Universidad de las Ciencias Informáticas

Facultad 3

Diseño e Implementación de una solución informática para el proceso Administrativo en los Tribunales Provinciales Cubanos.

Trabajo de Diploma para optar por el Título de Ingeniero en Ciencias Informáticas

Autor:

Alex Gómez Gutiérrez

Tutor:

Ing. Aray Pérez Deguez

Asesora:

Ing. Adilaraima Martínez Barrio

Ciudad de la Habana, Cuba

Curso: 2010-2011

Declaración de autoría

Declaro ser autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo. Para que así conste se firma la presente a los ____ días del mes de junio del año 2011.

_____ _____

Alex Gómez Gutiérrez
Autor

Ing. Aray Pérez Deguez
Tutor

Dedicatoria

Dedico este trabajo a mi familia que es lo más importante que hay en mi vida, en especial a mis padres por todo su apoyo y dedicación, y a los amigos que han estado en las buenas y en las malas.

Agradecimientos

Primeramente agradecer a las dos personas que son mi razón de ser y que son los responsables de que hoy este aquí, a mis padres, gracias por su apoyo, dedicación y confianza depositadas en mí.

A toda mi familia, a mis abuelos, tíos, primos, a mi hermana, gracias por estar ahí siempre que los necesité.

A los profesores del proyecto TPC, Aray que fue mi tutora, Adilaraima, Lisset, Jon, Misael, Maikel, Juan Carlos, en fin a todos los que me ayudaron en la realización de la tesis y con los cuales aprendí.

Al profesor Alain Eduardo Rodríguez mi primer profesor de programación y guía de grupo, que siempre ha estado ahí para brindarme su apoyo.

A los amigos que he hecho en la UCI, a los del grupo 4102 cuando empezamos este largo recorrido, con los cuales pase momentos buenos y malos.

Al piquete de la "Caliente", con los cuales nunca nos perdimos una fiesta y siempre estuvieron ahí para lo que hiciera falta.

Resumen

Entre los procesos que se llevan a cabo en los Tribunales Populares Cubanos se encuentra el Proceso Administrativo, el cual en término judicial se le denomina a la sucesión de trámites mediante los cuales una persona que se considere afectada por una decisión o acto de la administración pueda reclamar ante la Sala de lo Administrativo del Tribunal Provincial. Actualmente este proceso presenta dificultades en su tramitación, como son la introducción de errores y tachaduras en los documentos legales, demora en el cumplimiento de los términos así como en la búsqueda de un determinado expediente. El presente trabajo plantea una estrategia de cómo eliminar los problemas que existen en dicho proceso, a partir del diseño e implementación de una solución de software que satisfaga las necesidades evidentes. En la presente investigación se realiza una descripción de las principales tendencias y tecnologías usadas actualmente para el desarrollo de software, haciendo énfasis en las escogidas para el sistema que se propone como solución. Se exponen además los artefactos generados durante el diseño y la implementación del sistema sobre la base de la arquitectura planteada para la ejecución del mismo. Cada uno de estos artefactos se evalúa por medio de métricas mundialmente usadas mostrándose los resultados obtenidos. Con esta solución, los Tribunales Provinciales Cubanos contarán con una herramienta que permita una mejor calidad y celeridad en la tramitación del proceso Administrativo.

Tabla de contenido

Introducción

En el mundo actual el avance tecnológico de la computación ha alterado la vida cotidiana de la sociedad y ha cambiado los modos de proceder e inclusive de vivir. Las computadoras se han convertido en una herramienta indispensable para la realización de tareas y la solución a problemas, tanto así, que sin ellas en la actualidad muchas de las actividades colapsarían.

Las Tecnologías de la Información y las Comunicaciones (TIC) han penetrado en todos los ámbitos de la vida humana, estando presente en tiendas comerciales, universidades, hospitales, centros deportivos, etc.

Los tribunales no quedan excluidos, aplicando la informática Jurídica. Es necesario resaltar que la informatización de los órganos de justicia tiene una influencia positiva en los mismos, permitiendo aumentar la celeridad en los procesos judiciales y además, agilizar los asuntos de carácter jurídico administrativos.

El Poder Judicial es aquel poder del Estado que, de conformidad al ordenamiento jurídico, es el encargado de administrar justicia en la sociedad, mediante la aplicación de las normas jurídicas, en la resolución de conflictos. Por "poder", en el sentido de poder público, se entiende a la organización, institución o conjunto de órganos del Estado, que en el caso del Poder Judicial son los órganos judiciales o jurisdiccionales: juzgados y tribunales, que ejercen la potestad jurisdiccional y suelen gozar de imparcialidad y autonomía

En Cuba, el respeto y la garantía de los derechos ciudadanos, así como la proporcionalidad entre el delito y la sanción, son principios que rigen la administración de justicia, la cual está reinada por la imparcialidad, la colegiación y la participación popular. Los Tribunales constituyen un sistema de órganos estatales. La función de impartir justicia dimana del pueblo y es ejercida en su nombre por **(Sáez, 2008)**:

- ✓ El Tribunal Supremo Popular.
- ✓ Los Tribunales Provinciales Populares.
- ✓ Los Tribunales Municipales Populares

Cada uno cuenta con diferentes procesos distribuidos en cinco materias. Dichas materias son:

- ✓ Materia Administrativa.
- ✓ Materia Civil.
- ✓ Materia Económica.
- ✓ Materia Laboral.
- ✓ Materia Penal.

La informática es hoy, en Cuba, una vía de desarrollo social y económico, en la cual la industria de software desempeña un papel primordial. La Universidad de las Ciencias Informáticas (UCI), forma parte del mecanismo que se definió para lograr esta meta, pues sus productos están insertándose en el mercado nacional e internacional.

Cuba inició, a partir de 1970, un proceso de perfeccionamiento, modernización y sistematización de su sistema jurídico. Este proceso complejo y continuo, se dirigió a todas las ramas del derecho y puede afirmarse hoy que se ha creado un coherente conjunto de normas que son aplicables en la esfera judicial.

Los Tribunales Populares Cubanos (TPC) iniciaron un proceso de digitalización de su sistema judicial con miras a agilizar los trámites judiciales al reducir los términos en un 25 por ciento, elevar la calidad de los procesos y disminuir en un 30 por ciento el empleo de los recursos humanos**(EP-Internacional, 2010)**. Como parte de estas transformaciones y con el objetivo de posibilitar la rápida localización de los asuntos, el control y alerta de vencimiento de los términos, la ágil reproducción de resoluciones y la estandarización de los modelos utilizados en los juzgados, se concibió el proyecto TPC el cual le fue asignado a un grupo de profesores y estudiantes de la Facultad 3 de la UCI.

El proyecto está constituido por siete subsistemas o módulos como comúnmente se le conocen, entre los que se encuentra el subsistema Administrativo que se encarga de resolver los conflictos que se derivan por la inconformidad ante actos administrativos dictados por los órganos administrativos y que tienen prevista la reclamación en la vía judicial.

Actualmente, durante la tramitación del procedimiento Administrativo, los datos de los escritos son registrados manualmente en los libros de presentación de escritos, lo que provoca que el trabajo se prolongue y en muchas ocasiones no dé tiempo a tramitarlos en el término establecido, además que trae como consecuencias la introducción de

errores de repetición en el número de los asientos de los escritos, tachaduras, saltos en los espacios de las anotaciones, borraduras y problemas con el ahorro del papel como recurso. Muchos documentos son generados varias veces a causa del proceso manual, pues impide la tenencia de registros actualizados por la demora del llenado de datos. La búsqueda de un determinado expediente se torna complicada por la gran cantidad de archivos guardados, a pesar de que, en los estantes se guardan de manera organizada por año.

Con la implantación del nuevo sistema informático no se eliminará por completo la utilización de los archivos físicos. Los expedientes antiguos se seguirán utilizando, uso que se irá haciendo paulatinamente menor en cuanto el sistema vaya ganando en confianza y aceptación, y los archivos físicos sean digitalizados.

Teniendo en cuenta las deficiencias antes mencionadas surge la necesidad de crear un sistema capaz de gestionar los procesos que se llevan a cabo en los TPC, planteando el siguiente **problema de la investigación**. ¿Cómo contribuir a la gestión del proceso Administrativo en los Tribunales Provinciales de forma que permita elevar la calidad y la celeridad en su tramitación?

Para ello se tendrá como **objeto de estudio** de esta investigación el proceso de desarrollo de software en la creación de un sistema para la gestión judicial.

Constituye entonces el **objetivo general** desarrollar el diseño y la implementación de una solución informática para el proceso Administrativo en el proyecto Tribunales Populares Cubanos, que permita elevar la calidad y la celeridad en su tramitación.

Después de ser visto el problema planteado y el objeto de estudio se determina como **campo de acción** el diseño e implementación como fases especificas dentro del proceso de desarrollo de software en la gestión del proceso Administrativo en los Tribunales Provinciales Cubanos.

Como **idea a defender** se tiene que: realizando el diseño e implementación de una solución informática para el proceso Administrativo de los TPC se logrará elevar la calidad y la celeridad en su tramitación**.**

Objetivos Específicos:

- ✓ Definir el marco teórico-referencial para la justificación del trabajo mediante la investigación del estado del arte del proceso de diseño e implementación y de

los sistemas de gestión de procesos judiciales de tribunales en Cuba y el mundo.

- ✓ Obtener los diagramas del sistema.
- ✓ Obtener la implementación del sistema.
- ✓ Validar la solución de software.

Tareas a cumplir:

- ✓ Realización de un estudio del estado del arte sobre los patrones de diseño y buenas prácticas de programación.
- ✓ Realización de los diagramas de clases, de interacción, de componentes y de despliegue para la solución informática.
- ✓ Implementación de la aplicación informática a partir de los diagramas obtenidos en el diseño.
- ✓ Validación de los modelos de diseño por las distintas métricas de calidad que existen.
- ✓ Validación la solución de software mediante métricas para el diseño y pruebas de caja blanca y de caja negra, para lograr así una mejor calidad del producto.

Los **métodos científicos** son la forma de abordar la realidad, de estudiar la naturaleza, la sociedad y el pensamiento, con el propósito de descubrir su esencia y sus relaciones. Para la buena elaboración de este trabajo se utilizaron los siguientes métodos:

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

Histórico-lógico*:* Permitió la realización de un estudio detallado de los antecedentes y componentes de los Tribunales Populares Cubanos para su mejor análisis y comprensión, y así obtener una tendencia de cómo se comporta en la actualidad.

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

Modelación: Facilitó la creación de modelos que representan abstracciones con el objetivo de comprender el funcionamiento del proceso Administrativo que se lleva a cabo en los Tribunales Provinciales Cubanos.

Métodos Empíricos: Describieron y explicaron las características fenomenológicas del objeto, representaron un nivel de la investigación cuyo contenido procede de la experiencia y es sometido a cierta elaboración racional

Entrevista*:* Permitió recopilar información cualitativa de la investigación, mediante la cual se diagnostica el estado actual de los TPC y su funcionamiento.

El presente trabajo consta de 3 Capítulos que abordan los temas fundamentales distribuidos de la siguiente manera:

Capítulo 1: Fundamentación Teórica. En este capítulo se recoge toda la fundamentación teórica que sustenta el progreso de la investigación para el posterior desarrollo del sistema propuesto. En el mismo se hace un estudio del estado del arte de sistemas existentes con características similares en cuanto a objetivos de uso y de funcionamiento. Se fundamenta la utilización de la metodología, las herramientas y los tipos de patrones que se utilizarán para el desarrollo de la aplicación.

Capítulo 2: Solución Propuesta. En este capítulo se propone la solución técnica de la investigación. Se analiza la arquitectura del sistema a desarrollar, las diferentes capas que los componen y sus componentes. 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 previamente diseñados. 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 las de la medición de la calidad del diseño y la implementación de software.

Capítulo I: Fundamentación Teórica.

Introducción

En el presente capítulo se fundamenta la utilización de la metodología, las herramientas y los tipos de patrones que se utilizarán para el desarrollo de la aplicación. Se brinda una breve panorámica sobre los sistemas que soportan la informática jurídica, los proyectos y aplicaciones web que presentan relación con el presente trabajo a nivel mundial y nacional.

1.1. Proceso Administrativo.

Los tribunales atienden cinco procedimientos distribuidos en las instancias Supremo, Provincial y Municipal. Los procedimientos son: Laboral, Penal, Civil, Económico y Administrativo.

El presente trabajo se enfocará en el proceso Administrativo, el cual está compuesto por tres procedimientos: Administrativo, Recurso de Casación y Revisión. El proceso Administrativo se atiende en la instancia Provincial mientras que los procesos Recurso de Casación y Revisión en el Tribunal Supremo.

Se denomina proceso Administrativo, en término judicial, a la sucesión de trámites mediante los cuales una persona que se considere afectada por una decisión o acto de la administración pueda reclamar ante la Sala de lo Administrativo del Tribunal Provincial. Se considera Administración a los Organismos de la Administración Central del Estado y a los Órganos Locales del Poder Popular, aunque existen otros como Direcciones Municipales de la Vivienda, la Oficina Cubana de la Propiedad Industrial, la Oficina Nacional de Administración Tributaria, el Centro Nacional de Derecho de Autor, la Aduana General de la República, que contra su actuar también se puede reclamar. Las decisiones que adoptan las administraciones generalmente se plasman en resoluciones que son notificadas a los interesados y éstos cuentan con treinta días hábiles para establecer demanda ante el tribunal por hallarse inconformes con lo resuelto, porque consideran que con ella se lesiona un derecho, que en su opinión, tienen legalmente preestablecido.

1.2. Sistemas Judiciales existentes en el mundo

1.2.1. WinJuris

WinJuris es un programa flexible, diseñado para trabajar tanto en los tribunales penales, como en los civiles de cualquier tamaño. El programa documenta parte de la información básica que es gestionada en la sala, dígase los datos de los acusados y testigos, infracciones, multas, gastos, pagos, servicio de papel, además de otros documentos de la sala vinculados al proceso en conjunto con la correspondencia que se maneja. Este sistema permite generar Expedientes diarios del tribunal sobre las demandas tramitadas en la sala. Importa registros de numerosas fuentes, tales como citaciones, órdenes, reservas y detenciones.

WinJuris tiene un subsistema de contabilidad completo incorporado, que incluye la capacidad para distribuir automáticamente los pagos a los estatales o locales, crear informes diarios y mensuales de contabilidad, recibos de documento, y los controles de impresión. Posee numerosos informes y consultas que le permiten el seguimiento de una gran variedad de información **(Solution, 1988).**

1.2.2. IurisExplorer (Argentina).

Es un software que permite organizar toda la información que el usuario maneja y consultarla rápidamente. Además en él se gestionan las tareas cotidianas de todo Estudio Jurídico: administrar expedientes, redactar escritos, dejar notas a su secretaria, intercambiar documentación, llevar un orden de las causas, ordenar sus asuntos particulares.

1.2.3. Atlante

Sistema Informático desarrollado en Canarias que comenzó a instalarse en todas las dependencias judiciales del Archipiélago en el año 2007. Este sistema registra todas las diligencias realizadas por las distintas instituciones, remite la información adecuada al siguiente paso, aunque los funcionarios esperan que lleguen los documentos en papel para procesarlos. Actualmente se despliega Atlante II, pero este ha ocasionado problemas en los juzgados, donde los funcionarios deben trabajar "a mano" debido a las continuas "caídas" de los servidores.

1.2.4. Lexnet

Es un sistema de gestión de notificaciones telemáticas usado en la Administración de Justicia española que permite la comunicación bidimensional entre las oficinas judiciales y los distintos operadores jurídicos. Actúa sobre la base de la tecnología de los certificados digitales, permite el uso de la firma digital, en los documentos electrónicos, la presentación de escritos y notificaciones. Es un sistema de intercambio de documentos judiciales en formato electrónico **(Santos, 2010)**.

1.2.5. Infolex

Software de Gestión Jurídica (España). Este software es realizado por la empresa Jurisoft la cual es líder en el sector de la Informática Jurídica. Es un sistema multilingüe, parametrizable en función de las necesidades del usuario. Se adapta a la normativa de protección de datos y posee diferentes versiones (Servicios Jurídicos de Empresa, Asesorías Juridícas de la Administración) y motores de base de datos. Cuenta con acceso remoto y soluciones para entidades con delegaciones **(Infolex, 2007)**.

1.2.6. Soft Class para Abogados. (Barcelona, España).

Es un sistema integral de gestión para bufetes de abogados y despachos o asesorías jurídicas que permite llevar un completo control de todas las tareas relacionadas con los mismos, además es un software adaptable a otros ámbitos. La principal función de este software es la administración de expedientes, con varios apartados en el que se gestionan independiente o conjuntamente los datos que lo conforman. El programa funciona con una base de datos en formato MS Access (programa, utilizado en los sistemas operativos Microsoft Windows, para la gestión de bases de datos creado y modificado por Microsoft y orientado a ser usado en entorno personal o en pequeñas organizaciones). Eso da garantía de poder recuperar los datos en cualquier momento, y de poder efectuar exportaciones. Del mismo modo, pueden leer los datos de la base de datos para efectuar una combinación o realizar una selección de registros **(Abogados, 2004)**.

El uso de estos sistemas en el mundo ha traído una gran cantidad de ventajas, ayudando entre otras cosas, a tramitar correctamente los procesos que tienen lugar en los tribunales, así como a procesar la información que se precisa; sin embargo ninguno de ellos responde a las necesidades que hoy en día existen en el Sistema

Judicial en Cuba. Todos estos sistemas son propietarios, hay que pagar para poder utilizarlos, lo cual va en contra del paradigma de independencia tecnológica por el cual apuesta el país. Se puede señalar que Lexnet es sólo un sistema para notificaciones, recepción de escritos digitales y traslado de copias. Atlante mantiene la tramitación de forma manual y solo permite registrar diligencia y remite la información al siguiente paso que le corresponde. WinJuris está diseñado para trabajar en los procedimientos penales y civiles, pero no en el procedimiento administrativo, lo mismo que sucede con el sistema IurisExplorer. Es por ello que nos vemos en la necesidad de crear un sistema que cumpla con todas las características y requerimientos necesarios para una buena gestión en el Sistema Judicial Cubano.

1.3. Aplicaciones Web.

En la ingeniería de software se denomina aplicación web a aquellas aplicaciones que los usuarios pueden utilizar accediendo a un servidor web a través de Internet o de una intranet mediante un navegador. En otras palabras, es una aplicación de software que se codifica en un lenguaje soportado por los navegadores web en la que se confía la ejecución al navegador.

Para el desarrollo del sistema de informatización de los TPC se utilizará una aplicación web debido a lo práctico del navegador web como cliente ligero, a la independencia del sistema operativo, a la facilidad para actualizar y mantener aplicaciones web sin distribuir e instalar software a miles de usuarios potenciales, a la posibilidad de una administración centralizada así como una variedad de soluciones o framework de código libre que facilitan su desarrollo.

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. 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:
  - ✓ Orientada 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 complejo **(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)**.

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 orientación a objetos 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.

¿Cuáles son los beneficios de la POO?

La POO logra una reducción de la complejidad a la hora de construir el software ya que los objetos exponen solo su interfaz pública, escondiendo por lo tanto, los detalles de implementación y evitando las complejas interdependencias en el código.

Las interfaces promueven un código flexible y robusto y son ideales para particionar responsabilidades individuales y del equipo, logrando así un incremento de la productividad y la eficiencia.

La pérdida de acoplamiento y la modularidad facilitan la extensibilidad, la flexibilidad, la escalabilidad y la reusabilidad, así como que los cambios sean más fáciles de implementar y mantener.

Un buen diseño del conjunto de objetos permite que nuevas funcionalidades sean añadidas incrementalmente y de forma no invasiva.

1.5. Patrones de Diseño Orientado a Objeto.

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.

Un patrón de diseño es una solución a un problema de diseño. 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.

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, Fabricación Pura, Indirección, Variaciones Protegidas, No hables con extraños y Polimorfismo.

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

1.6. Framework de desarrollo

En el desarrollo de software, un marco de trabajo (framework), 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.

1.6.1. Zend Framework (ZF)

Se trata de un framework para desarrollo de aplicaciones Web y servicios Web con PHP, brinda soluciones para construir sitios web modernos, robustos y seguros. Además es Open Source y trabaja con PHP 5. ZF es una implementación que usa código 100% orientado a objetos. La estructura de los componentes de ZF es algo único; cada componente está construido con una baja dependencia de otros componentes. Esta arquitectura débilmente acoplada permite a los desarrolladores utilizar los componentes por separado.

1.6.2. Sauxe

A partir de la extensión de algunos componentes de Zend Framework surge Zend Ext, desarrollado por el Departamento de Tecnología y la UCID (Unidad de Compatibilización Integración y Desarrollo de Software para la Defensa), con el objetivo de crear un Marco de Trabajo extensible y configurable, centrando el desarrollo de las aplicaciones en la lógica del negocio, en las interfaces de usuario, alejando a los programadores de los detalles arquitectónicos, con soporte para entornos multientidad y para una arquitectura de sistema orientada a componentes **(Baryolo, 2008)**.

La unión de Zend Ext, Doctrine y Extjs dio lugar al Marco de Trabajo Sauxe.

Sauxe es 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 y agilidad en el proceso de desarrollo **(Baryolo, 2008)**. Siguiendo el paradigma de independencia tecnológica por el cual apuesta el país, reutiliza las siguientes tecnologías libres:

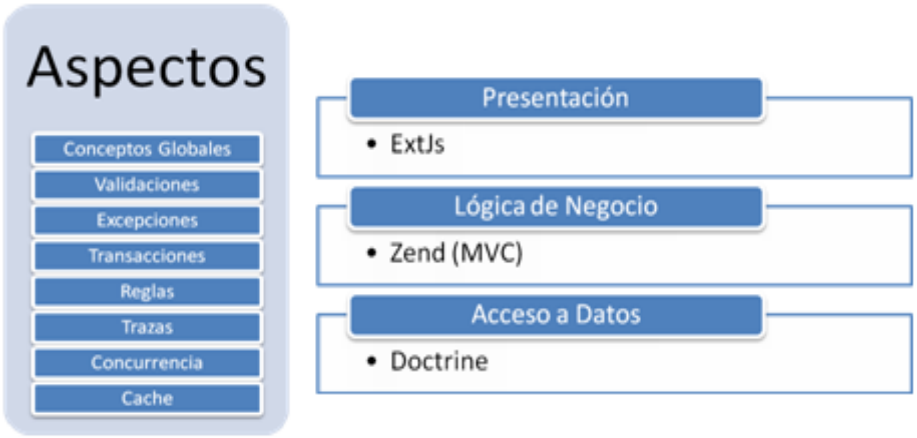

Figura 1: Arquitectura de Sauxe

El marco de trabajo Sauxe tiene definido para su uso una serie de herramientas específicas. Como sistema gestor de base datos el PostgreSQL 8.3 o 8.4 solamente, siendo esta una desventaja del framework e imposibilitando explotar al máximo las ventajas que brindan los ORM (Object Relation Mapper). Tiene como lenguajes de programación definidos ya, del lado del servidor PHP y del lado del cliente JavaScript. Todas estas herramientas son libres siguiendo el paradigma de independencia tecnológica por el cual apuesta el país

La arquitectura tecnológica de la plataforma tiene varias ventajas en diferentes tipos de escenarios arquitectónicos. Provee capacidad para configurar y gestionar de manera dinámica la cache del marco de trabajo. Posee capacidad para configurar y gestionar de manera dinámica la integración de las aplicaciones o dominios de soluciones que se instancien o construyan con la misma. Tiene un mecanismo de abstracción de la capa relacional de persistencia, este mecanismo de mapeo relacional de objeto, permite además contar con un mecanismo de SQL Helper. Provee un mecanismo de configuración y gestión dinámica de las características de concurrencia sobre entidades o dominios de la solución lógica que se modela, de manera que se puede configurar el esquema de concurrencia que se desee sobre las entidades (conceptos) o estructuras de entidades del esquema de persistencia de una solución específica. Ostenta un mecanismo de administración de transacciones, de manera que las operaciones de modificación sobre el modelo de dominio sean

transaccionales por defecto. Posee un mecanismo de administración y configuración dinámica de trazas de la solución.

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 dato, 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) subsistemas, esto garantizaría un sin número de bondades tanto de usabilidad como de configurabilidad a la solución en cuestión.

- ✓ **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.

El framework cuenta con un mecanismo de mensajería entre dominios o componentes, el cual permite la abstracción de la tecnología que instancia el mecanismo, usa para ello un estándar para la transmisión del paquete de información u objeto de dominio que se transmite. Provee un mecanismo de gestión, configuración y administración de excepciones de manera dinámica y declarativa. El mismo se integra con el elemento responsable de las trazas, cuando ocurra el suceso de una excepción. La solución incluye además la capacidad de definir familias de excepciones y dentro de las familias

de excepciones, excepciones tipo concretas, de cada excepción se puede definir características tipo tales como: línea de código e instrucción en se generó, traking de la excepción que la inició, fecha y hora entre otros metadatos que permiten una visualización y análisis de las trazas emitidas por el sistema por cada excepción.
La arquitectura tecnológica de la plataforma provee un mecanismo para la gestión de objetos a nivel de datos (bases de datos, esquemas, secuencias, trigger, funciones, roles, tablas, atributos, etc.) desde la capa de aplicación.
La misma provee además un mecanismo de configuración, generación, administración y visualización dinámica de reportes. Esta solución permite que los usuarios diseñen los reportes con los estándares definidos, imprimir dichos reportes en formatos como PDF, HTML, WORD entre otros. Permite configurar los atributos que se desean mostrar, los nombres que los identificarán y de que modelos de datos serán extraídos.

1.6.3. Object Relation Mapper (ORM)

Un ORM es una técnica de programación que nos permite convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional, es decir, las tablas de nuestra base de datos pasan a ser clases y los registros, objetos que podemos manejar con facilidad.

1.6.4. Doctrine

Doctrine es un ORM para PHP 5.2.3 y posterior. Además de todas las ventajas que conlleva un ORM, uno de sus puntos fuertes es su lenguaje DQL (Doctrine Query Language) inspirado en el HQL de Hibernate.

Cuando se trabaja con Doctrine, se necesita informar a su motor interno cuál es el modelo de la aplicación que esta está trabajando, para ello se puede hacer ingeniería inversa de la base de datos existente, o si se empieza la aplicación desde 0, crear el modelo en la sintaxis específica que propone Doctrine y luego generar toda la base de datos.

Selección del framework a utilizar

Para la realización del proyecto se utiliza el framework Sauxe integrado al Generador de reportes dinámicos desarrollado por el Centro de Datos, el primero tiene como framework base Zend Framework y el ORM Doctrine que este trae por defecto lográndose independencia del gestor de base de datos.

El Sauxe además de contar con todas las ventajas antes mencionadas también tiene un impacto económico y social en los días actuales. Con el desarrollo y reutilización de Sauxe se ahorran cuantiosos recurso 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 cada una de las aplicaciones que se desarrollen, se gastaría 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 tendrían asegurados todos los escenarios descritos anteriormente, ellos resuelven un porciento importante de los requisitos del sistema. 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. Cuanta 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 periodo de tiempo. Sauxe es una forma eficaz de construir sistemas con calidad guiados por estándares internacionales **(Ing. Oiner Gómez Baryolo, 2008)**.

1.7. Plataformas de desarrollo.

En informática, una plataforma de desarrollo es el entorno de software común en el cual se desenvuelve la programación de un grupo definido de aplicaciones. Comúnmente se encuentra relacionada directamente a un sistema operativo; sin

embargo, también es posible encontrarla ligada a una familia de lenguajes de programación o a una Interfaz de Programación de Aplicaciones (API por sus siglas en inglés).

1.7.1. Lenguajes de programación.

1.7.1.1. PHP

PHP (Hypertext Preprocessor) es un lenguaje de programación interpretado, diseñado originalmente para la creación de páginas web dinámicas. Es usado principalmente en interpretación del lado del servidor pero actualmente puede ser utilizado desde una interfaz de línea de comandos o en la creación de otros tipos de programas incluyendo aplicaciones con interfaz gráfica **(PHP, 2001-2011)**.

1.7.1.2. Java Script

Java Script es un lenguaje de programación interpretado, dialecto del estándar ECMAScript[1]. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Se utiliza principalmente en su forma del lado del cliente implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas **(Flanagan, 2002).**

Selección del lenguaje de programación a utilizar:

Se utilizan 2 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 aunque se posibilita la portabilidad para otros navegadores web que cumplan con el estándar ECMA 262 que norma al ECMAScript. Vale señalar que el uso de estos lenguajes de programación viene definido por el framework Sauxe.

1.8. Entornos de desarrollo integrados (IDE).

Un entorno de desarrollo integrado es un programa informático compuesto por un conjunto de herramientas de programación. Puede dedicarse en exclusiva a un sólo lenguaje de programación o bien, puede utilizarse para varios. Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir,

[1] ***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.*

consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica de usuario. Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes.

1.8.1.1. Aptana Studio.

Aptana Studio es un entorno de desarrollo integrado gratuito basado en eclipse y desarrollado por Aptana, Inc., que puede funcionar bajo Windows, Mac y Linux y provee soporte para lenguajes como: Php, Python, Ruby, CSS, Ajax, HTML y Adobe AIR. Tiene la posibilidad de incluir complementos para nuevos lenguajes y funcionalidades **(Alvarez, 2007)**.

Características:

- ✓ Asistente de código para HTML y Javascript.
- ✓ Librerías ajax (jQuery, prototype, Ext JS, dojo, YUI y Spry entre otras).
- ✓ Conexión vía FTP, SFTP, FTPS y Aptana Cloud.
- ✓ Herramientas para trabajo con base de datos.
- ✓ Marcado de sintaxis mediante colores.
- ✓ Compatible con extensiones para Eclipse (existen más de 1000).
- ✓ Es un software libre que posee la GNU General Public License o la Aptana Public License.

1.8.1.2. NetBeans.

NetBeans es un entorno de desarrollo, hecho principalmente para el lenguaje de programación Java. Existe además un número importante de módulos para extender el NetBeans IDE. NetBeans IDE es un producto libre y gratuito sin restricciones de uso.

NetBeans es un proyecto de código abierto de gran éxito con una gran base de usuarios, una comunidad en constante crecimiento, y con cerca de 100 socios en todo el mundo **(Corporation, 2011)**.

La plataforma NetBeans permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos. Un módulo es un archivo Java que contiene clases de java escritas para interactuar con las APIs de NetBeans y un archivo especial (manifest file) que lo identifica como módulo. Las aplicaciones

construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. Debido a que estos pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software.

El NetBeans IDE soporta el desarrollo de todos los tipos de aplicación Java (J2SE, web, EJB y aplicaciones móviles). Entre sus características se encuentra un sistema de proyectos basado en Ant, control de versiones y refactoring.

Selección del entorno de desarrollo a utilizar

Se utiliza como entorno de desarrollo Netbeans 6.9, debido a que es un es un proyecto de código abierto, soporta lenguajes dinámicos como PHP y Java Script tiene integración con el subversión, es multi-plataforma, tiene una interfaz amigable, tiene todas las herramientas para crear aplicaciones profesionales ya sean de escritorio, empresariales, web, móviles y aplicaciones SOA, no solo en Java sino también en C/C++ y Ruby, puedes hacer diagramas UML y es de múltiples lenguajes. Para la visualización de las interfaces se utiliza el Mozilla Firefox preferentemente, con el plugin Firebug para los entornos de desarrollo y prueba. El grupo de desarrollo del proyecto está familiarizado con dicho IDE.

1.9. Servidor de Aplicaciones

En informática, se denomina servidor de aplicaciones a un servidor en una red de computadores que ejecuta ciertas aplicaciones.

Se trata de un dispositivo de software que proporciona servicios de aplicación a las computadoras cliente. Un servidor de aplicaciones, generalmente gestiona la mayor parte (o la totalidad) de las funciones de lógica de negocio y de acceso a los datos de la aplicación. Los principales beneficios de la aplicación de la tecnología de servidores de aplicación son la centralización y la disminución de la complejidad en el desarrollo de aplicaciones. Es el corazón de un gran sistema distribuido **(Alicante, 2011)**.

1.9.1.1. BEA WebLogic.

BEA WebLogic es un servidor de aplicaciones J2EE (Java 2 Platform, Enterprise Edition) y también un servidor web HTTP de Oracle, para Unix, Linux, Microsoft Windows, y otras plataformas **(Oracle, 2011)**.

WebLogic puede utilizar Oracle, DB2, Microsoft SQL Server, y otras bases de datos que se ajusten al estándar JDBC.

WebLogic Server incluye interoperabilidad .NET y admite las siguientes capacidades de integración nativa:

- ✓ Mensajería nativa JMS a escala de empresa
- ✓ J2EE Connector Architecture
- ✓ Conector WebLogic/Tuxedo
- ✓ Conectividad COM+
- ✓ Conectividad CORBA
- ✓ Conectividad IBM WebSphere MQ

BEA WebLogic Server Process Edition también incluye una Administración de Procesos Empresariales y funcionalidad de mapeo de datos.

WebLogic admite políticas de seguridad administradas por Security Administrators. El modelo de seguridad de BEA WebLogic Server incluye:

- Separar la lógica de aplicaciones de negocio, del código de seguridad
- El rango completo de cobertura de seguridad tanto para los componentes J2EE y no J2EE.

1.9.1.2. Sun Java System Application Server.

Sun Java System Application Server, o SJSAS es un servidor de aplicaciones de la plataforma Java producido por Sun Microsystems. El servidor está basado en la plataforma Java EE y es el núcleo del sistema Java Enterprise, a manera de contenedor de EJB (Enterprise JavaBeans) o de proveedor de Web Service. Tiene soporte integrado para interfaces de desarrollo tales como Sun Java Studio Enterprise, Sun Java Studio Creator y NetBeans **(Oracle, 2011)**.

A partir de la versión 9, SJSAS Plataform Edition se encuentra siendo desarrollada bajo el proyecto de código libre GlassFish y bajo las licencias de CDDL y GPL. Este proyecto incluye código de otras compañías tales como Oracle y bases para el JEE5

1.9.1.3. Servidor Apache.

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)**.

Selección del servidor a utilizar

Se seleccionó Apache 2.0 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 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.

1.10. Herramientas CASE[2]

Las herramientas CASE son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el costo de las mismas en términos de tiempo y de dinero. Estas herramientas nos pueden ayudar en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de realizar un diseño del proyecto, cálculo de costes, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores entre otras. Existe actualmente gran variedad de dichas herramientas

1.10.1. Enterprise Architect

Enterprise Architect (EA) Professional es una herramienta CASE de Sparx Systems. Soporta ocho de los nueve diagramas estándares del UML: diagrama de casos de uso, de clases, de secuencia, de colaboración, de actividad, de estados, de implementación (componentes), de despliegue y varios perfiles del UML. Si fuera necesario, el diagrama de objetos se puede crear usando los diagramas de colaboración **(Systems, 2000-2007)**.

[2] ***C**omputer **A**ided **S**oftware **E**ngineering*, Ingeniería de Softw are Asistida por Computadora.

Características Principales:

- ✓ Enterprise Architect tiene un mecanismo de perfil UML genérico para cargar y trabajar con diferentes perfiles UML. En Enterprise Architect, estos perfiles se especifican en archivos XML con un formato específico.
- ✓ Permite ingeniería de código (directa e inversa) para ANSI C++, Visual Basic 6, Java, C#, VB.NET, Delphi y Bases de datos: Ingeniería directa desde el modelo de datos al script DDL. La ingeniería reversa usa la fuente de datos ODBC.
- ✓ Provee trazabilidad completa desde los modelos de requisitos, análisis y diseño, hasta la implementación y despliegue.
- ✓ Permite una administración efectiva del proyecto mediante la asignación de recursos a los elementos, la medición del esfuerzo y riesgos, la estimación del tamaño y complejidad del proyecto y la implementación del control de cambios y de procedimientos de mantenimiento.

1.10.2. Rational Rose Enterprise Edition 2003

Rational Rose es una herramienta de producción y comercialización establecidas por Rational Software Corporation (actualmente parte de IBM). Rose es un instrumento operativo conjunto que utiliza el Lenguaje Unificado (UML) como medio para facilitar la captura de dominio de la semántica, la arquitectura y el diseño. Este software tiene la capacidad de crear, ver, modificar y manipular los componentes de un modelo.

Características Principales:

- ✓ No es gratuito, se debe hacer un previo pago para poder adquirir el producto.
- ✓ Rational Rose habilita asistentes para crear clases y provee plantillas de código que pueden aumentar significativamente la cantidad de código fuente generada. Adicionalmente, se pueden aplicar los patrones de diseño, Racional Rose ha provisto 20 de los patrones de diseño GOF para Java.
- ✓ Permite el diseño de bases de datos.
- ✓ Brinda la posibilidad de generar y realizar ingeniería inversa en una buena cantidad de lenguajes de programación.
- ✓ Interfaz poco amigable.

1.10.3. Visual Paradigm

Visual Paradigm para UML 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. La herramienta UML CASE también proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML. Es fácil de instalar y actualizar y compatible entre ediciones **(Paradigm, 2011).**

Características Principales:

- ✓ Soporte de UML versión 2.1.
- ✓ Diagramas de Procesos de Negocio-Proceso, Decisión, Actor de negocio, Documento.
- ✓ Modelado colaborativo con CVS y Subversión (control de versiones).
- ✓ Ingeniería de ida y vuelta.
- ✓ Ingeniería inversa - Código a modelo, código a diagrama.
- ✓ Ingeniería inversa Java, C++, Esquemas XML, XML, NET exe/dll, CORBA IDL.
- ✓ 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.
- ✓ Generación de código y despliegue de EJB - Generación de beans para el desarrollo y despliegue de aplicaciones.
- ✓ 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.
- ✓ Distribución automática de diagramas -Reorganización de las figuras y conectores de los diagramas UML.
- ✓ Importación y exportación de ficheros XMI.

Selección de la herramienta a utilizar

Después de haber estudiado y analizado algunas de las herramientas CASE se eligió para la realización del proyecto el Visual Paradigm 6.4 ya que posee alta capacidad de integración con el lenguaje orientado a procesos BPMN y UML que serán los utilizados para el desarrollo del sistema. La aplicación a desarrollar será programada en PHP y Visual Paradigm se integra fácilmente con este lenguaje. Además el equipo de desarrollo cuenta con un mayor conocimiento de esta herramienta que de otras descritas como por ejemplo del Enterprise Architect que a pesar de ofrecer similares funcionalidades es desconocido por el equipo de desarrollo, posibilitando ahorro de tiempo en el desarrollo del software ofreciendo capacitación de los desarrolladores. Se decidió no utilizar la herramienta Rational Rose debido a que no permite la integración con el lenguaje BPMN ni con el lenguaje de programación seleccionado.

1.11. Sistemas gestores de Base de datos

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

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

1.11.1. Oracle

Es un manejador de base de datos relacional que hace uso de los recursos del sistema informático en todas las arquitecturas de hardware, para garantizar su aprovechamiento al máximo en ambientes cargados de información.

Es el conjunto de datos que proporciona la capacidad de almacenar y acudir a estos de forma recurrente con un modelo definido como relacional. Además es una suite de productos que ofrece una gran variedad de herramientas **(Oracle, 2011)**.

Características Principales:

- ✓ Es una herramienta de administración gráfica que es mucho más intuitiva y cómoda de utilizar.
- ✓ Ayuda a analizar datos y efectuar recomendaciones concernientes a mejorar el rendimiento y la eficiencia en el manejo de aquellos datos que se encuentran almacenados.
- ✓ Apoya en el diseño y optimización de modelos de datos.
- ✓ Asiste a los desarrolladores con sus conocimientos de SQL y de construcción de procedimientos almacenados y triggers, entre otros.
- ✓ Apoya en la definición de estándares de diseño y nomenclatura de objetos.
- ✓ Documenta y mantiene un registro periódico de las mantenciones, actualizaciones de hardware y software, cambios en las aplicaciones y, en general, todos aquellos eventos relacionados con cambios en el entorno de utilización de una base de datos.
- ✓ Oracle es un producto vendido a nivel mundial, aunque la gran potencia que tiene y su elevado precio hace que solo se vea en empresas muy grandes y multinacionales, por norma general. Se ejecuta en computadoras personales (PC), microcomputadoras, mainframes y computadoras con procesamiento paralelo masivo. Soporta unos 17 idiomas, se ejecuta automáticamente en más de 80 arquitecturas de hardware y software distinto sin tener la necesidad de cambiar una sola línea de código. Esto es porque más el 80% de los códigos internos de Oracle son iguales a los establecidos en todas las plataformas de sistemas operativos.

1.11.2. PostgreSQL

PostgreSQL es un sistema de gestión de base de datos relacional orientada a objetos y libre, publicado bajo la licencia BSD.

Como muchos otros proyectos de código abierto, el desarrollo de PostgreSQL no es manejado por una empresa y/o persona, sino que es dirigido por una comunidad de desarrolladores que trabajan de forma desinteresada, altruista, libre y/o apoyada por organizaciones comerciales. Dicha comunidad es denominada el PGDG (PostgreSQL Global Development Group).

Propiedades de PostgreSQL:

- ✓ Atomicidad (Indivisible) es la propiedad que asegura que la operación se ha realizado o no, y por lo tanto ante un fallo del sistema no puede quedar a medias.
- ✓ Consistencia es la propiedad que asegura que sólo se empieza aquello que se puede acabar. Por lo tanto se ejecutan aquellas operaciones que no van a romper la reglas y directrices de integridad de la base de datos.
- ✓ Aislamiento es la propiedad que asegura que una operación no puede afectar a otras. Esto asegura que dos transacciones sobre la misma información nunca generarán ningún tipo de error.
- ✓ Durabilidad es la propiedad que asegura que una vez realizada la operación, ésta persistirá y no se podrá deshacer aunque falle el sistema.

Características Principales:

- ✓ Se adapta en casi todos los principales sistemas operativos: Linux, Unix, Mac OS, Beos, Windows, etc.
- ✓ Documentación muy bien organizada, pública y libre, con comentarios de los propios usuarios.
- ✓ Comunidades muy activas, varias comunidades en castellano.
- ✓ Bajo “Costo de Propiedad Total” (TCO) y rápido “Retorno de la Inversión Inicial” (ROI)
- ✓ Altamente adaptable a las necesidades del cliente.

- ✓ Soporte nativo para los lenguajes más populares del medio: PHP, C, C++, Perl, Python, etc.
- ✓ Soporte de todas las características de una base de datos profesional (triggers, store procedures –funciones, secuencias, relaciones, reglas, tipos de datos definidos por usuarios, vistas, vistas materializadas, etc.)
- ✓ Soporte de tipos de datos de SQL92 y SQL99.
- ✓ Soporte de protocolo de comunicación encriptado por SSL.
- ✓ Extensiones para alta disponibilidad, nuevos tipos de índices, datos espaciales, minería de datos, etc.

Selección del SGBD a utilizar

Se utiliza como gestor de base de datos PostgreSql 8.3 que es un servidor de base de datos relacional y libre. Es el servidor definido por la arquitectura del framework Sauxe, pudiendo utilizar hasta la versión 8.4 del mismo. Tiene soporte total para transacciones, disparadores, vistas, procedimientos almacenados, almacenamiento de objetos de gran tamaño. Se destaca en ejecutar consultas complejas, consultas sobre vistas, subconsultas y joins[3] de gran tamaño. Permite la definición de tipos de datos personalizados e incluye un modelo de seguridad completo. Como toda herramienta de software libre PostgreSQL tiene entre otras ventajas las de contar con una gran comunidad de desarrollo en Internet, su código fuente está disponible sin costo alguno y algo muy importante es que dicha herramienta es multiplataforma. Además de sus ofertas de soporte, cuenta con una importante comunidad de profesionales y entusiastas de PostgreSQL de los que los centros de desarrollos pueden obtener beneficios y contribuir. Como cliente tiene el PgAdmin III que es distribuido junto a PostgreSQL que también es libre.

1.12. Metodologías de desarrollo de software.

Una metodología de desarrollo de software se refiere a un marco de trabajo (framework) que es usado para estructurar, planear y controlar el proceso de desarrollo en sistemas de información.

[3] ***(JOIN)***. *En lenguaje SQL permite combinar registros de dos o más tablas en una base de datos relacional.*

A lo largo del desarrollo de la Ingeniería de Software, se han propuesto muchas metodologías, que han sido efectivas en su momento, cada una con sus características y alcance. A grandes rasgos, se pueden dividir las metodologías en dos grandes grupos: las ágiles y las tradicionales:

1.12.1. Metodologías ágiles.

Las metodologías ágiles surgen como una extensión a las metodologías tradicionales para mejorar el desarrollo de sistemas, según el tipo de proyecto y empresa, añadiendo y mejorando (optimizando) las practicas de desarrollo de software. Son muy usadas y están para evolucionar y revolucionar las prácticas de ingeniería.

Desde el surgimiento de la crisis del software en la década del 70 hasta nuestros días han tenido las metodologías ágiles nuevas modificaciones y representantes. Dentro de las metodologías ágiles se pueden mencionar:

- ✓ Extreme Programming.
- ✓ Scrum.
- ✓ Familia de Metodologías Crystal.
- ✓ Feature Driven Development.
- ✓ Proceso Unificado Rational, una configuración ágil.
- ✓ Dynamic Systems Development Method.
- ✓ Adaptive Software Development.
- ✓ Open Source Software Development.

Una vez analizadas las características generales de las metodologías de desarrollo ágiles, se deduce que no se ajustan a las características del proyecto debido a que en las mismas el cliente es parte del equipo de desarrollo, cosa que no sucede en el proyecto TPC. Son metodologías diseñadas para grupos pequeños, menores de 10 integrantes, y el proyecto cuenta con un personal superior. Generan pocos artefactos, opuestamente a lo que sucede en el proyecto. Estas metodologías se caracterizan por la intervención de escasos roles y una generación poco significativa de artefactos para el desarrollo del software. Igualmente cabe destacar el poco protagonismo dado a la arquitectura de software en estas metodologías.

1.12.2. Metodologías tradicionales.

Teniendo en cuenta la filosofía de desarrollo de las metodologías, aquellas con mayor énfasis en la planificación y control del proyecto, en especificación precisa de requisitos y modelado, reciben el apelativo de Metodologías Tradicionales o Pesadas

Las metodologías tradicionales se diferencian de las ágiles en muchos aspectos, sus características son las siguientes:

- ✓ Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo.
- ✓ Cierta resistencia a los cambios.
- ✓ Impuestas externamente.
- ✓ Proceso mucho más controlado, con numerosas políticas/normas.
- ✓ Existe un contrato prefijado.
- ✓ El cliente interactúa con el equipo de desarrollo mediante reuniones.
- ✓ Grupos grandes y posiblemente distribuidos.
- ✓ Más artefactos.
- ✓ Más roles
- ✓ La arquitectura del software es esencial y se expresa mediante modelos.

Dentro de las metodologías tradicionales se pueden citar:

- ✓ MSF (Microsoft Solution Framework)
- ✓ Win-Win Spiral Model
- ✓ Iconix
- ✓ RUP (Rational Unified Procces)

1.12.2.1. MSF (Microsoft Solution Framework)

MSF es una metodología desarrollada por Microsoft Consulting Services en conjunto con varios grupos de negocios de Microsoft y otras fuentes de la industria. MSF provee los principios, modelos y disciplinas para un correcto desarrollo de proyectos en cualquier plataforma (Linux, Citrix, Microsoft, Unix).

Esta es una metodología flexible e interrelacionada con una serie de conceptos, modelos y prácticas de uso, que controlan la planificación, el desarrollo y la gestión de proyectos tecnológicos. MSF se centra en los modelos de proceso y de equipo dejando en un segundo plano las elecciones tecnológicas.

Tiene como características:

- ✓ Adaptable: es parecido a un compás, usado en cualquier parte como un mapa, del cual su uso es limitado a un específico lugar.
- ✓ Escalable: puede organizar equipos tan pequeños entre 3 ó 4 personas, así como también, proyectos que requieren 50 personas a más.
- ✓ Flexible: es utilizada en el ambiente de desarrollo de cualquier cliente.
- ✓ Tecnología Agnóstica: porque puede ser usada para desarrollar soluciones basadas sobre cualquier tecnología.

1.12.2.2. Win-Win Spiral Model

Win-Win Spiral Model propuesto por Barry Boehm es un nuevo logro en un proceso tradicional de software. Manteniendo al mismo tiempo muchos de los elementos tradicionales la versión Win-Win Spiral Model se esfuerza por comprender a todos los interesados en el proceso de desarrollo. Se trata de un motor de colaboración que establece que "ganar" las condiciones establecidas por los usuarios, clientes, desarrolladores, ingenieros de sistemas y con el fin de evolucionar y priorizar los requisitos durante todo el proceso. Las prácticas tradicionales, como los requisitos de ingeniería, diseño, código, y probar, aún están presentes en cada iteración de la espiral, pero el paso de colaboración durante el proceso de desarrollo hace que sea claramente de adaptación. Esta colaboración ofrece el software más rápido, con mayor calidad, menos costosa y, debido a la inicial de satisfacción de las necesidades de los usuarios y la cantidad reducida de mantenimiento.

Estabilización: En esta fase se conducen pruebas sobre la solución, las pruebas de esta etapa enfatizan el uso y operación bajo condiciones realistas. El equipo se enfoca en priorizar y resolver errores y preparar la solución para el lanzamiento.

Implantación: Durante esta fase el equipo implanta la tecnología base y los componentes relacionados, estabiliza la instalación, traspasa el proyecto al personal soporte y operaciones, y obtiene la aprobación final del cliente.

1.12.2.3. Iconix

Es un proceso simplificado en comparación con otros procesos más tradicionales, que unifica un conjunto de métodos de orientación a objetos con el objetivo de abarcar el ciclo de vida de un proyecto. Presenta claramente las actividades de cada etapa y exhibe una secuencia de pasos que deben ser seguidos. Esta entre la complejidad del RUP (Rational Unified Procces) y la simplicidad de XP (Extreme Programming).

Tiene como características:

- ✓ Iterativo e incremental: Varias iteraciones ocurren entre el desarrollo del modelo del dominio y la identificación de los casos de uso. El modelo estático es incremental refinado por los modelos dinámicos.
- ✓ Trazabilidad: Cada paso esta referenciado por algún requisito. Se define trazabilidad como la capacidad de seguir una relación entre los diferentes "artefactos de software" producidos.
- ✓ Dinámica del UML: la metodología ofrece un uso "dinámico" del UML, por que utiliza algunos diagramas del UML, sin exigir la utilización de todos, como en el caso de RUP.

1.12.2.4. Proceso Unificado de Desarrollo (RUP).

RUP fue creado por el mismo grupo de expertos que crearon UML, Ivar Jacobson, Grady Booch y James Rumbaugh en el año 1998. El objetivo que se perseguía con esta metodología era producir software de alta calidad, es decir, que cumpla con los requerimientos de los usuarios dentro de una planificación y presupuesto establecidos. Como se expresaba anteriormente, esta metodología concibió desde sus inicios el uso de UML como lenguaje de modelado.

Es un proceso dirigido por casos de uso, este avanza a través de una serie de flujos de trabajo (requisitos, análisis, diseño, implementación, prueba) que parten de los casos de uso; está centrado en la arquitectura y es iterativo e incremental. RUP cubre el ciclo de vida de desarrollo de un proyecto y toma en cuenta las siguientes buenas prácticas, utilizándolas en el modelo de desarrollo de software.

- ✓ Desarrollo de software en forma iterativa.
- ✓ Manejo de requerimientos.

- ✓ Utiliza arquitectura basada en componentes.
- ✓ Modela el software visualmente.
- ✓ Verifica la calidad del software.
- ✓ Controla los cambios.

Selección de la metodología a utilizar.

Una vez analizadas las metodologías anteriores se ha seleccionado la metodología RUP por tener varios aspectos a su favor. En RUP se genera gran cantidad de documentación que es requerida por la dinámica del proyecto y exigido por parte del cliente, ya que en el mismo el personal varía y se necesita tener documentado lo que se ha hecho anteriormente por otros miembros del grupo de trabajo que pueden no encontrarse presente. Es una metodología robusta que se adapta muy bien a proyectos de larga duración, complejos y con un gran equipo de desarrollo como es el caso del proyecto TPC.

El equipo está familiarizado con la herramienta, y la introducción de una nueva técnica produciría un atraso del cronograma de ejecución, ya que a diferencia de las metodologías ágiles, las metodologías tradicionales requieren de un estudio previo a su utilización. Otras razones son que RUP es una metodología altamente configurable que ha dado buenos resultados a lo largo de su utilización, y existe bastante documentación de la misma.

1.13. Arquitectura de Software

La arquitectura de software se refiere a un grupo de abstracciones y patrones que nos brindan un esquema de referencia útil para guiarnos en el desarrollo de software dentro de un sistema informático.

Así, los programadores, diseñadores, ingenieros y analistas pueden trabajar bajo una línea común que les posibilite la compatibilidad necesaria para lograr el objetivo deseado.

1.10.1. Estilos Arquitectónicos

Se identifican los estilos arquitectónicos como un “conjunto de reglas de diseño que identifica 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" **(Carlos Reynoso, 2004)**.

1.13.1.1. Estilos de Flujo de Datos

Esta familia de estilos enfatiza la reutilización y la modificabilidad. Es apropiada para sistemas que implementan transformaciones de datos en pasos sucesivos. Ejemplares de la misma serían las arquitecturas de tubería-filtros y las de proceso secuencial en lote.

1.13.1.1.1. Tubería y filtros

Esta consta de un conjunto de componentes denominados "filtros" conectados entre sí por "tuberías" que transmiten datos desde un componente al siguiente.

Cada filtro trabaja de manera independiente de los componentes que se encuentran situados antes o después de ella. Se diseñan de tal modo que esperan un conjunto de datos en un determinado formato y obtienen como resultado otros datos de salida en un formato específico.

A pesar de que es fácil de entender e implementar, al igual que enfatiza la reutilización y la modificabilidad, este estilo de arquitectura no se recomienda para la realización del sistema, puesto que es demasiado simplista y no permite el desarrollo de aplicaciones con lógica de negocios complicadas, como es el caso del presente sistema.

1.13.1.2. Estilos Centrados en Datos

Esta familia de estilos enfatiza la integrabilidad de los datos. Se estima apropiada para sistemas que se fundan en acceso y actualización de datos en estructuras de almacenamiento. Sub-estilos característicos de la familia serían los repositorios, las bases de datos, las arquitecturas basadas en hipertextos y las arquitecturas de pizarra.

1.13.1.2.1. Arquitecturas de Pizarra o Repositorio

El estilo de repositorio o pizarra tiene dos componentes fundamentales: una estructura de datos que representa el estado actual y una colección de componentes independientes que operan sobre él. Se han utilizado principalmente en aplicaciones

de inteligencia artificial como reconocimiento de patrones, reconocimiento de habla, en exploraciones recientes de inteligencia artificial distribuida o cooperativa, en robótica, en modelos multi-agentes, en programación evolutiva, en gramáticas complejas, en modelos de crecimiento afines a los L-Systems de Lindenmayer, entre otros. Muchos más sistemas de los que se cree están organizados como repositorios: bibliotecas de componentes reutilizables, grandes bases de datos y motores de búsqueda.

Por lo que se ha podido apreciar, este estilo arquitectónico no es afín con el tipo de aplicación que se desea desarrollar, que es una aplicación de gestión.

1.13.1.3. Estilos de Llamada y Retorno

Esta familia de estilos enfatiza la modificabilidad y la escalabilidad. Son los estilos más generalizados en sistemas en gran escala. Dentro de esta familia se encuentran el Modelo-Vista-Controlador (MVC), la Arquitectura Orientada a Objetos y la Arquitectura en Capas.

1.13.1.3.1. Modelo-Vista-Controlador

Fue diseñada para reducir el esfuerzo de programación necesario en la implementación de sistemas múltiples y sincronizados de los mismos datos. Sus características principales son que el Modelo, las Vistas y los Controladores se tratan como entidades separadas; esto hace que cualquier cambio producido en el Modelo se refleje automáticamente en cada una de las Vistas.
Este modelo de arquitectura se puede emplear en sistemas de representación gráfica de datos, como se ha citado, o en sistemas CAD[4], en donde se presentan partes del diseño con diferente escala de aumento, en ventanas separadas.
Este modelo de arquitectura presenta varias ventajas:

- ✓ Hay una clara separación entre los componentes de un programa; lo cual nos permite implementarlos por separado
- ✓ Hay un API (interfaz de programación de aplicaciones) muy bien definido; cualquiera que use el API, podrá reemplazar el Modelo, la Vista o el Controlador, sin aparente dificultad.

[4] *CAD (Computer Aided Designo DiseñoAsistido por Ordenador) es el nombre genérico que se les da a cualquier tipo de software que se refiera a, dibujo asistido por computadora, y permita hacer dibujos bidimensionales, tridimensionales, y/o técnicos.*

- ✓ La conexión entre el Modelo y sus Vistas es dinámica; se produce en tiempo de ejecución, no en tiempo de compilación.

Al incorporar el modelo de arquitectura MVC a un diseño, las piezas de un programa se pueden construir por separado y luego unirlas en tiempo de ejecución. Si uno de los Componentes, posteriormente, se observa que funciona mal, puede reemplazarse sin que las otras piezas se vean afectadas. Este escenario contrasta con la aproximación monolítica típica de muchos programas Java. Todos tienen un marco que contiene todos los elementos, un controlador de eventos, un montón de cálculos y la presentación del resultado. Ante esta perspectiva, hacer un cambio aquí no es nada trivial.

1.13.1.3.2. Arquitectura en Capa

La división en capas es una de las técnicas más comunes que utilizan los diseñadores de software para separar un sistema de software complicado. Se puede ver en arquitecturas de máquina, donde las capas descienden de un lenguaje de programación con llamadas de sistema operativo en los controladores de dispositivos y conjuntos de instrucciones del microprocesador y en las puertas lógicas dentro de los chips. En este esquema, las capas superiores se sirven de los servicios brindados por las capas inferiores, pero las capas inferiores desconocen de las capas superiores. Aun más, cada capa usualmente esconde sus capas inferiores a las capas superiores por lo tanto la capa 4 usa los servicios de capa 3, que utiliza los servicios de capa 2, pero la capa 4 no es consciente de la capa 2. (No todas las arquitecturas en capas son opacas como esta, pero la mayoría lo son). Entre los beneficios de esta arquitectura se encuentran:

- ✓ 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 mejora el rendimiento.
- ✓ Seguridad: permite aislar componentes.

También presentan sus desventajas como son:

Las capas encapsulan algunas cosas bien, pero no todas. Como resultado, se obtienen a veces cambios en cascada. El ejemplo clásico de esto en una aplicación empresarial en capas es agregar un campo que necesita para mostrar en la interfaz de usuario, debe estar en la base de datos y por lo tanto debe agregarse a cada capa intermedia.
Las capas adicionales pueden perjudicar el rendimiento. En cada capa, la información normalmente necesita ser transformada de una representación a otra.

1.14. Conclusiones parciales

Luego del análisis realizado a lo largo de este capítulo se han desarrollado los principales aspectos abordados durante la investigación, se han reflejado muchos de los conceptos más importantes a tener en cuenta para la continuidad del trabajo, por lo que se considera que los objetivos propuestos con el desarrollo de este capítulo han sido cumplidos.

De manera general, se analizaron los principios que rigen el funcionamiento del proceso administrativo en los TPC. Se lleva a cabo un estudio de los sistemas judiciales existentes en el mundo y sus características particulares. Se realiza un análisis de las herramientas de desarrollo, metodologías y tendencias a utilizar en la solución propuesta destacando sus características esenciales.

Capítulo II: Solución Propuesta

Introducción

En el presente capitulo se propone la solución técnica de la investigación. Se analiza la arquitectura del sistema a desarrollar, las diferentes capas que los componen y sus componentes. Se presenta el modelo de diseño conformado por los diagramas de clases y los diagramas de secuencia (Diagramas de Interacción) , el modelo de implementación con los diagramas de componentes y el diagrama de despliegue previamente diseñados. Se exponen los patrones de diseño utilizados en la solución propuesta.

2.1. Arquitectura del sistema

Los sistemas o arquitecturas en capas se definen como una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior. En la práctica, las capas suelen ser entidades complejas, compuestas de varios paquetes o subsistemas.
Las ventajas del estilo en capa son:

- ✓ 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.
- ✓ El estilo admite muy naturalmente optimizaciones y refinamientos.
- ✓ 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.

La arquitectura seleccionada para la realización de la aplicación es la arquitectura en capas por todas las ventajas antes mencionadas, compuesta por la Capa de Presentación; Capa de Control o Negocio; Capa Acceso a Datos; Capa de Dato.

Capa de presentación

En esta capa se emplea las facilidades que brinda el Marco de Trabajo ExtJS para la construcción de interfaces amigables a la vista de los usuarios. Ext 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, también estará un persistidor de configuración que es el encargado de comunicarse vía XML con los ficheros de configuración del sistema denominado FastResponse.

Capa de Dato

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

2.2. Estructura Organizativa.

Es importante conocer que para el desarrollo exitoso del proyecto se propone la realización de los siguientes modelos:

- Modelo de Negocio.
- Modelo de Casos de Uso.
- Modelo de Diseño.
- Modelo de Implementación.
- Modelo de Despliegue.
- Modelo de Datos.
- Modelo de Prueba.

Es por ello que existirá un paquete para cada modelo donde internamente cada uno de estos tendrá un paquete para cada módulo. Habrá un paquete donde estarán todas las clases del proyecto, su organización estará orientada a la arquitectura que propone el

framework SAUXE. Dentro de "Clases del proyecto" existirán dos paquetes más, "apps" y "web" donde se encuentran principalmente las clases necesarias para el desarrollo de la aplicación. Dentro de cada uno de estos habrá un paquete para cada módulo para organizar las clases correspondientes a cada uno de ellos, ya sean clases de presentación, de negocio o clase de datos. Dentro del paquete "web" estarán todas las clases de presentación o sea las clases del tipo js, y dentro del paquete "apps" estarán todas las clases del negocio y las clases de datos. Dentro del paquete "apps", en el paquete "Controllers" estarán las clases controladoras. Luego paralelo al paquete "Controllers" estará el paquete "Models" quien tendrá todas las clases que se generan por cada tabla de la base de datos. Se puede encontrar otro paquete llamado "Estereotipos" donde tendremos todos los estereotipos usados para el modelado **(Urrutia, 2010)**.

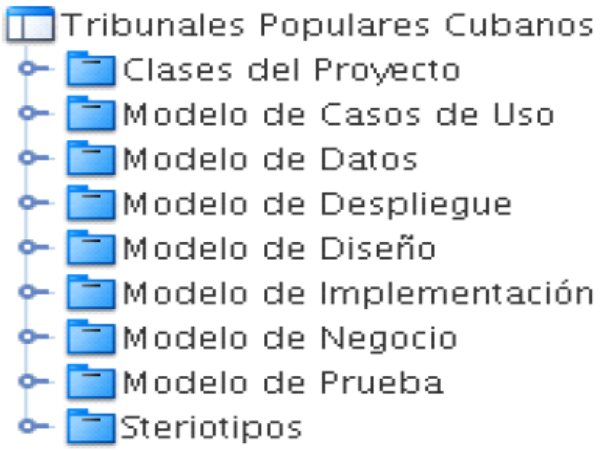

Figura 2: Estructura organizativa general en Visual Paradigm

2.3. 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.

2.3.1. Modelo de diseño

El modelo de diseño es un modelo de objetos que describe la realización de los casos de usos y sirve como abstracción del modelo de implementación y su código fuente.

Es usado tanto para concebir como para documentar el diseño de un sistema de software. Incluye los diagramas de interacción, de clases y el modelo de datos.

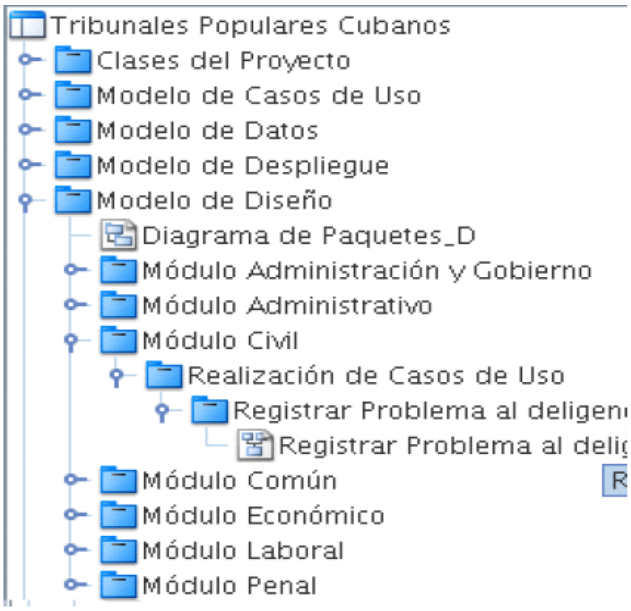

Figura 3: Estructura interna de un módulo dentro del paquete "Modelo de Diseño".

2.3.1.1. Diagramas de clase

Los diagramas de clases son muy utilizados durante el diseño para la representación de la estructura estática del sistema, mostrando las clases, sus atributos y las relaciones que se establecen entre las mismas. Un diagrama de clases del diseño contiene la siguiente información: clases, asociaciones y atributos, interfaces, con sus operaciones y constantes, métodos, información sobre los tipos de atributos, navegabilidad y dependencias.

A continuación se muestra una descripción de un caso de uso de una prioridad alta dentro del sistema y el diagrama de clase correspondiente:

Caso de uso Registrar escrito de demanda:

Caso de Uso:	Registrar escrito de demanda
Actores:	Abogado (inicia).
Resumen:	El caso de uso se inicia cuando el abogado necesita registrar un escrito de demanda. Consiste en que el abogado selecciona la opción registrar escrito de demanda insertando los datos necesarios. El caso de uso termina con la creación de la nueva demanda pasando esta a pendiente a admisión.

Precondiciones:	✓ El sistema debe estar instalado y ejecutándose correctamente. ✓ El usuario debe estar autenticado con los permisos necesarios.
Reglas del negocio:	3
Referencias:	RF.1, RF.2, RF.5, RF.8, RF.10, RF.12, CU Gestionar demandante (incluido), CU Gestionar demandado (incluido).
Prioridad:	Alta

Tabla 1: Descripción del Caso de Uso Registrar escrito de demanda

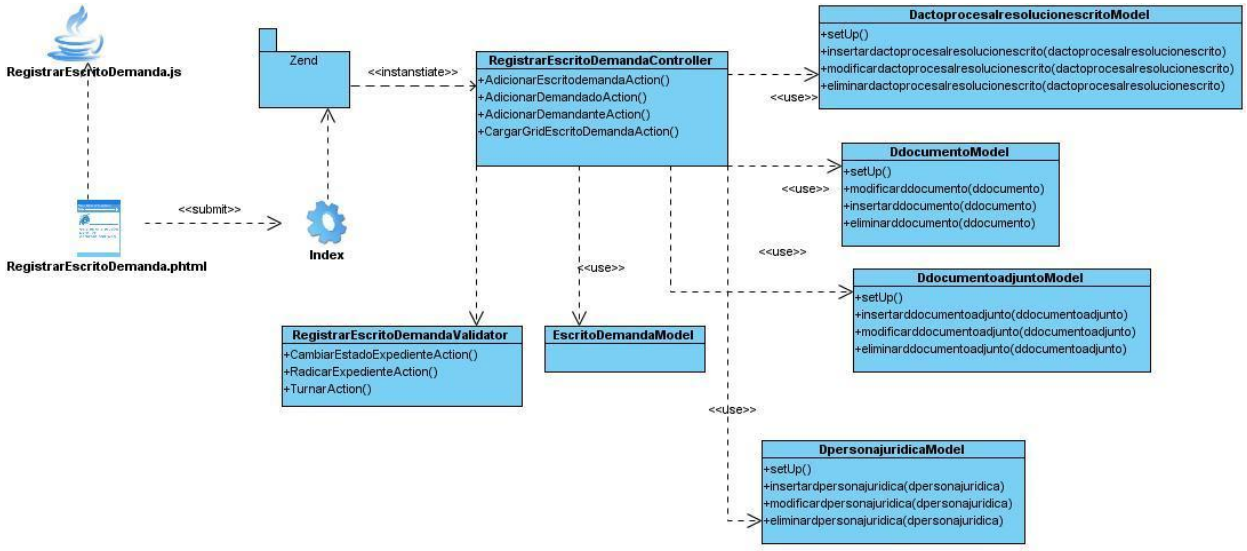

Figura 4: Diagrama de Clase: "Registrar escrito demanda".

2.3.1.2. Diagramas de interacción

Muestran gráficamente la relación entre los objetos a fin de cumplir con los requerimientos. Estos diagramas 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.

A continuación se muestra el diagrama de secuencia correspondiente al caso de uso descrito anteriormente:

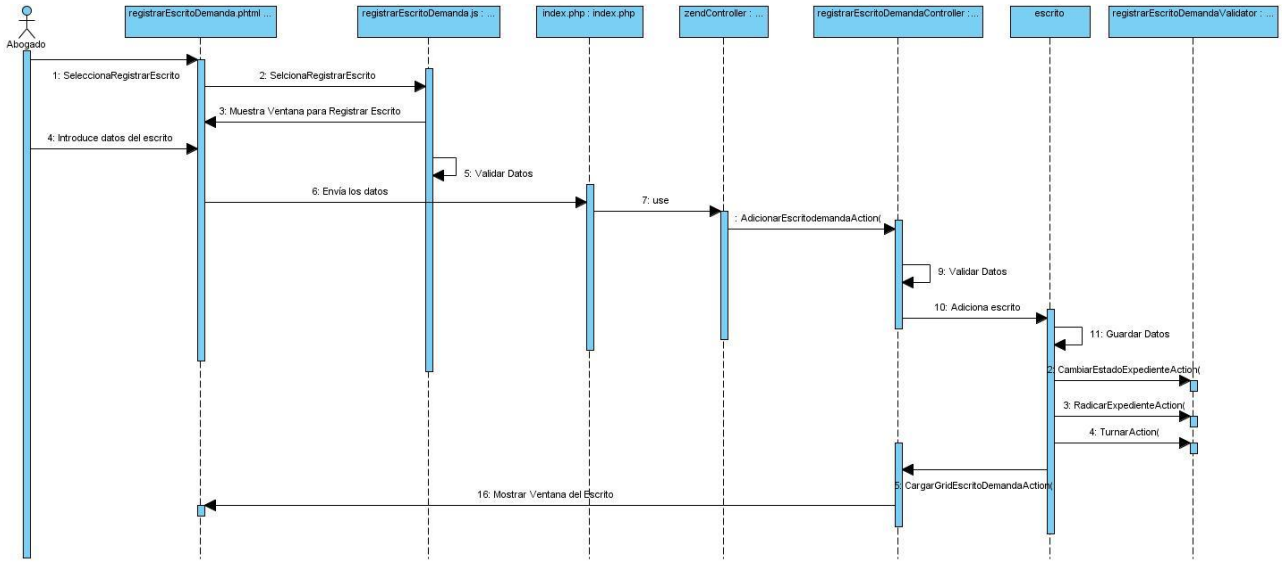

Figura 5: Diagrama de secuencia: "Registrar escrito demanda".

2.3.1.3. Modelo de datos

Un modelo de datos es la representación abstracta de los datos en un sistema gestor de base de datos. Básicamente el modelo de datos está formado por tres elementos fundamentales que son: los objetos, que son todas las entidades que manipulan los datos a persistir; los atributos, que son las características básicas de los objetos antes mencionados; y las relaciones, que son las que enlazan a dichos objetos entre sí.

El modelo de datos correspondiente al módulo consta con un total de 72 tablas, de ellas 29 nomencladores y las 43 restantes para el almacenamiento de la información correspondiente al flujo de trabajo del módulo. Para su construcción se tuvo en cuenta todos los datos que de alguna manera tienen que persistir en el sistema y en los nomencladores se agruparon los estándares predeterminados del negocio del proceso Administrativo.

Para ver el modelo de datos del proceso Administrativo referirse al Anexo#1: Modelo de Datos del proceso Administrativo.

2.3.2. Modelo de implementación

Un modelo de implementación muestra las dependencias entre las partes de código del sistema y la estructura del sistema en ejecución. Como parte del Modelo de Implementación se encuentran los Diagramas de Componentes y los Diagramas de Despliegue.

2.3.2.1. Diagrama de componentes

El diagrama de componentes es un diagrama que muestra un conjunto de elementos del modelo, tales como componentes, subsistemas de implementación y sus relaciones. Se utiliza para modelar la vista estática de un sistema. Muestra la organización y las dependencias entre un conjunto de componentes software, sean estos componentes de código fuente, librerías, binarios o ejecutables.

Seguidamente se muestra el diagrama de componentes del sistema:

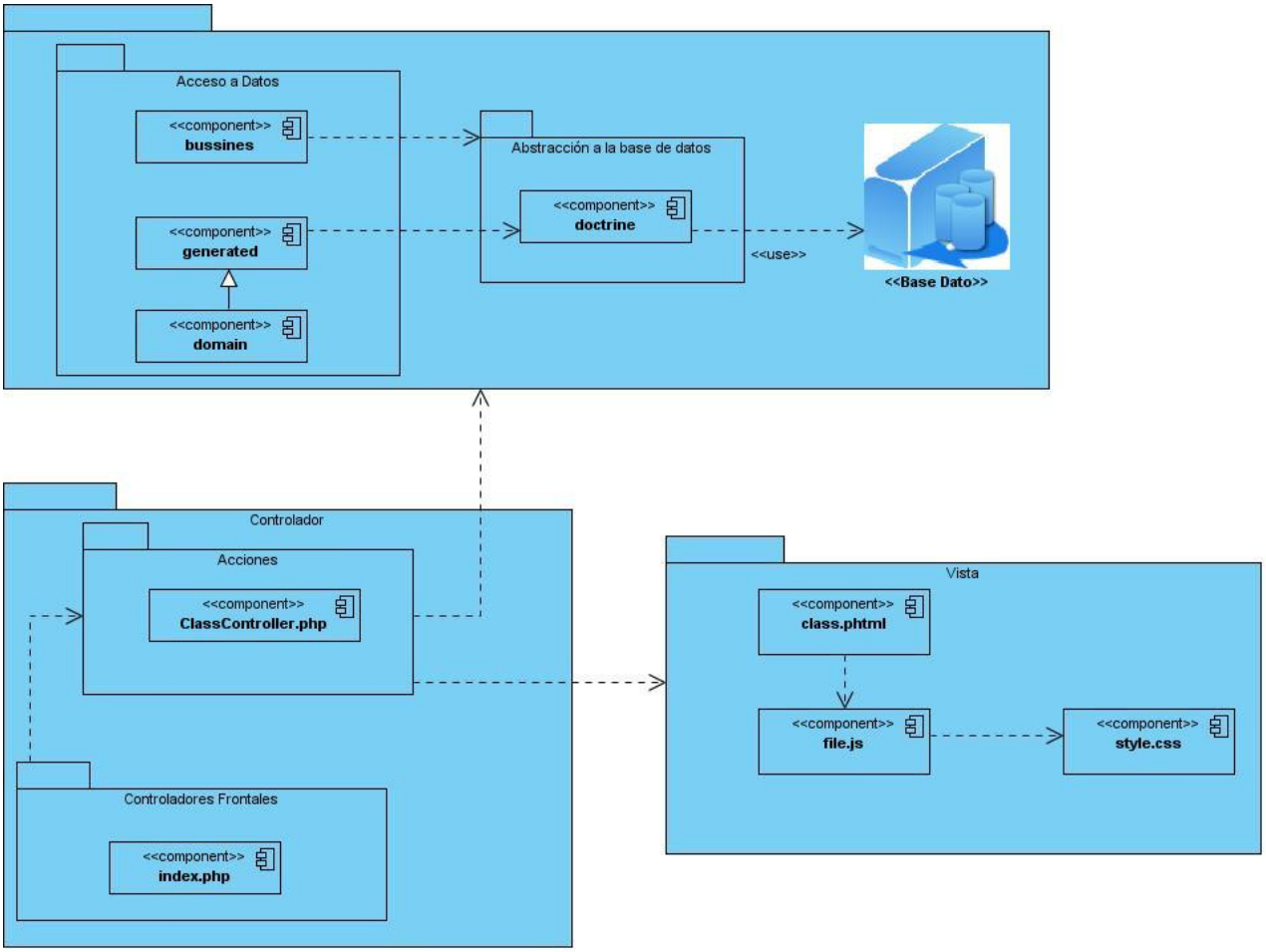

Figura 6: Diagrama de componentes de la solución informática

2.3.2.2. Diagrama de despliegue

Los diagramas de despliegue muestran las relaciones físicas de los distintos nodos 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. Un nodo es un recurso de ejecución tal como un computador, un dispositivo o memoria.

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:

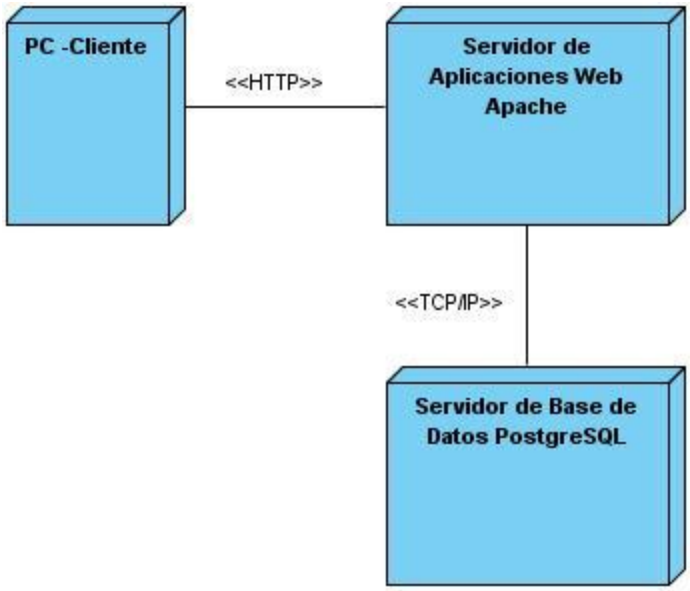

Figura 7: Diagrama de Despliegue de la solución informática

Algunos de los diagramas de Clases del diseño correspondientes al proceso Administrativo aparecen en el Anexo # 2: Descripción del Caso de Uso Gestionar Demandado, Anexo # 3: Descripción del CU Gestionar Expediente Pendiente de Admisión y Anexo #4: Descripción del CU Disponer sobre Escritos de Personería.

2.4. Estándar de Codificación

Un estándar de codificación completo comprende todos los aspectos de la generación de código. Si bien los programadores deben implementar un estándar de forma prudente, éste debe tender siempre a lo práctico. Un código fuente completo debe reflejar un estilo armonioso, como si un único programador hubiera escrito todo el código de una sola vez. La legibilidad del código fuente repercute directamente en lo bien que un programador comprende un sistema de software. El mejor método para asegurarse de que un equipo de programadores mantenga un código de calidad es establecer un estándar de codificación sobre el que se efectuarán luego revisiones del código de rutinas. Usar técnicas de codificación sólidas y realizar buenas prácticas de programación con vistas a generar un código de alta calidad es de gran importancia

para la calidad del software y para obtener un buen rendimiento y mantenimiento del software.

La adopción de un estándar de codificación sólo es viable si se sigue desde el principio hasta el final del proyecto de software. No es práctico, ni prudente, imponer un estándar de codificación una vez iniciado el trabajo. En el caso del proyecto Tribunales Populares Cubanos es necesario aclarar que tanto el diseño para PHP como para Java Script se debe hacer orientado a objetos a fin de potenciar todas las ventajas que este paradigma implica, el estándar de codificación fue definido por la dirección del proyecto en sus inicios y es basado en los estándares de codificación para PHP **(Urrutia, 2010)**:

- ✓ **Identación.**

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

- ✓ **Cabecera del archivo.**

Todos los archivos .php inician con una cabecera específica que indica la información de la versión, autor de los últimos cambios, etc. Cada equipo decide si se quiere o no agregar más datos.

```
/**
*
* @Control de presentación de los weblogs. "weblog.php"
* @versión: 5.4.2      @modificado: 3 de febrero del 2011
* @autor: Alex
*
*/
```

- ✓ **Comentarios en las funciones.**

Todas las funciones deben tener un comentario, antes de su declaración, explicando que hacen. Ningún programador debe 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 son colocadas en un archivo .php aparte, donde sólo se coloca el código de la clase. El nombre del archivo será el mismo del de la clase y siempre empezará en mayúscula. Se debe 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.

- ✓ **Nombres de variables.**

No se debe adoptar la ***notación húngara***[5] en el código. Muchos proyectos afirman que es una de las técnicas de ofuscación de código más ampliamente usadas en la actualidad.

Los nombres deben ser descriptivos y concisos. No usar ni grandes frases ni pequeñas abreviaciones para las variables. Siempre es mejor saber qué hace una variable con sólo conocer su nombre. Esto aplica para los nombres de variables, funciones, argumentos de funciones y clases.

Todos los nombres deben estar en minúscula (Excepto con las clases, donde la primera letra ha de ser mayúscula). En caso de usar más de una palabra, ésta será separada por un signo de underscore "_".

En las **funciones**, es importante que el nombre denote su función inmediatamente. Cosas como *imprimir_datos* están bien, pero estaría mejor *imprimir_datos_usuario*. De igual manera, en los **argumentos** de las funciones se quiere saber inmediatamente que se está usando. Es mejor *crear_usuario($nick, $email)* que *crear($n, $e)*.

La filosofía es sencilla. No se debe dañar la legibilidad del código por pereza, aplicar el sentido común y no crear funciones de más de 4 palabras.

- ✓ **Siempre incluir las llaves.**

[5] Es aquella donde se coloca el tipo de dato antes del nombre de variable, ej: strNombre para una variable de tipo string.

En todo momento a la hora de codificar un bloque de instrucciones, éste debe ir encerrado entre llaves, aun cuando conste de una sola línea.

```
if ($cosa)
{
        funcion();
}
```

No se gasta mucho tiempo adicional y se gana muchísimo en legibilidad.

- ✓ **Llaves. Donde colocarlas.**

Todos los corchetes van en una línea propia.

```
if (algo)
{
        for (iteracion)
        {
                //código
        }
}
while (condición)
{
        funcion();
}
```

- ✓ **Poner espacios entre signos.**

Si se tiene un signo y operador binario, se colocan espacios a ambos lados. Si se tiene un signo unario, se colocan espacios a uno de sus lados. En términos más simples, se programa como si se escribiera ***bien*** en español. Este elemento es algo muy sencillo que ayuda a la legibilidad del código.

Esto está mal:

```
$a=0;
for($i=5;$i<=$j;$i++)
```

Esto está bien:

```
$a = 0;
```

```
for ($i = 5; $i <= $j; $i++)
```

- ✓ **Precedencia de operadores**

Lo mejor es siempre usar paréntesis para estar seguro de la precedencia de los operadores. Básicamente, la idea es no codificar operaciones complejas y estar seguros que nuestros compañeros de equipo con menos "habilidad" comprendan todo sin problemas:

```
//Incorrecto
$bool = ($i < 7 && $j > 8 || $k == 4);

//Correcto
$bool = (($i < 7) && (($j < 8) || ($k == 4)));

//Incluso mejor
$bool = ($i < 7 && ($j < 8 || $k == 4));
```

- ✓ **Operadores unarios de suma y resta.**

Se deben utilizar en una sola línea y a la derecha, por ejemplo:

```
//Esto está MAL
$cosa = $matriz[$i--];
$otra = $matriz[++$y];

//Esto está BIEN
$y++;
$cosa = $matriz[$i];
$otra = $matriz[$y];
$i--;
```

2.5. Patrones de diseño utilizados

Modelo Vista Controlador

Este patrón de diseño viene definido por el framework Sauxe. Está estructurado para crear las clases por separadas, las clases del modelo, que son la lógica del negocio y la cual representa la información en la cual la aplicación opera, están en el paquete **app\administrativo\Model**. Las clases de la vista, que son las que rende rizan el modelo dentro de una página web apropiada para que el usuario pueda interactuar, se encuentran en el paquete **web\administrativo\Views** y 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\administrativo\Controllers**.

Este patrón separa la lógica de negocio (model) y la presentación (view), resultando en un código muy mantenible.

Figura 8: Estructura del paquete Apps, donde se encuentran las clases controladoras y las del modelo, aplicando el patrón MVC.

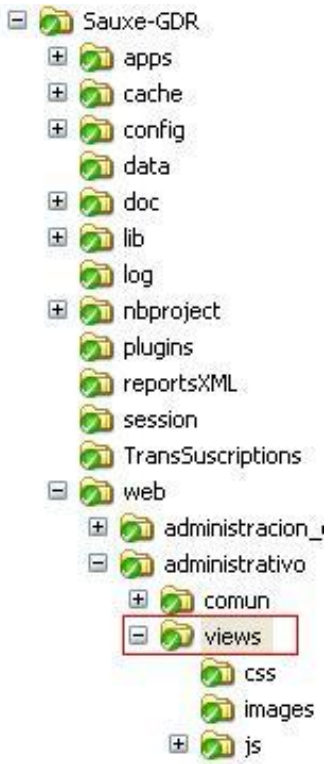

Figura 9: Estructura del paquete Web, donde se encuentran las clases de la vista, aplicando el patrón MVC.

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. 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, además se desea testar las clases aisladamente sin sus dependencias y se quiere desacoplar sus clases de ser responsables de localizar y gestionar el tiempo de vida de sus dependencias **(Guillerón, 2009)**.

Este patrón es utilizado por el framework para crear y brindar 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 framework a través del patrón IoC, define la creación de clases, para de ellas brindar los servicios necesarios.

```
<?php
class ProcedimientosService {

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

}
?>
```

Figura 10: Representación de la clase ProcedimientosService, ubicada en el paquete apps\administrativo\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.

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

Figura 11: Fichero ioc-general.xml en al paquete apps\administrativo\común\recursos\xml, donde se crea el servicio ObtenerProcedimientos.

```
function cargarComboProcesosAction() {
        $idninstancia = $this->_request->getPost('idinstancia');
        $materia = $this->_request->getPost('materia');
        if($materia == 'Penal')
            $procedimientos = $this->integrator->penal->ObtenerProcedimientos($idninstancia);
        else if($materia == 'Civil')
            {print_r("Llamar a servicio de civil");die();}
        else if($materia == 'Laboral')
            {print_r("Llamar a servicio de laboral");die();}
        else if($materia == 'Administrativo')
            $procedimientos = $this->integrator->administrativo->ObtenerProcedimientos($idninstancia);
        else if($materia == 'Economico')
            {print_r("Llamar a servicio de economico");die();}
        echo json_encode($procedimientos);
    }
```

Figura 12: Representación de la función cargarComboProcesosAction en la clase InicioController.php del módulo Común, la cual consume el servicio ObtenerProcedimientos brindado por el módulo Administrativo.

Singlenton

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)**.

En el sistema propuesto se utiliza el patrón singleton con el objetivo que de una clase solo exista una sola instancia.

```
class NescritoExt extends Nescrito
{
    public function setUp()
    {
     parent::setUp();
    }

    /**
     * constructor
     *
     * this is private constructor (use getInstance to get an instance of this class)
     */
    private function __construct(){}

    /**
     * Returns an instance of this class
     * (this class uses the singleton pattern)
     *
     */
    public static function getInstance()
    {
        static $instance;
        if ( ! isset($instance)) {
            $instance = new self();
        }
        return $instance;
    }
}
```

Figura 13: Representación de la clase NescritoExt, donde se usa el patrón Singleton.

```
function cargarComboTipoEscritoAction()
        {
            $obj = NescritoExt::getInstance();
                    $tipoescrito=$obj->ListarEscritos();
                    $datos=$tipoescrito->toArray();
                    echo json_encode($datos);return;
        }
```

Figura 14: Representación de la función cargarComboTipoEscritoAction, donde se solicita una instancia de la clase NescritoExt.

2.6. Estrategia trazada para el diseño

Para la realización de la solución propuesta en el proyecto implementó una estrategia para el diseño, basada en los patrones arquitectónicos y de diseño propuestos para el desarrollo **(Urrutia, 2010)**. A continuación se exponen los pasos a seguir en dicha estrategia.

1. Confección de los modelos conceptuales para cada uno de las fases dentro de la inscripción de documentos.

2. Confección del modelo de datos para la solución informática para el subsistema Administrativo.

3. Realización de los diagramas de clases correspondientes a cada caso de uso que se implementa, unido a los diagramas de secuencia correspondientes para cada uno de los escenarios que se necesiten.

4. Generación de código de las clases del dominio, y los gestores del negocio y de acceso a datos.

5. Capacitación al programador de los diagramas hechos para cada caso de uso, en específico aquel que le toque implementar.

2.7. Conclusiones parciales

En este capítulo se expuso la solución propuesta para el sistema, mostrándose cada uno de los elementos que componen la arquitectura definida, algunos de los patrones de diseño y los componentes usados para la implementación, los artefactos propios de los flujos de trabajo bases a defender en este trabajo, el modelo de diseño y el de implementación así como el estándar a seguir para la codificación del proyecto y las estrategias para el diseño.

Capítulo III: Análisis de Resultados

3.1. Introducción

En este capítulo se evalúa el grado de calidad y fiabilidad de los resultados obtenidos con el desarrollo de este trabajo, teniendo por objeto 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.

3.2 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:

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

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, así como los propios internos.

Valoración directa: Es una valoración del producto, de forma indirecta o 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.

3.2.1. Valoraciones aplicadas a la solución propuesta.

Dentro de los distintos tipos de validación del software existentes se escogieron para la solución propuesta las de tipo cualitativa y directa. Debido a los problemas actuales que existen con las licencias de software el sistema se realizó sobre plataforma libre implementándose en el lenguaje PHP utilizando el Zend Framework por decisión del grupo de arquitectura del Proyecto TPC donde fue desarrollado el sistema. Se empleó como gestor de base de datos la herramienta PostgreSQL debido a que es una herramienta con características importantes como: presenta interfaz amigable y ágil de manipular. Para el acceso a datos se utilizó el framework de acceso a datos Doctrine. Se utilizó el framework Ext-JS para el desarrollo de la interfaz visual. Todas estas herramientas fueron una decisión del grupo de arquitectura del proyecto TPC.

3.3. 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. Siendo esto la principal razón de la concepción de las métricas inspiradas en lo propuesto por Pressman **(Pressman, 2005)**. Los atributos de calidad que se abarcan son:

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 de 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 del 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, modulo, clase, conjunto de clases, etc.) diseñado.

Las métricas concebidas como instrumento para evaluar la calidad del diseño del proceso Administrativo del Sistema Integral de Tribunales y su relación con los atributos de calidad definidos en este trabajo son las siguientes:

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

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: Tabla (Tamaño operacional de clase (TOC))

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

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 3: Tabla (Relaciones entre clases (RC))

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

Ver instrumentos y tabla de resultados en (Anexo # 5: Instrumento de medición de la métrica Tamaño operacional de clase (TOC)).

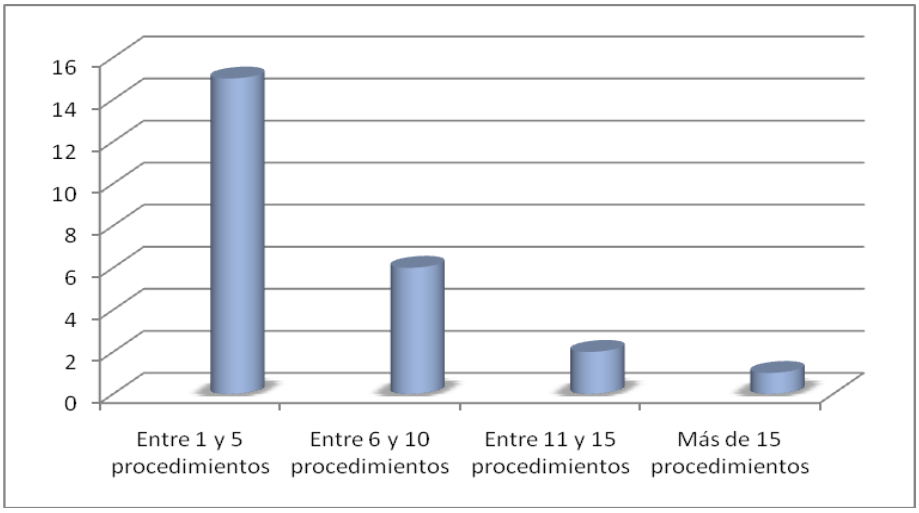

Figura 15: Representación de los resultados obtenidos en el instrumento agrupados en los intervalos definidos.

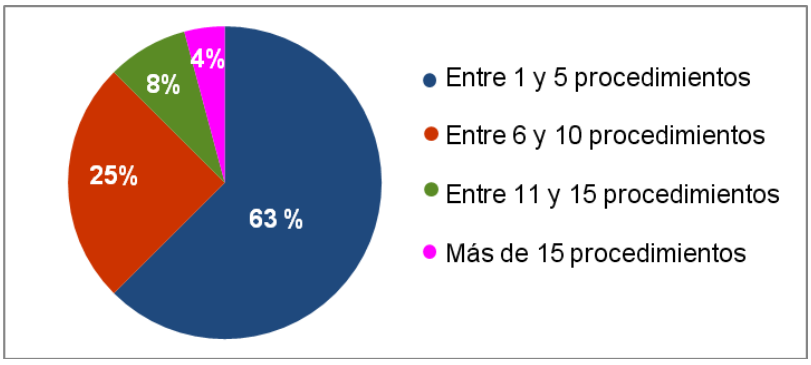

Figura 16: Representación en % de los resultados obtenidos en el instrumento agrupados en los intervalos definidos.

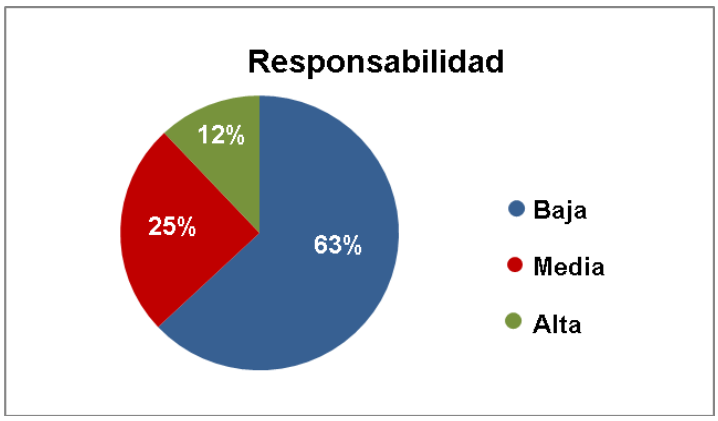

Figura 17: Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo responsabilidad

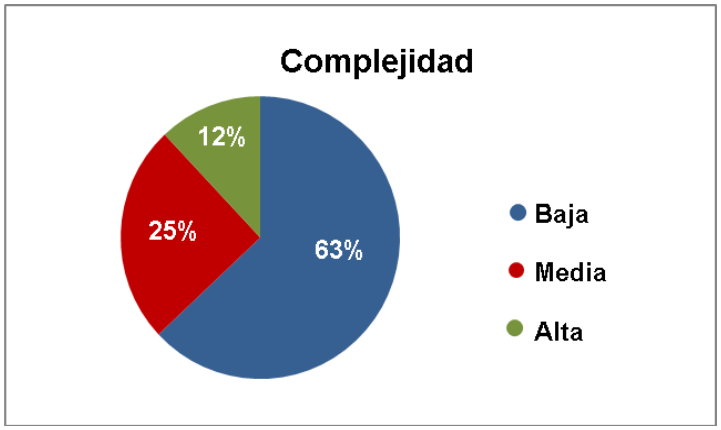

Figura 18: Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Complejidad de Implementación.

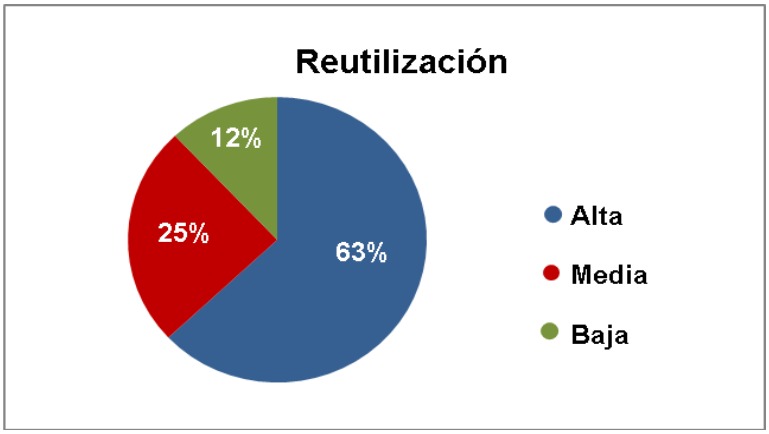

Figura 19: Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Reutilización.

Haciendo un análisis de los resultados obtenidos en la evaluación del instrumento de medición de la métrica TOC, se puede concluir que el diseño del proceso administrativo del Sistema Integral de Tribunales tiene una buena calidad teniendo en cuenta que el 87,5 % de las clases incluidas en este sistema posee menos cantidad de operaciones que la mitad del valor máximo registrado en las mediciones que fue de 21 procedimientos. Esto influye de manera positiva en el hecho de que predomine una responsabilidad baja de las clases en un 63%. Además el 63% de las clases poseen evaluaciones positivas en los atributos de calidad complejidad de implementación y reutilización, lo cual garantiza una solidez en el diseño del sistema puesto que habrá gran reutilización de las clases y su implementación no resulta complicada.

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

Ver instrumentos y tabla de resultados en (Anexo # 6: Instrumento de medición de la métrica Relaciones entre clases (RC)).

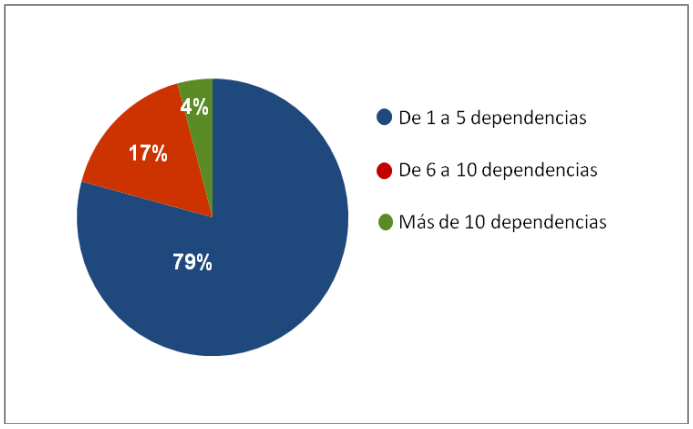

Figura 20: Representación en % de los resultados obtenidos en el instrumento agrupados en los intervalos definidos.

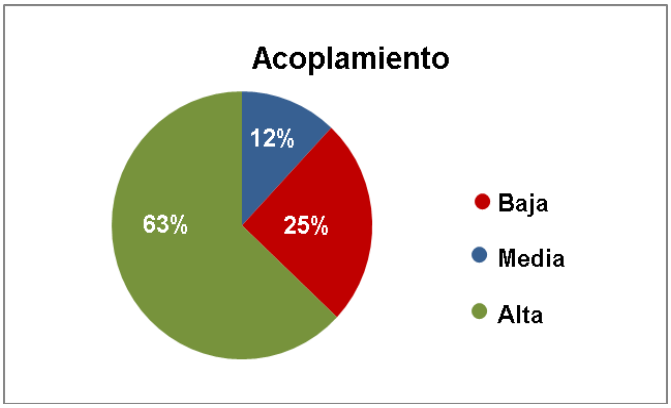

Figura 21: Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Acoplamiento.

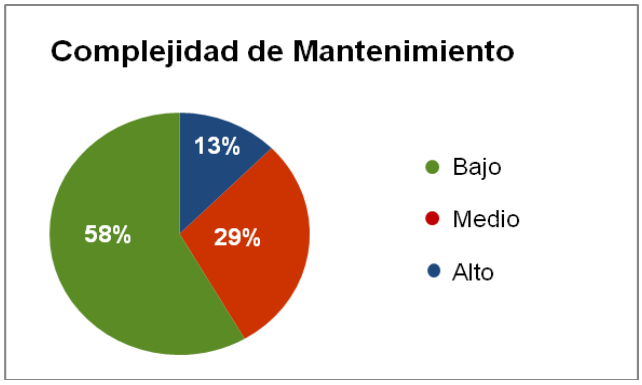

Figura 22: Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Complejidad de Mantenimiento.

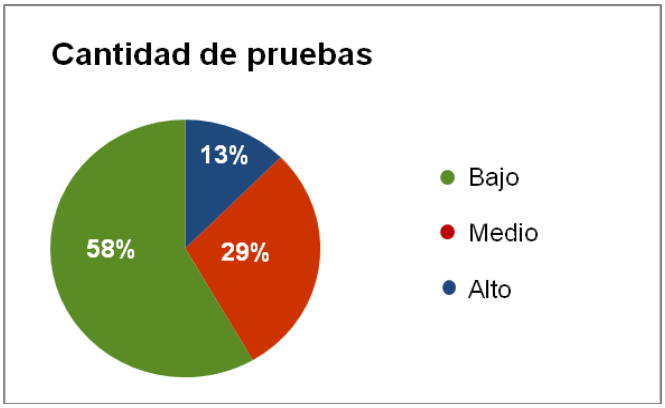

Figura 23: Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Cantidad de Pruebas

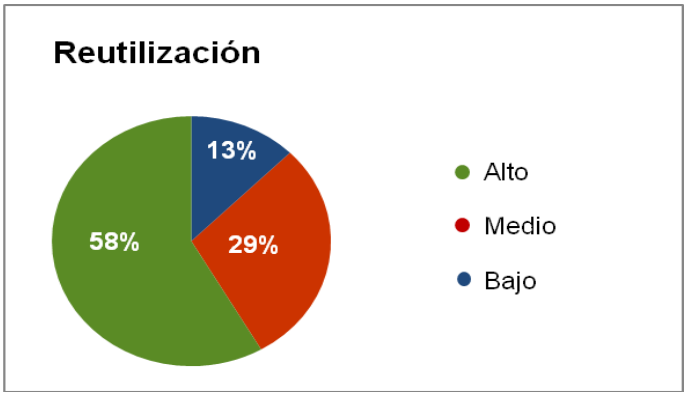

Figura 24: Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Reutilización.

Haciendo un análisis de los resultados obtenidos en la evaluación del instrumento de medición de la métrica RC, se puede concluir que el diseño del proceso Administrativo del Sistema Integral de Tribunales tiene una buena calidad teniendo en cuenta que el 55% de las clases incluidas en este subsistema posee 3 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 25% de las clases posee bajos índices en cuanto a Acoplamiento, este es un factor en el cual se debe trabajar, para posibilitar que se realicen modificaciones en el sistema teniendo poco impacto en el mismo y que entre las clases exista un bajo acoplamiento. Por su parte los atributos de calidad Complejidad de Mantenimiento, Cantidad de Pruebas y Reutilización se comportan satisfactoriamente en un 79% de las clases, teniendo 5 o menos dependencias de otras clases, fomentando el reúso de las mismas y facilitando el mantenimiento del sistema así como las pruebas necesarias para la comprobación del software.

3.4 Pruebas de Caja Blanca o Estructurales.

Se denomina cajas blancas a un tipo de prueba de software que se realiza sobre las funciones internas de un módulo. 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, para luego realizar las de caja negra sobre varios subsistemas.

Algunas técnicas de prueba de Caja Blanca son:

- ✓ **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 selecciona caminos de prueba 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.

Para realizar la prueba se usará la del Camino Básico, debido a que permite obtener una medida de la complejidad lógica del diseño y usar la misma como guía para la definición de un conjunto de caminos básicos, garantizando que durante la prueba se ejecute al menos una vez cada sentencia del programa.

Para ello es necesario conocer el número de caminos independientes de un determinado algoritmo mediante el cálculo de la complejidad ciclomática. Se debe comenzar por un análisis del código, posteriormente son enumeradas cada una de las instrucciones, se construye el grafo de flujo asociado y según las fórmulas pertinentes se calcula dicha complejidad.

A continuación se analizan y enumeran las sentencias de código de uno de los métodos contenidos en la clase RegistrarescritodemandaController, específicamente: cargarGridDemandadosAction, este por su parte se encarga de cargar en un gridpanel los datos enviados por el caso de uso Gestionar Demandado. 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 cargarGridDemandadosAction()
        {
            $datos = array();//1
            if(isset($_SESSION['Tipo de Sujeto']))//2
            for($i = 0;$i<count($_SESSION['Tipo de Sujeto']);$i++)//3

            {
                $datos[$i]['tiposujetodemandado']= $_SESSION['Tipo de Sujeto'][$i];//4
                $datos[$i]['nombredenominacion']= $_SESSION['Dpersonanatural'][$i]['primernombre'];//4
                $datos[$i]['domiciliolegal']= $_SESSION['Ddireccion'][$i]['direccion'];//4
            }
            $result =  array('cantidad_filas'=> count($datos), 'datos' => $datos);//5
            echo json_encode($result);return;//5

            }
```

Figura 25: Método cargarGridDemandadosAction.

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:

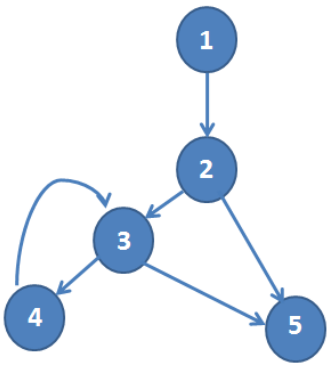

Figura 26: Grafo de flujo asociado al algoritmo cargarGridDemandadosAction.

Una vez construido el grafo de flujo asociado al procedimiento anterior se determina la complejidad ciclomática, el cálculo 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) = (6 -5) + 2

V (G) = 3.

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) = 2 + 1

V (G) = 3.

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) = 3

El cálculo efectuado mediante las fórmulas antes presentadas muestran una complejidad ciclomática de valor 3, de manera que existen 3 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 -- 5

Camino básico #2: 1 – 2 – 3 – 4 – 3 – 5

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: Se deben cargar en el gridpanel los datos que envía el CU GestionarDemandado.

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

Entrada: El objeto SESSION no esta seteado.

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 # 1.

Condición de ejecución: Para el algoritmo es necesario que en el objeto SESSION en la posición "tipo de sujeto" 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 salgan los demandados.

El resultado obtenido fue correcto.

3.5. Pruebas de Caja Negra o Funcionales.

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.

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:

- ✓ 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**.**

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, descubre de

forma inmediata una clase de errores 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.

En la sección de anexos aparece el Diseño de Caso de Prueba aplicado a la funcionalidad Gestionar Expedientes Pendientes de Admisión. Anexo # 7: Caso de prueba de Caja Negra para validar el requisito Gestionar Expedientes Pendientes de Admisión.

Los resultados obtenidos a través de la realización de los métodos de prueba expuestos con anterioridad como parte de las pruebas internas realizadas a los subsistemas, fueron satisfactorios desde el punto de vista interno y funcional, atendiendo al correcto comportamiento del mismo ante diferentes situaciones.

3.6. Conclusiones parciales

En el desarrollo de este capítulo se efectuó un análisis de las diferentes pruebas que se le pueden realizar a un software, haciendo énfasis en sus objetivos y alcance. Se realizó una descripción de los test de unidad, dentro de los cuales se analizaron los tipos de prueba: Caja Blanca y Caja Negra, la ejecución de las mismas y los resultados obtenidos, además se evaluó mediante métricas el diseño propuesto. Resulta importante explicar que los casos de pruebas presentados no son los únicos diseñados para las aplicaciones sino que el objetivo es plasmar la forma de la realización de los mismos.

Conclusiones

Como conclusiones generales de la investigación se tiene que:

- ✓ Con el estudio de los patrones de diseño y buenas prácticas de programación se logró definir el estándar de codificación, así como los patrones a usar para una mayor organización y claridad a la hora de desarrollar el sistema.
- ✓ Con la realización de los diagramas de clases, interacción, componentes y despliegue se orientaron tanto al desarrollador, como a los arquitectos y clientes en la organización y en los elementos necesarios para la puesta en marcha del sistema.
- ✓ A partir de los diagramas de clases del diseño se implementó el flujo principal del proceso administrativo para el sistema de tribunales en Cuba.
- ✓ La aplicación de métricas permitió determinar que el diseño y la implementación de la solución propuesta presentan el nivel de calidad requerida.

Recomendaciones

Con la realización de este trabajo se recomienda:

1- Continuar con el diseño y la implementación de los casos de uso del proceso Administrativo.

2- Comenzar con el análisis, diseño y posterior implementación de los procesos Casación y Revisión, de la Materia Administrativa, en la instancia superior.

Referencias bibliográficas

1. **Oracle.** Java EE at a Glance. [Online] http://www.oracle.com/technetwork/java/javaee/overview/index.html.
2. **2003-2004.** Servidores de Aplicaciones. [Online] 2003-2004. http://www.jtech.ua.es/j2ee/2003-2004/abierto-j2ee-2003-2004/sa/sesion1-apuntes.htm.
3. **Ivar Jacobson, Grady Booch, James Rumbaugh. 2000.** *El Proceso Unificado de Desarrollo de Software.* Madrid : Pearson Addisson-Wesley, 2000.
4. **Infolex. 2007.** Infolex, Gestión Jurídica. [Online] 2007. http://www.infolex.es/ie/index.aspx.
5. **Corporation, Oracle. 2011.** NetBeans. [Online] 2011. http://netbeans.org/.
6. **PHP. 2001-2011.** PHP: Hypertext Preprocessor. [Online] 2001-2011. http://www.php.net/.
7. **Flanagan, David. 2002.** *JavaScript: The Definitive Guide (4 ª Edición edición).* 2002. ISBN 0-596-00048-0.
8. **Paradigm, Visual. 2011.** Visual Paradigm. [Online] Visual Paradigm, 2011. http://www.visual-paradigm.com.
9. **Systems, Sparx. 2000-2007.** Enterprise Architect - Herramienta de diseño UML. [Online] 2000-2007. http://www.sparxsystems.com.ar/products/ea.html.
10. **Abogados, Soft Class para. 2004.** Soft Class para Abogados 2004. [Online] 2004. http://www.softclass.com/.
11. **Santos, Alberto Martínez de. 2010.** Justicia y Prehistoria: Otra de LEXNET en el tránsito a la NOJ y al más allá. [Online] 4 26, 2010. http://justiciayprehistoria.blogspot.com/2010/04/otra-de-lexnet-en-el-transito-la-noj-y.html.
12. **Baryolo, Gómez, Tenrero, Cabrera, Marianela y Silega, Martínez, Nemuris. 2008.** *Plantilla Registro de la Propiedad intelectual(Sauxe).* 2008.
13. **Ing. Oiner Gómez Baryolo, Ing. Yoandry Morejón Borbón , Ing. Darien García Tejo. 2008.** *ARQUITECTURA TECNOLÓGICA PARA EL DESARROLLO DE SOFTWARE.* Habana : s.n., 2008.
14. **Budd, Timothy. 1991.** *An introduction to Object-Oriented Programming.* 1991. ISBN: 0-201-54709-0.

15. **Carlos Reynoso, Nicolás Kiccillof. 2004.** Estilos y Patrones en la Estrategia de Arquitectura de Microsoft. [Online] 3 2004. http://cic.javerianacali.edu.co/wiki/lib/exe/fetch.php?media=materias:estiloypatron.pdf.

16. Computer Glossary, Computer Terms. [Online] http://whatis.techtarget.com/.

17. **Alvarez, Miguel Angel. 2007.** Aptana Studio. [Online] 11 16, 2007. http://www.desarrolloweb.com/articulos/aptana-studio.html.

18. **Alicante, Universidad de. 2011.** Especialista Universitario: Developing Java Enterprise Applications. [Online] 2011. http://www.jtech.ua.es/j2ee/ejemplos/sa/sesion1-apuntes.htm.

19. **Oracle. 2011.** Oracle and Bea Systems. [Online] 2011. http://www.oracle.com/us/products/middleware/application-server/index.html.

20. **Oracle. 2011.** Sun GlassFish Enterprise Server. [Online] 2011. http://www.sun.com/software/products/appsrvr/index.xml.

21. **Foundation., The Apache Software. 2011.** Documentación del Servidor HTTP Apache 2.0. [Online] 2011. http://httpd.apache.org/docs/2.0/es/.

22. **Oracle. 2011.** Oracle Database. [Online] 2011. http://www.oracle.com/es/products/database/index.html.

23. **EP-Internacional. 2010.** Globedia. [Online] 6 1, 2010. [Cited: 10 12, 2010.] http://cu.globedia.com/cuba-comenzara-2011-digitalizacion-sistema-judicial.

24. **Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. 1995.** *Design Patterns: Elements of Reusable Object-Oriented Software.* 1995. ISBN 0-201-63361-2.

25. **Pressman, Roger S. 2005.** *La Ingeniería del Software, un enfoque práctico.* 2005. ISBN: 9701054733.

26. **Sáez, Dora Pérez. 2008.** Tribunales populares: garantía de justicia. *Juventud Rebelde, Edición digital.* 2008.

27. **Solution, PTS. 1988.** Professional & Technical Software Solutions. [Online] 1988. http://www.ptssolutions.com/court_records_software.shtml.

28. **Alfonso, Jorge. 2010.** Tratando de entenderlo: Patrones de Diseño: Singleton. [Online]1 26, 2010. http://tratandodeentenderlo.blogspot.com/2010/01/patrones-de-diseno-singleton.html.

29. **Urrutia, Ing. Misael Rodriguez. 2010.** *Manual de Diseño del proyecto TPC.* Habana : s.n., 2010.

30. **Urrutia, Ing. Misael Rodríguez. 2010.** *Pautas a seguir para la codificación del proyecto Tribunales Populares Cubanos en el lenguaje PHP.* Habana : s.n., 2010.

31. **Guillerón, Gastón. 2009.** Gln-Blog. [Online] 2009. http://glnconsultora.com/blog/?p=3.