir una aplicación, cada capa puede residir en un computador distinto y agregar máquinas para mejorar el rendimiento.
- Seguridad: permite aislar componentes.

1.4 Paradigmas de programación

Un paradigma de programación es una propuesta tecnológica que es adoptada por una comunidad de programadores cuyo núcleo central es incuestionable en cuanto a que unívocamente trata de resolver uno o varios problemas claramente delimitados. (Budd, 1991)

La resolución de estos problemas debe suponer consecuentemente un avance significativo en al menos un parámetro que afecte a la ingeniería de software. Tiene una estrecha relación con la formalización de determinados lenguajes en su momento de definición. Un paradigma de programación está delimitado en el tiempo en cuanto a aceptación y uso ya que nuevos paradigmas aportan nuevas o mejores soluciones que la sustituyen parcial o totalmente.

Aunque existe un gran número de paradigmas, los más esenciales se podrían dividir en dos grandes grupos:

- **Programación Declarativa**: le dice al ordenador qué hacer, pero no cómo hacerlo, o sea, se describe el problema que se quiere solucionar, pero no las instrucciones necesarias para solucionarlo. La solución se logrará mediante mecanismos internos de inferencia de información a partir de la descripción realizada. A este grupo corresponden los paradigmas:
  - ✓ Funcional
  - ✓ Lógica
- **Programación Imperativa**: describe la programación en términos del estado del programa y sentencias que cambian dicho estado. Los programas imperativos son un conjunto de instrucciones que le indican al computador cómo realizar una tarea. A este grupo corresponden los paradigmas:
  - ✓ Orientado a Objetos
  - ✓ Visual, Orientada a eventos, Orientada a Aspectos.

Pero sin lugar a dudas, el paradigma de Programación Orientada a Objetos (POO) desde su aparición revolucionó la forma de pensar a la hora de programar y trajo consigo muchos beneficios, que dieron gran impulso a la construcción de software cada vez más complejos. (Budd, 1991)

1.4.1 Programación Orientada a Objetos (POO)

La programación orientada a objetos o POO es un paradigma de programación que usa objetos y sus interacciones para diseñar aplicaciones y programas informáticos. Una de las características principales de la POO es que sigue con frecuencia el mismo método que se aplica en la resolución de problemas de la vida diaria. El diseño está dirigido por las responsabilidades. Está basado en varias técnicas, incluyendo herencia, abstracción, polimorfismo y encapsulamiento. (Budd, 1991)

Los elementos fundamentales de la POO son:

Objetos: El concepto fundamental de la POO es el objeto. Los objetos representan cosas reales o abstractas del universo del problema a resolver y poseen un nombre que los identifica. Tienen responsabilidades y comportamientos bien definidos. Son consistentes, coherentes y completos; y pertenecen a una categoría o clase.

Clases: Las clases son declaraciones de objetos, también se podrían definir como abstracciones de objetos. La definición de un objeto es la clase. Cuando se programa un objeto y se define sus características y funcionalidades en realidad lo que se está haciendo es programar una clase.

Métodos: Son las funcionalidades asociadas a los objetos.

Herencia: En la POO la herencia es el mecanismo fundamental para implementar la reutilización y extensibilidad del software. A través de ella los diseñadores pueden construir nuevas clases partiendo de una jerarquía de clases ya existente (comprobadas y verificadas) evitando con ello el rediseño, la modificación y verificación de la parte ya implementada. La herencia facilita la creación de objetos a partir de otros ya existentes, obteniendo características (métodos y atributos) similares a los ya existentes.

Abstracción: La abstracción permite realizar generalizaciones, simplifica la realidad ignorando los detalles que no tienen relevancia en el universo del problema que se modela y se enfoca en las cosas comunes, pero permitiendo las variaciones.

Polimorfismo: Se refiere a la capacidad para que varias clases derivadas de una antecesora utilicen un mismo método de forma diferente.

Encapsulamiento: Se denomina al ocultamiento del estado, es decir, de los datos miembro, de un objeto de manera que sólo se puede cambiar mediante las operaciones definidas para ese objeto.

1.5 Marco de Trabajo de desarrollo

En el desarrollo de software un Marco de Trabajo es una estructura conceptual y tecnológica de soporte definida normalmente con artefactos o módulos de software concretos, con base en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado, entre otros programas para ayudar a desarrollar y unir los diferentes componentes de un proyecto. Representa una arquitectura de software que modela las relaciones generales de las entidades del dominio. Provee una estructura y una metodología de trabajo la cual extiende o utiliza las aplicaciones del dominio. En sistemas informáticos, un Marco de Trabajo es con frecuencia una estructura que indica cuáles tipos de programas de aplicación[2] deben ser construidos y cómo estos se interrelacionarán.

Básicamente un Marco de Trabajo evita construir una aplicación desde cero, los marcos de trabajos orientados a objetos en particular están estructurados en librerías de clases y ahorran de cierta forma el trabajo largo y a veces tedioso de la programación. Se debe tener en cuenta que para comenzar a desarrollar en un Marco de Trabajo se deben conocer sus especificaciones. (Saavedra López, 2008)

La unión de ZendExt, Doctrine y ExtJS dio como resultado un Marco de Trabajo que contiene un conjunto de componentes reutilizables que provee la estructura genérica y el comportamiento para una familia de abstracciones, logrando una mayor estandarización, flexibilidad, integración, agilidad en el proceso de desarrollo, este Marco de Trabajo no es más que Sauxe, como se presenta en la Figura 2. (Gómez Baryolo, 2010)

[2] Un Programa de Aplicación es un tipo de programa informático diseñado como herramienta para permitir a un usuario realizar uno o diversos tipos de trabajo.

Figura 2: Aspectos de Sauxe.

En los siguientes subepigrafes se realizará una descripción de los Marcos de Trabajos utilizados para la realización del sistema en desarrollo.

1.5.1 Marco de Trabajo ExtJS

Es una librería Java Script para el desarrollo de aplicaciones web interactivas usando tecnologías como AJAX, DHTML y DOM. Incluye un alto rendimiento, interfaces de usuario personalizables, un bien diseñado y extensible modelo de componentes, una interfaz intuitiva y fácil de utilizar con licencias de código abierto y comercial. (ExtJS, 2011)

Sauxe utiliza ExtJS 2.2 en la capa de presentación, por la gran gama de componentes que se pueden reutilizar para agilizar el proceso de desarrollo y mostrarle al usuario una interfaz más amigable y funcional. (Gómez Baryolo, 2010)

1.5.2 Marco de Trabajo ZendExt 1.0

Es una extensión del Marco de Trabajo Zend desarrollada por el Centro de Informatización de Gestión Empresarial y el centro de Desarrollo y Asimilación de Tecnologías de la UCID (UCI-Defensa) con el objetivo de crear un Marco de Trabajo extensible y configurable, centrando el desarrollo de las aplicaciones en la lógica del negocio y en las interfaces de usuario y alejando a los programadores de los detalles arquitectónicos, con soporte para entornos multi-entidad[3] y para una arquitectura de sistema orientada a componentes. (Gómez Baryolo, 2010)

[3] El objetivo de la implantación de un sistema multientidad es el de utilizar la misma plataforma, herramientas y servicios por distintas entidades o aplicaciones, como si se tratara de un sistema exclusivo de cada una de ellas.

1.5.3 Doctrine 1.2.1

Doctrine 1.2.1 es una potente multiplataforma para el mapeo de objeto-relacional (ORM[4]) para PHP 5.2 o superior. (doctrine-project, 2008)

Sauxe utiliza en la capa de acceso a datos el Lenguaje de Consulta de Datos (DQL[5]) que implementa Doctrine. La documentación de este tiene todas las características necesarias para ser funcional en casi cualquier proyecto. Entre otros elementos se tiene la posibilidad de exportar una base de datos existente a sus clases correspondientes y también a la inversa, es decir, convertir clases (convenientemente creadas) a tablas de una base de datos. (Gómez Baryolo, 2010)

1.5.4 Marco de Trabajo Sauxe

Sauxe ha tenido un gran impacto económico, tecnológico y social en los días actuales ya que con su desarrollo y reutilización se ahorran cuantiosos recursos humanos y materiales, puesto que adquirir un Marco de Trabajo de este tipo en el mercado internacional resultaría costoso y si se decide implementar un sistema que garantice cada uno de los escenarios arquitectónicos que deben soportar las aplicaciones que se desarrollen, se gastarían millones de dólares innecesariamente y nunca se lograría estandarizar y agilizar el proceso de desarrollo y las aplicaciones no estarían centradas en los requerimientos del usuario sino en la tecnología. Como Sauxe garantiza un entorno de desarrollo sustentado por soluciones estables y de alta calidad debido a la estandarización que este introduce al desarrollo, al iniciar un desarrollo sobre este Marco de Trabajo se asegurarían todos los escenarios descritos anteriormente. Estos resuelven un porciento importante de los requisitos del sistema.

En el ámbito tecnológico ha tenido una gran relevancia ya que es un producto novedoso desarrollado sobre tecnologías libres como el lenguaje PHP, gestor de base de datos PostgreSQL, servidor web Apache, entre otros. Su administración centralizada de todos los aspectos necesarios a tener en cuenta en el desarrollo de aplicaciones de gestión lo convierte en un producto de punta en esta rama. Permite realizar un número de funcionalidades que hace esta Arquitectura muy aplicable para cualquier entorno web en PHP. Es la primera vez que un producto de este tipo permite la gestión de multi-entidad y con esta la compartimentación de la información de cada una de ellas. Además, hasta hoy no se permitía la administración de conexiones a las bases de datos y los permisos sobre estas de forma centralizada. Algo novedoso también resulta la incorporación de la auditoría, que permita consultar todas las acciones realizadas en los sistemas, con toda la información asociada, dígase tiempo, valores, interacción. El tener incorporado un componente que

[4] ORM: es un acrónimo Object-Relational Mapping, mapeo de objeto-relacional.
[5] DQL(Doctrine query Languaje): es un enguaje que utiliza Doctrine para ejecutar sus consultas

se encarga de la gestión dinámica de estructuras de un país lo convierte en una tecnología aún más potente y reutilizable. (Gómez Baryolo, 2010)

Sauxe además de contar con todas las ventajas con las que cuentan las tecnologías libres mencionadas anteriormente, también cuenta con un componente llamado Traza que permite gestionar todas las acciones que se ejecutan dentro de un sistema. Con esta información no solo se pueden identificar ataques, puesto que permite evaluar el rendimiento de cada acción para reimplementarla en caso que el tiempo de respuesta sea muy grande. (Gómez Baryolo, 2010)

La arquitectura tecnológica de la plataforma provee un mecanismo de seguridad que cuenta con:

- **Autenticación:** la autenticación (o autentificación) es el proceso de verificar formalmente la identidad de las entidades participantes en una comunicación o intercambio de información. Por entidad se entiende tanto personas, como procesos o computadoras.
- **Autorización:** es la parte del sistema que protege los recursos del sistema permitiendo que sólo sean usados por aquellos consumidores a los que se les ha concedido autorización para ello. Los recursos incluyen archivos y otros objetos de datos, programas, dispositivos y funcionalidades provistas por aplicaciones.
- **Administración de perfil:** debe garantizarse la personalización de las aplicaciones de este dominio a nivel de cada usuario, se define como perfil los datos únicos de cada recurso dentro del sistema que define el comportamiento del mismo ante las entradas emitidas por este recurso y las salidas entregadas por el (los) subsistema(s).
- **Administración de conexiones:** consiste en un grupo de procesos dedicados a la gestión de las conexiones a la base de datos de un sistema determinado ubicado en un servidor de bases de datos definido también como un parámetro configurable, así como el gestor en uso. Esta solución debe incluir la gestión dinámica de usuarios de bases de datos y permisos sobre ellas.

Se puede concluir que sin iniciar el desarrollo ya se cuenta con una parte importante del sistema. Este sistema ha tenido un gran impacto social en la formación de desarrolladores puesto que estos han aprendido todo lo referente al desarrollo de aplicaciones web de alta complejidad e integración, el trabajo con grandes equipos de forma organizada, ahorrando tiempo, recursos y mejorando la calidad de los productos finales. Ha posibilitado a los desarrolladores de aplicaciones web de gestión de diferentes lugares del país, una rápida adquisición de conocimientos sobre la tecnología tipo que se propone, debido a que el Marco de Trabajo se encuentra debidamente documentado.

Cuenta con un curso de postgrado registrado en la UCI de 96 horas, este curso ha posibilitado formar desarrolladores de diversas entidades del país y estos a su vez han logrado implementar aplicaciones en un corto período de tiempo. Sauxe es una forma eficaz de construir sistemas con calidad guiados por estándares internacionales. (Gómez Baryolo, 2010)

1.5.4.1 Componentes del Marco de Trabajo Sauxe

Los componentes que originaron el surgimiento de ZendExt y que constituyen la base del núcleo del Marco de Trabajo Sauxe son 8 de los 51 que posee el Marco de Trabajo Zend, posibilitan la obtención de una poderosa aplicación, con los requerimientos y capacidades necesarias que debe cumplir todo sistema, principalmente de gestión gubernamental como es el caso del SIT. Sauxe es una integración de estos componentes, en conjunto con los de la librería ExtJS y los del ORM Doctrine. Con su correcto uso y configuración permiten dar solución a las dificultades que puedan surgir durante el desarrollo. A continuación se muestra una relación de los componentes más utilizados y una breve descripción de su funcionalidad: (Yadira, 2010)

- Zend_View: Es la clase que permite trabajar con la vista en el patrón Modelo-Vista-Controlador. El uso de esta clase, ocurre en dos grandes pasos: su controlador de secuencia de comandos crea una instancia de Zend_View y asigna variables a esa instancia, luego el controlador le dice al Zend_View que emita una vista, y devuelve el control al script de la vista, lo que genera la vista saliente. Su principal responsabilidad es mantener el script de la vista separado de los scripts del modelo y de los controladores e identificar cuál es la vista o phtml que se está instanciando, una vez identificada procede a su ejecución.
- Zend_Controller_Front: Todas las peticiones que se ejecutan en un subsistema son validadas por el controlador frontal, el mismo contiene un método llamado init() que tiene la responsabilidad de realizar un conjunto de configuraciones y validaciones. Implementa un Modelo de Controlador Frontal usado en aplicaciones que utilizan la arquitectura Modelo-Vista-Controlador (MVC). Su propósito es inicializar el entorno de la solicitud, rutear la solicitud entrante, y luego hacer un envío de cualquiera de las acciones descubiertas; le agrega las respuestas y las regresa cuando se completa el proceso.
- Zend_Loader: Garantiza que a partir de una ruta de inclusión, este sea el responsable de la inclusión de los recursos requeridos en el proceso.
- Zend_Controller_Action: Es una clase abstracta que se puede utilizar para implementar controladores de acción para usar con el Controlador Frontal. Al usar este componente, es necesario hacer una subclase en sus clases actuales de controladores de acción (o hacer una subclase para crear su propia clase base de acción de controladores). La operación más elemental es hacer una subclase, y crear métodos de acción que corresponden a las posibles

acciones que se desee que el controlador maneje, que serán cualquier método que termine en 'Action'.

- ZendExt_Controller: Permite lograr que todas las peticiones sean procesadas únicamente por un archivo index.php, lo que ofrece un punto central para todas las páginas de la aplicación y asegura la instalación de un ambiente correcto para ejecutar la aplicación.
- ZendExt_Validation: Este componente debe validar todos los datos y acciones que se activan ya sean por un usuario o por un sistema externo, que serán ejecutadas en un controlador de una aplicación en específico. La acción será validada antes de llegar al controlador para no hacer peticiones innecesarias a este.
- Doctrine_Query: Representa una consulta DQL que se utiliza para consultar bases de datos en una forma orientada a objetos.
- Doctrine_Record: Presenta la capacidad para hacer referencia al valor de los hijos de un registro, gracias a la carga diferida que hace Doctrine, que permite mayor flexibilidad al no añadir cargas extras.
- ext-base.js: Componente encargado del manejo de las solicitudes y respuestas, trabajo con Ajax y manejo de componentes de ExtJS.
- ext-all.js: Componente encargado de la creación de los componentes visuales de la vista. Está incluida dentro de las clases que trae ExtJS.

1.6 Patrones de Diseño

Un patrón de diseño describe una estructura de diseño que resuelve un problema de diseño particular dentro de un contexto específico y en ¨medio¨ de fuerzas que pueden tener un impacto en la manera en que se aplica y utiliza el patrón. Para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser reusable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias. (Presman, 2007).

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces.

1.6.1 Patrones de Diseño orientados a Objetos

En la tecnología de objetos, un patrón es una descripción del problema y la solución, a la que se da un nombre, y que se puede aplicar a nuevos contextos; idealmente, proporciona consejos sobre el

modo de aplicarlo en varias circunstancias, y considera los puntos fuertes y compromisos. (Larman, 2005)

Entre los patrones de diseño se pueden mencionar los patrones de asignación de responsabilidades GRASP (patrones generales de software para asignación de responsabilidades, siglas de General Responsibility Assignment Software Patterns) y los patrones GOF (siglas de Gang of Four) que es el nombre con el que se conoce comúnmente a los autores del libro Design Patterns. (Erich Gamma, 1995)

Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. Dentro de este grupo de patrones se encuentran los siguientes: Experto, Creador, Bajo Acoplamiento, Alta Cohesión, Controlador, entre otros.

Los patrones de diseño GOF son 23 y se clasifican según su propósito en Creacionales, Estructurales, de Comportamiento, según su ámbito en Objeto y de Clase.

1.6.2 Patrones de arquitectura y de diseño utilizados

- **Patrón de arquitectura modelo-vista-controlador (MVC)**

El patrón MVC es un patrón de arquitectura de software encargado de separar la lógica de negocio de la interfaz del usuario y es el más utilizado en aplicaciones Web, ya que facilita la funcionalidad, mantenibilidad y escalabilidad del sistema, de forma simple y sencilla, a la vez que permite no mezclar lenguajes de programación en el mismo código.

- El **modelo** representa la información con la que trabaja la aplicación, es decir, su lógica de negocio.
- La **vista** transforma el modelo en una página web que permite al usuario interactuar con ella.
- El **controlador** se encarga de procesar las interacciones del usuario y realiza los cambios apropiados en el modelo o en la vista. (Morín, 2009)

A continuación se realiza un resumen del funcionamiento básico del patrón MVC de forma representativa:

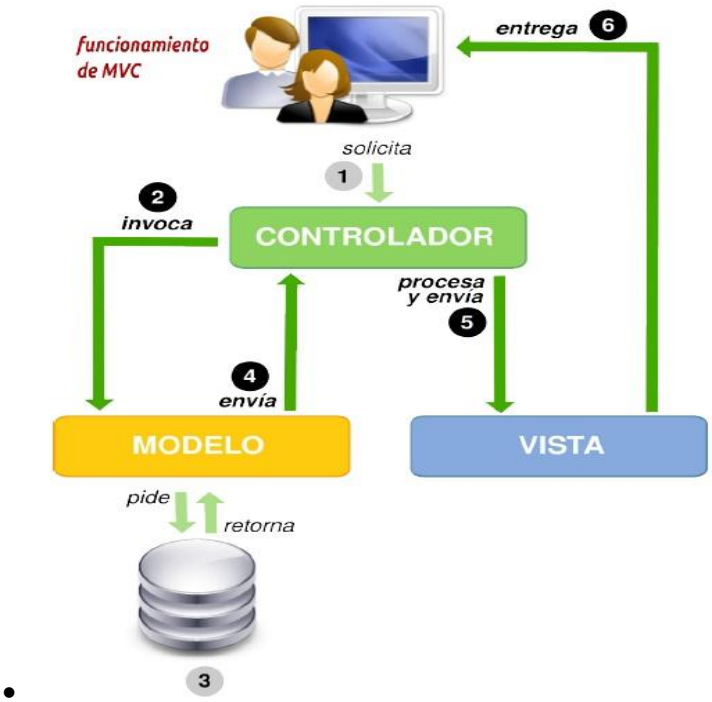

Figura 3: Representación gráfica del funcionamiento del patrón arquitectónico MVC.

- **Experto**

  En este patrón específicamente la asignación de la responsabilidad debe recaer sobre la clase que conoce toda la información necesaria para cumplir con la misma, como la creación de un objeto o la implementación de un método.

- **Alta cohesión**

  Alta cohesión consiste en que la información que almacena una clase debe ser coherente y estar relacionada con la clase de manera que esta tenga responsabilidades moderadas en un área funcional y sólo colabora con las otras clases necesarias para realizar las tareas que se le asignan.

- **Bajo acoplamiento**

  Este patrón asigna una responsabilidad para mantener bajo acoplamiento. El acoplamiento es una medida de la fuerza con que una clase está conectada a otras clases, con que las conoce y con que recurre a ellas. Una clase con bajo (o débil) acoplamiento no depende de muchas otras.

- **Controlador**

El patrón controlador es un patrón que sirve de intermediario entre una interfaz específica y el algoritmo que la implementa, de tal manera que es quien recibe los datos del usuario y los envía a las distintas clases según el método llamado. Este patrón propone que la lógica de negocios debe estar separada de la capa de presentación, con el objetivo de aumentar la reutilización de código y a la vez tener un mayor control.

- **Patrón de diseño Singleton**

El patrón de diseño creacional Singleton[6] está diseñado para restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto. Su intención consiste en garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella. (Welicki, 2012)

- **Patrón de diseño Inversión de Control (IoC)**

La Inversión de Control es un patrón de diseño pensado para permitir un menor acoplamiento entre componentes de una aplicación y fomentar así el reúso de los mismos. Se utiliza entre otras cosas para desacoplar las clases de sus dependencias de manera de que las mismas puedan ser reemplazadas o actualizadas con muy pocos o casi ningún cambio en el código fuente de sus clases, cuando se desea escribir clases que dependan de clases cuyas implementaciones no son conocidas en tiempo de compilación, si se desea testar las clases aisladamente sin sus dependencias y si se quiere desacoplar sus clases de ser responsables de localizar y gestionar el tiempo de vida de sus dependencias. (Guillerón, 2009)

1.7 Lenguajes utilizados

Se utilizan dos lenguajes interpretados, del lado del servidor php (PHP 5.2.5 con las extensiones php-pgsql, php-xsl, php-soap, php-gdi) cuyo intérprete es el motor de Zend y Javascript del lado del cliente usando como intérprete Mozilla Firefox[7] el cual cuenta con Firebug que es una extensión creada y diseñada especialmente para desarrolladores y programadores web. Esta extensión cuenta con un paquete de utilidades con el que se puede analizar (revisar velocidad de carga, estructura DOM[8]), editar, monitorizar y depurar el código fuente, CSS, HTML y JavaScript de una página web de manera instantánea y online. Más allá de estas potentes utilidades que presenta el navegador web Mozilla Firefox se posibilita la portabilidad para otros navegadores web que

[6] Instancia única
[7] Mozilla Firefox es un navegador web de software libre el cual cuenta con varias extensiones que lo hacen un potente intérprete para el desarrollo web.
[8] DOM: Modelo de Objetos del Documento es esencialmente una interfaz de programación de aplicaciones que proporciona un conjunto estándar de objetos para representar documentos HTML y XML, un modelo estándar sobre cómo pueden combinarse dichos objetos, y una interfaz estándar para acceder a ellos y manipularlos.

cumplan con el estándar ECMA 262 que norma al ECMAScript[9]. Vale señalar que el uso de estos lenguajes de programación vienen definido por el Marco de Trabajo Sauxe.

1.7.1 Lenguaje del lado del cliente

Un lenguaje del lado cliente es totalmente independiente del servidor, lo cual permite que la página pueda ser albergada en cualquier sitio. El código, tanto del hipertexto como de los ***scripts***, es accesible a cualquiera y ello puede afectar a la seguridad.

JavaScript

JavaScript es un lenguaje que permite a los desarrolladores crear acciones en sus páginas web. Puede ser utilizado por profesionales y para quienes se inician en el desarrollo y diseño de sitios web. No requiere de compilación ya que JavaScript funciona del lado del cliente, los navegadores son los encargados de interpretar estos códigos. Entre los diferentes servicios que se encuentran realizados con JavaScript en Internet se encuentran: Correo, Chat, Buscadores de Información entre otros. (Flanagan, 2001)

1.7.2 Lenguaje del lado del servidor

Se les clasifica como lenguajes del lado del servidor a los lenguajes de programación en la tecnología cliente servidor que se ejecutan del lado del servidor y de los cuales los usuarios solo obtienen el beneficio del procesamiento de la información. (DesarrolloWeb, 2007)

PHP 5.2.5

PHP (inicialmente PHP[10] Tools o Personal Home Page Tools) es un lenguaje de código abierto interpretado de alto nivel embebido en páginas HTML y ejecutado en el servidor, que fue escrito originalmente por Rasmus Lerdorf. Entre sus principales ventajas se pueden encontrar:

- Es un lenguaje multiplataforma.
- Completamente orientado al desarrollo de aplicaciones web dinámicas[11] con acceso a información almacenada en una base de datos.

[9] ECMAScript: es una especificación de lenguaje de programación publicada por ECMA International. El desarrollo empezó en 1996 y estuvo basado en el popular lenguaje JavaScript propuesto como estándar por Netscape Communications Corporation. Actualmente está aceptado como el estándar ISO 16262.
[10] Hypertext Preprocessor, Preprocesador de Hipertexto.
[11] Una página web dinámica es aquella que incorpora efectos como texto que aparece y desaparece, animaciones, acciones que se activan al pulsar botones y ventanas con mensajes de aviso al usuario.

- **Rendimiento**: El motor de PHP 5 fue rediseñado completamente con un administrador de memoria optimizado para mejorar el rendimiento.
- **Portabilidad**: PHP está disponible para sistemas operativos como UNIX, Microsoft Windows, Mac OS y OS/2. Los programas escritos en este lenguaje son portables entre plataformas.
- **Facilidad de uso:** Su sintaxis es clara y consistente, cuenta además con documentación exhaustiva para todas las funciones de su núcleo.
- **Código Abierto:** El lenguaje es desarrollado por una comunidad de programadores voluntarios que publican su código libremente en la Web y puede ser usado sin el pago de licencias o inversiones en hardware costoso. Esto reduce el costo de la producción del software sin afectar la flexibilidad o confiabilidad.
- **Soporte comunitario:** Cuenta con amplio soporte gracias a la numerosa comunidad de programadores que lo usan en todo el mundo.

1.8 Herramientas

Para el desarrollo de este trabajo de diploma se estudiaron las herramientas definidas y utilizadas por el proyecto de Informatización de los TPC, las cuales se describen a continuación.

1.8.1 Herramienta de modelado

Una Herramienta de Modelado o CASE[12] "*es un conjunto de ayudas para el desarrollo de programas informáticos, desde la planificación, pasando por el análisis y el diseño, hasta la generación del código fuente de los programas y la documentación*". (Luis Giraldo, 2005)

Visual Paradigm 6.4 es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software, análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. VisualParading también proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML. (VisualParading, 2004)

Entre las principales características de VisualParading se pueden encontrar:

[12] Computer Aided Software Engineering, Ingeniería de Software Asistida por Computadora.

- Soporte de UML versión 2.1.
- Permite la realización de Diagramas de Procesos de Negocio.
- Permite el Modelado colaborativo con CVS y Subversion.
- Interoperabilidad con modelos UML2 (meta modelos UML 2.x para plataforma Eclipse) a través de XML[13].
- Ingeniería de ida y vuelta. Ingeniería inversa.
- Generación de código - Modelo a código, diagrama a código.
- Editor de detalles de Casos de Uso - Entorno todo-en-uno para la especificación de los detalles de los casos de uso, incluyendo la especificación del modelo general y de las descripciones de los casos de uso.
- Diagramas de flujo de datos.
- Soporte ORM - Generación de objetos Java desde la base de datos.
- Generación de bases de datos - Transformación de diagramas de Entidad- Relación en tablas de base de datos.
- Ingeniería inversa de bases de datos - Desde Sistemas Gestores de Bases de Datos (DBMS) existentes a diagramas de Entidad-Relación.
- Generador de informes para generación de documentación.
- Distribución automática de diagramas - Reorganización de las figuras y conectores de los diagramas UML.

Por estas razones y porque permite modelar todos lo diagramas como parte de los artefactos especificados por las fases de la metodología RUP para el ciclo de vida del software, se definió como Herramienta CASE a utilizar Visual Paradigm.

1.8.2 Ambiente de desarrollo integrado

Un entorno de desarrollo integrado o Integrated Development Environment (IDE), en inglés, es un programa compuesto por un conjunto de herramientas para un programador. Puede dedicarse en exclusiva a un solo lenguaje de programación o bien, poder utilizarse para varios. Un IDE es un entorno de programación que cuenta con un editor de código, un compilador, un depurador y un

[13] XML Metadata Interchange

constructor de interfaz gráfica de usuario (GUI). Los IDEs pueden ser aplicaciones por si solas o pueden ser parte de aplicaciones existentes.
Eclipse, ZendStudio y Netbeans son entornos de desarrollo integrados que pueden utilizar PHP como lenguaje de programación.

Por parte del equipo de arquitectura se decidió utilizar Netbeans 7.0.1 debido a que es un proyecto de código abierto, soporta lenguajes dinámicos como PHP y Java Script, tiene integración con el subversión y el Marco de Trabajo Zend, es multi-plataforma, tiene una interfaz muy amigable e intuitiva, tiene todas las herramientas para crear aplicaciones profesionales ya sean de escritorio, empresariales, web, móviles y aplicaciones SOA[14], no solo en Java sino también en C/C++ y Ruby, se pueden hacer diagramas UML y es de múltiples lenguajes.

Netbeans 7.0.1: Es una plataforma de programación utilizada para crear entornos de desarrollo. Sus creadores lo definen como un IDE para todo y nada en particular. La arquitectura de plugins de Netbeans permite, además de integrar diversos lenguajes sobre un mismo IDE, introducir otras aplicaciones accesorias que pueden resultar útiles durante el proceso de desarrollo como: herramientas UML, editores visuales de interfaces y ayuda en línea para librerías. (netbeans.org, 2008)

1.8.3 Sistema Gestor de Base de Datos

Para que los usuarios puedan procesar, describir, administrar y recuperar los datos almacenados en una base de datos se utiliza un sistema gestor de bases de datos o SGBD. En estos sistemas se proporciona un conjunto coordinado de programas, procedimientos y lenguajes que permiten a los distintos usuarios realizar sus tareas habituales con los datos, garantizando además la seguridad de los mismos. Dentro de este concepto hoy son más populares debido a sus funcionalidades los Sistemas de Gestión de Bases de Datos Relacionales (SGBDR). Bajo este concepto se oculta toda la complejidad informática necesaria para gestionar (DesarrolloWeb, 2007)**:**

- El acceso controlado de los procesos de los usuarios a los datos.
- La gestión del almacenamiento de los datos.
- La gestión de las comunicaciones entre procesos clientes y servidores.
- Interconexión con distintos protocolos y sistemas operativos.
- Acceso a los recursos conectados a la red.

[14] Arquitectura Orientada a Servicios

El SGBD que se decidió utilizar por parte de la dirección del proyecto y del equipo de arquitectura es: **PostgreSQL 8.4.**

PostgreSQL es un poderoso Sistema de Base de Datos Objeto Relacional de código abierto. Cuenta con más de 15 años de desarrollo activo y una arquitectura probada que se ha ganado una sólida reputación de confiabilidad, integridad de datos y corrección. Funciona en todos los principales sistemas operativos, incluyendo Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64), y Windows. Es totalmente compatible con ACID[15], tiene soporte completo para claves foráneas, uniones, vistas, disparadores y procedimientos almacenados (en varios idiomas). También es compatible con el almacenamiento de objetos binarios, incluyendo imágenes, sonidos o vídeo. Tiene interfaces de programación nativa de C / C++, Java, NET, Perl, Python, Ruby, Tcl, ODBC, entre otros, y la documentación en varios idiomas.

PostgreSQL cuenta con sofisticadas funciones como el Control de Concurrencia Multi-Versión (MVCC), punto en el tiempo de recuperación, tablespaces, replicación asincrónica, transacciones anidadas (puntos de retorno), backups en línea y un planificador optimizador de consultas sofisticadas. Es compatible con conjuntos de caracteres internacionales, codificación de caracteres multi-byte, Unicode, y es consciente de la configuración regional para la clasificación, caso-sensibilidad, y el formato. Es altamente escalable, tanto en la enorme cantidad de datos que puede manejar y en el número de usuarios concurrentes que puede acomodar.

1.8.4 Servidor Web

Un servidor web es un programa informático que procesa una aplicación del lado del servidor realizando conexiones bidireccionales y/o unidireccionales y síncronas o asíncronas con el cliente generando o cediendo una respuesta en cualquier lenguaje o aplicación del lado del cliente.

Por parte de la dirección del proyecto se tomó la decisión de utilizar como servidor web Apache 2.2.3.

Apache: es un servidor web gratuito, potente y que ofrece un servicio estable y sencillo de mantener y configurar. Es indiscutiblemente uno de los mayores logros del Software Libre. (Foundation, 2011)

Apache 2.2.3 como servidor web y de aplicaciones al mismo tiempo. Es multiplataforma, aunque idealmente está preparado para funcionar bajo Linux. Es muy sencillo de configurar además de ser

[15] **ACID** es un acrónimo de **A**tomicity, **C**onsistency, **I**solation and **D**urability: Atomicidad, Consistencia, Aislamiento y Durabilidad en español.

Open-source. Tiene amplias librerías de PHP a disposición de los programadores. Posee diversos módulos que permiten incorporarle nuevas funcionalidades, estos son muy simples de cargar. Cuenta con abundante documentación y es capaz de utilizar varios lenguajes de programación.

1.9 Métricas para la medición de la calidad del diseño del software

Cuando se construye un sistema informático es de gran importancia tener en cuenta todos los aspectos necesarios para obtener un producto de software con la mayor calidad posible.

Es importante entonces destacar que una métrica se puede definir como la aplicación continua de mediciones basadas en técnicas para el proceso de desarrollo del software y sus productos para suministrar información relevante a tiempo, así el administrador junto con el empleo de estas técnicas mejorará el proceso y sus productos. (Muñóz, 2008)

Los sistemas de software orientados a objetos, como el sistema propuesto en este trabajo, poseen características especiales que lo diferencian de otros sistemas de software desarrollados utilizando métodos convencionales, es por ello que las métricas aplicadas a sistemas orientados a objetos deben ajustarse a estas características.

1.9.1 Métricas orientadas a Clases

Son métricas especializadas en el diseño orientado a objetos, cuya función consiste en medir las características correspondientes a la comunicación y la colaboración entre los objetos.

De esta colección las métricas que más referencias poseen son las orientadas a clases o la colección de métricas de CK [16] como también se les conoce:

- **Árbol de profundidad de herencia (APH):** Esta métrica se basa en la longitud máxima desde el nodo hasta la raíz del árbol, por lo que a medida que crece el APH, las clases de nivel inferior heredarán más métodos y la complejidad del diseño será mayor. Mientras que un valor grande de APH indica que el nivel de reutilización de los métodos es alto.

- **Número de descendientes (NDD):** Un descendiente es una subclase que se encuentra inmediatamente subordinada a otra en la jerarquía de clases. Por lo que a medida que crece el NDD, se incrementa la reutilización y el número de pruebas requeridas para cada descendiente.

Otro conjunto de métricas de este tipo son las propuestas por Lorenz y Kidd, que separan las métricas basadas en clases en cuatro categorías: tamaño, herencia, valores internos y valores externos. Algunas de métricas propuestas por Lorenz y Kidd son:

[16] Chidamber y Kemerer

- **Tamaño Operacional de Clases (TOC):** El tamaño operacional de una clase se determina a partir del número total de operaciones que están encapsuladas dentro de una clase y el número de atributos que están encapsulados por la clase.

- **Relaciones entre clases (RC):** Está dado por el número de relaciones de uso de una clase con otra.

1.10 Métricas de Prueba

Las métricas de prueba que existen se concentran en el proceso de prueba y no en las características técnicas de la prueba. Los responsables de la prueba se guían por las métricas de análisis, diseño y código. Entre ellas vale destacar la métrica de Puntos de Función, Bang, Halstead y la Complejidad Ciclomática. (Pressman, 2005)

Otras a destacar dentro de las métricas de pruebas son:

1.10.1 Pruebas de caja blanca

La prueba de caja blanca también denominada prueba de caja de cristal es un método de diseño de casos de prueba que usa la estructura de diseño procedimental para obtener los casos de pruebas. Esta prueba consiste específicamente en diseñar los Casos de prueba atendiendo al comportamiento interno y la estructura del programa examinándose la lógica interna sin considerar los aspectos de rendimiento. Las pruebas de caja blanca se llevan a cabo en primer lugar, sobre un módulo concreto. Mediante los métodos de prueba de caja blanca el ingeniero del software puede obtener casos de prueba que (1) garanticen que se ejercita por lo menos una vez todos los caminos independientes de cada módulo; (2) ejerciten todas las decisiones lógicas en sus vertientes verdadera y falsa; (3) ejecuten todos los bucles en sus límites y con sus límites operacionales; y (4) ejerciten las estructuras internas de datos para asegurar su validez. (Pressman, 2005)

Algunas técnicas de prueba de Caja Blanca son (Pressman, 2005):

- **Prueba de Condición:** Es un método de diseño de casos de prueba que ejercita las condiciones lógicas contenidas en el módulo de un programa.
- **Prueba de Flujo de Datos:** Se seleccionan caminos de pruebas de un programa de acuerdo con la ubicación de las definiciones y los usos de las variables del programa.
- **Prueba de Bucles:** Es una técnica de prueba de caja blanca que se centra exclusivamente en la validez de las construcciones de bucles.

- **Prueba del Camino Básico:** Esta técnica permite obtener una medida de la complejidad lógica de un diseño y usar la misma 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 circula el flujo de control. Para obtener dicho conjunto de caminos independientes se construye el grafo de flujo asociado y se calcula su complejidad ciclomática[17].

  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. (Pressman, 2005)

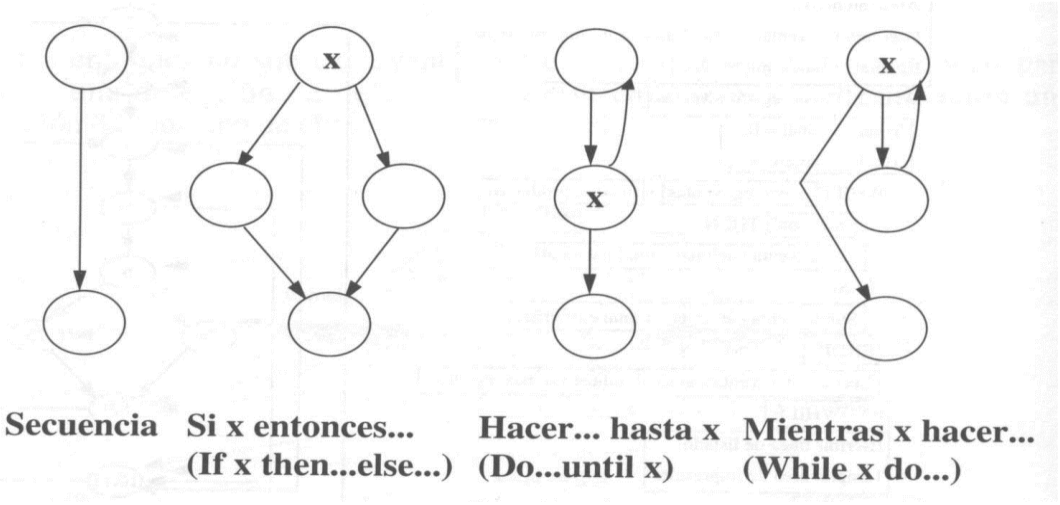

Figura 4: Notación de grafos de flujo para las instrucciones: Secuenciales, If, While.

Un grafo de flujo está formado por 3 componentes fundamentales que ayudan a su elaboración y comprensión, estos brindan información para confirmar que el trabajo se está haciendo adecuadamente. Componentes del grafo de flujo (Pressman, 2005):

- ✓ **Nodo**: son los círculos representados en el grafo de flujo, el cual representa una o más secuencias del procedimiento, donde un nodo corresponde a una secuencia de procesos

17 La Complejidad Ciclomática (en inglés, Cyclomatic Complexity) es una métrica del software que proporciona una medición cuantitativa de la complejidad lógica de un programa. Es una de las métricas de software de mayor aceptación, ya que ha sido concebida para ser independiente del lenguaje.

o a una sentencia de decisión. Los nodos que no están asociados se utilizan al inicio y final del grafo.

- ✓ **Aristas**: son constituidas por las flechas del grafo, son iguales a las representadas en un diagrama de flujo y constituyen el flujo de control del procedimiento. Las aristas terminan en un nodo, aún cuando el nodo no representa la sentencia de un procedimiento.

- ✓ **Regiones**: son las áreas delimitadas por las aristas y nodos donde se incluye el área exterior del grafo, como una región más. Las regiones se enumeran siendo la cantidad de regiones equivalente a la cantidad de caminos independientes del conjunto básico de un procedimiento.

1.10.2 Pruebas de caja negra

Son realizadas sobre la interfaz de usuario y no interesa el funcionamiento interno del software, están orientadas a demostrar que las funciones del sistema son operativas, que los valores de entrada se aceptan adecuadamente y que se produce una salida correcta. (Pressman, 2005)

Este tipo de prueba se centra en lo que se espera del software, es decir, los casos de prueba pretenden demostrar que las funciones del sistema son operativas, que los valores de entradas se aceptan adecuadamente y que se produce una salida correcta, sin preocuparse de lo que pueda estar haciendo el software internamente, es decir, todas las pruebas se realizan sobre la interfaz de usuario. (Pressman, 2005)

La prueba de Caja Negra no es una alternativa a las técnicas de prueba de la Caja Blanca, sino un enfoque complementario que intenta descubrir diferentes tipos de errores a los encontrados en los métodos de la Caja Blanca.

Con este tipo de pruebas se intenta encontrar (Pressman, 2005):

- Funciones incorrectas o ausentes.
- Errores de interfaz.
- Errores en estructuras de datos o en accesos a las bases de datos externas.
- Errores de rendimiento.
- Errores de inicialización y terminación.

Para el correcto desarrollo de las pruebas existen diferentes técnicas: Partición de equivalencia, Análisis de valores límites y Grafos de Causa-Efecto**.**

Conclusiones parciales

Al finalizar este capítulo se arribó a las siguientes conclusiones:
Los conceptos y definiciones estudiadas resultaron de gran importancia para una mejor comprensión del objetivo de este trabajo y así darle continuidad al mismo.
De manera general se analizaron los principios que rigen el funcionamiento del Proceso Ordinario de la Materia Civil en los TPC, así como las metodologías, herramientas, lenguajes de programación que serán utilizados para el desarrollo del sistema y su arquitectura, como se resumen a continuación:

- RUP, como metodología de desarrollo.
- Visual Paradigm con UML, como herramienta de modelado CASE.
- UML, como lenguaje de modelado.
- PHP, como lenguaje de programación por la parte del servidor y JavaScript por la parte del cliente.
- Netbeans, como entorno de desarrollo.
- SAUXE, como Marco de Trabajo de desarrollo.
- PostgreSQL y Apache, como servidores de Base de Datos y Web, respectivamente.

Además de las métricas a utilizar para validar el diseño propuesto como son Tamaño Operacional de Clase (TOC) y Relaciones Entre Clases (RC).

Capítulo 2: Solución Propuesta

Introducción

En el presente capítulo se presenta la propuesta de solución. Inicialmente se realizará un análisis profundo de la arquitectura del sistema a desarrollar y se tratará además la estructura organizativa para la implementación del sistema. A partir de los requisitos identificados para el Proceso Ordinario de la Materia Civil se modelarán los Diagramas de Clases del diseño, los Diagramas de Componentes y de Despliegue, los cuales permiten obtener una visión más clara de la implementación del sistema la cual se realizará regida por los estándares de codificación que se abordan en este capítulo.

La solución a desarrollar consta de 30 casos de uso los cuales responden a 86 Requisitos Funcionales. La descripción de los mismos se encuentra en el Modelos del sistema v2.0 del Subsistema Civil.

2.1 Arquitectura del sistema

El desarrollo del sistema informático responde a una arquitectura basada en capas compuesta por tres capas lógicas: la Capa de Presentación, Capa de Control o Negocio, Capa Acceso a Datos y una capa física: Capa de Datos.

Capa de presentación

En esta capa se emplean las facilidades que brinda el Marco de Trabajo ExtJS para la construcción de interfaces amigables a la vista de los usuarios. ExtJS centra su desarrollo en el uso de JavaScript, CSS y AJAX. Esta capa se comunica únicamente con la capa de negocio.

Capa de Control o Negocio

Es donde se encuentran los programas que se ejecutan, se reciben las peticiones del usuario y se envían las respuestas tras el proceso. Se denomina capa de negocio (e incluso de lógica del negocio) porque es aquí donde se establecen todas las reglas que deben cumplirse. Esta capa se comunica con la capa de presentación para recibir las solicitudes y presentar los resultados, y con la capa de datos, para solicitar al gestor de base de datos almacenar o recuperar datos de él. También se consideran aquí los programas de aplicación.

Capa acceso a Datos

En esta capa estará presente el ORM (Object Relational Mapping) Doctrine como Marco de Trabajo de persistencia para la comunicación con el servidor de datos mediante el protocolo PDO[18].

Capa de Datos

En esta capa estará ubicado como servidor de base de datos PostgreSQL y un conjunto de Ficheros de Configuración de la arquitectura tecnológica.

A continuación se muestra una figura en la que se ven relacionadas estas 4 capas.

Figura 5: Representación gráfica de las relaciones entre capas.

2.2 Estructura organizativa para la implementación del sistema

Teniendo en cuenta los principios establecidos por Sauxe, la estructura está basada en el modelo en tres capas (Capa de presentación, Capa de acceso a datos y Capa de lógica del negocio), que define una organización jerárquica de manera tal que cada capa proporciona servicios a la capa superior y se sirve de las prestaciones que le brinda la capa inferior. Estas están integradas por elementos compuestos de varios paquetes o subsistemas. Según la materia, el módulo creado para su gestión posee la estructura necesaria para el desarrollo de las funcionalidades específicas de cada uno. (Pupo, 2010)

[18] PHP Data Objects

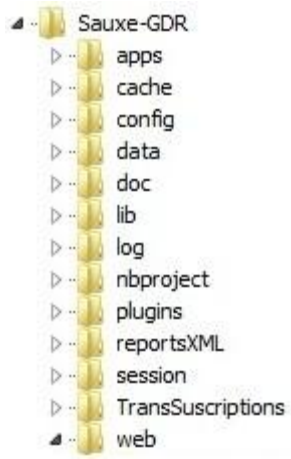

Figura 6: Organización propuesta por Sauxe

En la carpeta Web se controla la vista, específicamente en la carpeta Views (Vistas), se recopilan los ficheros que van a gestionar la capa de presentación: (Pupo, 2010)

- Css: dentro de esta carpeta se encuentran las plantillas para el diseño del módulo, aunque ExtJS proporciona los componentes necesarios para la presentación.
- js: dentro de esta carpeta se encuentran los ficheros de extensión js donde se escribirá el código correspondiente a la capa de presentación, aquí es donde se carga el diseño de la aplicación, para cada clase controladora se crea una carpeta que tendrá incluido el fichero js correspondiente a dicha clase control. (Pupo, 2010)

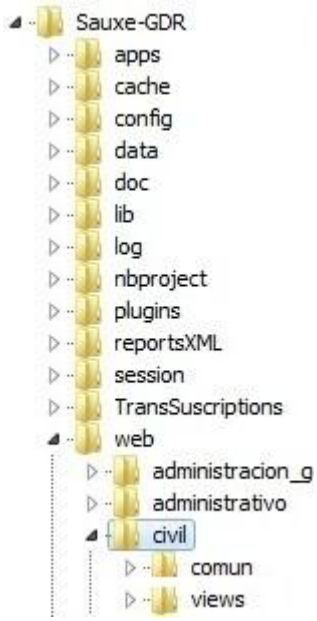

Figura 7: Estructura de la Capa de Presentación

En la carpeta Apps se maneja el control de la lógica del negocio y del modelo, así como la prestación y publicación de servicios, dentro de esta se encuentran las siguientes carpetas: (Pupo, 2010)

- Común: contiene los ficheros xml de las excepciones del módulo (exception.xml) y para la publicación de los servicios creados (ioc-general.xml).
- Controllers (Controladoras): contiene las clases que manejan el control de la lógica del negocio, en las que se programan las funcionalidades que son utilizadas cuando el usuario hace alguna solicitud al sistema, a través de acciones sobre los componentes de la interfaz, como oprimir un botón o seleccionar un valor de una lista.
- Models (Modelo): el control del modelo se realiza a partir de las tres clases generadas por el mapeador de datos Doctrine; las business, en las que se realizan los gestionar de los datos (Insertar, Eliminar, Modificar), las domain, donde se programan las consultas a la Base de Datos y que poseen herencia de las generated, que son las clases que reflejan la abstracción de las tablas de la Base de Datos y su relación con el resto de las tablas.
- Services: contiene los servicios que se utilizan, aunque el módulo Administración y Gobierno es el encargado de la creación de los servicios que son utilizados en conjunto por el resto de los módulos, cada uno crea los servicios que requiera en la carpeta services y luego estos son publicados en su fichero ioc-general.xml, con un nombre que identifique la función que realiza el servicio.
- Validators: esta carpeta contiene las clases de tipo php que van a realizar acciones de validación en el componente, como por ejemplo las precondiciones que se deben cumplir antes de que un determinado método sea ejecutado.
- Views: contiene los archivos de extensión phtml donde se especifica el título de la página que se gestiona y se carga el archivo js que mostrará la presentación y se especifica el orden en que estos serán utilizados pues la presentación generalmente se segmenta en varios js, para optimizar la organización del código.

Figura 8: Estructura de la Capa de Control de la Lógica del Negocio y del Modelo

2.3 Estándares de codificación

En el caso del proyecto Tribunales Populares Cubanos el estándar de codificación fue definido en sus inicios por la dirección del proyecto, a continuación se muestra el formato para los siguientes puntos:

Identación

- En el contenido siempre se identará este con tabs, nunca utilizando espacios en blanco.

Cabecera del archivo

- Es importante que todos los archivos .php inicien con una cabecera específica que indique información de la versión y autor de los últimos cambios. Es de cada equipo decidir si se quieren o no agregar más datos.

A continuación se muestra un ejemplo de cómo debe quedar la cabecera de los archivos:

```
/*** Descripción de RegistrarEscritoDemanda
*@Versión: 5.4.2 @modificado: 11 de mayo del 2012
* @Autor Reinier
*/
```

Comentarios en las funciones

- Todas las funciones deben tener un comentario antes de su declaración explicando que hacen. Ningún programador debería tener que analizar el código de una función para conocer su utilidad. Tanto el nombre como el comentario que acompañe a la función deben bastar para ello.

Clases

- Las clases serán colocadas en un archivo .php aparte donde sólo se colocará el código de la clase. El nombre del archivo es el mismo del de la clase y siempre empezará en mayúscula. En lo posible, procurar que los nombres de clase tengan una sola palabra. Las clases siguen las mismas reglas de las funciones, por tanto, debe colocarse un comentario antes de la declaración de la clase explicando su utilidad.

Rutinas y Métodos

- Los métodos comenzarán su nombre con minúscula y si es más de una palabra la segunda comenzará con mayúscula.
- Un buen nombre para una función o método es aquel que describe todo lo que la rutina hace.
- Es recomendable que los nombres de los métodos comiencen con un verbo seguido del objeto al que afecta. Por ejemplo: incluirDiligencia, verDiligencia, obtenerInforme.
- Cuando existan grupos de funciones que realicen operaciones similares con pequeñas diferencias, se deberá establecer un sistema de creación de nombres coherente.

Números de métodos y grados de responsabilidad

- El número de métodos de una clase debe ser inferior a 40. Esta regla debe aplicarse a todas las clases del proyecto. (Rosenberg, 1999)

Nombre de variables

- Excepto las constantes, todas las instancias y variables de clase o método empezarán con minúscula. Las palabras internas que lo forman (si son compuestas) empiezan con su primera letra en mayúsculas (lowerCamelCase[19]).
- No se utilizarán nombres de variables que puedan ser ambiguos. Por ejemplo col es un nombre ambiguo ya que puede ser columna o color.
- Se procurará evitar dar nombres sin sentido a variables temporales. Por ejemplo: temp, i, tmp.
- Las variables booleanas deben tener nombres que sugieran respuestas o contenidos de tipo S/N, por ejemplo: Éxito, Correcto, Realizado.
- Los nombres de las variables booleanas deben ser positivos, por ejemplo: encontrado en lugar de noEncontrado.
- Se introducirán variables booleanas temporales cuando en una estructura de control (if, case, while) la expresión sea excesivamente compleja.

[19] lowerCamelCase: cuando la primera letra de cada una de las palabras es minúscula.

2.4 Diseño e implementación

El diseño es un refinamiento que toma en cuenta los requerimientos no funcionales, por lo cual se centra en cómo el sistema cumple sus objetivos. El modelo de implementación describe cómo los elementos del modelo del diseño se implementan en términos de componentes y cómo estos se organizan de acuerdo a los nodos específicos en el modelo de despliegue. (MeRinde, 2012)

2.4.1 Patrones de diseño y de arquitectura utilizados

Para el desarrollo del sistema se utilizaron los siguientes patrones de diseño:

Patrón Bajo Acoplamiento:

Este patrón está evidenciado en el marco de trabajo Sauxe ya que dentro de la capa modelo, las clases de abstracción de datos son las más reutilizables y no tienen asociaciones con las clases de la capa de la vista ni con el controlador.

En la siguiente figura se muestra un ejemplo del patrón Bajo Acoplamiento cuando la clase de abstracción de datos CivDdemandado solo tiene relación con su clase base BaseCivDdemandado.

```php
<?php
class CivDdemandado extends BaseCivDdemandado
{
    public function setUp()
    {
        parent::setUp();

        $this->hasOne('CivDpersona', array('local'=>'idrepresentantelegal','foreign'=>'idpersona'));
        $this->hasOne('CivDpersona', array('local'=>'idpersona','foreign'=>'idpersona'));
        $this->hasOne('CivDexpediente', array('local'=>'idexpediente','foreign'=>'idexpediente'));
    }

}
?>
```

Figura 9: Clase del modelo que solo depende de su clase Base.

Patrón Controlador:

En la estructura que propone Sauxe este patrón se evidencia dentro de la carpeta controllers que se encuentra en “/app/Civil/controllers”, dentro de esta carpeta se encuentran todas las clases controladoras del módulo y dichas clases son las responsables de implementar todas las funcionalidades pertenecientes a una interfaz de un componente determinado, además son las que reciben los datos del usuario y los utilizan de acuerdo a la acción solicitada. En la siguiente imagen se muestra un ejemplo de una clase controladora:

```php
<?php

class RegistrarescritodemandaController extends ZendExt_Controller_Secure {

    function init() {

    function registrarescritodemandaAction() {

    function cargarComboProvinciaAction() {

    function cargarComboMunicipioAction() {

    function cargarComboTipoDemandaAction() {

    function cargarComboPaisesAction() {

    function adicionarEnSessionAction() {
```

Figura 10: Clase controladora RegistrarescritodemandaController.

Patrón Inversión de Control (IoC):

La Inversión de Control es un patrón de diseño pensado para permitir un menor acoplamiento entre componentes de una aplicación y fomentar así el reuso de los mismos. Este patrón es utilizado por el Marco de Trabajo para el trabajo de dependencia entre módulos que se hace mediante la creación y consumo de servicios. Cada módulo del proyecto cuenta con un esquema propio en la base de datos. Cuando un módulo necesita acceder al esquema de otro lo que se hace es crear y brindar un servicio para que el que lo necesite lo consuma.

El Marco de Trabajo a través del patrón IoC define la creación de clases, para de ellas brindar los servicios necesarios. En la siguiente figura se muestra la clase ProcedimientosService ubicada en el paquete apps/civil/services donde se crea la función ObtenerProcedimientos.

```php
<?php

class ProcedimientosService {

    function ObtenerProcedimientos($idinstancia)
    {
    return NprocedimientoExt::ObtenerProcedimientos($idinstancia);
    }

    function devolverProcedimiento($idexpediente)
    {
    return NprocedimientoExt::devolverProcedimiento($idexpediente);
    }

}
?>
```

Figura 11: Representación de la clase ProcedimientosService ubicada en el paquete apps/civil/services, donde se crea la función ObtenerProcedimientos.

Cada módulo cuenta con un fichero ioc-general.xml en el cual se brindan los servicios deseados, para que otros lo consuman. La siguiente figura muestra el fichero ioc-general.xml que se encuentra en el paquete apps/civil/comun/recursos/xml.

```
<civil src="civil">
    <ObtenerProcedimientos reference="">
        <inyector clase="ProcedimientosService" metodo="ObtenerProcedimientos" />
        <prototipo>
            <parametro nombre="idinstancia" tipo="integer" />
            <resultado tipo="array" />
        </prototipo>
    </ObtenerProcedimientos>
```

Figura 12: Fichero ioc-general.xml en el paquete apps/civil/comun/recursos/xml.

La siguiente figura muestra la función cargarComboProcesosAction() en la clase InicioController.php del módulo Común, la cual consume el servicio ObtenerProcedimientos brindado por el módulo Civil.

```
function cargarComboProcesosAction() {
    $idninstancia = $this->_request->getPost('idinstancia');
    $materia = $this->_request->getPost('materia');
    if ($materia == 'Penal')
        $procedimientos = $this->integrator->penal->ObtenerProcedimientos($idninstancia);
    else if ($materia == 'Civil')
        $procedimientos = $this->integrator->civil->ObtenerProcedimientos($idninstancia);
    else if ($materia == 'Laboral')
        $procedimientos = $this->integrator->laboral->ObtenerProcedimientos($idninstancia);
    else if ($materia == 'Administrativo')
        $procedimientos = $this->integrator->administrativo->ObtenerProcedimientos($idninstancia)
    else if ($materia == 'Económico')
        $procedimientos = $this->integrator->economico->ObtenerProcedimientos($idninstancia);
    echo json_encode($procedimientos);
}
```

Figura 13: Representación de cómo se consume el servicio ObtenerProcedimientos brindado por el módulo Civil.

Singleton:

El patrón de diseño Singleton (instancia única) está diseñado para restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto. Su intención consiste en garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella.

Se implementa creando en nuestra clase un método que crea una instancia del objeto sólo si todavía no existe alguna. Para asegurar que la clase no puede ser instanciada nuevamente se regula el alcance del constructor (con atributos como protegido o privado). La instrumentación del patrón puede ser delicada en programas con múltiples hilos de ejecución. Si dos hilos de ejecución intentan crear la instancia al mismo tiempo y esta no existe todavía, sólo uno de ellos debe lograr crear el objeto. (Alfonso, 2010)

```
class ZendExt_Event {
    /**
     * Instancia para la implementación del patrón Singleton
     *
     * @var ZendExt_Event
     */
    private static $_instance;
    /**
     * Constructor
     */
    private function __construct() {
        $this->_xml = ZendExt_FastResponse::getXML('events');
        $this->_ioc = ZendExt_IoC::getInstance();
    }

    /**
     * Retorna una instancia de ZendExt_Event para el singleton
     *
     * @return ZendExt_Event
     */
    static function getInstance () {
        if (self :: $_instance == null)
            self :: $_instance = new self ();

        return self :: $_instance;
    }
```

Figura 14: Representación del patrón Singleton en la clase Event.php en el paquete lib/ZendExt/ Event.php.

Con la implementación del patrón Singleton que se muestra en la anterior se fuerza a no crear objetos de forma directa mediante el operador "new", pues se puede apreciar en la implementación de la clase ZendExt_Event, primero se crea la instancia que será utilizada en la implementación del patrón, que se declara privada, con el objetivo de que dicha clase sea la única que tenga permisos para modificarla, luego se declara privado el constructor, con el fin de que este no tenga una visibilidad pública eliminando de esta manera la posibilidad de que esta clase sea instanciada mediante el operador "new" y finalmente se verifica si esta instancia ha sido creada, si no ha sido creada se crea y se retorna, y si ya fue creada en el momento que se llamó la función getInstance() solamente se retorna.

Patrón de arquitectura MVC

Este patrón de arquitectura viene definido por el Marco de Trabajo Sauxe. Está estructurado para crear las clases por separado como se detalla a continuación:

- Las clases del modelo, que son la lógica del negocio, están en el paquete **app\civil\Model**.

- Las clases de la vista, que son las que renderizan el modelo dentro de una página web apropiada para que el usuario pueda interactuar, se encuentran en el paquete **app\civil\views.**
- Las clases controladoras encargadas de responder a las acciones del usuario e invocar cambios en el modelo o generar la vista apropiada, se encuentran en el paquete **app\civil\Controllers**.

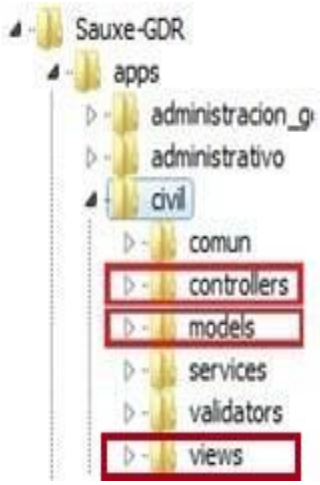

Figura 15: Estructura del paquete Apps, donde se encuentran las clases controladoras, las del modelo y las de la vista, aplicando el patrón MVC.

2.4.2 Modelo de Diseño

El Modelo de Diseño es un modelo de objetos que describe la realización física de los casos de uso, centrándose en cómo los requisitos funcionales junto con otras restricciones relacionadas con el entorno de implementación tienen impacto en el sistema a considerar. Además, el modelo de diseño sirve de abstracción de la implementación del sistema y es de ese modo utilizado como una entrada fundamental de las actividades de implementación. (Jacobson, 1992)

2.4.2.1 Diagramas de Clase

Un Diagrama de Clases es un tipo de diagrama estático que describe la estructura de un sistema mostrando sus clases, atributos y las relaciones entre ellos. Los diagramas de clases son utilizados durante el proceso de análisis y diseño de los sistemas, donde se crea el diseño conceptual de la información que se manejará en el sistema, y los componentes que se encargarán del funcionamiento y la relación entre uno y otro. (Jacobson, 1992)

Se realizaron diagramas de clases a partir de la descripción de cada uno de los casos de uso definidos para informatizar el proceso Ordinario. A continuación se presenta la descripción detallada

del caso de uso Crear providencia por no evacuar dúplica, con el propósito de mostrar más adelante la modelación del mismo.

Caso de Uso:	Crear providencia por no evacuar dúplica
Actores:	Juez Ponente
Resumen:	El caso de uso comienza cuando el juez necesita crear la providencia por no evacuar dúplica, debido a que la parte demandada no propuso un escrito de dúplica en el término establecido. Consiste en que el juez ponente selecciona el expediente a trabajar y genera la resolución correspondiente. El caso de uso termina con la creación de la providencia abriendo a pruebas y teniendo por no evacuado la dúplica.
Precondiciones:	• El sistema debe estar instalado y ejecutándose correctamente. • El usuario debe estar autenticado con los permisos necesarios. • Los expedientes tienen estado: Trasladados para Dúplica. • Que se haya vencido el término establecido para que la parte actora presente el escrito de dúplica.
Referencias	RF: 41, 135, 136, 137, 138 Cu extendido Visualizar detalles de expediente.
Reglas de Negocio	1, 2, 4, 6, 9, 22, 29, 38
Prioridad	Crítico

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. Ordena visualizar los expedientes Trasladados para Dúplica.	2. Muestra una interfaz con los siguientes datos: • Número de Expediente • Tipo de demanda • Trámite (Proveer renuncia a dúplica) • Fecha de Vencimiento del Término
3. Selecciona el expediente que se desea trabajar.	
4. Selecciona la opción Pasar a Definitivo.	5. Muestra la providencia abriendo el proceso a pruebas.

	Finaliza así el caso de uso.
Flujo Alterno 1 "Cancelar"	
Acción del Actor	**Respuesta del Sistema**
4. Presiona la opción Cancelar.	4. 1 Cierra la interfaz correspondiente. **Finaliza así el caso de uso.**
Secciones	
Sección 1 "No abrir a pruebas"	
Acción del Actor	**Respuesta del Sistema**
4. Presiona la opción para no abrir a pruebas el proceso.	
4.1 Presiona la opción Aceptar.	4.2 Muestra un mensaje indicando si está seguro de que el proceso debe pasar a estar concluso para sentencia.
4.3 Presiona la opción afirmativa.	4.4 Muestra la providencia declarando proceso concluso para sentencia. **Finaliza así el caso de uso.**
Flujo Alterno 1 "No"	
Acción del Actor	**Respuesta del Sistema**
4. Presiona la opción Negativa.	4. 1 Cierra la interfaz correspondiente.
Poscondiciones	• El expediente pasa al estado: Pendiente de proposición de pruebas. • Queda elaborada la providencia abriendo a pruebas y teniendo por no evacuado la dúplica. • Si se selecciona la opción No abrir a pruebas: El expediente pasa al estado: Concluso para sentencia.

Tabla 1: Descripción del caso de uso Crear providencia por no evacuar dúplica.

A partir de la descripción anterior, y teniendo en cuenta las clases del modelo implicadas, se realizó el siguiente diagrama de clases, vale aclarar que el resto de los diagramas realizados se encuentra en el Anexo # 3 Modelo de Diseño:

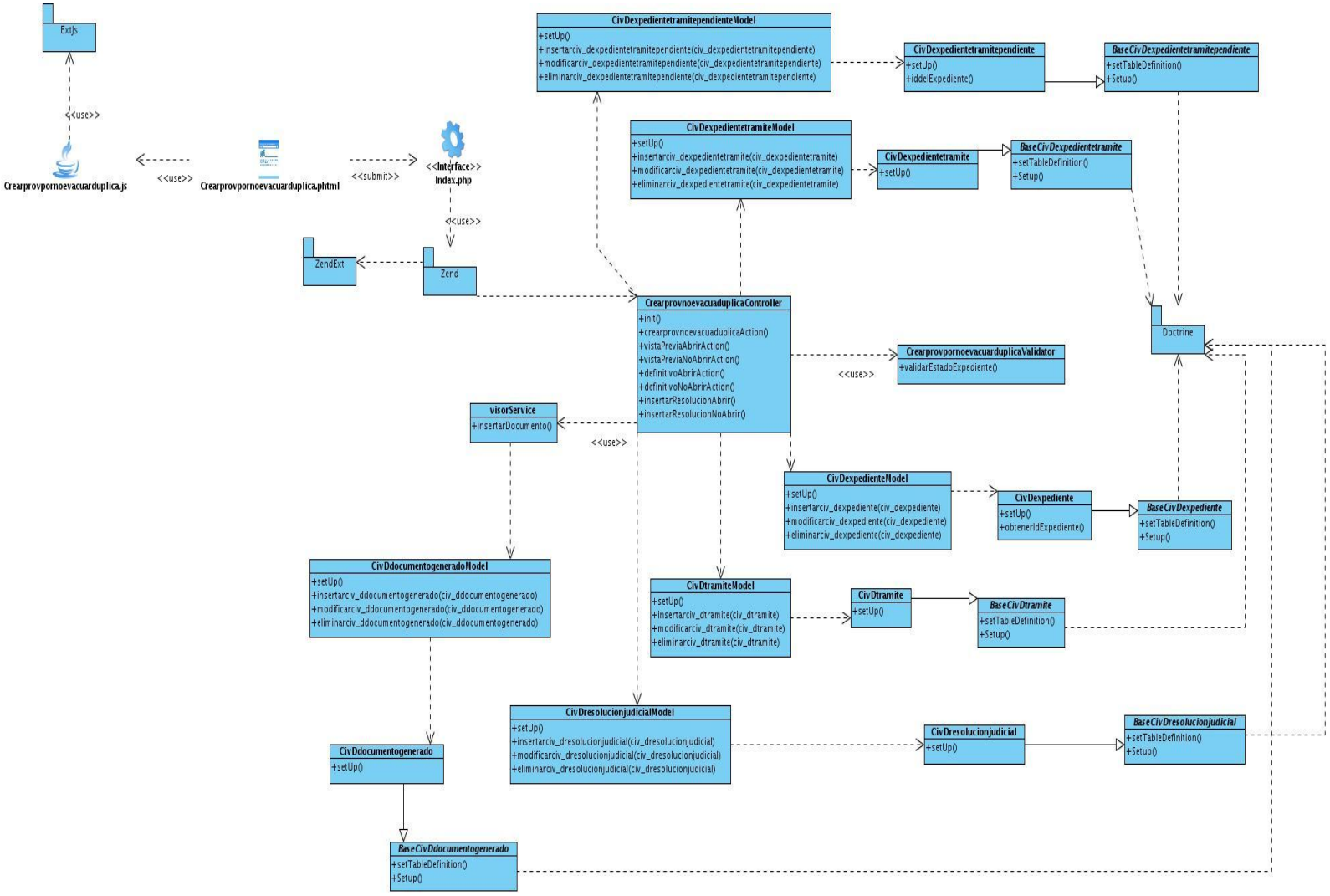

Figura 16: Diagrama de Clase: Crear providencia por no evacuar dúplica.

2.4.2.2 Diagramas de Interacción

Los Diagramas de Iteración se pueden expresar en Diagramas de Secuencia y en Diagramas de Colaboración. Los primeros muestran las interacciones entre objetos, ordenadas en secuencia temporal durante un escenario concreto y son los que encontrará en el Modelo de Diseño, los segundos destacan la organización de los objetos que participan en una interacción. Los diagramas de interacción son importantes para modelar los aspectos dinámicos de un sistema y para construir sistemas ejecutables a través de ingeniería hacia adelante e ingeniería inversa. (States, 2005)

Una vez realizados los diagramas de clases, se procedió a elaborar diagramas de secuencia. A continuación se presenta el diagrama de secuencia para el caso de uso Disponer sobre Demanda, el resto de los diagrama de secuencia se encuentran en el Anexo # 3 Modelo de Diseño:

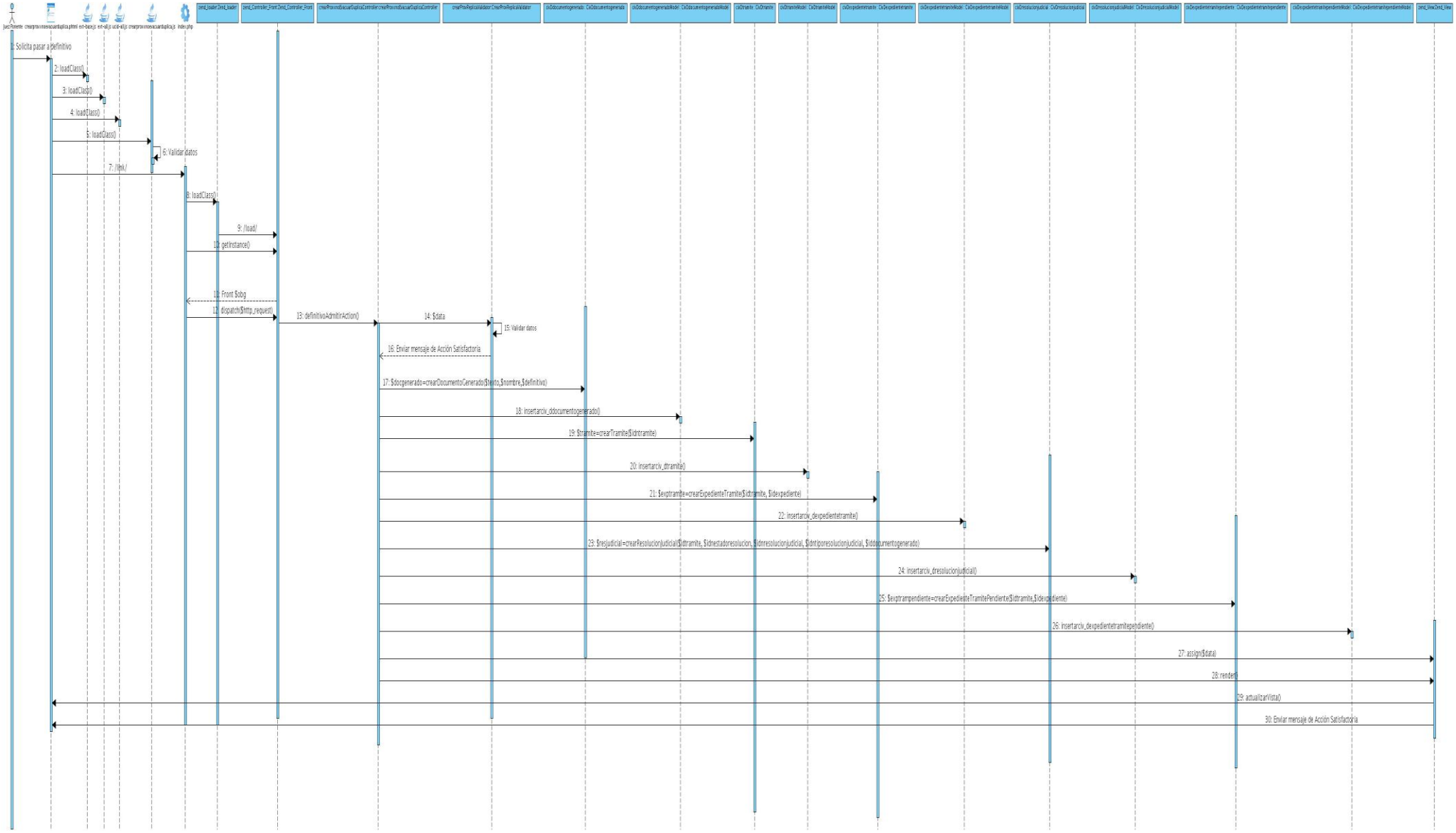

Figura 17: Diagrama de secuencia para el caso de uso Crear providencia por no evacuar dúplica.

2.4.3 Modelo de Implementación

El Modelo de Implementación está conformado por los diagramas de despliegue y diagramas de componentes, que son artefactos generados durante el flujo de trabajo de Implementación ya que es en este donde se describe cómo los elementos del modelo del diseño se implementan en términos de componentes y cómo estos se organizan de acuerdo a los nodos específicos en el modelo de despliegue.

2.4.3.1 Diagrama de Componentes

Un Diagrama de Componentes representa gráficamente cómo un sistema de software es dividido en componentes mostrando las dependencias existentes entre estos. El diagrama contiene componentes, interfaces, relaciones entre ellos y puede contener paquetes utilizados para agrupar elementos del modelo; mostrando las dependencias lógicas entre componentes de software. Los diagramas de componentes son usados para estructurar el modelo de implementación en términos de subsistemas de implementación y mostrar las relaciones entre sus elementos. (States, 2005)

A continuación se muestra el Diagrama de Componentes, en el cual se presentan algunos de los componentes del Marco de Trabajo Sauxe como son Doctrine_Query, Doctrine_Record y otros que fueron previamente descritos en el epígrafe **1.5.4.1** del Capítulo 1 del presente trabajo.

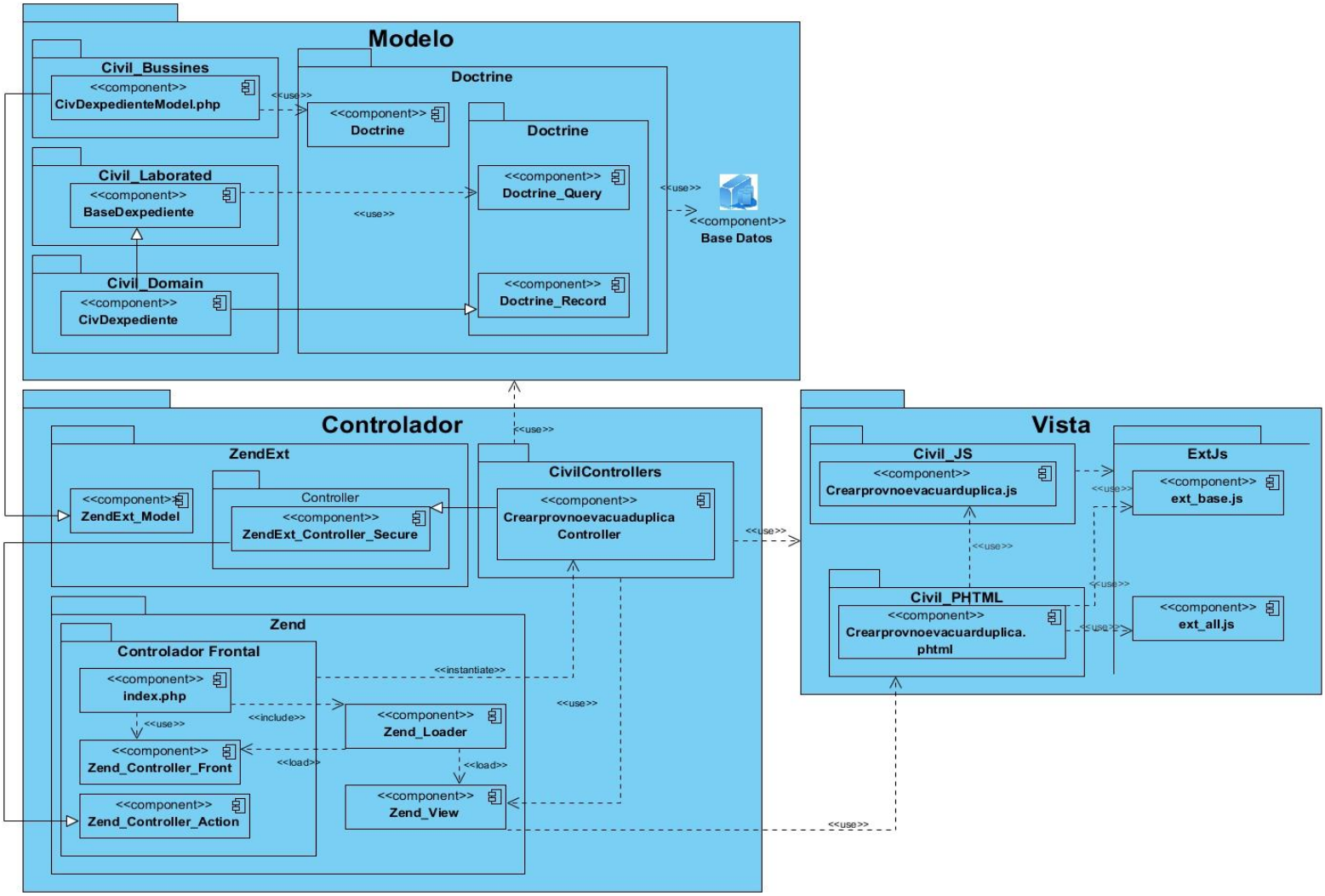

Figura 18: Diagrama de Componentes para el caso de uso Crear providencia por no evacuar dúplica.

2.4.3.2 Diagrama de Despliegue

Los Diagramas de Despliegue muestran las relaciones físicas de los distintos nodos[20] que componen un sistema y el reparto de los componentes sobre dichos nodos. La vista de despliegue representa la disposición de las instancias de componentes de ejecución en instancias de nodos conectados por enlaces de comunicación.

Para la elaboración del Diagrama de Despliegue de la solución propuesta se consideraron los requisitos de hardware que requiere el sistema para su implantación; así como la estructura organizacional de cada instancia de los tribunales y sus respectivas entidades, por lo que es necesario garantizar que el sistema se despliegue en todos los municipios y provincias del país. El Diagrama de Despliegue que se presenta deberá cumplir con las especificaciones siguientes (Arquitectura, 2010):

[20] Un nodo es un recurso de ejecución tal como un computador, un dispositivo o memoria los cuales tienen relaciones que representan medios de comunicación entre ellos así como una Intranet a través de protocolos.

Se requieren de varias PC Clientes, en las que se procesará toda la información que se genera como parte de la tramitación de los procesos según la instancia y la materia, donde al menos uno estará conectado a una impresora por el puerto USB de la máquina para la impresión de los documentos. Por cada instancia el sistema contará con 1 servidor de Base de Datos y con 1 servidor Web Apache2.

Los requerimientos de Hardware estimados para el correcto funcionamiento del sistema consistirán en:

- Las Máquinas Clientes, serán clientes ligeros que utilizarán el Sistema Operativo Linux en la distribución Debian, con al menos 512 MB de memoria RAM.
- Los servidores de aplicación de las instancias del Tribunal poseerán Micro Dual Quad Core Xeon E5440 con 12 megas de cache, 8GB de memoria RAM y 40GB de disco duro y usarán el Sistema Operativo Linux en la distribución Ubuntu.
- Para los servidores de Base de Datos y del Centro de datos se requiere que posean Core I3, 2 GB de memoria RAM, 1 terabyte de disco duro y el Sistema Operativo Linux en la distribución Ubuntu.

A continuación se muestra el Diagrama de Despliegue correspondiente a la estructura interna tanto del Tribunal Supremo Popular, como de los Tribunales Provinciales y Municipales:

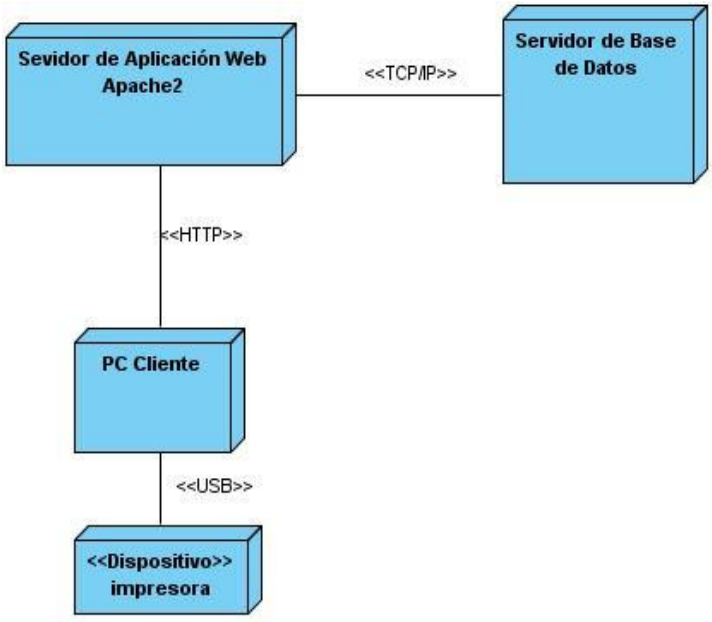

Figura 19: Diagrama de Despliegue de la solución informática.

2.5 Interfaces del sistema

Para la realización de la aplicación se desarrollaron interfaces amigables y seguras ya que cuenta con las facilidades que brinda el Marco de Trabajo ExtJS. A continuación se presenta la interfaz del caso de uso Crear Providencia por no Evacuar Dúplica.

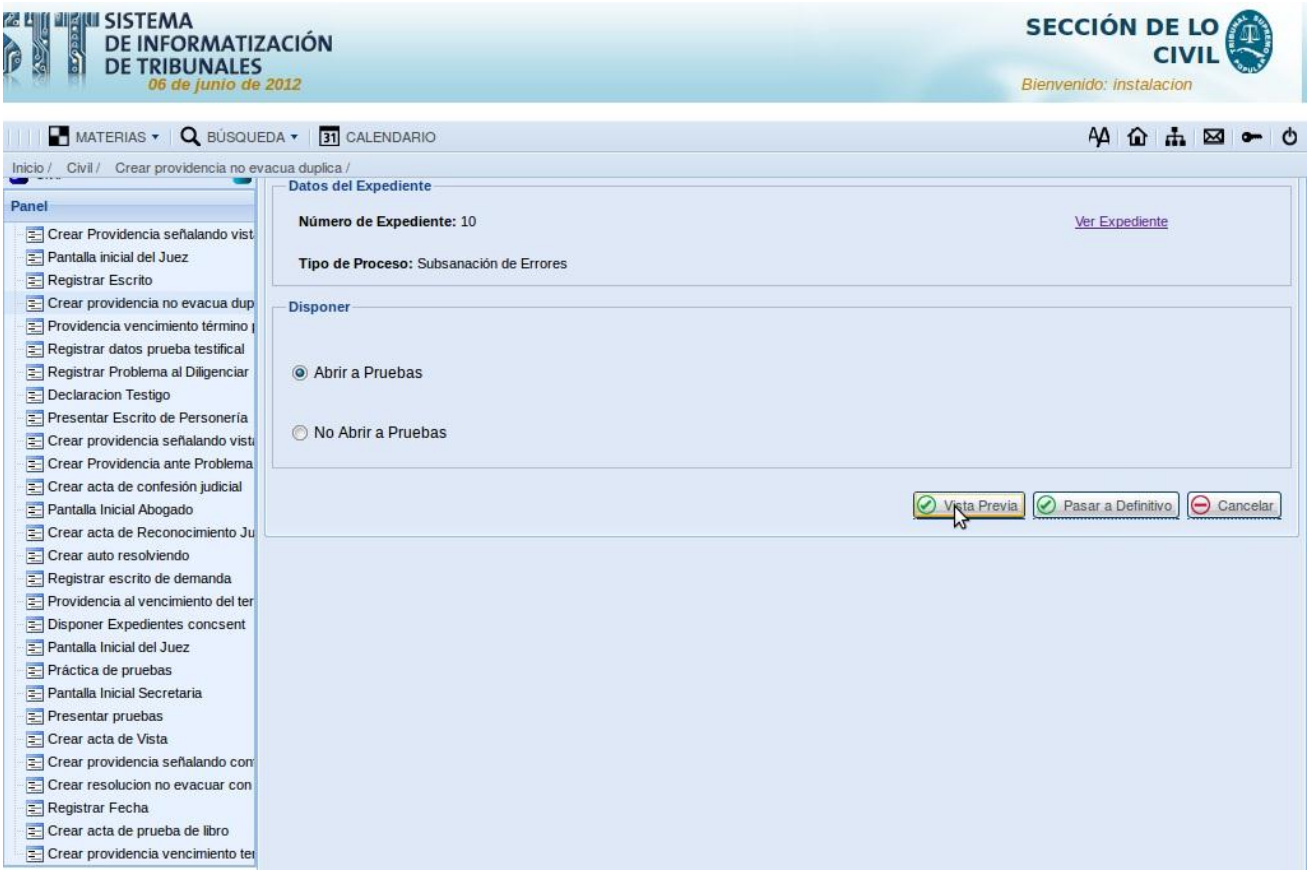

Figura 20: Interfaz gráfica del caso de uso Crear Providencia por no Evacuar Dúplica.

Conclusiones parciales

En este capítulo se presentó la solución propuesta para el sistema, mostrándose el estándar a seguir para la codificación del proyecto, la estructura organizativa de la aplicación, además de elementos que componen la arquitectura definida como los patrones de diseño utilizados para el desarrollo. Así como interfaces desarrolladas en el sistema. Los artefactos obtenidos en este capítulo son de vital importancia para la construcción del sistema. En particular:

- El Modelo del diseño permitió materializar con precisión los requerimientos del cliente y sirvió como guía para las actividades de implementación al producir una representación técnica del software desarrollado.

- El Modelo de implementación permitió obtener una visión general de los componentes y subsistemas a implementar, facilitando que el proceso de implementación se realizara de forma simple con una mayor calidad.

Capítulo 3: Análisis de resultados

Introducción

En este capítulo se evalúa el grado de calidad y fiabilidad de los resultados obtenidos en el desarrollo de este trabajo mediante la aplicación de algunas de las métricas empleadas internacionalmente para tal fin, específicamente las asociadas a las de la medición de la calidad del diseño y la implementación de software.

3.1 Valoración de la solución.

En diseño y desarrollo, la validación está relacionada con el proceso de reexaminación de un producto para determinar la conformidad con las necesidades del usuario. La validación es realizada normalmente sobre el producto final bajo condiciones operacionales definidas. La valoración de la solución es el paso final en el proceso de evaluación del software. Una solución puede ser evaluada de acuerdo a los siguientes criterios: (Ministros, 2004)

- Valoración cualitativa.
- Valoración directa.
- Valoración indirecta.

Una valoración se realiza mediante una métrica para asignar uno de los valores de una escala (el mismo puede ser número o categoría) al atributo de la entidad (sistema, subsistemas o componentes).

Valoración cualitativa*:* Es una evaluación sistemática del grado o capacidad de una entidad para satisfacer necesidades o requerimientos específicos. Además, se emplean categorías como algunos de los atributos más importantes de una entidad, ejemplo: el lenguaje de desarrollo del programa (C, C ++, C #, PHP, JAVA).

Valoración indirecta*:* Es la valoración de un atributo derivada del valor de uno o más atributos diferentes. Es la valoración externa de un atributo de un sistema, ejemplo: el tiempo de respuesta a la información alimentada por el usuario, es una valoración indirecta de los atributos del software, debido a que esta medida se verá influenciada por los atributos externos del sistema (Facilidad de mantenimiento, Confiabilidad, Eficacia, Usabilidad, Reusabilidad y Portabilidad), así como los propios internos (Facilidad de traza, Modularidad, Tolerancia a fallos, Eficiencia de ejecución, Eficiencia de almacenamiento entre otros).

Valoración directa: Es una valoración del producto de forma directa. Ejemplo: El número de líneas de código, las valoraciones de la complejidad, el número de fallas encontradas durante el proceso y el índice de señales o alertas, son todas las valoraciones internas propias del producto en sí.

Valoraciones aplicadas a la solución propuesta.

Dentro de los distintos tipos de valoración del software existentes se seleccionaron para la solución propuesta las de tipo cualitativa y directa.

3.2 Métricas de Software.

Un aspecto importante a tener en cuenta en la fase de evaluación de la calidad del diseño ha sido la creación de métricas básicas inspiradas en el estudio de la calidad del diseño orientado a objeto referenciadas por Pressman, teniendo en cuenta que este estudio brinda un esquema sencillo de implementar y que a la vez cubre los principales atributos de calidad de software. (Pressman, 2005)

Los atributos de calidad que se abarcan son (Pressman, 2005):

- **Responsabilidad.** Consiste en la responsabilidad asignada a una clase en un marco de modelado de un dominio o concepto de la problemática propuesta.
- **Complejidad de implementación.** Consiste en el grado de dificultad que tiene implementar un diseño de clases determinado.
- **Reutilización.** Consiste en el grado de reutilización presente en una clase o estructura de clase dentro de un diseño de software.
- **Acoplamiento.** Consiste en el grado de dependencia o interconexión de una clase o estructura de clase, con otras, está muy ligada a la característica de Reutilización.
- **Complejidad de mantenimiento.** Consiste en el grado de esfuerzo necesario a realizar para desarrollar un arreglo, una mejora o una rectificación de algún error de un diseño de software. Puede influir indirecta pero fuertemente en los costes y la planificación del proyecto.
- **Cantidad de pruebas.** Consiste en el número o el grado de esfuerzo para realizar las pruebas de calidad (Unidad) del producto (Componente, módulo, clase, conjunto de clases, etc.) diseñado.

Las métricas concebidas como instrumento para evaluar la calidad del diseño del Proceso Ordinario de la Materia Civil en el SIT y su relación con los atributos de calidad definidos en este trabajo son:

- Tamaño operacional de clase (TOC)

- Relaciones entre Clases(RC)

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

Tamaño operacional de clase (TOC): Está dado por el número de métodos asignados a una clase.

En la tabla que se muestra a continuación se pueden observar los atributos de calidad y el modo en que afectan a las clases medidas.

Atributo de calidad	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 del grado de reutilización de la clase.

Tabla 2: Tamaño Operacional de Clase (TOC)

El TOC será calculado por la cantidad de operaciones con las que cuenta cada una de las clases medidas. Los altos valores para la métrica TOC indican que las clases medidas deben tener bastante responsabilidad, lo que propicia un bajo nivel de reutilización de la clase y complicará la implementación y las pruebas. En general, operaciones y atributos heredados o públicos deben ser ponderados con mayor importancia cuando se determina el tamaño de clase, pues las operaciones y atributos privados, permiten la especialización. También se pueden calcular los promedios para el número de operaciones de cada clase. Cuanto más pequeño sea el valor promedio para el tamaño, será más viable para que las clases dentro del sistema puedan reutilizarse. (Lorenz y Kidd, 1994)

Luego de aplicar dicha métrica a 38 clases de la solución, la misma arrojó los siguientes resultados:

Clase	Procedi-mientos	Respon-sabilidad	Complejidad	Reutilización
RegistrarescritodemandaController	18	Alta	Alta	Baja
NtipoescritoExt	2	Baja	Baja	Alta
NestadocivilExt	1	Baja	Baja	Alta
NformacomparecerExt	2	Baja	Baja	Alta
DpersonanaturalExt	2	Baja	Baja	Alta

CrearprovidenciaduplicaController	5	Media	Media	Media
DexpedienteExt	10	Alta	Alta	Baja
DtramiteExt	2	Baja	Baja	Alta
DexpedienteTramiteExt	1	Baja	Baja	Alta
DresolucionjudicialExt	3	Baja	Baja	Alta
DexpedientetramitependienteExt	4	Baja	Baja	Alta
CrearprovnoevacreplicaController	6	Media	Media	Media
DtramiteExt	2	Baja	Baja	Alta
DexpedienteTramiteExt	1	Baja	Baja	Alta
DresolucionjudicialExt	3	Baja	Baja	Alta
DexpedientetramitependienteExt	4	Baja	Baja	Alta
CrearactavistaController	18	Alta	Alta	Baja
NCorrecionExt	2	Baja	Baja	Alta
DexpedienteExt	10	Alta	Alta	Baja
DtramiteExt	2	Baja	Baja	Alta
DexpedienteTramiteExt	1	Baja	Baja	Alta
DActaExt	1	Baja	Baja	Alta
DabogadocomparecidoExt	1	Baja	Baja	Alta
DpersonacomparecidaExt	1	Baja	Baja	Alta
DActaVistaExt	1	Baja	Baja	Alta
DpersonaExt	5	Media	Media	Media
DpersonajuridicaExt	4	Baja	Baja	Alta
DpersonanaturalExt	2	Baja	Baja	Alta
DabogadoExt	4	Baja	Baja	Alta
CrearactapruebalibroController	17	Alta	Alta	Baja
DexpedienteExt	10	Alta	Alta	Baja
DpersonaExt	5	Media	Media	Media
DpersonajuridicaExt	4	Baja	Baja	Alta
DpersonanaturalExt	2	Baja	Baja	Alta
NprocedimientoExt	2	Baja	Baja	Alta
DpruebalibroExt	2	Baja	Baja	Alta
DexpedienteTramiteExt	1	Baja	Baja	Alta
DActaExt	1	Baja	Baja	Alta

Tabla 3: Resultados de la evaluación de la métrica TOC y su influencia en los atributos de calidad (Responsabilidad, Complejidad de Implementación y Reutilización).

Para ver la especificación de los umbrales[21] que se emplean para medir la categoría (Baja, Media y Alta) ver (Anexo # 1: Instrumento de medición de la métrica Tamaño operacional de clase (TOC)).

A continuación se muestran distintas graficas con los resultados en porciento que devuelve la aplicación de la métrica TOC sobre los atributos de calidad (Responsabilidad, Complejidad y Reutilización). La figura que se muestra a continuación representa la relación de la cantidad de clases y su porciento por los intervalos de procedimiento definidos.

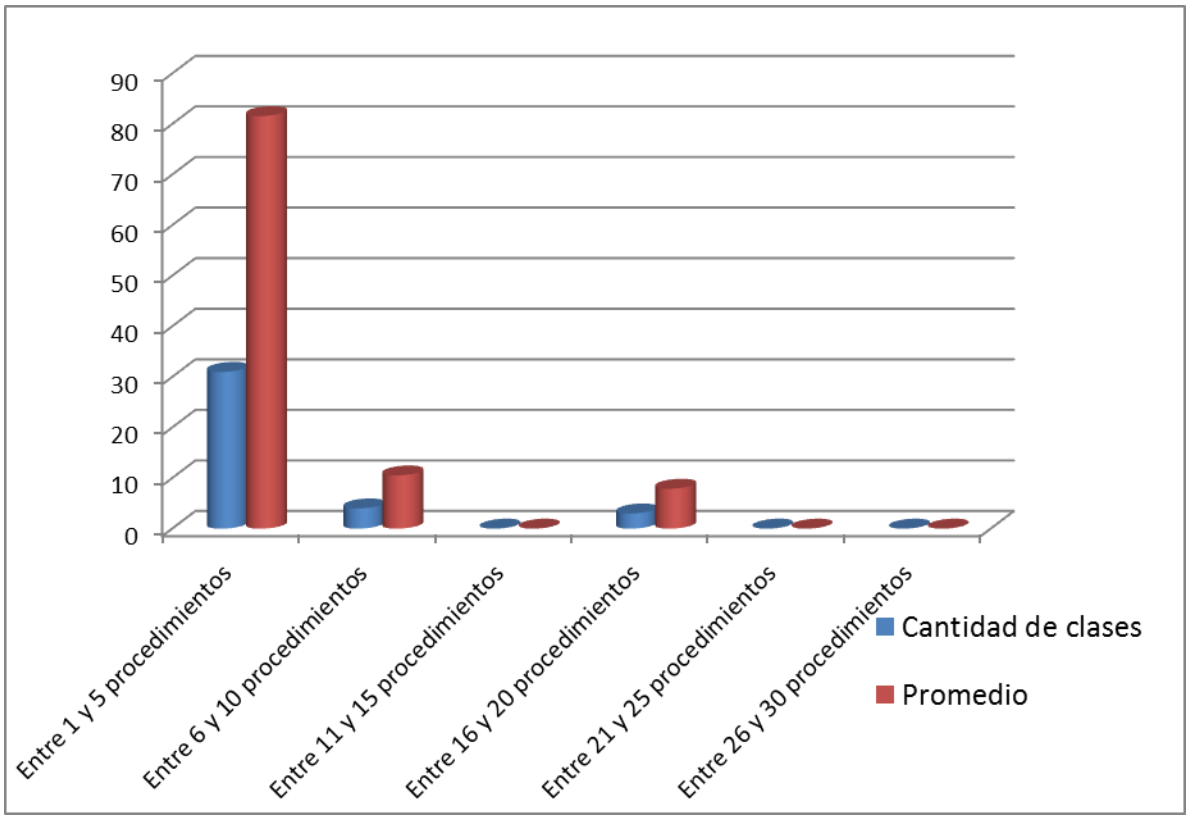

Figura 21: Representación de la Cantidad de Clases y su Promedio por los intervalos de procedimientos.

[21] Valores heurísticas usados para fijar rangos de valores deseables y no deseables de métricas, para el software medido

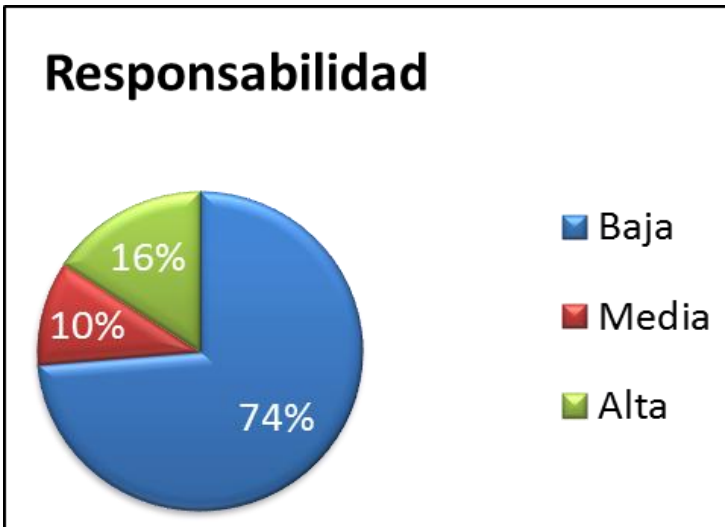

Figura 22: Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo responsabilidad.

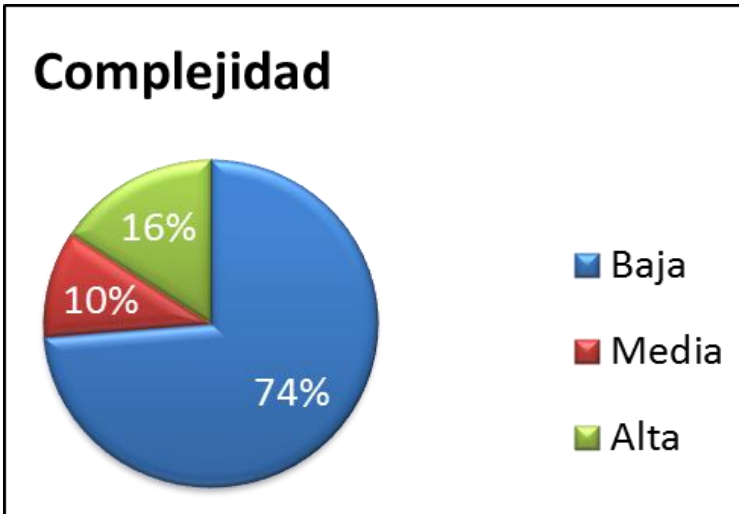

Figura 23: Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Complejidad de Implementación.

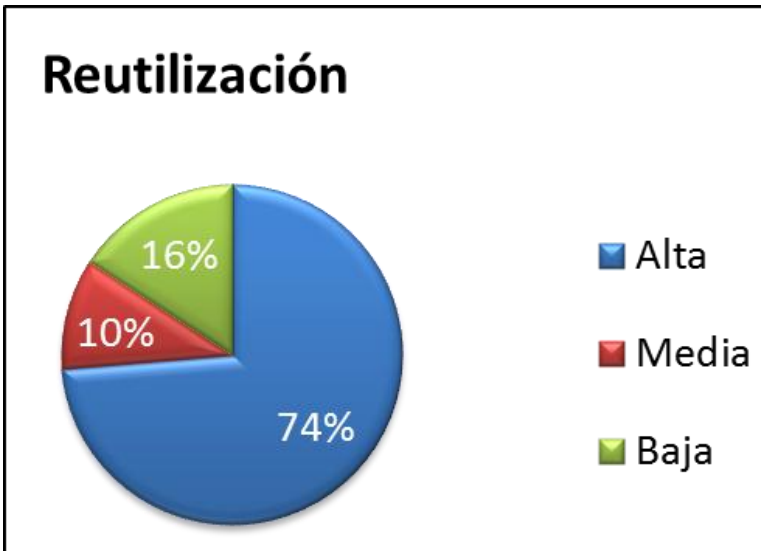

Figura 24: Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Reutilización.

Haciendo un análisis de los resultados obtenidos en la evaluación del instrumento de medición de la métrica TOC, se puede concluir que el diseño realizado para los casos de uso identificados para informatizar el proceso Ordinario de la Materia Civil tiene una buena calidad teniendo en cuenta que se obtuvieron resultados satisfactorios en los diferentes atributos de calidad medidos, estos resultados van unidos al hecho de que el 92% de las 38 clases medidas (35 clases) tienen 10 o menos operaciones. Por lo que se puede afirmar que se obtuvo una solución eficiente con altos niveles de reutilización (74%) lo que garantiza que las operaciones realizadas puedan servir de ayuda a otros subsistemas de la aplicación. También se evidencia que el diseño realizado fue correcto ya que se obtuvieron bajos niveles de responsabilidad (74%) y complejidad de implementación (74%), pues se le asignaron correctamente las responsabilidades a las clases involucradas en la solución.

3.2.2 Resultados del instrumento de evaluación de la métrica Relaciones entre Clases (RC).

El resultado de la aplicación de la métrica **RC** está dado por el número de relaciones de uso de una clase con otras. La aplicación de dicha métrica permite evaluar atributos de calidad como el Acoplamiento, la Complejidad de mantenimiento, la Reutilización y la Cantidad de pruebas. De forma tal que mientras mayor sea las relaciones de uso entre clases mayor será el Acoplamiento, la Complejidad de Mantenimiento y la Cantidad de pruebas, mientras que su Reutilización disminuye. (Lorenz y Kidd, 1994)

En la tabla que se muestra a continuación se pueden observar los atributos de calidad y el modo en que afectan a las clases medidas.

Atributo de calidad	Modo en que lo afecta
Acoplamiento	Un aumento del RC implica un aumento del Acoplamiento de la clase.
Complejidad de mantenimiento	Un aumento del RC implica un aumento de la complejidad del 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.

Tabla 4: Relaciones entre Clases (RC).

Luego de aplicar dicha métrica a 16 clases de la solución, la misma arrojó los siguientes resultados:

Clase	Cantidad de Relaciones de Uso	Acopla-miento	Complejidad Mant.	Reutili-zación	Cantidad de Pruebas
RegistrarescritodemandaController	16	Alto	Alta	Baja	Alta
NtipoescritoExt	0	Ninguno	Baja	Alta	Baja
NestadocivilExt	0	Ninguno	Baja	Alta	Baja
CrearprovidenciaduplicaController	5	Alto	Media	Media	Media
DexpedienteExt	1	Bajo	Baja	Alta	Baja
DtramiteExt	2	Medio	Baja	Alta	Baja
CrearactavistaController	14	Alto	Alta	Baja	Alta
NCorrecionExt	0	Ninguno	Baja	Alta	Baja
DpersonajuridicaExt	1	Bajo	Baja	Alta	Baja
CrearactapruebalibroController	12	Alto	Alta	Baja	Alta
DpersonanaturalExt	1	Bajo	Baja	Alta	Baja
DpersonaExt	1	Bajo	Baja	Alta	Baja
DpruebalibroExt	1	Bajo	Baja	Alta	Baja
DActaExt	1	Bajo	Baja	Alta	Baja
DexpedienteTramiteExt	2	Medio	Baja	Alta	Baja
NprocedimientoExt	0	Ninguno	Baja	Alta	Baja

Tabla 5: Resultados de la evaluación de la métrica RC y su influencia en los atributos de calidad utilizados en esta métrica.

Para ver la especificación del umbral que se emplea para medir la categoría (Baja, Media y Alta) ver (Anexo # 2: Instrumento de medición de la métrica Relaciones entre Clases (RC)).

A continuación se muestran distintas graficas con los resultados en porciento que devuelve la aplicación de la métrica RC sobre los atributos de calidad anteriormente mencionados. La figura que se muestra a continuación representa la relación de la cantidad de clases y su porciento por la cantidad de dependencias definidas.

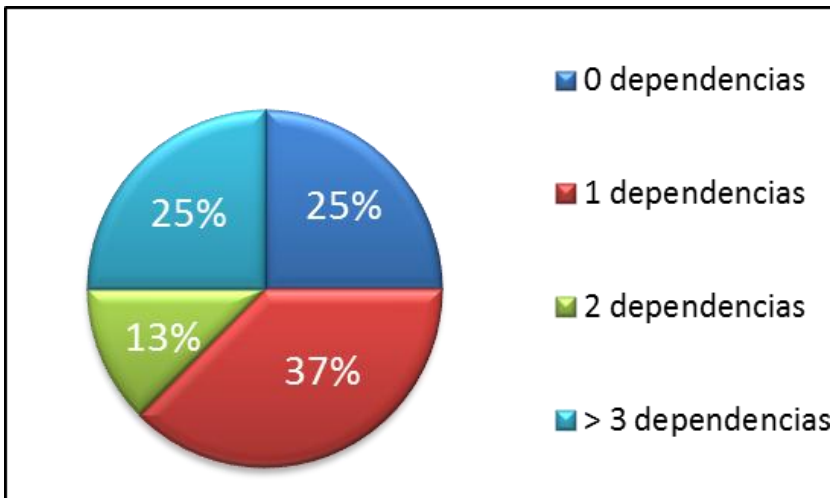

Figura 25: Representación en % de los resultados obtenidos en el instrumento agrupando la cantidad de clases por cantidad de dependencias.

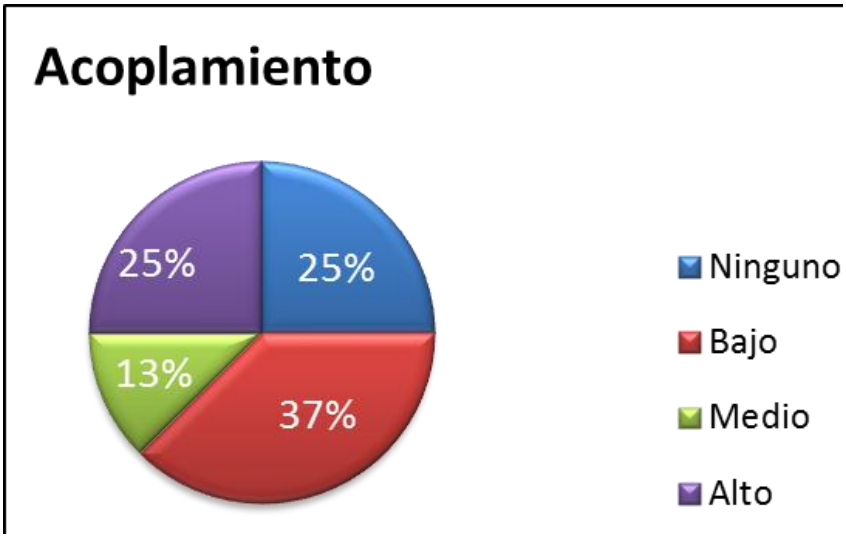

Figura 26: Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Acoplamiento.

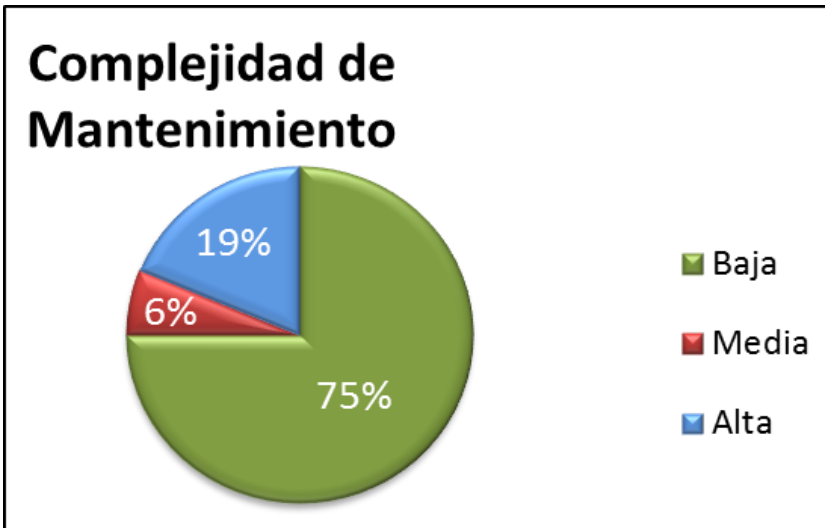

Figura 27: Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Complejidad de Mantenimiento.

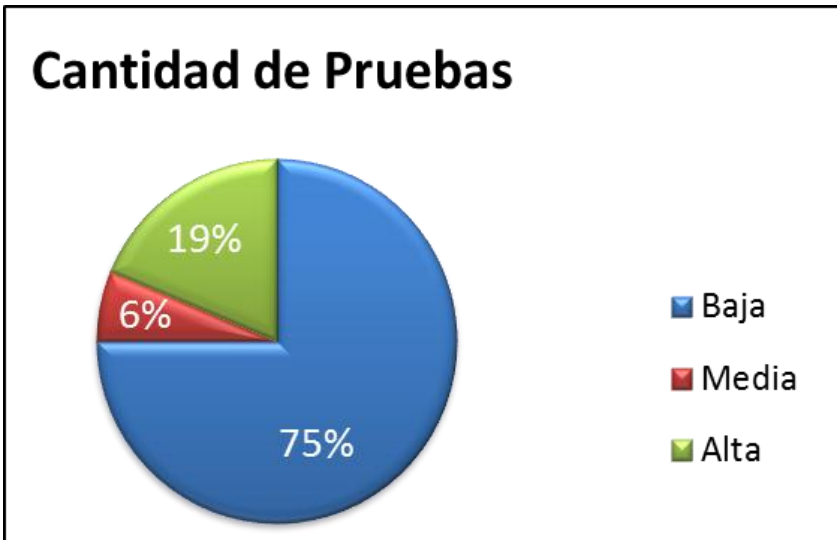

Figura 28: Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Cantidad de Pruebas.

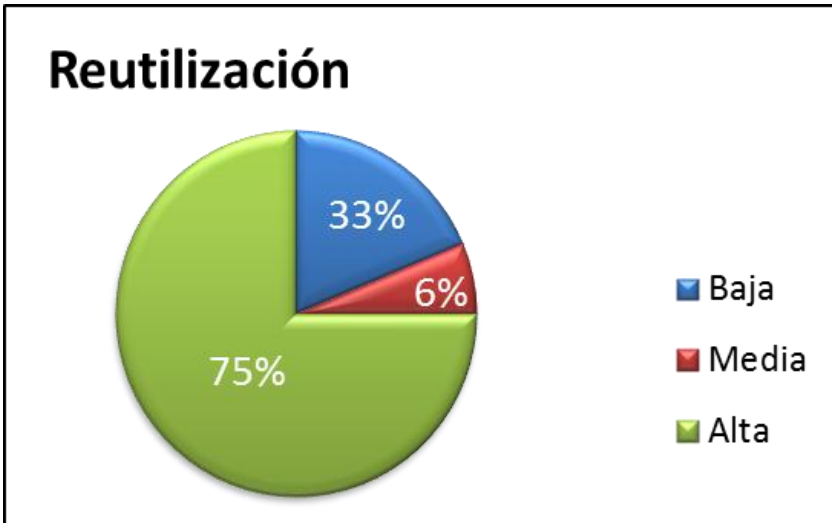

Figura 29: Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Reutilización.

Haciendo un análisis de los resultados obtenidos en la evaluación del instrumento de medición de la métrica **RC**, se puede concluir que el diseño realizado para los casos de uso identificados para informatizar el proceso Ordinario de la Materia Civil tiene una buena calidad teniendo en cuenta que el 75% de las clases incluidas en este subsistema posee 2 o menos dependencias de otras clases. Esto favorece el hecho que al ocurrir un cambio de alguna de las clases, la afectación en las restantes sea de poca importancia. El 62% de las clases poseen bajos índices en cuanto a Acoplamiento, lo cual nos demuestra que una clase no depende de muchas clases solo de las necesarias, vale destacar que las clases que presentan mayor índice de acoplamiento son las clases Controladoras ya que son las clases que manejan el control de la lógica del negocio. También la aplicación de la métrica RC arrojó resultados satisfactorios en el atributo Complejidad de mantenimiento con un 75% de las clases con índices bajos, lo que facilita las tareas de corrección, modificación y mantenimiento de las clases. El número o el grado de esfuerzo para realizar las pruebas de calidad del producto (clase, conjunto de clases, componente, módulo, entre otros) diseñado es bajo dado que la métrica aplicada nos arroja un 75% de baja Cantidad de pruebas fomentando así un alto índice de Reutilización con un 75%.

3.3 Pruebas de Caja Blanca o Estructurales.

Al sistema desarrollado se le aplicó la prueba de caja blanca, específicamente la técnica de prueba del camino básico, pues se desea evaluar la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución. Esta prueba permite garantizar que en los casos de prueba obtenidos a través del camino básico se ejecute cada sentencia del programa por lo menos una vez.

A continuación se analizan y enumeran las sentencias de código de uno de los métodos contenidos en la clase RegistrarescritodemandaController, específicamente: cargarGridDemandantesAction, este por su parte se encarga de adicionar en una sesión los demandantes que se encuentran en base de datos para luego cargarlos y mostrarlos en el GridDemandantes. Vale señalar que solo se muestra el resultado de una de las pruebas realizadas a un método en específico, con el objetivo de mostrar cómo es que se realiza este tipo de procedimiento.

```
function cargarGridDemandantesAction() {
    $form = json_decode(stripcslashes($this->_request->get('formulario5')));1
    $datos = array();1
    if (isset($_SESSION['iddemandante'])) {2
        $con = count($_SESSION['iddemandante']);3
        for ($i = 0; $i < $con; $i++) {4
            $idpersona = $_SESSION['iddemandante'][$i]['iddemandante'];5
            $persona = Doctrine::getTable('Dpersona')->find($idpersona);5
            //Si es una persona natural
            if ($persona->idnpersona == 2) {6
                $personanatural = Doctrine::getTable('Dpersonanatural')->find($idpersona);7
                $dato['segapelldte'] = $personanatural->primernombre . ' ' . $personanatural->segundonombre; 7
                $pasaporte = 'PASAPORTE:' . $personanatural->pasaporte;7
                $ci = 'CI:' . $personanatural->ci;7
                $dato['idpersona'] = $persona->idpersona;7
                $dato['idnpersona'] = $persona->idnpersona;7
            }
            else
            {
                $personanajuridica = Doctrine::getTable('Dpersonajuridica')->find($idpersona);8
                $dato['segapelldte'] = $personanajuridica->nombre;8
                $dato['infDte'] = $personanajuridica->organismo;8
            }
            array_push($datos, $dato);9
        }
    }
    $result = array('cantidad_filas' => count(datos), 'datos' => $datos);10
    echo json_encode($result);10
    return;10
}
```

Figura 30: Método cargarGridDemandantesAction.

Después de este paso, es necesario representar el grafo de flujo asociado al código antes presentado a través de nodos, aristas y regiones, en ese caso:

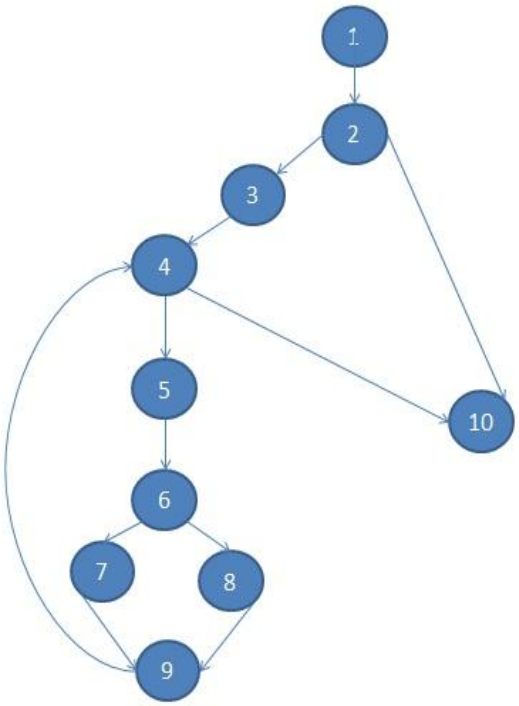

Figura 31: Grafo de flujo asociado al algoritmo cargarGridDemandantesAction.

Una vez construido el grafo de flujo asociado al procedimiento anterior se determina la complejidad ciclomática la cual es una métrica de software extremadamente útil pues 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 da un límite superior para el número de pruebas que se deben realizar para asegurar que se ejecute cada sentencia al menos una vez. Para realizar el cálculo de la complejidad ciclomática del código es necesario efectuarlo mediante tres vías o fórmulas de manera tal que quede justificado el resultado, siendo el mismo en cada caso:

1. V (G) = (A – N) + 2

Siendo “A” la cantidad total de aristas y “N” la cantidad total de nodos.

V (G) = (12 -10) + 2

V (G) = 4.

2. V (G) = P + 1

Siendo “P” la cantidad total de nodos predicados (son los nodos de los cuales parten dos o más aristas).

V (G) = 3 + 1

V (G) = 4.

3. V (G) = R

Siendo “R” la cantidad total de regiones, se incluye el área exterior del grafo, contando como una región más.

V (G) = 4

El cálculo efectuado mediante las fórmulas antes presentadas muestran una complejidad ciclomática de valor 4, de manera que existen 4 posibles caminos por donde el flujo puede circular, este valor representa el número mínimo de casos de pruebas para el procedimiento tratado.

Seguidamente, es necesario especificar los caminos básicos que puede tomar el algoritmo durante su ejecución.

Camino básico #1: 1 – 2 – 10

Camino básico #2: 1 – 2 – 3 – 4 – 5 –6 – 7 – 9 – 4 – 10

Camino básico #3: 1 – 2 – 3 – 4 – 5 –6 – 8 – 9 – 4 -- 10

Se procede a ejecutar los casos de pruebas para cada uno de los caminos básicos determinados en el grafo de flujo. Para definir los casos de prueba es necesario tener en cuenta:

Descripción: Se describe el caso de prueba y de forma general se tratan los aspectos fundamentales de los datos de entrada.

Condición de ejecución: Se especifica cada parámetro para que cumpla una condición deseada y así ver el funcionamiento del procedimiento.

Entrada: Se muestran los parámetros que serán la entrada al procedimiento.

Resultados Esperados: Se expone el resultado esperado que debe devolver el procedimiento después de efectuado el caso de prueba.

Las descripciones y las condiciones de ejecución coincidirán en cada uno de los casos de prueba.

1- Caso de prueba para el camino básico # 1.

Descripción: Los datos de entrada cumplirán con los siguientes requisitos:

- El objeto SESSION no está seteado[22].

Condición de ejecución: Para el algoritmo es necesario que en el objeto SESSION en la posición “iddemandante” no haya datos.

[22] Determina si una variable está definida y no es **NULL**.

Entrada: El objeto SESSION no está seteado, es decir no contiene ningún demandante.

Resultados esperados: Teniendo en cuenta los datos pasados por parámetro se espera que el gridpanel salga sin datos.

El resultado obtenido fue correcto.

2- Caso de prueba para el camino básico # 2.

Descripción: Los datos de entrada cumplirán con los siguientes requisitos:

- El objeto SESSION está seteado, es decir contiene al menos 1 demandante.

Condición de ejecución: Para el algoritmo es necesario que en el objeto SESSION en la posición "iddemandante" existan datos.

Entrada: En el objeto SESSION existen datos.

Resultados esperados: Teniendo en cuenta los datos pasados por parámetro se espera que en el gridpanel de demandantes salgan los demandantes que se encuentran previamente guardados en el objeto SESSION.

El resultado obtenido fue correcto.

2- Caso de prueba para el camino básico # 3.

Descripción: Se deben cargar en el gridpanel Demandantes los datos cargados desde el objeto SESSION.

Condición de ejecución: Para el algoritmo es necesario que en el objeto SESSION en la posición "iddemandante" existan datos.

Entrada: En el objeto SESSION existen datos.
Resultados esperados: Teniendo en cuenta los datos pasados por parámetro se espera que en el gridpanel de demandantes salgan los demandantes que se encuentran previamente guardados en el objeto SESSION.

El resultado obtenido fue correcto.

3.4 Pruebas de Caja Negra o Funcionales.

Para la realización de las pruebas de caja negra se empleó la técnica Partición de Equivalencia que representa una de las más efectivas pues permite examinar los valores válidos e inválidos de las entradas existentes en el software. Un caso de prueba ideal descubre de forma inmediata una clase de errores (por ejemplo, procesamiento incorrecto de todos los datos de caracteres) que de otro modo requerirían la ejecución de muchos casos antes de detectar el error genérico. Dirige la definición de casos de pruebas que descubran clases de errores, reduciendo el número de clases de prueba que hay que desarrollar. (Pressman, 2002)

Los datos de los casos de prueba correspondientes a cada uno de los casos de uso desarrollados se presentan como uno de los artefactos generados de la presente investigación. Las pruebas fueron aplicadas por el equipo de calidad del proyecto de informatización de los TPC y durante la ejecución de las mismas se realizaron tres iteraciones a cada uno de los casos de uso. Entre los errores detectados se encuentran errores ortográficos, de validación de interfaces, de correspondencia con la documentación, de funcionalidad y de carga de datos, los cuales fueron corregidos y en una tercera iteración no se detectó ninguno. Los resultados de cada una de estas iteraciones se aprecian en la siguiente gráfica. Resulta importante resaltar que los casos de pruebas presentados no son los únicos diseñados para la solución propuesta sino que el objetivo es plasmar la forma en que se realizaron los mismos.

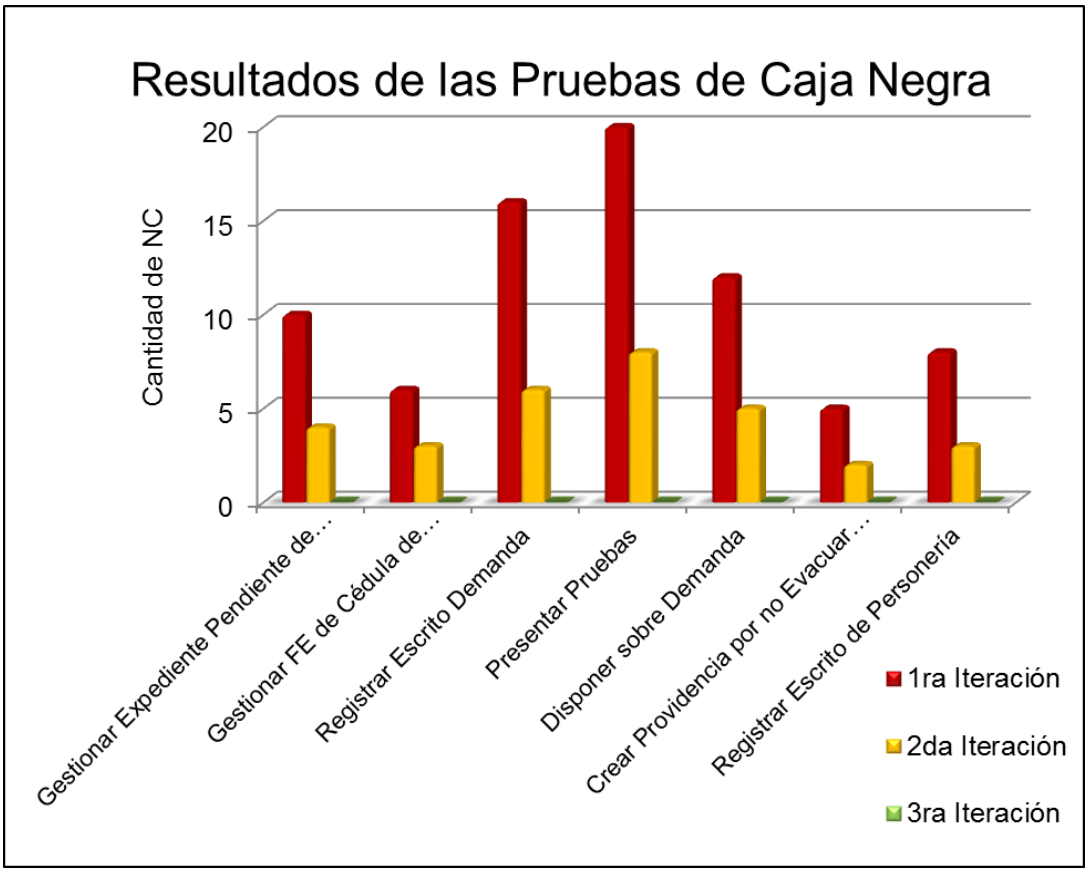

Figura 32: Gráfica que muestra la cantidad de no conformidades detectadas en algunos casos de uso por iteración.

Conclusiones parciales

En el desarrollo de este capítulo se efectuó una valoración del diseño propuesto mediante métricas las cuales arrojaron buenos resultados en cuanto a los atributos de calidad medidos como son: la Complejidad de Implementación y Reutilización en la métrica TOC y en los atributos Acoplamiento, Cantidad de Pruebas y Reutilización de la métrica RC. Además, se realizaron diferentes pruebas a la solución propuesta. Se realizó una aplicación de las pruebas de Caja Blanca y Caja Negra, la ejecución de las mismas arrojaron buenos resultados, ya que en el caso de las pruebas de Caja Negra se detectaron un conjunto de no conformidades a las interfaces del sistema que luego fueron corregidas en una segunda y tercera iteración.

Conclusiones

La realización de la presente investigación propició un estudio de las herramientas definidas por el equipo de arquitectura del proyecto de informatización de los TPC , dentro de las cuales se encuentra el Marco de Trabajo Sauxe en conjunto al paradigma de Programación Orientada a Objetos y las ventajas que ofrece el lenguaje de programación PHP.

La elaboración del Modelo de Diseño el cual incluye los Diagramas de Clases del Diseño y los Diagramas de Secuencia, y del Modelo de Implementación que a su vez cuenta con los Diagramas de Componentes y Despliegue se orientaron tanto al desarrollador, como a los arquitectos y clientes en la organización, dando paso a la implementación del proceso Ordinario de la Materia Civil en el SIT.

Finalmente se realizaron un conjunto de pruebas a la solución propuesta para comprobar la calidad de la misma. Las pruebas realizadas fueron pruebas de caja negra y caja blanca, además se evaluaron un conjunto de atributos de calidad a través de métricas que arrojaron resultados satisfactorios.

Recomendaciones

Con la realización de este trabajo se recomienda:

1. Comenzar con el diseño y la implementación luego de identificar los casos de uso del Proceso Ordinario de la Materia Civil para instancias superiores.

2. Comenzar con el análisis, diseño y posterior implementación de los restantes procesos de la Materia Civil.

3. Tomar esta investigación como material de estudio para la realización de aplicaciones similares.

Referencias Bibliográficas

1. **Alfonso, Jorge. 2010.** Tratando de entenderlo: Patrones de Diseño: Singleton. [En línea] 2010. [Citado el: 5 de 2 de 2012.] http://tratandodeentenderlo.blogspot.com/2010/01/patrones-de-diseno-singleton.html.
2. **Anabel Gómez Albear, Juan José Rodríguez Caballero. 2011.** *Ingeniería de Requisitos del Proceso Ordinario para el subsistema Civil de la solución informática SIT.* Ciudad de la Habana : s.n., 2011.
3. **Arquitectura de Software. 2010.** *Arquitectura de Software del Sistema de Informatización de los Tribunales Populares Cubanos, Versión 1.0. Dirección de Calidad de la Infraestructura Productiva.* 2010.
4. **Bahit, Eugenia. 2011.** POO y MVC en PHP. [En línea] Julio de 2011. http://eugeniabahit.blogspot.com/2011/07/poo-y-mvc-en-php.html.
5. **Budd, Timothy. 1991.** *An introduction to Object-Oriented Programming.* 1991. ISBN:0-201-54709-0.
6. **DesarrolloWeb. 2007.** DesarrolloWeb.com. [En línea] 2007. [Citado el: 29 de Enero de 2012.] www.desarrolloweb.com/php/.
7. **Doctrine. 2010.** Manual de Doctrine para php. [En línea] Febrero de 2010. http://phpdoc.org/.
8. **doctrine-project. 2008.** *Doctrine: ORM Open Source para PHP 5.2.* 2008.
9. **Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. 1995.** *Design Patterns: Elements of Reusable Object-Oriented Software.* 1995. ISBN 0-201-63361-2.
10. **Eugenia, Bahit.** *POO y MVC en PHP.*
11. **ExtJS. 2011.** IDATI:Ext js Framework javascript/ajax poderoso para aplicaciones RIA. [En línea] 2 de Marzo de 2011. [Citado el: 27 de Enero de 2012.] http://www.idati.cl/2011/03/02/ext-js-framework-javascriptajax-poderoso-para-aplicaciones-ria/.
12. **Flanagan, David. 2001.** *The Definitive Guide.* 4ta Edición. s.l. : O'Reilly Media, 2001.
13. **Foundation. 2011.** The Apache Software. *Documentación del Servidor HTTP Apache 2.0.* [En línea] 2011. [Citado el: 10 de Enero de 2012.] http://httpd.apache.org/docs/2.0/es/.
14. **Gómez Baryolo, Oiner. 2010.** *Marco de Trabajo SAUXE.* Habana : s.n., 2010.
15. **Guerra, Prof: Lic. Roberto. 2010.** Scribd Inc. [En línea] 20 de Mayo de 2010. [Citado el: 10 de Enero de 2012.] http://es.scribd.com/doc/31440864/Metodologia-RUP.
16. **Guillerón, Gastón. 2009.** Gln-Blog. [En línea] 2009. [Citado el: 16 de 2 de 2012.] http://glnconsultora.com/blog/?p=3.
17. **Jacobson, Ivar and y otros. 1992.** *Object-Oriented Software Engineering—A Use Case.* s.l. : Addison-Wesley, 1992.
18. **KevinZhang.** *Design Patterns.Elements of Reusable Object-Orinted Software.*
19. **La web del programador. 2011.** PostgreSQL PE | La base de datos libre mas avanzada del mundo. [En línea] 15 de Enero de 2011. [Citado el: 2 de Febrero de 2012.] www.lawebdelprogramador.com.
20. **LARMAN, Craig.** UML y Patrones. [En línea] [Citado el: 20 de Enero de 2012.] http://bibliodoc.uci.cu....
21. **Larman, Craig. 2005.** *UML y Patrones.Introducción al análisis y diseño orientado a objetos.* s.l. : Prentice Hall, 2005.
22. **Lorenz y Kidd. 1994.** *Object Oriented Metrics.* Englewood : Prentice Hall, 1994.

23. **MeRinde. 2012.** MeRinde. [En línea] 2012. http://merinde.net/index.php?option=com_content&task=view&id=137&Itemid=192.
24. **Microsoft. 2008.** ¿Qué es la Arquitectura de Software? [En línea] 2008. [Citado el: 16 de 2 de 2012.] www.arqsoft.com.
25. **Ministros, Presidencia del Consejo de. 2004.** ONGEI. [En línea] 28 de Mayo de 2004. [Citado el: 10 de Marzo de 2012.] http://www.ongei.gob.pe/bancos/banco_normas/archivos/Guia-Evaluacion-SW.pdf.
26. **Montané Izaguirre, Ing Juan Carlos. 2010.** *Plan de Desarrollo de Software.* Habana : s.n., 2010.
27. **Morín, Arley Triana. 2009.** Blog de Arley Triana Morín. [En línea] 22 de Julio de 2009. [Citado el: 3 de Febrero de 2012.] http://arleytriana.blogspot.com/2009/07/implementacion-del-patron-clasico-de.html.
28. **Muñóz, Dra.Coral. 2008.** *MÉTRICAS DE LA CALIDAD DEL SOFTWARE.* La Mancha : s.n., 2008.
29. **netbeans.org. 2008.** *NetBeans un Entorno de Desarrollo Integrado.* 2008.
30. **Presman, Roger S. 2007.** *Ingeniería de Software: un enfoque práctico.* 6ta Edición. Nueva York : Editorial McGraw-Hill, 2007.
31. **Pressman. 2002.** *Ingenieria_del_Software._Un_enfoque_practico.* 5ta. 2002.
32. **Pressman, Roger.S.** *La ingeniería de software un enfoque práctico.* Quinta Edición. s.l. : Editorial:Mc Graw Hill.
33. **Pupo, Yanisledy Canete. 2010.** *Libro de Ayuda del Marco de Trabajo Sauxe.* Ciudad de la Habana : s.n., 2010.
34. **Rosenberg, Stapko and Gallo. 1999.** MANUAL DE ESTILO DE PROGRAMACIÓN. [En línea] 24 de Enero de 1999. [Citado el: 25 de Febrero de 2012.] www.ahristov.com/taller/blog/Estilo-de-Programacion-9-01.pdf -.
35. **Rumbaugh, et al. 2000.** *El Proceso Unificado de Desarrollo de Software.* s.l. : Addison Wesley, 2000.
36. **RUP. 2010.** La tecla de escape. [En línea] 17 de Octubre de 2010. [Citado el: 12 de Enero de 2012.] http://latecladeescape.com/ingenieria-del-software/metodologias-de-desarrollo-del-software.html.
37. **Stapko, Rosenberg and Gallo. 1999.** MANUAL DE ESTILO DE PROGRAMACIÓN. [En línea] 24 de Enero de 1999. [Citado el: 4 de Febrero de 2012.] www.ahristov.com/taller/blog/Estilo-de-Programacion-9-01.pdf.
38. **Sussman, Collins y y otros. 2008.** Control de versiones con Subversion. [En línea] 2008. [Citado el: 15 de Enero de 2012.] http://svnbook.red-bean.com/nightly/es/svn-book.pdf.
39. **Tigris.org. 2001.** Tigris.org. [En línea] 2001. [Citado el: 15 de Enero de 2012.] http://subversion.tigris.org/.
40. **VisualParading. 2004.** *Visual Paradigm for UML.* 2004.
41. **Welicki, León. 2012.** MSDN. [En línea] 2012. http://msdn.microsoft.com/es-es/library/bb972272.aspx.
42. **Yadira Piñera Andux. 2010.** *Trabajo de Diploma :Formalización y estandarización de la documentación técnica de la arquitectura tecnológica del Marco de Trabajo Sauxe versión 2.0.* Universidad de las Ciencias Informáticas, Ciudad de la Habana : s.n., 2010.