Universidad de las Ciencias Informáticas

Facultad III

Título: Paquete de componentes para el sistema de los Tribunales Populares Cubanos.

Trabajo de diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autores: Manuel Urquiza Rodríguez
Antonio Nodal Delgado

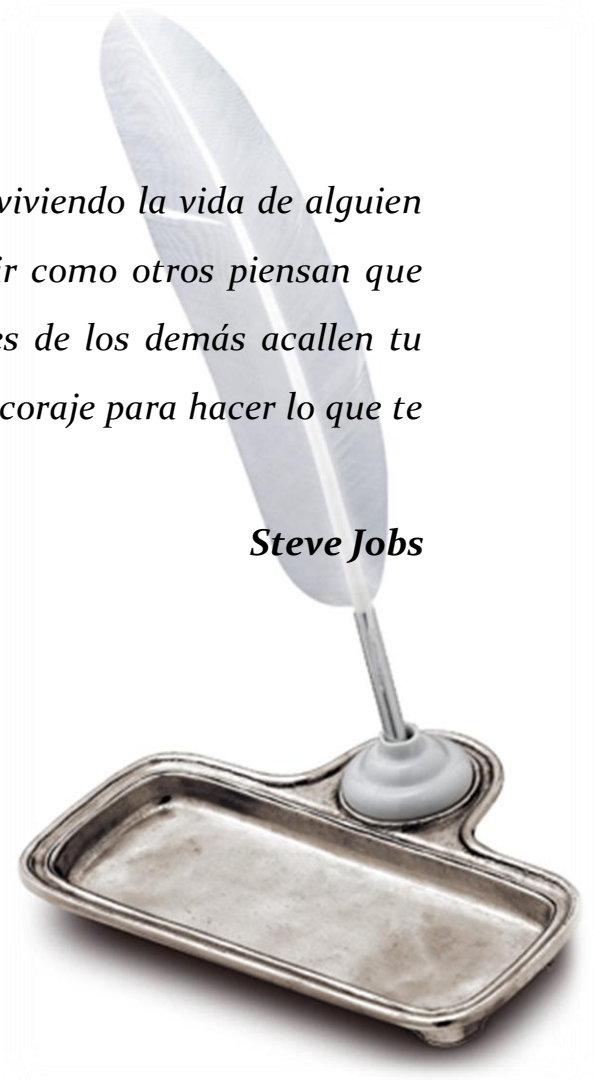
Tutor: Ing. Martín Villalón Cruzata

Co-Tutor: Ing. Daimi Lamorú Marciel

La Habana, Cuba
Curso: 2011-2012

“Tu tiempo es limitado, de modo que no lo malgastes viviendo la vida de alguien distinto. No quedes atrapado en el dogma, que es vivir como otros piensan que deberías vivir. No dejes que los ruidos de las opiniones de los demás acallen tu propia voz interior. Y, lo que es más importante, ten el coraje para hacer lo que te dicen tu corazón y tu intuición”.

Steve Jobs



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 _____.

Autores:

_____ _____

Manuel Urquiza Rodríguez Antonio Nodal Delgado

Tutor: Co-Tutor:

_____ _____

Ing. Martín Villalón Cruzata Ing. Daimi Lamorú Marciel

Datos de Contacto

Tutor: Ing. Martín Villalón Cruzata

Categoría Docente: Instructor.

Síntesis: Graduado en la Universidad de las Ciencias Informáticas. Profesor de Programación.

Correo electrónico: mvillalon@uci.cu

Teléfono: 837 2269

Co-Tutora: Ing. Daimi Lamorú Marciel

Categoría Docente: Instructor.

Síntesis: Graduado en la Universidad de las Ciencias Informáticas. Actualmente desempeña el rol de analista del Módulo Común del Proyecto Tribunales Populares Cubanos.

Correo electrónico: dlamoru@uci.cu

Teléfono: 8358878

Agradecimientos

A mi madre, por ser un ejemplo para mí, por ser madre y padre, y por ser ese motor impulsor que me guió durante todo este largo camino.

A mi padrastro, por ser como un padre para mí, por el apoyo incondicional que siempre me brindaste y por transmitirme esa seguridad que siempre hace falta.

A mi hermano menor por ser esa persona con quien me puedo sentar y contarle cualquier cosa. Gracias por estar ahí siempre para mí.

A mi hermano mayor, por ser ejemplo para mí en todas esas cosas de la vida en las que a veces unos somos menos espabilados que otros.

A mi tío Evelio por ser tan atento, por ser esa persona que siempre salva el día, gracias por todas esas veces que me ayudaste en muchas cosas.

A mis abuelos maternos que aunque quisquillosos siempre me han demostrado su cariño y confianza.

A toda mi familia por darme ese gran apoyo, por ser tan preocupados y atentos conmigo.

A mi novia por soportarme cada día, por ser tan comprensiva y por saber cómo llegar a mí.

A mi cuñada y su esposo por ser tan buenos conmigo y en especial a Samuel.

A mi compañero de tesis, por ayudarme en la realización de este trabajo, y ser mi amigo.

A mis amigos del fútbol, del ocio y de las buenas y las malas (Yaidel, Reinier, Lisbet y Pompa).

A mis amigos de la producción por soportarme y ayudarme esa cantidad de veces que los fui a molestar.

A mi tutor y cotutora por estar siempre ahí, por su atención incondicional y eficiente.

A mis compañeros de estudio por ser esas personas que te enseñan cosas nuevas todos los días y por compartir todo este tiempo.

A mis amigos del Comité Primario UJC de la facultad, que siempre saben ayudar y guiar a todo el que se le acerque.

A todos los educadores que han contribuido a mi formación como profesional en esta institución en especial a Dariela Elvira Espinosa Leyva por contribuir a mi formación como profesional.

Antonio

Agradecimientos

A mi mamá, por siempre haber intentado que yo no tuviese otra preocupación que atender mis estudios, por siempre apoyarme, guiarme, quererme mucho y mantenerse ahí para mí.

A mi papá que nunca se ha despreocupado de mis cosas y siempre ha estado al tanto de todo, apoyándome y guiándome.

A mis hermanos, mi tía Estela, mi primo Reynol, mi primo Guillermo y en sí toda la familia que me han apoyado y ayudado en momentos que los he necesitado, principalmente en estos años lejos de mi casa.

A mi compañero de tesis, que fue un gran impulso en este trabajo siempre demostrándome que se podía hacer más.

Al tutor Martin y la co-tutora Daimi que fueron de gran ayuda, siempre estuvieron ahí y les robamos bastante tiempo.

A todos los amigos del Comité Primario de la UJC de la facultad, con Yadián al frente, fue una escuela para mi desde que entré, me enseñaron muchísimo y siempre estuvieron dispuestos a ayudar a quien se le acercara

A todo el equipo del proyecto tribunales que fueron de gran ayuda en el desarrollo de la tesis.

A Yannier, Eddy, Alexander que desde el servicio se convirtieron en muy buenos amigos y están ahí para lo que se les necesite.

A Bárbara, Giorgy mis amigos y las personas que me demostraron que podía obtener mejores resultados y ayudaron cada vez que lo necesité.

En fin todos con los que he compartido fiestas, que hemos sido compañeros de grupo, que hemos compartido apartamento, Pedro Frank, Pedro Enrique, Claudia, Lino, José Manuel, Felipe, Rafael, Carlos, Dayron. En fin muchas personas…

A todos los educadores que contribuyeron en mi formación, y especialmente aquellos que supieron marcar la diferencia y realmente influyeron muy positivamente en mi formación y fueron un ejemplo como Rolan R Bullain, Yalice Gómez, Leidily Cardoso.

A todas las personas que se sienten felices por este logro y que creyeron en mí.

Manuel

Dedicatoria

A mi mamá y mi padrastro por todo lo que han sacrificado por sus hijos durante toda su vida, por ser excelentes padres y aún mejores personas y por su apoyo incondicional durante estos largos 5 años de sacrificio y separación.

A mis tres hermanos por estar siempre ahí a pesar de la distancia que nos ha separado durante estos 5 años.

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

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

Antonio

A toda mi familia, especialmente a mi mamá.

A la revolución por crear el proyecto UCI y por dejarme formar parte de él.

Manuel

Resumen

En los Tribunales Populares Cubanos (TPC) se desarrollan cotidianamente muchos procesos. Estos son muy variados y están en dependencia de las materias[1] en que se desarrollen. Actualmente la creación y utilización de los documentos legales necesarios en estos procesos presentan algunas dificultades como la duplicidad y deterioro físico de los mismos, esto en conjunto a la necesidad de disponer de un sistema que estandarice dichos procesos y documentos, además que ejecute y supervise los procesos de manera auténtica conforme con las disposiciones legales que los regulan y que posibilite generar reportes estadísticos con mayor celeridad y facilidad es lo que da surgimiento al Proyecto Tribunales Populares Cubanos (PTPC) fase 1, el mismo está compuesto por 8 subsistemas, dentro de ellos se encuentra el subsistema Común, este se encarga de centralizar el desarrollo de los actos procesales comunes en todas las materias de los TPC. El presente trabajo plantea una solución que permitirá economizar esfuerzos, mediante el desarrollo de un Paquete de Componentes de Software (PCS), el cual consiste en implementar un grupo de 8 componentes de software (Visualizar documentos, Configurar turnado de expedientes, Configurar turnado de expedientes por salas, Gestionar plantillas, Ver detalles de expedientes, Returnar expedientes, Registrar notificación y Reincorporar expedientes) que agruparán gran parte del trabajo común para todas las materias permitiendo reutilizar el funcionamiento de los mismos por los restantes subsistemas disminuyendo así el tiempo de desarrollo del proyecto, mejorar la mantenibilidad del código y ahorrar un enorme grupo de recursos humanos y materiales.

En esta investigación se realiza una descripción de las principales tecnologías y tendencias de desarrollo de software que se emplean en el PTPC en su primera fase. Se ofrecen todos los artefactos obtenidos durante el diseño e implementación de los componentes, los cuales son evaluados mediante un conjunto de pruebas y métricas, a través de las cuales se validará su diseño y funcionamiento.

[1] Administrativa, Económica, Civil, Laboral y Penal.

Índice de contenido

Índice de tablas

Índice de figuras

Introducción

En la actualidad los avances tecnológicos son cada vez más sorprendentes y han revolucionado la forma de vivir, de comunicarse y hasta de resolver los problemas. Hoy en día con estos novedosos avances el ordenador se ha convertido en un pilar fundamental para el desarrollo de esferas como la economía, la salud, la educación y la sociedad en general. Las Tecnologías de la Informática y las Comunicaciones (TIC) también han cobrado auge, a diario se puede apreciar su variada aplicación en diferentes lugares como escuelas, hospitales, centros biotecnológicos y hasta en la forma de impartir justicia en la sociedad.

La rama del Derecho en Cuba no queda excluida de la aplicación de estos beneficios, pues de no utilizarlos se estaría alejando de la realidad tecnológica, es por eso, que en la actualidad se aplica la Informática Jurídica, técnica interdisciplinaria que tiene por objeto el estudio e investigación de los conocimientos de la informática en general, aplicables a la recuperación de información jurídica, así como la elaboración y aprovechamiento de los instrumentos de análisis y tratamiento de información jurídica necesarios para lograr dicha recuperación (1). Los TPC son muy jóvenes en la aplicación de esta técnica interdisciplinaria. Comparando sus logros con los obtenidos en el primer mundo se evidencia que poseen un atraso. En consecuencia con esto, el país se ha propuesto desarrollar soluciones internas a problemas de gran impacto en la sociedad, es aquí donde juega un papel fundamental la Universidad de las Ciencias Informáticas (UCI) con todos sus proyectos productivos, la gran mayoría de repercusión nacional, ofreciendo así soluciones propias a procesos que se realizan manualmente en los tribunales y otras entidades estatales, contribuyendo así con el proceso de informatización de la sociedad cubana.

La UCI cuenta con varios centros de desarrollo de software con el fin de suplir las necesidades de sus clientes nacionales e internacionales. Cada uno se especializa en la producción de software de una rama en particular. El Centro de Gobierno Electrónico (CEGEL) es el encargado de brindar soluciones informáticas en la esfera Gobierno Electrónico[2]. El mismo, en conjunto con el Ministerio de Justicia (MINJUS) apreciaron la necesidad de desarrollar un Sistema para la Informatización de Tribunales (SIT) en busca de mejorar la eficiencia y eficacia de los trámites judiciales y economizar el empleo de recursos materiales y humanos, surgiendo así el PTPC fase1.

[2] El gobierno electrónico es la continua optimización en la prestación de servicios públicos, acceso a la información pública y participación ciudadana mediante la transformación interna y externa de las relaciones con base en el uso de las TIC.

El proyecto está constituido por ocho subsistemas o módulos, el Administrativo, Laboral, Civil, Económico, Administración y Gobierno, Reportes, Penal y Común, este último es el encargado de elaborar un PCS para centralizar los actos procesales que son comunes en el resto de los subsistemas del PTPC fase 1.
Durante el proceso de desarrollo de software cada subsistema es el responsable de implementar los actos procesales relacionados con su materia, en estos subsistemas existen más de un 60% de actos procesales que son comunes, si cada módulo del proyecto se dedica a implementar estos actos procesales de forma independiente, esto provocaría la no uniformidad en la solución, la duplicidad de funcionalidades, un mayor tiempo de desarrollo del producto final y dificultaría la mantenibilidad del código.
Por las deficiencias anteriormente planteadas surge el siguiente **problema de investigación** ¿Cuál es la especificación de los artefactos necesarios para la informatización del conjunto de actos procesales identificados como comunes en todas las materias de los TPC?
Se tiene como **objeto de estudio** de esta investigación el proceso de desarrollo de software orientado a componentes, constituyendo entonces el **objetivo general** implementar un PCS reutilizable para centralizar los actos procesales comunes de todos los subsistemas del PTPC.
Después de haber planteado el problema en cuestión y el objeto de estudio se determina como **campo de acción** el diseño e implementación como parte del proceso de desarrollo de software orientado a componentes en la gestión de los actos procesales comunes en los TPC.
Como **idea a defender** se plantea que: Desarrollando el diseño y la implementación de los actos procesales identificados como comunes de todas las materias de los TPC, entonces se obtendrá la especificación de los artefactos necesarios para la informatización.

Objetivos Específicos

- Realizar el modelo del diseño.
- Realizar la implementación del PCS.
- Validar la solución propuesta.

Tareas a cumplir

- Realización de un estudio del proceso de desarrollo de software orientado a componentes.
- Diseño de los diagramas de clases.
- Diseño de los diagramas de interacción.

- Diseño de los diagramas de componentes.
- Implementación de los elementos del diseño.
- Realización de las pruebas de software correspondientes al PCS.
- Análisis de los resultados de las pruebas realizadas.

La realización de las tareas estuvo sustentada por un conjunto de métodos de investigación:

Teóricos

Analítico-Sintético: Posibilitó la realización de un estudio del proceso de software orientado a componentes, permitiendo la extracción de los elementos más importantes.

Histórico-Lógico: Facilitó el estudio de algunos marcos de trabajo y modelos de desarrollo de componentes, así como identificar los principales patrones de diseño e implementación más eficaces en el proceso de software orientado a componentes.

Modelación: Posibilitó la creación de modelos que representan abstracciones de los actos procesales comunes de los Tribunales Populares Cubanos para comprender su funcionamiento y realizar su posterior implementación.

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 hace un estudio del proceso de construcción de componentes, se exponen conceptos, patrones de diseño, así como el marco de trabajo, y arquitectura orientada a componentes que se emplea en el proceso de desarrollo del PTPC. También se fundamenta el empleo de las herramientas y tecnologías que se emplean en la actualidad para el desarrollo de este tipo de aplicaciones.

Capítulo 2: Propuesta de Solución. En este capítulo se realiza la descripción de la solución propuesta, especificando los patrones arquitectónicos y de diseño, el estándar de codificación, diagramas de clases del diseño, diagramas de interacción, diagramas de componentes y se documentan las principales funcionalidades a informatizar.

Capítulo 3: Validación de la solución propuesta, aquí se realiza la validación de la solución al problema mediante la aplicación de pruebas de Caja Negra, pruebas Unitarias a los algoritmos que por su complejidad lo requieran y pruebas de Integración. Además se aplican las métricas Tamaño Operacional de Clases (**TOC**) y Relación entre Clases (**RC**) que permiten conocer las dependencias y relaciones entre clases, así como la complejidad y reutilización de las mismas.

Capítulo 1

Fundamentación Teórica

1.1 Introducción

En la actualidad existen numerosas tendencias de desarrollo de software que han surgido para dar respuesta a las necesidades de los desarrolladores. Una de las más empleadas es el desarrollo basado en componentes. Esta nueva forma de concebir el desarrollo de proyectos ha revolucionado la manera de elaborar software de muchos programadores, ya que se apoya en componentes de software muchas veces ya desarrollados que combinados correctamente cumplirían con los requerimientos del sistema, esto está dado por una de las ventajas fundamentales del desarrollo de software basado en componentes que es la reutilización, pues los componentes pueden ser desarrollados de forma independiente al contexto donde serán ejecutados y luego pueden ser utilizados por diversas aplicaciones reduciendo el tiempo de desarrollo y mejorando la fiabilidad de estas aplicaciones, ya que los componentes se habrán probados con anterioridad. Otra de las considerables ventajas es que permite realizar un eficiente mantenimiento del código, pues al modificar la estructura o funcionamiento de un componente, se actualizarán los cambios realizados automáticamente en todos los lugares de la aplicación donde se emplee el mismo.
Es importante destacar que este nueva tendencia ha provocado un especial interés en muchos o casi todos los desarrolladores, pues les ofrece muchos beneficios y eso se puede apreciar en las bibliotecas de software y los repositorios de componentes que no son más que herramientas que contribuyen a ahorrar recursos en el desarrollo, ya que brindan un grupo de componentes reutilizables para cualquier aplicación.

1.2 Desarrollo de componentes

Una de las piedras angulares en el desarrollo de software en entornos de desarrollo rápido de aplicaciones (RAD por sus siglas en inglés) es la programación basada en componentes. En estos entornos, la tarea de un programador se asemeja más a la de un “ensamblador” de piezas de software que la de un “constructor” de software. Con el desarrollo basado en componentes se consigue un desarrollo muy ágil y más simple, además permite que se puedan alcanzar altos niveles

de reutilización y mantenimiento del código. Para conocer qué es un componente se debe tener un conocimiento básico de qué es la programación orientada a objetos, ya que la programación basada en componentes (**PBC** de aquí en adelante) se apoya sobre ella.

Se pueden encontrar varias definiciones de un componente en las diversas literaturas existentes, la mayoría de las cuales no dan una definición intuitiva de un componente, sino que se centran en los aspectos generales de un componente. Por ejemplo, en un Modelo de Objetos de Componentes (COM) en un resumen técnico de Microsoft, “un ***componente*** se define como una pieza de software compilada, que ofrece un servicio” (2). Se conoce que un componente es una pieza de software, y es evidente que ofrece un servicio, pero esta definición es demasiado amplia, ya que, por ejemplo, incluso las bibliotecas compiladas (.O y. Dll) se pueden definir de esta manera. En esta sección, primeramente se aclarará el concepto de componente teniendo en cuenta las diferentes definiciones que se encuentran en las literaturas.

1.2.1 Diferentes definiciones de componentes

- Un **componente** es una parte reemplazable, casi independiente y no trivial de un sistema que cumple una función clara en el contexto de una arquitectura bien definida (3).
- Un **Componente de software** es una unidad de composición que solo depende del contexto contractual de forma específica y explícita (4).
- Szyperski define un componente, precisamente, mediante la enumeración de sus propiedades características: Un ***componente de software*** es una unidad de composición con interfaces especificadas contractualmente y dependencias explícitas en un único contexto. Se puede implementar de forma independiente y está sujeto a la composición por terceras partes (5).La implicación de estas propiedades es la siguiente: Para que un componente pueda ser desplegado de forma independiente, se requiere una clara distinción de su entorno y otros componentes. Por lo tanto debe tener interfaces correctamente especificadas, mientras que la implementación tiene que ser encapsulada en el componente y no es directamente accesible desde el entorno en el que se encuentra. Esto es lo que hace que pueda ser una unidad de implementación desarrollada por terceros.
- D'Souza y Wills definen un ***componente*** como una parte reutilizable de software que se desarrolló de forma independiente y puede combinarse con otros componentes para construir unidades más grandes. Puede ser adaptado, pero no puede ser modificado (6).

Existen muchas otras definiciones de componentes, estas comienzan con la consideración desde diferentes puntos de vista de la Ingeniería de Software Basada en Componentes (ISBC) y se centran en diferentes aspectos, por ejemplo, las distintas fases de desarrollo del software. Después de analizar estas definiciones y de acuerdo al paquete de componentes que se desea desarrollar en el PTPC fase 1 con las circunstancias especificadas anteriormente se puede concluir que:

- Un componente es una pieza de software reutilizable con bajos niveles de dependencia de otros componentes o librerías, que puede ser reconfigurado de acuerdo a las necesidades de la aplicación que se desea desarrollar y suple una necesidad evidente de la arquitectura definida.

1.2.2 Características de un componente

- La integración y despliegue de componentes debe ser independiente del ciclo de vida de los mismos y no debería existir la necesidad de recompilar la aplicación que los utilice cuando estos se actualicen (7).
- Su visibilidad es exclusivamente a través de su interfaz, una implicación importante de esto es la necesidad de una especificación completa de un componente incluida su interfaz funcional, las características no funcionales (rendimiento, los recursos necesarios, etc.), casos de uso, pruebas, y así sucesivamente (7).
- Puede estar implementado en cualquier lenguaje de programación, aunque los lenguajes orientados a objetos son especialmente adecuados para este fin.
- Un componente puede ser tan pequeño como un botón en una interfaz gráfica de usuario o tan grande como un servidor Web.
- La interfaz de un componente, o sea, su vista externa, se describe como un conjunto de puertos que son los puntos de interacción entre un componente y su entorno.
- La característica más importante de un componente es la separación de sus interfaces de su implementación. Esta separación es diferente a la que se puede encontrar en muchos lenguajes de programación como ADA o Modula-2 en los que la declaración está separada de la implementación, o en aquellos lenguajes orientado a objetos de programación en los que las definiciones de clases se separan de las implementaciones de las mismas (7).

1.2.3 Principales modelos de componentes existentes

Las numerosas definiciones que existen en las literaturas de lo que es un componente a menudo tratan de definir un componente en oposición a los lenguajes de programación orientados a objetos o lenguajes de descripción de arquitecturas (LDA). En esta sección no se ofrece una nueva definición, sino que se trata de aclarar la situación mostrando los modelos de las diferentes tecnologías de componentes disponibles en la actualidad: JavaBeans (8), Enterprise Java Beans (9 pág. 73) y el COM + (10) (9 pág. 71).

Los modelos de componentes proveen estándares para la implementación e interoperabilidad entre los componentes. Además brindan funcionalidades e infraestructuras como facilitadores de meta información, utilizando e intercambiando servicios (9). Estos estándares de implementación de componentes definen también cómo las interfaces externas de los componentes son accedidas, ya que estas interfaces son la única manera de acceder a su funcionalidad. Un modelo de componentes es un elemento indispensable para la tecnología DSBC, sin embargo los modelos estrictos también puede ser un factor limitante en este contexto (11). En un escenario ideal, un desarrollador podría utilizar los mejores componentes disponibles sin tener que pensar en el modelo de componentes que se han implementado.

1.2.3.1 JavaBeans

El modelo de componentes JavaBeans fue propuesto por Sun en 1997, esta es la primera noción de un componente en el lenguaje Java. Un **bean** es un componente[3]. La principal cualidad de este modelo de componentes es su simplicidad, que se evidencia cuando se compara con otros modelos de componentes industriales; la especificación general del mismo es un documento de 114 páginas (8).

El alcance de este modelo de componentes es bastante limitado y no escala grandes rasgos del desarrollo basado en componentes, pero a pesar de sus limitaciones, ha sido ampliamente utilizado y es influyente y popular en la comunidad Java.

Uno de los factores que explican el éxito de JavaBeans es la Caja Bean, una herramienta que proporciona la especificación para demostrar la viabilidad de una idea atractiva. Un aspecto interesante de este modelo es la importancia dada a los diversos contextos en que puede ser considerado un componente. "En primer lugar un **bean** debe ser capaz de funcionar dentro de una

[3] Sun posteriormente liberó un modelo de componentes diferentes llamado Enterprise JavaBeans (EJB). Estos dos modelos son bastante distintos y no se debe confundir porque sus nombres son engañosos.

herramienta de desarrollo y en segundo lugar, cada **bean** debe ser utilizable en tiempo de ejecución dentro de la aplicación generada" (8). Este es uno de los pocos estándares en el cual el componente está explícitamente adaptado para interactuar en dos contextos diferentes: en el momento de la composición, dentro de la herramienta de desarrollo, y en tiempo de ejecución, en el ambiente de ejecución. El objetivo de la API de JavaBeans es definir un modelo de componentes de software para Java, aunque ha sido diseñado para la construcción de interfaces gráficas de usuario (IGU) donde la personalización de los componentes desempeña un papel esencial (7).

1.2.3.2 Enterprise Java Beans

Enterprise Java Beans (EJB) (12) es un modelo de componentes esencial del entorno de Sun J2EE y utiliza el sistema de tipos de Java. Los EJBs son componentes que residen dentro de un contenedor en un servidor de aplicaciones. La implementación de un EJB consta de clases de Java que se implementan en el contenedor, donde los clientes acceden desde una interfaz remota a este contenedor para invocar sus métodos.

EJB puede implementar diferentes conceptos. Los conceptos de entidades **Bean** del modelo de negocio están representados en las tablas de la base de datos (9).

A diferencia de otros estándares de componentes los Enterprise Java Beans no soportan el trabajo con eventos y además, no existe un mecanismo estandarizado para saber el tiempo de ejecución de un componente si este utiliza un servicio en particular brindado por el servidor de aplicaciones EJB (9).

1.2.3.3 COM+

El Modelo de Objetos de Componentes (COM por sus siglas en inglés) de Microsoft (13) es utilizado mayormente dentro de los sistemas operativos de Microsoft. Los componentes del modelo COM son declarados usando el Lenguaje de Definición de Interfaces de Microsoft (LDIM) que soporta las clases e interfaces del modelo de componentes COM. Las propiedades se declaran utilizando métodos **set** y **get**, además LDIM utiliza su propio sistema de tipos que se basa en el sistema de tipos de C incluyendo punteros a interfaces. Los componentes son generalmente implementados con clases C++ o con otro lenguaje compatible con COM. Las recientes adiciones a COM han sido del lado del servidor, para mejorar el balance de carga y monitoreo de transacciones. Con la instalación de estas nuevas actualizaciones este modelo ha ido creciendo y es conocido como COM +; sin embargo, en el lado del cliente el modelo de programación sigue siendo el mismo. El uso de COM +

se puede observar fácilmente en la plataforma .NET y los propios lenguajes de Microsoft, por ejemplo, Visual Basic.NET y C #. NET (9).

1.2.3.4 Selección del Modelo de Componentes a utilizar

Generalmente los modelos de desarrollo de componentes existentes en la actualidad tienen sus características peculiares, ya que cada modelo surge debido a la necesidad de satisfacer una línea de producción de software determinada que responde a un grupo de necesidades específicas. Para el desarrollo del Módulo Común del PTPC fase 1 no se adoptó ninguno de los modelos anteriormente mencionados, debido a que estos no responden totalmente a las especificidades del subsistema. Para la construcción de este módulo se acogió a un desarrollo Orientado a Objetos guiado por la utilización del marco de trabajo Sauxe, el cual brinda una fuerte implementación del patrón arquitectónico **Modelo**-**Vista**-**Controlador**, lo que posibilita la separación de las diferentes capas de los componentes que se pretenden construir, permitiendo una mayor organización en el paquete de componentes que se obtendrá como resultado final. Además, el empleo de este marco de trabajo facilita la interoperabilidad entre componentes, debido a que posibilita el consumo de servicios entre componentes, es por eso, que se puede plantear que el modelo de componentes adoptado para el desarrollo del Módulo Común está sustentado por el marco de trabajo seleccionado en conjunto al paradigma de Programación Orientado a Objetos (**POO**), las ventajas que ofrece el lenguaje de programación empleado (**PHP**).

1.2.4 Atributos a tener en cuenta en la construcción de componentes

Para una mejor clasificación de los componentes, es necesario aclarar algunos atributos relacionados con su definición .De acuerdo con Sametinger (14), seis atributos son esenciales para el desarrollo basado en componentes: La funcionalidad, interactividad, concurrencia, distribución, formas de adaptación y de control de calidad. A continuación se hace referencia a cada atributo citado.

Funcionalidad

La funcionalidad de los componentes es esencial para su reutilización en algunos contextos. Dependiendo de la cantidad de operaciones ejecutadas por el componente, la funcionalidad puede ser demasiado específica para un contexto, demasiado general, o incompleta. El concepto de

funcionalidad se complementa con los conceptos de aplicabilidad, la generalidad y completitud (9 pág. 62).

- La **aplicabilidad** de un componente indica la probabilidad con la que puede ser candidato a reutilización en la gama de sistemas de software para la que fue diseñado para ser reutilizado. Un componente puede tener un alto nivel de aplicabilidad para un dominio de aplicaciones determinado y puede tener un nivel bajo o nulo para otros (9 págs. 62-63).
- La **generalidad** indica si un componente tiene una funcionalidad más específica o no. Por ejemplo, un componente que ordena números posee menos generalidad que un componente que ordena objetos arbitrarios. Una alta generalidad también influye en que exista una alta aplicabilidad, sin embargo hay que tener cuidado con una generalidad excesiva, porque implica el desarrollo de componentes más complejos y estos pueden consumir una mayor cantidad de tiempo y recursos innecesariamente (9 pág. 63).
- La **completitud** de un componente describe si este ofrece la funcionalidad esperada en el dominio solicitado. Un ejemplo clásico de un componente incompleto es una pantalla de menú que necesita ser pintada en un dispositivo, como un teléfono móvil, y este no implementa el método pintar para mostrar el contenido (9 pág. 63).

Interactividad

Un atributo que influye directamente en la reutilización de un componente es su interactividad, debido a que, puede interactuar con otros o con el usuario. El objetivo principal cuando se habla de componentes interactivos es: una alta cohesión y un bajo acoplamiento, lo que significa que los componentes tienen que tener un alto grado de unidad conceptual y una baja o nula dependencia de otros componentes (9 pág. 63).

Concurrencia

La ejecución en paralelo de componentes es lo que se denomina concurrencia. Un componente concurrente puede ser no determinista, es decir, puede dar resultados diferentes para la misma entrada de datos. Esto sucede porque un componente no sabe exactamente qué secuencia de instrucciones se está ejecutando en un momento dado, de modo que este atributo se emplea para resolver problemas no deterministas que no requieren una ejecución secuencial. La concurrencia también se utiliza para ganar velocidad de ejecución y para eliminar el tiempo de inactividad del procesador (9 pág. 64).

Distribución

Los sistemas distribuidos por definición son sistemas físicos o lógicos separados (15). Las principales razones por la que los sistemas distribuidos han alcanzado la popularidad es que no incluyen consideraciones de costo, pero permiten incrementar las capacidades de respuesta de estos sistemas y aumentan la flexibilidad de la expansión incremental. Un componente que posea este atributo debe poder acoplarse en un ambiente distribuido.

Un sistema distribuido consiste en ejecutar de forma independiente un grupo de componentes. Esos componentes pueden ser implementados en el mismo lenguaje de programación, utilizando los mecanismos de comunicación que brinda el mismo o pueden ser implementados en diferentes lenguajes y paradigmas de programación haciendo uso solamente de un conjunto de mecanismos de comunicación. Además el compartimiento de datos y el intercambio de mensajes constituyen el mecanismo principal de comunicación de estos componentes (9 pág. 64).

Formas de adaptación

Antes de que un componente sea acoplado a un sistema, este requiere que se le hagan algunas adaptaciones. Esta fase es llamada “Optimización del componente”, y en la misma, el componente sufrirá un grupo adaptaciones que no afectarán nunca su comportamiento esencial. Después de esta fase, un componente puede pasar a través de un grupo de “modificaciones”, lo que no es recomendable porque en la mayoría de los casos estas modificaciones disminuyen la reusabilidad del componente, debido a que las funcionalidades primarias sufren cambios (9 págs. 64-65).

Un ejemplo simple de optimización puede ser el caso de tener que exhibir la interfaz gráfica de un componente del PTPC fase 1 en la pantalla de un teléfono móvil que tiene dimensiones diferentes a las de un ordenador, sería necesario optimizar dicho componente para trabajar en esa resolución de pantalla específicamente.

Control de la calidad

El aseguramiento de la calidad de componentes es una tarea extremadamente compleja. Por otro lado verificar la calidad es aún más difícil. El proceso es tan complicado que no existe una verificación formal de la calidad incluso para componentes pequeños. Algunos trabajos en esta área han estado investigando y han logrado resultados importantes, como también los ha alcanzado el trabajo presentado por Álvaro (16), el cual tiene como objetivo principal definir un “Proceso de Certificación de Componentes de Software”, sin embargo, en los métodos actuales y procesos de

desarrollo de componentes, la idea que se tiene es desarrollar un sistema tolerante a fallos, aún si este está compuesto por componentes poco fiables.

Existen algunos trucos para asegurar que un componente también es tolerante a fallos como: las precondiciones y post-condiciones y las excepciones (14).

- **Precondiciones**: Son expresiones que toman valores de verdadero o falso, que chequean si los parámetros de entrada son válidos y si los objetos se encuentran en un estado razonable para ejecutar las tareas solicitadas.
- **Post-condiciones**: Son las encargadas de chequear si, después de terminada la tarea solicitada, esta terminó con éxito de acuerdo a los métodos o funciones que supuestamente se debían ejecutar para cumplir con la misma.
- **Excepciones**: Las excepciones representan otro tipo de garantía de la calidad para el componente, ya que son usadas para manejar las situaciones atípicas durante la ejecución del código (9 pág. 65).

1.2.5 Clasificación de componentes

Los componentes pueden clasificarse de diferentes formas por su **composición** en simples o compuestos y por su **tipología** en visuales o controladores. Un componente simple es aquel que no necesita de ningún otro para completar su estructura funcional mientras que, uno compuesto sí necesita de otro componente o servicio que brinde otra entidad o aplicación externa para completar su funcionamiento.

Los componentes visuales son aquellos que interactúan directamente con el usuario, y ayudan a conformar parte de una interfaz de usuario que le posibilitará al mismo introducir o mostrar datos.

Los componentes controladores son aquellos que hacen algún tipo de procesamiento de información, muchas veces se pueden encontrar trabajando en conjunto con un componente visual, ya que le permite recolectar los datos con los que puede posteriormente procesar o modificar información de un dominio determinado, aunque no necesariamente tienen que poseer interfaz gráfica, ya que se pueden utilizar además mediante el consumo de servicios desde otros componentes.

1.2.5.1 Tipos de componentes visuales

- **Presentación de datos**
  - Incluye todos los datos presentados en una pantalla visible.

- Existen numerosas formas de presentación, como gráficos, hojas de cálculo, listados, etc.

- **Formularios de datos**
  - Los formularios de entrada de datos también se consideran componentes visuales.
- **Ayudas de navegación**
  - Este tipo de vistas incluye menús, hipervínculos, mapas del sitio, etc.
- **Pantallas informativas o ventanas emergentes**
  - Este tipo de vistas incluye texto de bienvenida, instrucciones, pantallas de ayuda, mensajes de error, cuadros de confirmación, etc.

1.2.5.2 Tipos de componentes controladores

- **Permiten navegar por la pantalla**
  - Algunos mecanismos de navegación pueden requerir datos (como usar una lista desplegable) para seleccionar la próxima pantalla donde navegar.
- **Clases controladoras**
  - Posibilitan preparar datos de negocio para presentarlos en una vista. Un buen ejemplo de ello son los informes.
  - Contienen funciones que pueden modificar datos de negocio al ser llamadas desde la vista.

1.3 Aplicaciones web

Una aplicación web es un sistema informático que se enmarca dentro de la arquitectura cliente/ servidor, un ordenador solicita servicios (el cliente) y otro está a la espera de recibir solicitudes y las responde (el servidor).

Al desarrollar el Sistema de Informatización de Tribunales como aplicación web se están aprovechando muchas de sus ventajas tales como: evitar gran parte de los problemas de compatibilidad ya que solo se necesita un navegador actualizado, no ocupa espacio en el disco duro de los clientes, las actualizaciones son inmediatas pues no existe la necesidad de instalar y mantener la aplicación de cada uno de los disímiles usuarios potenciales, estos solo deben conectarse mediante el navegador al servidor donde se actualizarán los cambios, posibilitando un alto grado de comunicación e iteración de forma activa entre la información y los usuarios, y la creación de páginas

personalizadas según el perfil de usuario, en fin la gestión de los procesos de los TPC desde todas sus sedes y la notificación a todos los involucrados de forma inmediata de realizada la acción.

1.4 Metodologías de desarrollo de software

Una metodología de desarrollo de software es un conjunto de procedimientos, técnicas, herramientas y un soporte documental para el desarrollo de productos software, puede seguir uno o varios modelos de ciclo de vida, el ciclo de vida indica qué es lo que hay que obtener a lo largo del desarrollo del proyecto pero no cómo hacerlo, mientras que la metodología indica cómo hay que obtener los distintos productos parciales y finales. Para el desarrollo del proyecto se eligió la metodología RUP.

1.4.1 Proceso Unificado de Desarrollo (RUP)

Es una metodología robusta apropiada para proyectos de gran envergadura similares al PTPC. El objetivo principal que se persigue con esta metodología es crear software de alta calidad. Dicha metodología concibió desde sus inicios el uso de UML como lenguaje de modelado, es un proceso dirigido por casos de uso, 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. Además el equipo de trabajo se encuentra familiarizado con la misma y la utilización de dicha metodología provee la generación de una gran documentación solicitada por el cliente y necesaria para el proyecto por los continuos cambios de personal.

Dentro de sus características principales se pueden citar:

- Forma disciplinada de asignar tareas y responsabilidades (quién hace qué, cuándo y cómo).
- Desarrollo iterativo, lo que permite ir evolucionando el producto final.
- Uso de arquitectura basada en componentes, lo que facilita un alto nivel de reutilización de componentes e interfaces permitiendo así ahorrar recursos.
- Modelado visual del software.
- Verificación de la calidad del software.

1.5 Arquitectura de Software

En el año 2000 se acordó definir oficialmente la Arquitectura de Software según figura en el documento IEEE Std 1471-2000 (17): “La arquitectura del software es la organización fundamental

de un sistema encarnada en sus componentes, las relaciones entre ellos y el entorno, y los principios que dirigen su diseño y evolución".

Para elaborar una Arquitectura de Software hay que tener presente dos elementos fundamentales, los **objetivos** y las **restricciones** que se manejan para la construcción del software que se desea obtener, puesto que son los elementos rectores que condicionan las herramientas y tecnologías a emplear para su desarrollo. Durante el proceso de elaboración de la arquitectura del PTPC se analizaron un grupo significativo de particularidades, objetivos y restricciones que traería consigo el desarrollo del mismo, pues al ser el Sistema de Informatización de Tribunales (SIT) el producto final, debe garantizar un alto nivel de fiabilidad, seguridad y eficiencia en la gestión de todos los procesos judiciales en Cuba.

Con el objetivo de asegurar los parámetros mencionados anteriormente y cumplir con los objetivos fundamentales del proyecto, el equipo de arquitectos tomó la decisión de emplear un conjunto de ***Frameworks***[4] que posibilitan un desarrollo seguro, eficiente y controlado de los requerimientos identificados que debe tener y cumplir el producto final.

1.5.1 Marcos de trabajo

Los marcos de trabajos (framework) son una estructura de soporte definida mediante la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software 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 y provee una estructura y una metodología de trabajo la cual extiende o utiliza las aplicaciones del dominio (18).

1.5.1.1 Zend Framework (ZF)

Es una herramienta de código abierto construida con el objetivo de desarrollar aplicaciones web con alto grado de seguridad y consistencia, además posibilita el desarrollo de servicios web. Emplea el lenguaje PHP 5 y el paradigma de programación orientado a objetos con una estructura de sus componentes muy típica, ya que cada uno de ellos está implementado con una baja dependencia de los demás componentes. Su arquitectura posee un bajo acoplamiento, lo que le permite a los

[4] Es una estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos de *software* concretos, con base a la cual otro proyecto de *software* puede ser más fácilmente organizado y desarrollado. También se conoce como Marco de Trabajo.

desarrolladores reutilizar los componentes por separados y agilizar las tareas de desarrollo de software.

1.5.1.2 Sauxe

El departamento de tecnologías de la UCI en conjunto con la UCID (Unidad de Compatibilización Integración y Desarrollo de Software para la Defensa) constituyen el equipo de desarrollo de ZendExt, resultado de la extensión de algunos componentes de ZendFramework, con el objetivo de crear un marco de trabajo extensible y configurable centrado en el desarrollo de aplicaciones web, en la lógica del negocio y en las interfaces de usuario, alejando a los programadores de los detalles arquitectónicos y brindando una arquitectura de sistema orientada a componentes (19). Posteriormente la unión de ZendExt, Doctrine y Extjs provoca el nacimiento del Marco de Trabajo Sauxe. El mismo está conformado por un grupo 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 (20). La versión de este producto (2.0) que se emplea en el PTPC consta de los siguientes componentes:

Nombre del Componente	Nombre del Componente
ZendExt_App	ZendExt_GlobalConcept
ZendExt_Aspect	ZendExt_IoC
ZendExt_ADT	ZendExt_Nomencladores
ZendExt_Cache	ZendExt_Trace
ZendExt_ExpImp	ZendExt_Validation
ZendExt_MVC	ZendExt_Portal
ZendExt_Exception	ZendExt_TransactionManager
ZendExt_FastResponse	

Tabla 1 Componentes que posee el marco de trabajo Sauxe

Todas las herramientas empleadas en la construcción de Sauxe son herramientas libres, siguiendo de esta manera el paradigma de independencia tecnológica por el que tanto aboga el país, la única desventaja que tiene el marco de trabajo es que solamente maneja como sistema gestor de base datos el PostgreSQL 8.3 o 8.4, pero consta de una gran organización de los elementos del proyecto, favoreciendo el desarrollo de componentes de forma organizada puesto que cada componente tendrá una estructura bien definida en las diferentes capas (Capa de Presentación, Capa de Control o Negocio, Capa de Acceso a Dato, Capa de Datos, Capa de Servicios) que define en su arquitectura

y facilita de esta manera las labores de mantenimiento de los componentes a desarrollar. Su estructura es la siguiente:

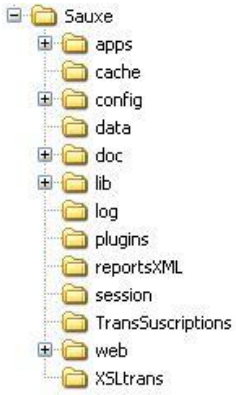


Figura 1 Estructura de carpetas de Sauxe

Este marco de trabajo brinda un conjunto de ventajas significativas:

- Permite configurar y gestionar de manera dinámica la cache del marco de trabajo.
- Posibilita configurar y gestionar de manera dinámica la integración de las aplicaciones o dominios de soluciones que se instancien o construyan con la misma.
- Posee 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.
- Tiene 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 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.
- Tiene un mecanismo de administración y configuración dinámica de trazas de la solución.

1.5.1.3 Extjs

ExtJS es una librería JavaScript que permite construir aplicaciones web complejas y dinámicas, provee un grupo de componentes con un acabado visual extraordinario y es muy potente en los temas de validación, ya que brinda la opción de crear validadores de datos propios para los campos de los formularios, lo que es útil si se tiene validaciones que se repiten a lo largo de la aplicación.
Características principales:

- Posee componentes de interfaces de usuario con un alto performance y fáciles de personalizar.
- Brinda un modelo de componentes extensibles.
- Posee un API fácil de usar.
- Tiene licencias Open Source y comerciales.

Utilizar esta librería para la construcción de interfaces gráficas de usuario permite tener relevantes beneficios:

- **Existe un balance entre Cliente – Servidor**. La carga de procesamiento se distribuye, permitiendo que el servidor, al tener menor carga, pueda manejar más clientes al mismo tiempo.
- **Comunicación asíncrona**. En este tipo de aplicación el motor de construcción puede comunicarse con el servidor sin necesidad de estar sujeta a un clic o una acción del usuario, dándole la libertad de cargar información sin que el cliente se dé cuenta (21).
- **Eficiencia de la red**. El tráfico de red puede disminuir al permitir que la aplicación elija que información desea transmitir al servidor y viceversa, sin embargo la aplicación que haga uso de la pre-carga de datos puede que revierta este beneficio por el incremento del tráfico (21).

1.5.2 Object Relation Mapper (ORM)

Un ORM es la herramienta que permite acceder de forma efectiva a las bases de datos relacionales desde un ambiente orientado a objetos, ya que es necesaria una interfaz que traduzca la lógica de los objetos a la lógica relacional y viceversa, esta interfaz permite realizar lo que se conoce como el *"mapeo objeto-relacional"*, y está formada por objetos que permiten acceder a los datos y que contienen en sí mismos el código necesario para hacerlo. Un ORM crea una base de datos orientada a objetos virtuales sobre la base de datos relacional, lo que facilita el uso de las características nativas de la orientación a objetos (herencia y polimorfismo fundamentalmente).

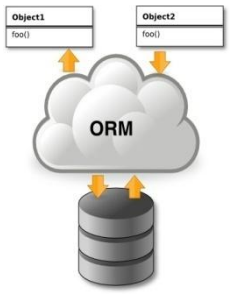


Figura 2 Funcionamiento de un ORM

1.5.2.1 Doctrine

Doctrine es un ORM muy completo y configurable, permite generar de forma automática el modelo de clases basándose en el modelo relacional de tablas, compatible con PHP 5.2.3+ e incorpora una Capa de Abstracción a Base de Datos (DBL por sus siglas en inglés).

El marco de trabajo Sauxe utiliza en la capa de acceso a datos el Lenguaje de Consulta de Datos (DQL) que implementa Doctrine, lo que permite utilizar un grupo de funcionalidades nativas propias del ORM que agilizan las labores de desarrollo y facilitan el trabajo de los desarrolladores. Posee una amplia documentación y tiene las características necesarias para ser útil en el desarrollo de la mayoría de los proyectos.

Principales ventajas:

- **Reutilización:** La principal ventaja que aporta un ORM es la reutilización permitiendo llamar a los métodos de un objeto de datos desde distintas partes de la aplicación e incluso desde diferentes aplicaciones.
- **Portabilidad:** Utilizar una capa de abstracción permite cambiar en mitad de un proyecto de una base de datos MySQL a una Oracle sin ningún tipo de complicación. Esto es debido a que no se utiliza una sintaxis MySQL, Oracle o SQLite para acceder al modelo, sino una sintaxis propia del ORM utilizado que es capaz de traducir a diferentes tipos de bases de datos.
- **Seguridad:** Los ORM suelen implementar mecanismos de seguridad que protegen la aplicación de los ataques más comunes como inyecciones SQL.
- **Mantenimiento del código:** Gracias a la correcta ordenación de la capa de datos, modificar y mantener el código es una tarea sencilla (22).

1.6 Paradigmas de programación

Un Paradigma de Programación es una propuesta tecnológica que es adoptada por una comunidad de programadores, constituye un modelo básico de diseño y desarrollo de programas que permite producir estos con directrices especificadas. Este modelo está pensado de forma que determina la estructura correcta de los programas y controla el modo en que se piensa y formula la solución. Es importante conocer que cada uno de ellos se instaura tras una revolución científica, que aporta respuestas a enigmas que no podían resolverse con los ya existentes, pero también que existen varios, de los cuales no se puede decir que uno sea mejor que el otro, sino que cada uno tiene sus ventajas y desventajas y generalmente su utilidad está dada por el entorno donde se vayan a utilizar.

Como en todos los sistemas la selección del paradigma de programación debe facilitar la elaboración del producto final, debe ser la propuesta tecnológica más cómoda y eficiente para formular la solución. Atendiendo a las características que debe presentar el sistema para los TPC el paradigma base empleado es el Paradigma de Programación Orientado a Objetos.

1.6.1 Paradigma de Programación Orientado a Objetos (PPOO)

Este es un paradigma que para diseñar sistemas y aplicaciones informáticas simula el comportamiento de los objetos reales, intenta amoldarse al modo de pensar del hombre y no al de la máquina, esto facilita la implementación, ya que estos objetos pasan a ser el concepto fundamental de la POO, ellos se van a convertir en el modelado de lo que se quiere realizar, que pueden ser entidades reales o abstractas del universo o específicamente del problema que se quiere resolver, van a poseer atributos que representan sus características o propiedades y métodos que representan su comportamiento o acciones que realizan. Todas las propiedades y métodos comunes a los objetos se encapsulan o se agrupan en clases. Una clase es una plantilla o un prototipo para crear objetos, por eso se dice que los objetos son instancias de clases. Este paradigma está basado en varias técnicas, como son: abstracción, encapsulamiento, herencia y polimorfismo.

Abstracción: Permite realizar generalizaciones, es la que permite modelar la realidad a través de objetos en la programación, simplifica la realidad ignorando los detalles que no tienen relevancia en el problema que se modela y se enfoca en las cosas comunes, pero permitiendo las variaciones.

Encapsulamiento: Es la propiedad que permite que los objetos presenten la misma interfaz pero oculten información de su funcionamiento, es decir ocultan sus datos de manera que sólo se puede cambiar mediante las operaciones definidas para ese objeto.

Herencia: Es el mecanismo fundamental para implementar la reutilización y extensibilidad del software. Esta facilita la clasificación que se hace a distintos objetos, permitiendo construir nuevas clases partiendo de una jerarquía ya existente, ya que permite crear objetos que guarden propiedades y formas de interactuar comunes, de forma tal que evite el volverlo a implementar porque los ya existentes lo contienen y el nuevo va a ser un objeto que tomará las mismas características del anterior.

Polimorfismo: Se refiere a la capacidad que se utiliza cuando se definen clases derivadas de otras y estas necesitan redefinir alguno de sus métodos, ya que el comportamiento presenta algunas variaciones.

1.6.2 Paradigma de programación orientado a componentes (PPOC)

El PPOO es muy potente, pero en el desarrollo del módulo común fue necesaria también la utilización de muchas de las características que brinda el PPOC para facilitar la implementación y hacerla de una forma más eficiente. Se puede decir que este paradigma es una extensión natural de la orientación a objetos que va a permitir el desarrollo y la utilización de componentes reutilizables y utilizar las características fundamentales del PPOO, pero esta vez orientadas a componentes, permitiendo la herencia, polimorfismo, encapsulamiento y abstracción entre los componentes. Este paradigma es el que guía la construcción de cada uno de los componentes de forma tal que luego cada uno de los restantes módulos puedan acoplar en sus funcionalidades los mismos de forma sencilla.

1.7 Patrones de diseño orientados a objetos

Un **patrón** es una solución a un problema determinado, al que se le da un nombre, y que se puede aplicar a nuevos contextos; idealmente, proporciona consejos sobre el modo de aplicarlo en varias circunstancias. Durante la etapa de diseño de un software se conciben un grupo de patrones que de acuerdo a la solución que brindan cubren una necesidad determinada en el diseño.

Existen dos grupos de patrones de diseño principales, los Patrones Generales de Software para Asignación de Responsabilidades (GRASP siglas de General Responsibility Assignment Software Patterns en inglés) y los patrones correspondientes al Grupo de los Cuatro (GOF siglas de Gang of Four en inglés) que es el nombre con el que se conoce por lo general a los autores del libro DesignPatterns (23).

Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. Los patrones GRASP que se utilizan en la solución son:

- Bajo Acoplamiento
- Controlador

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. De este grupo los patrones utilizados son:

- Singleton.

Además otro de los patrones que se emplean son:

- IoC (Inversión de control).
- Modelo Vista Controlador

1.8 Herramientas CASE

Las Herramientas CASE (Computer Aided Software Engineering, Ingeniería de Software Asistida por Computadora) son un conjunto de programas y ayudas que facilitan la automatización del ciclo de vida del desarrollo de un Software, apoyando de esta manera a los analistas, ingenieros de software y desarrolladores. Fueron desarrolladas para organizar y estructurar de forma correcta y eficiente un grupo de actividades propias del proceso de desarrollo de software y facilitar las tareas de coordinación de los eventos que necesitan ser mejorados en el ciclo de desarrollo de software.
Algunas de estas herramientas son:

- Enterprise Architect
- Rational Rose Enterprise Edition 2003
- Visual Paradigm.

Durante los inicios del proyecto el equipo de arquitectura decidió el empleo de Visual Paradigm como herramienta CASE[5], ya que la aplicación a desarrollar será escrita en PHP y esta se integra fácilmente con este lenguaje, también el equipo de desarrollo cuenta con una mayor experiencia en el empleo de esta herramienta que con otras existentes, por lo que esta elección posibilitará un considerable ahorro de tiempo en el desarrollo del software debido a que no hay que capacitar a los desarrolladores.

1.8.1 Visual Paradigm 3.4

La herramienta Visual Paradigm para UML ha sido creada para soportar el ciclo de vida completo del proceso de desarrollo del software (análisis y diseño orientados a objetos, construcción, pruebas y despliegue) a través de la representación de todo tipo de diagramas. Constituye una herramienta de software libre de probada utilidad para el analista. Fue diseñado para una amplia gama de usuarios interesados en la construcción de sistemas de software de forma fiable a través de la utilización de un enfoque Orientado a Objetos.
Visual Paradigm es una herramienta CASE que soporta las últimas versiones del Lenguaje de Modelado Unificado (UML) y la Notación y Modelado de Procesos de Negocios. En adición al soporte de Modelado UML esta herramienta provee el modelado de procesos de negocios, además de un generador de mapeo de objetos-relacionales para los lenguajes de programación Java, .NET y PHP.
Características principales:

[5] Disponible en : http://alfresco.cegel.prod.uci.cu/alfresco/faces/jsp/browse/browse.jsp

- Disponibilidad en múltiples plataformas (Windows, Linux).
- Diseño centrado en casos de uso y enfocado al negocio que generan un software de mayor calidad.
- Modelo y código que permanece sincronizado en todo el ciclo de desarrollo.
- Soporta aplicaciones Web.
- Generación de código - Modelo a código, diagrama a código.
- Editor de figuras.
- Interoperabilidad con modelos UML2.
- Diagramas de flujo de datos.

1.9 Plataformas de desarrollo

El proceso de desarrollo de software del PTPC está sustentado por el empleo de una **plataforma de desarrollo,** que no es más que el entorno de software común en el cual se desenlaza la codificación de un conjunto determinado de aplicaciones que se necesitan para el funcionamiento del mismo. Las plataformas generalmente se encuentran relacionadas 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. La plataforma empleada en el PTPC está ligada a la utilización de una rama de lenguajes y es independiente del sistema operativo que se decida utilizar. Los lenguajes que se utilizan son PHP del lado del servidor y JavaScript para la construcción de interfaces de usuarios del lado del cliente.

1.9.1 Lenguajes de programación

Un **lenguaje de programación** es diseñado para describir un grupo de acciones o tareas secuenciales que un ordenador debe ejecutar. Este consiste en un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. Por lo que constituye una manera práctica para que los seres humanos puedan influir sobre el comportamiento de una computadora bajo determinadas circunstancias. Los lenguajes utilizados en el desarrollo del proyecto se describen a continuación en los próximos epígrafes.

1.9.1.1 PHP 5.2.5

PHP es un lenguaje de programación interpretado, ideado originalmente para la construcción de páginas web dinámicas, su acrónimo (Hypertext Preprocessor) significa preprocesador de hipertexto.

El gran parecido que posee con los lenguajes más comunes de programación estructurada, como C y Perl, permiten a la mayoría de los programadores crear aplicaciones complejas con una curva de aprendizaje muy corta. También les permite involucrarse con aplicaciones de contenido dinámico sin tener que aprender todo un nuevo grupo de funciones. Su funcionamiento es muy básico, cuando el cliente hace una petición al servidor para que le envíe una página web, el servidor ejecuta el intérprete de PHP, este procesa el script solicitado que generará el contenido de manera dinámica y el resultado es enviado por el intérprete al servidor, quien a su vez se lo envía al cliente. También mediante extensiones es posible la generación de archivos PDF, Flash, así como imágenes en diferentes formatos.

Este lenguaje posee muchas ventajas:

- Orientado al desarrollo de aplicaciones web dinámicas con acceso a información almacenada en una base de datos.
- El código fuente escrito en PHP es invisible al navegador web y al cliente ya que es el servidor el que se encarga de ejecutar el código y enviar su resultado HTML al navegador. Esto hace que la programación en PHP sea segura y confiable.
- Capacidad de conexión con la mayoría de los motores de base de datos que se utilizan en la actualidad, destaca su conectividad con MySQL y PostgreSQL.
- Capacidad de expandir su potencial utilizando módulos (llamados extensiones).
- Posee una amplia documentación en su sitio web oficial, entre la cual se destaca que todas las funciones del sistema están explicadas y ejemplificadas en un único archivo de ayuda.
- Es libre, por lo que se presenta como una alternativa de fácil acceso para todos.
- Permite aplicar técnicas de programación orientada a objetos.
- Biblioteca nativa de funciones sumamente amplia e incluida.
- No requiere definición de tipos de variables aunque sus variables se pueden evaluar también por el tipo que estén manejando en tiempo de ejecución.
- Tiene manejo de excepciones (desde PHP5).

1.9.1.2 JavaScript

Es un lenguaje de programación interpretado, surgido como resultado del estándar ECMAScript [6] y desarrollado por **Netscape**, es orientado a objetos, imperativo, débilmente tipado y dinámico. Se utiliza esencialmente del lado del cliente, implementado como parte del navegador web permitiendo mejoras significantes en las interfaces de usuario puesto que es muy eficiente en los temas de validaciones y para el desarrollo de web dinámicas.

Sus características más significativas son:

- Es un lenguaje orientado a eventos. Cuando un usuario interactúa con un enlace o mueve el puntero sobre algún otro elemento HTML se pueden producir diferentes eventos y a través de JavaScript se pueden desarrollar scripts que ejecuten funciones en respuesta a estos eventos.
- Es un lenguaje interpretado, o sea, no necesita ser compilado. El navegador del usuario se encarga de interpretar las sentencias JavaScript contenidas en una página HTML y ejecutarlas adecuadamente.
- Es soportado por la mayoría de los navegadores como Internet Explorer, Google Chrome, Opera, Mozilla Firefox, entre otros.

1.9.1.3 HTML

Es el lenguaje de marcado de hipertexto usado para la construcción de páginas web. Fue creado en 1986 por Tim Berners Lee; el cual tomó dos herramientas preexistentes: El concepto de Hipertexto el cual permite conectar dos elementos entre si y el SGML (Lenguaje Estándar de Marcación General) el cual sirve para colocar etiquetas o marcas en un texto que indique como debe verse. HTML no presenta ningún compilador, por lo tanto algún error de sintaxis que se presente éste no lo detectará y se visualizará en la forma como éste lo entienda. El conjunto de etiquetas que se creen para la construcción de una página web, se deben guardar con la extensión *.htm* o *.html*. Estos documentos pueden ser mostrados por los visores o "browsers" de páginas Web en Internet, como Netscape Navigator, Mosaic, Firefox, Opera entre otros (24).

[6] **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.

1.10 Entornos de Desarrollo Integrados

Un Entorno de Desarrollo Integrado (IDE siglas de Integrated Development Environment en inglés), consiste básicamente en un programa informático que previamente ha sido instalado en la máquina del desarrollador y cuyo principal objetivo es la elaboración de otras aplicaciones, es decir es un entorno de programación que ha sido empaquetado como *"Software",* que va a contener un editor de código, un compilador, un depurador y un constructor de interfaz gráfica, que van a facilitar las diferentes tareas necesarias en el desarrollo y mantenimiento de cualquier tipo de aplicación. En la actualidad hay infinidades de entornos de desarrollos integrados, entre los cuales el equipo de arquitectura del proyecto decidió utilizar para el desarrollo del sistema para los TPC NetBeans en su versión 7.0.1.

1.10.1 NetBeans 7.0.1

NetBeans IDE es un producto libre y gratuito sin restricciones de uso, de código abierto, escrito completamente en Java, teniendo el código fuente disponible para su reutilización bajo las licencias CDDL y la GPL. Es multiplataforma, presenta una interfaz muy amigable e intuitiva, soporta disímiles lenguajes, además en sus versiones más recientes se le han ido acoplando nuevas funcionalidades que resultan de gran utilidad para el desarrollo web, facilitando la creación de proyectos en PHP. NetBeans en su versión 7.0.1 presenta un sistema para examinar todos los directorios de cada proyecto, haciendo reconocimiento y carga de clases, métodos y objetos, para acelerar la programación, proponiendo un esqueleto para organizar el código fuente, el editor conjuntamente integra los lenguajes como HTML, JavaScript y CSS, permitiendo la refactorización y búsqueda de usos para CSS y lenguajes de tipo HTML, y el completamiento de código y links para atributos de CSS. Por otra parte el editor de PHP es muy ágil y robusto, siendo eficiente en el completamiento de código, reconocimiento de sintaxis y presentando un potente depurador que facilita la programación. También desde este potente IDE es posible realizar la administración del Sistema de Control de Versiones, en este caso SVN, que va a facilitar las actualizaciones diarias del trabajo realizado.

1.11 Sistemas gestores de Base de datos

Un sistema gestor de bases de datos o SGBD(conocido también por las siglas DBMS procedentes del inglés, "*Data Base Management System*"), es un software o conjunto de programas que permite definir, crear y mantener una base de datos, permitiendo un acceso controlado a la misma. EL SGBD es la aplicación que interactúa con los usuarios de los programas de aplicación y la base de datos,

teniendo entre sus objetivos fundamentales proporcionar un entorno eficiente a la hora de almacenar y recuperar los datos, garantizando la seguridad de los mismos, incluyendo la especificación de tipos, estructura y restricciones de los datos, también el guardado y la manipulación mediante actualizaciones de los mismos y la realización de consultas y generación de informes.

Para el desarrollo del SIT el equipo de arquitectura definió que de los diversos SGBD existentes en la actualidad se utilizará PostgreSQL en su versión 8.4, basándose en sus características, ventajas y en que es el soportado por el marco de trabajo seleccionado.

1.11.1 PostgreSQL 8.4

PostgreSQL es un sistema gestor de base de datos objeto-relacional, bajo licencia BSD. Este se ha convertido en el sistema de gestión de bases de datos de código abierto más potente del mercado, constando ya con una arquitectura suficientemente probada, que se ha ganado una sólida reputación de confiabilidad, integridad de datos y corrección.

PostgreSQL es multiplataforma, brinda soporte para los lenguajes más populares, incluyendo PHP que es el utilizado en el PTPC. Este utiliza un modelo cliente/servidor y usa multiprocesos en vez de multi-hilos para garantizar la estabilidad del sistema. Un fallo en uno de los procesos no afectará el resto, de esta manera el sistema continúa funcionando. Es un sistema muy seguro que funciona muy bien con grandes cantidades de datos (25). Entre sus características fundamentales se puede mencionar que:

- Es altamente escalable. Tanto en la cantidad bruta de datos que puede manejar como en el número de usuarios concurrentes que puede atender, realizando esto de forma más eficiente que muchos otros gestores.
- Implementa el uso de deshacer (rollback's), subconsultas y transacciones, haciendo su funcionamiento mucho más eficaz, ofreciendo soluciones en disimiles campos.
- Soporta distintos tipos de datos: además del soporte para los tipos base, también soporta datos de tipo fecha, monetarios, elementos gráficos, datos sobre redes, cadenas de bits, entre otros. También permite la creación de tipos propios.
- Incorpora una estructura de datos arreglo.
- Implementación del estándar SQL92/SQL99.
- Permite la declaración de funciones propias, así como la definición de disparadores.
- Permite la gestión de diferentes usuarios, como también los permisos asignados a cada uno de ellos.

- Incluye copias de seguridad en caliente/en línea.

Estas son algunas de sus características las cuales facilitan el manejo de los datos referente a la aplicación para los TPC. Donde este gestor será gestionado por una aplicación gráfica llamada PGAdmin III que es una de las más completas y populares con licencia Open Source. Está escrita en C++ y utiliza la librería gráfica multiplataforma wxWidgets, permitiendo que se pueda usar en sistemas operativos como: GNU/Linux, FreeBSD, Solaris, Mac OS y Windows. Es capaz de gestionar versiones a partir de PostgreSQL 7.3, ejecutándose en cualquier plataforma. Posee una interfaz gráfica que soporta todas las características de PostgreSQL y facilita enormemente la administración.

1.12 Servidor web

Un servidor Web puede definirse como un programa que escucha las peticiones de los clientes (usuarios o navegantes), realizadas a través de un navegador web y las atiende o satisface, proporcionando los recursos que soliciten usando el protocolo HTTP o el protocolo HTTPS (la versión cifrada y autenticada). Esta aplicación es la que se encarga de gestionar cualquier sistema web en el lado del servidor, permitiendo realizar conexiones bidireccionales y/o unidireccionales así como síncronas o asíncronas. En otras palabras el servidor web se mantiene a la espera de la solicitud de un navegador, resuelve la petición de este, y da respuesta a la misma, generalmente enviando una página HTML, la cual el navegador interpreta y muestra en la pantalla al usuario.

Como en la mayoría de los desarrollos de aplicaciones en el del sistema para los TPC también existió un momento en el que fue crucial seleccionar entre muchos servidores web existentes el que se utilizaría. En ese momento el equipo de arquitectura decidió que se trabajaría con el servidor apache en su versión 2.0.

1.12.1 Servidor Apache 2.0

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. Alguna de sus características:

- Es multiplataforma, aunque idealmente está preparado para funcionar bajo Linux.
- Muy sencillo de configurar.
- Es Open-Source.
- Amplias librerías de PHP y Perl a disposición de los programadores.

- Posee diversos módulos que permiten incorporarle nuevas funcionalidades, estos son muy simples de cargar.
- Es capaz de utilizar lenguajes como PHP, TCL, Python, entre otros.

Apache tiene amplia aceptación en la red: desde 1996, es el servidor HTTP más usado en el mundo y alcanzó su máxima cuota de mercado en 2005 siendo el servidor empleado en el 70% de los sitios web en el mundo (26).

1.13 Control de versiones

El Control de Versiones es la técnica que posibilita gestionar los diversos cambios que se realizan sobre los elementos de algún proyecto. Una versión o revisión de un proyecto es el estado en que se encuentra el mismo en un momento determinado de su desarrollo. A pesar que también puede realizarse un control de versiones de forma manual, es aconsejable utilizar una herramienta que facilite realizar esta gestión. Para realizar el control de versiones en el PTPC el equipo de arquitectura decidió utilizar la herramienta RapidSVN.

1.13.1 RapidSVN 0.12.0

RapidSVN es una plataforma de interfaz gráfica de usuario para el sistema de revisión de Subversion escrito en C++ utilizando el marco de trabajo wxWidgets, el mismo está licenciado bajo la Licencia Pública General (GPL de General Public License en inglés).
Algunas de sus principales características se muestran a continuación:

- Proporciona una interfaz fácil de usar para las funciones de Subversion.
- Sencillo para los principiantes, pero lo suficientemente flexible como para aumentar la productividad de los usuarios de Subversion con experiencia.
- Se ejecuta en cualquier plataforma en la que Subversion y wxWidgets se puedan ejecutar: Linux, Windows, Mac OS / X, Solaris, etc (27).

1.14 Conclusiones

En este capítulo se mostró una panorámica sobre algunos de los modelos de desarrollo de componentes creados por corporaciones destacadas en el desarrollo de software, pero ninguno de estos modelos satisface las necesidades específicas del Módulo Común del PTPC, por lo que se decidió desarrollar dicho módulo atendiendo a las especificaciones propias del negocio, de forma tal que cumpla con todas las necesidades del resto de los módulos. Además se abordaron conceptos

que facilitaron la comprensión del trabajo y características de las herramientas, tecnologías y metodología utilizada para el desarrollo. En la elaboración del módulo todas las herramientas que se utilizaron son libres:

- PHP 5.2.5 como lenguaje de programación.
- Sauxe 2.0 como marco de trabajo.
- Visual Paradigm 3.4 como herramienta case para el modelado.
- Apache 2.0 como servidor Web.
- PostgresSQL 8.4 como Sistema Gestor de Base de Datos.
- Netbeans 7.0.1 como IDE de desarrollo.
- RapidSVN 0.12.0 para el control de versiones.

Capítulo 2

Propuesta de Solución

2.1 Introducción

En el presente capítulo se tratan los elementos y características relacionadas con el diseño e implementación del PCS. Se expone además el estándar de codificación y los patrones empleados durante el desarrollo así como el modelo de diseño e implementación que se siguió para la elaboración del PCS. También se muestra la concepción de la arquitectura y algunos rasgos esenciales en el funcionamiento del marco de trabajo, con el objetivo de brindar una mayor comprensión de los componentes implementados.

2.2 Arquitectura del Sistema de Informatización de Tribunales

Como se menciona en el capítulo anterior, el desarrollo del SIT responde a una arquitectura en **n capas**, la cual define 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, además se emplea el patrón arquitectónico Modelo–Vista–Controlador (MVC). Este es de los más utilizados en aplicaciones web, ya que permite desarrollar sistemas fácilmente escalables, sencillos de mantener y que no mesclen lenguajes de programación. Dicho patrón divide la aplicación en tres capas o niveles de abstracción las cuales son: Modelo, Vista y Controlador. Se puede decir que la principal característica de la aplicación de MVC es aislar la vista del modelo.

La estructura que define el marco de trabajo Sauxe para la separación de estas capas deja bien estructurado y delimitado donde quedan las clases del modelo, las de la vista y las controladoras.

Las clases que se encuentran en el paquete “TPC/apps/moduloComun/model”, van a ser las que responden al modelo como se muestra en la figura 3, que son las que representan la información con la que trabaja la aplicación, es decir la lógica del negocio. Dentro del paquete “*bussines*” (negocio) se encuentran las clases “*Model*” que son las encargadas de insertar, modificar o eliminar objetos de las clases entidades del modelo de datos, dentro del paquete “*domain*” (dominio) se encuentran las

clases del dominio que heredan de sus respectivas clases *Base*[7] que se encuentran en el paquete "*generated*" que también pertenece al paquete "*domain*".

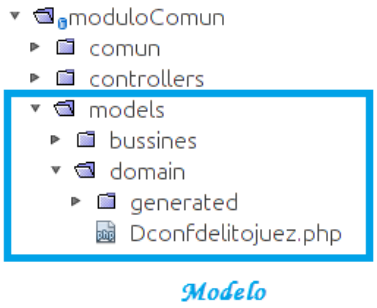


Figura 3 Capa del Modelo del Módulo Común

Las clases que se encuentran dentro del paquete "TPC/apps/moduloComun/views", en conjunto con las que importan que se encuentran en "TPC/web/moduloComun/views" son las que responden a la capa de la vista como se aprecia en las figuras 4 y 5 respectivamente, que son las que transforman el modelo en una página web que le permite al usuario interactuar con ella, para esto se apoya en la utilización de la librería ExtJS, que facilita la construcción de interfaces con un buen acabado y agradables a la vista del usuario.

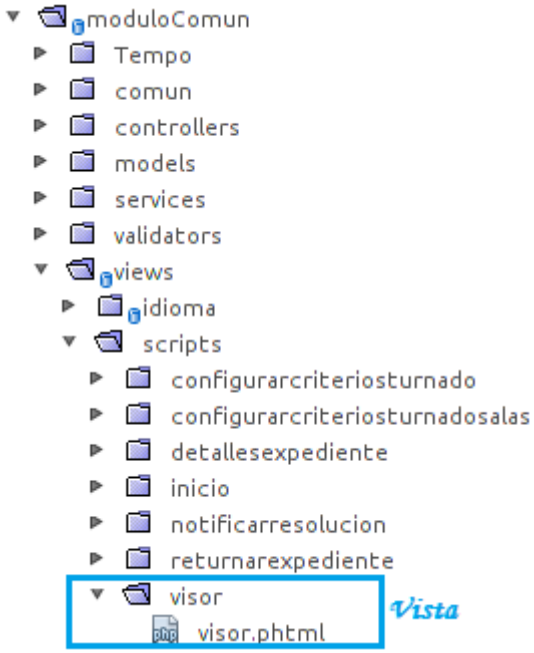


Figura 4 Capa de la vista paquete apps

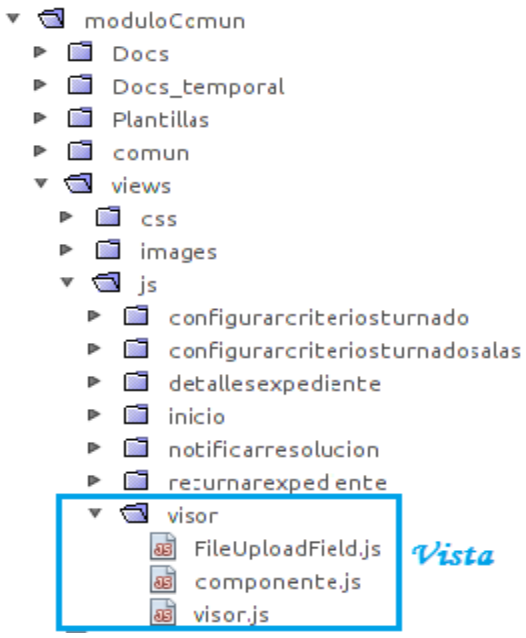


Figura 5 Capa de la vista paquete web

[7] Son las clases que genera el ORM Doctrine, que contienen los mismos atributos y relaciones que existen en el modelo de datos.

Las controladoras son las que se encuentran dentro del paquete "TPC/apps/moduloComun/controllers", estas son las encargadas de procesar las interacciones del usuario, recibir peticiones de la vista y gestionar y procesar la vista, además si es necesario solicita acciones del modelo y realizan cambios apropiados en la capa del modelo o la vista según sea el caso.

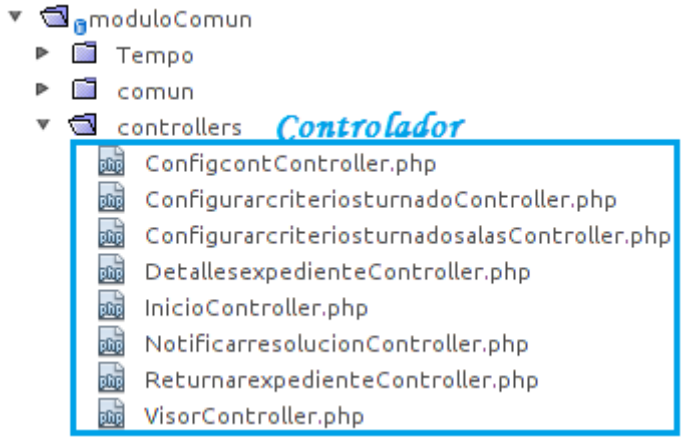


Figura 6 Capa del controlador del Módulo Común

Esta estructura en la construcción de los componentes comunes es muy útil puesto que permitió sin ningún inconveniente el desarrollo de varias interfaces comunes, a las cuales cada módulo le realizaría el resto según el negocio de cada uno de ellos. Ejemplo de algunas de estas interfaces son "Notificar resolución a las partes" y "Ver Detalles del Expediente", las cuales se pueden desarrollar por la independencia que brinda esta estructura. Además dicha estructura ha permitido en el desarrollo realizar cambios en cada una de las capas sin necesidad que las demás tengan que variar. También posibilita desarrollar varias vistas con el mismo modelo de datos puesto que en varios casos de usos como son el "Returnar Expedientes" y el "Reincorporar expedientes returnados" las vistas son distintas pero en esencia la capa de modelo es la misma.

2.3 Rasgos particulares en el desarrollo del SIT

El **SIT** posee un modelo de datos muy grande debido a que tiene 8 subsistemas que generan una enorme cantidad de información, es por eso, que se decidió separar en esquemas, donde cada esquema responde al modelo de datos de cada subsistema. Esta separación provee un incremento de la seguridad necesario para cualquier aplicación de gestión de información judicial, ya que al ser independiente cada uno de los diferentes modelos de datos, estos mantendrán su consistencia

debido a que las funcionalidades o servicios de otros módulos ajenos no podrán influir directamente sobre el modelo de un módulo específico, logrando de esta manera que el propio subsistema sea el responsable de su modelo de datos. Producto de esta separación durante el desarrollo del **SIT** se hizo necesario la aplicación del patrón IoC[8] lo que posibilita a módulos externos modificar u obtener datos de un módulo determinado con su previo conocimiento.

2.4 Patrones utilizados

2.4.1 Patrón IoC

La Inversión de Control es un patrón de diseño pensado para permitir un menor acoplamiento entre componentes de una aplicación y fomentar así la reutilización de los mismos. Este patrón se emplea en Sauxe con el objetivo de que cada módulo publique los servicios que otros módulos necesitan consumir de él, ya que es el único que posee los permisos para acceder a su modelo de datos, logrando de esta manera una interoperabilidad eficaz entre los diferentes componentes de cada módulo. Dentro de la estructura del proyecto en "TPC/app/Nombre_Módulo/Services" se encuentran todas las clases que implementan o consumen los servicios de cada uno de los módulos del proyecto. En la Figura 7 se muestra una imagen donde se puede apreciar la implementación del servicio turnar, este aparece con el nombre *proximoEnTurno* pero posteriormente es especificado con el nombre de turnar en ioc-general del Módulo Común (Figura 8) del PTPC.

```
class TurnarService {
    function proximoEnTurno($materia,$sala,$expediente)
        {
            $idtribunal= $_SESSION['UCID_Cedrux_UCI']['entidad']->idestructura;
            $sala=$_SESSION['UCID_Cedrux_UCI']['entidad']->sala;
            //Leer la configuracion de turnado que tiene el tribunal y la sala en cuestion
            $confTurnado = new DconfturnadoExt();
            $confTurnado = $confTurnado->buscarConfTurnado($idtribunal, 1);
            foreach ($confTurnado as $configuracion) {
                $criterio = $configuracion['idncriterio'];
                switch ($criterio){
                    case 1://-------Criterio de turnado uno a uno
                        return $this->TurnadoUnoaUno($configuracion,$sala);
                    case 2://-------Criterio de turnado por porcedimientos
                        return $this->TurnadoPorProcedimientos($materia,$configuracion, $sala, $expediente);
                    case 3://-------Criterio de turnado por delitos
                        return $this->TurnadoPorDelitos($materia, $configuracion, $sala, $expediente);
                    case 4://-------Criterio de turnado por territorios
                        return $this->TurnadoPorTerritorios($materia, $configuracion, $sala, $expediente);
                    default:return;

                }
            }

            return $juezTurnado;
        }
```

Figura 7 Implementación del servicio Turnar utilizado para turnar expedientes

[8] Inversion of Control por sus siglas en inglés.

Además en "TPC/app/Nombre_Módulo/comun/recursos/xml" se encuentra un fichero de configuración llamado "ioc-general.xml" que es donde cada módulo publica los servicios correspondientes a sus componentes.

```
<moduloComun src="moduloComun">
        <FechaCompleta>
        <ObtenerFechaHoy>
        <ListarPaises>
        <ListarDpa>
        <ObtenerMac>
        <Pasar_definitivo>
        <insertar_notificacion>
        <turnar reference="">
                <inyector clase="TurnarService" metodo="proximoEnTurno" />
                <prototipo>
                        <parametro nombre ='materia' tipo='integer'/>
                        <parametro nombre ='sala' tipo='integer'/>
                        <parametro nombre ='expediente' tipo='integer'/>
                        <resultado tipo="array" />
                </prototipo>
        </turnar>
         <visualizarDocumento>
        <calcularEdad>
        <LeerNumero>

         <GuardarReturnado>

</moduloComun>
```

Figura 8 Especificación del servicio turnar en el ioc-general del Módulo Común

2.4.2 Patrón Singleton

El patrón de diseño **singleton** (que significa 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 propósito consiste en garantizar que una clase sólo tenga una instancia y proporcionar un único punto de acceso global a ella. Dicho patrón provee una **única instancia** global debido a que si se solicita una instancia de la clase:

- La propia clase es la responsable de crear la única instancia, la cual debe garantizar que no se pueda crear ninguna otra (interceptando todas las peticiones para crear nuevos objetos) y proporcionando un solo punto de acceso a ella.
- Si no se ha creado anteriormente (o sea, es la primera vez que se usa esta clase) se crea la instancia.
- Si existe ya anteriormente una instancia (es la segunda o más veces que se usa), se retorna la existente todas las veces que se solicite.
- El constructor de la clase debe declararse como privado para que la clase no pueda ser instanciada directamente.

- Además en **PHP5** no se permite que la clase sea clonada con el método __clone ().

```
class ZendExt_Event {

    /**
     * Instancia para la implementación del patrón Sigleton
     *
     * @var ZendExt_Event
     */
    private static $_instance;

    /**
     * Constructor
     */
    private function __construct() {
        $this->_xml = ZendExt_FastResponse::getXML('events');
        $this->_ioc = ZendExt_IoC::getInstance();
    }
    /**
     * Retorna una instancia de ZendExt_Event para el singleton
     *
     * @return ZendExt_Event
     */
    static function getInstance () {
        if (self :: $_instance == null)
            self :: $_instance = new self ();

        return self :: $_instance;
    }
```

Figura 9 Implementación del patrón Singleton que se emplea en la clase ZendExt_Event

Con la implementación del patrón Singleton que se muestra en la figura 9 se fuerza a que **no se puedan crear objetos de forma directa** mediante el operador **"new"** (28), pues se puede apreciar en la implementación de la clase ZendExt_Event primero se crea la instancia que será utilizada en la implementación del patrón, que se declara privada para con el objetivo de que dicha clase sea la única que tenga permisos para modificarla, luego se declara privado el constructor, con el fin de que este no tenga una visibilidad pública eliminando de esta manera la posibilidad de que esta clase sea instanciada mediante el operador **"new"** y finalmente se verifica si esta instancia ha sido creada, si

no ha sido creada se crea y se retorna y si ya fue creada en el momento que se llamó la función **getInstance()** solamente se retorna.

2.4.3 Patrón Controlador

El patrón controlador es un patrón que sirve de intermediario entre una interfaz específica y el algoritmo que la implementa, de tal manera que es quien recibe los datos del usuario y los envía a las distintas clases según el método llamado. Este patrón propone que la lógica de negocios debe estar separada de la capa de presentación, con el objetivo de aumentar la reutilización de código y a la vez tener un mayor control. En la estructura que propone Sauxe este patrón se evidencia dentro de la carpeta controllers que se encuentra en "TPC/app/Nombre_Módulo/controllers", dentro de esta carpeta se encuentran todas las clases controladoras de cada uno de los módulos y dichas clases son las responsables de implementar todas las funcionalidades pertenecientes a una interfaz de un componente determinado, además son las que reciben los datos del usuario y los utilizan de acuerdo a la acción solicitada.

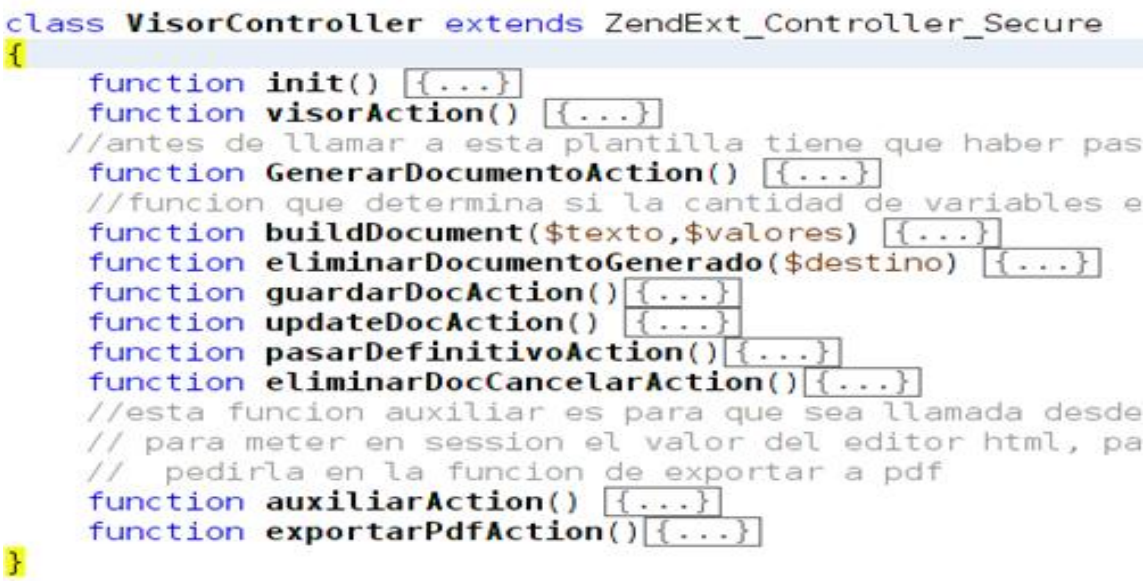


Figura 10 Clase VisorController encargada de implementar las funcionalidades de la interfaz del visor de documentos

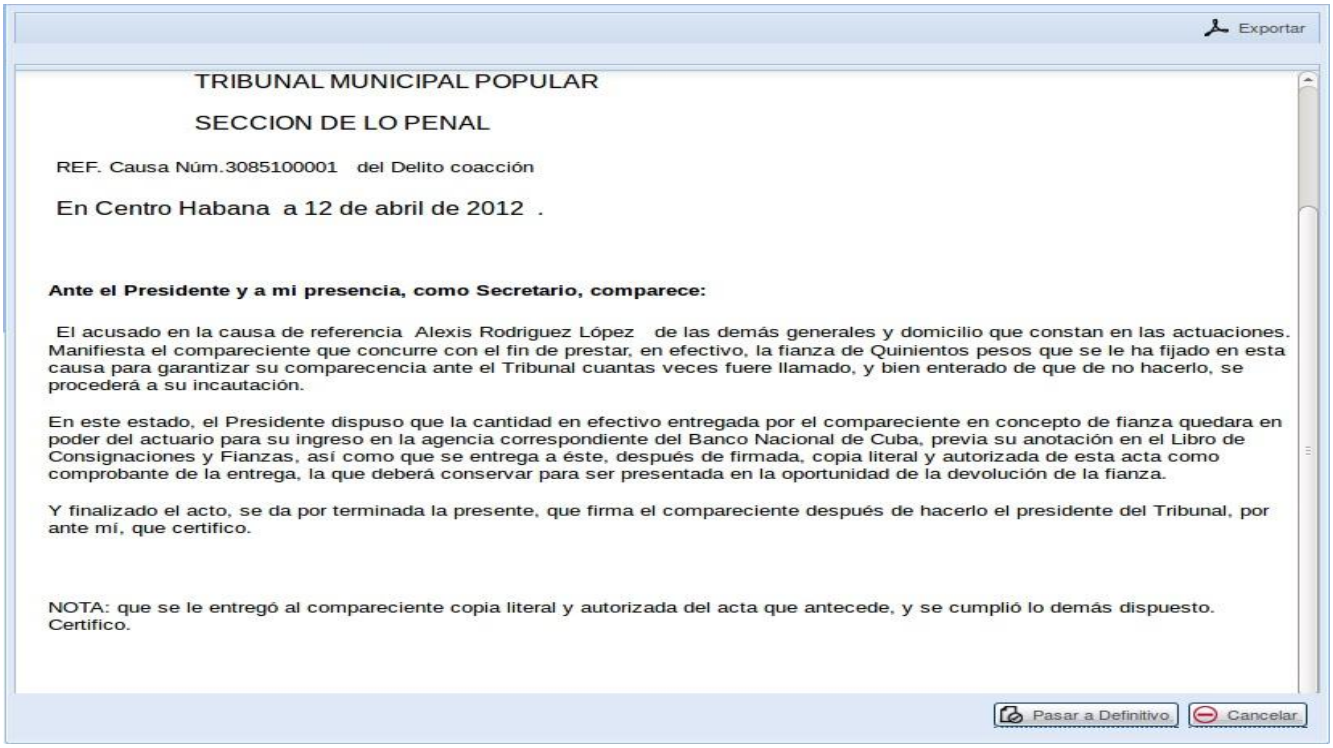


Figura 11 Interfaz visual del Visor de documentos

2.4.4 Patrón Bajo acoplamiento

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

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

```php
<?php
class Dconfdelitojuez extends BaseDconfdelitojuez
{
   public function setUp()
   {
                parent::setUp();

        $this->hasOne('Dconfturnado', array('local'=>'idconfturnado','foreign'=>'idconfturnado'));
   }

}
?>
```

Figura 12 Clase del modelo que solo depende de su clase Base

Como se aprecia en la figura anterior la clase del modelo "**Dconfdelitojuez**" solo depende de su clase Base y no de ninguna clase de la vista o la capa del controlador.

2.5 Modelo de diseño

El modelo de diseño es un modelo de objetos que describe la realización física de los casos de uso, centrándose en los requisitos funcionales y restricciones importantes. Este modelo como el diseño, tiene como entrada esencial el resultado del análisis, o sea el modelo de análisis y define una abstracción de la implementación, siendo utilizado como entrada fundamental de las actividades de implementación, puesto que es considerado como un plano para estas. Este modelo lo componen varios artefactos, de los cuales se abordan los Diagramas de Clases del Diseño y los Diagramas de Interacción.

2.5.1 Diagramas de clases del diseño

Los diagrama de Clases de la fase del Diseño son una representación estática de los elementos de la solución del sistema, mostrando la especificación para las clases software de una aplicación, presentando como notación de los elementos que lo forman (clases e Interfaces) y las relaciones que existen entre los mismos (asociaciones).

Las clases se representan mediante una caja subdividida en tres partes, en la parte superior se muestra el nombre, en el medio los atributos y debajo las operaciones (los métodos) (29). Se van a utilizar además estereotipos web para la representación de los formularios, los archivos JavaScript, los archivos CSS, las páginas clientes y las páginas servidoras.

A continuación se muestra la representación del diagrama de clases del diseño del componente: **Configurar Criterio de Turnado (CCT)[9].**

[9] Ver modelo de diseño en los artefactos entregados

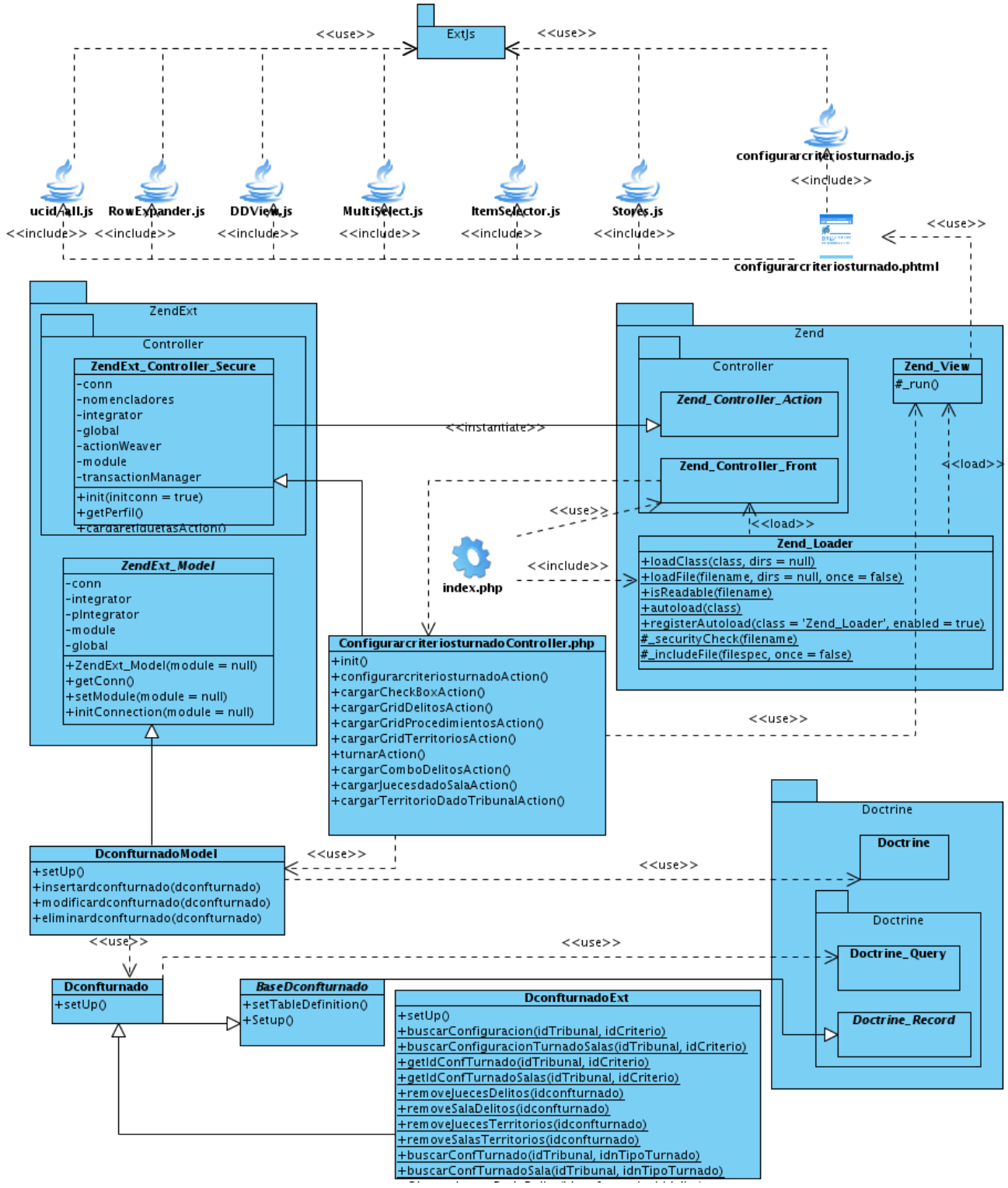


Figura 13 Diagrama de clases del diseño correspondiente al componente CCT

2.5.2 Diagramas de interacción

Primeramente es bueno hacer la salvedad de que existen dos tipos de diagramas de interacción, ambos se basan en la misma información y muestra un patrón de interacción entre objetos, pero cada uno de ellos se enfatiza a un aspecto en particular. Estos diagramas son: el diagrama de colaboración y el diagrama de secuencia.

2.5.2.1 Diagramas de secuencia

En el caso de los diagramas de secuencia que son los que se presentan en la investigación, muestran una iteración ordenada según la secuencia temporal de los eventos, es decir muestran los objetos que interactúan y los mensajes que se intercambian ordenados según la secuencia de tiempo en que se realiza cada uno de ellos, quedando establecido para el eje vertical la representación del tiempo y en el horizontal es donde se colocan los objetos y actores participantes en la interacción, estos últimos no tienen que tener un orden prefijado pues cada uno tiene una línea vertical hacia donde se establecen los mensajes que se muestran mediante flechas, estableciendo el tiempo de arriba hacia abajo (29).

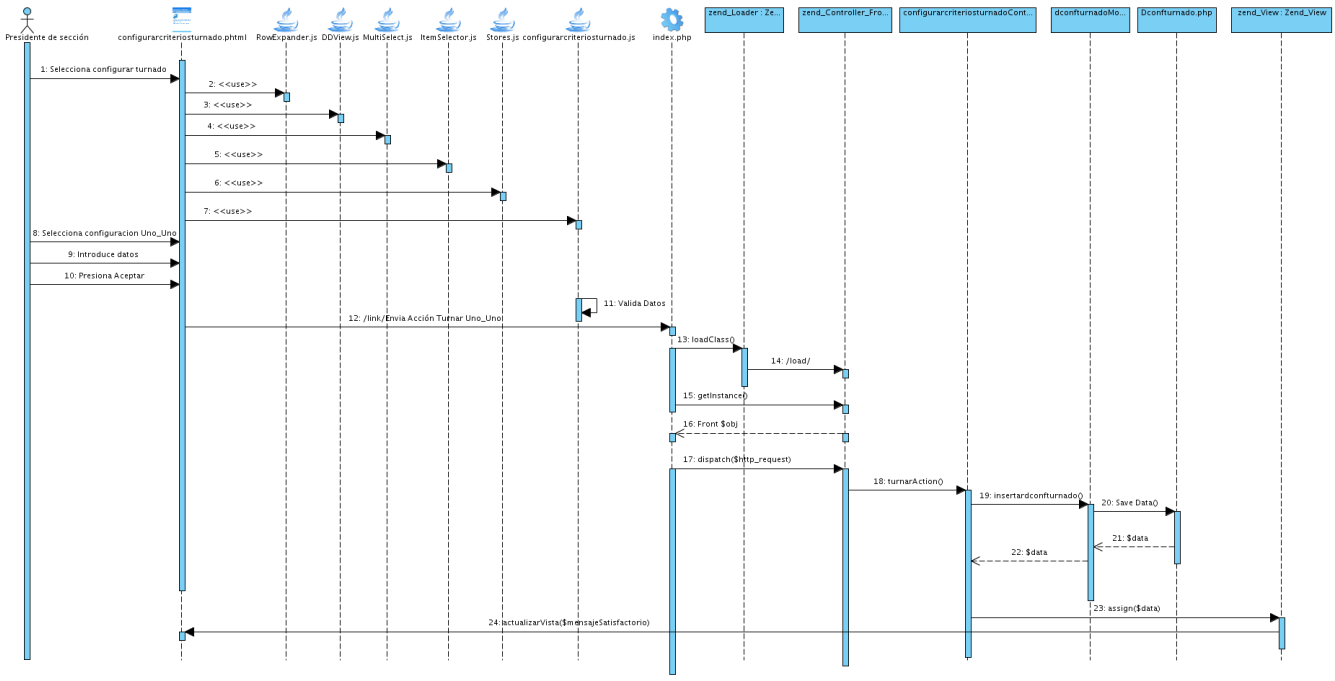


Figura 14 Diagrama de secuencia del escenario configurar criterios de turnado uno a uno del componente CCT

2.6 Modelo de datos

El modelo de datos es el proceso que implica crear una representación que tienen los usuarios de los datos. Básicamente está formado por tres elementos fundamentales: 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 a él Modulo Común consta de un total de 17 tablas, de ellas 2 nomencladores y las 15 restantes para el almacenamiento de la información correspondiente al flujo de trabajo del módulo. Para la construcción se tuvo en cuenta todos los datos que se necesitan persistan en el sistema y en los nomencladores se agruparon los estándares definidos por el negocio común. La nomenclatura de dichas entidades es de la siguiente forma: para las tablas de datos d<Nombre>, para las nomencladoras n<Nombre> y para las que almacenan relaciones entre entidades d<Nombre1><Nombre2>. A continuación se muestra el modelo de datos del Módulo Común.

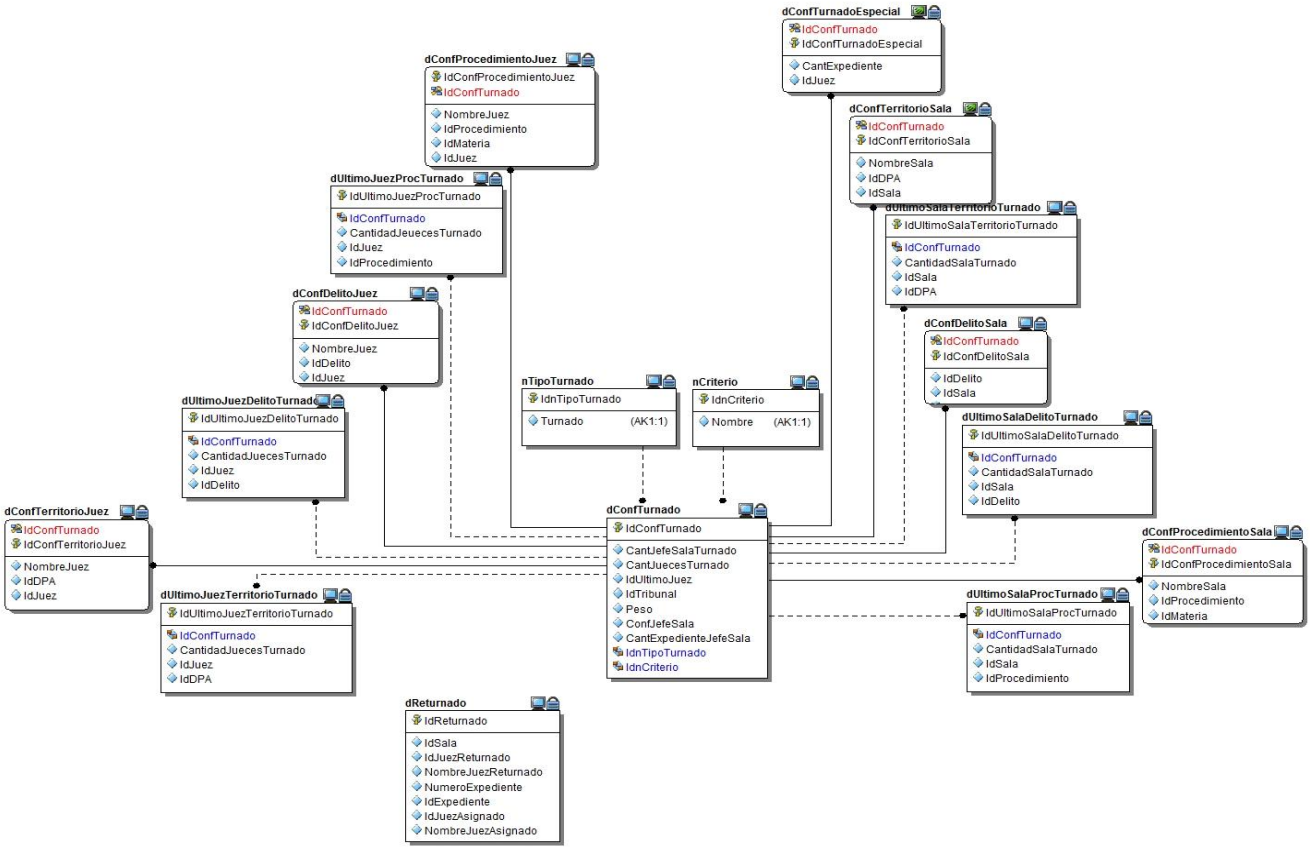


Figura 15 Modelo de Datos del Módulo Común

2.7 Modelo de implementación

El modelo de implementación describe cómo los elementos del modelo de diseño se implementan en términos de componentes. Describe también cómo se organizan los componentes de acuerdo con los mecanismos de estructuración y modularización disponibles en el entorno de implementación y en el lenguaje de programación utilizado y cómo dependen los componentes unos de otros. Como parte del Modelo de Implementación se encuentran los Diagramas de Componentes.

2.7.1 Diagramas de componentes

Los diagramas de componentes describen los elementos físicos de un sistema y sus relaciones. Muestran las opciones de realización incluyendo código fuente, binarios y ejecutables. Los componentes representan todos los tipos de elementos software que entran en la fabricación de la aplicación informática. Pueden ser simples archivos, paquetes, bibliotecas cargadas dinámicamente, etc. A continuación se muestra el Diagrama de Componentes del componente “Configurar Criterios de Turnado por Expedientes” del Módulo Común:

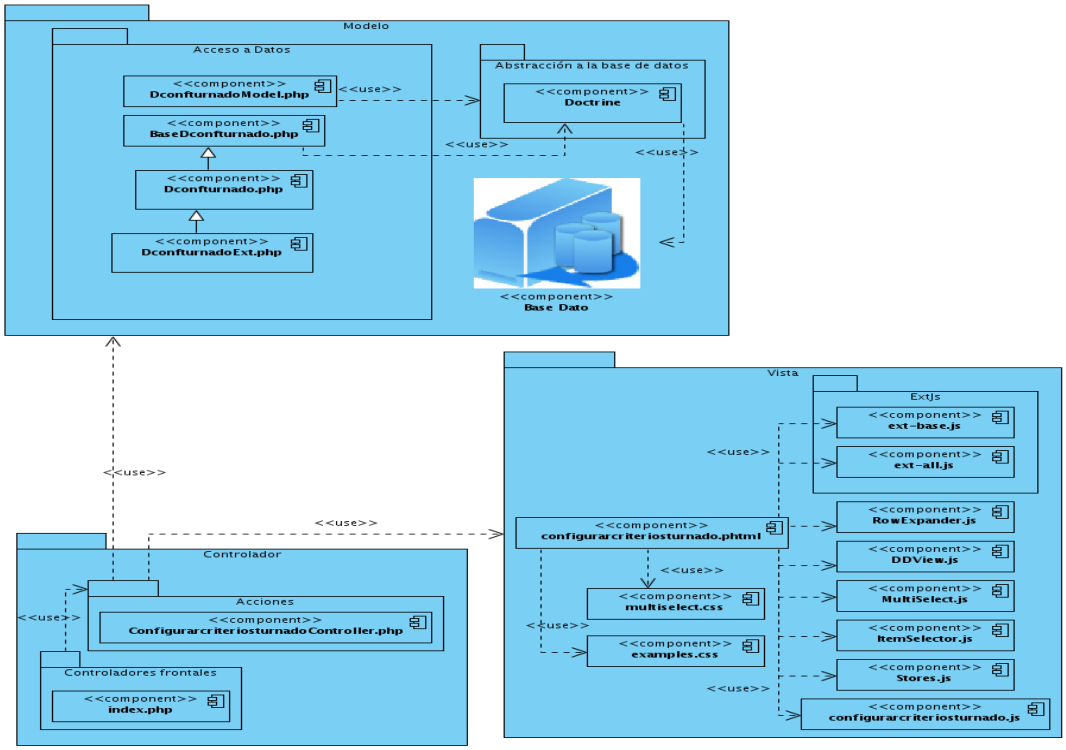


Figura 16 Diagrama de componentes del componente CCT

2.8 Estándares de nomenclatura y codificación utilizados

Los estándares de codificación son un conjunto de directrices, normas y reglamentos enfocados a la especificación de cómo debe escribirse el código fuente de la aplicación. Estos incluyen pautas sobre la nomenclatura de las variables, clases y paquetes; la correcta identación del código, como escribir estructuras de control e incluso las formas de poner los comentarios, en sí todo lo referente a la generación del código.

La correcta elaboración y utilización de este permite que todos los desarrolladores implementen siguiendo las mismas pautas y así puedan entender el código del resto como si estuvieran mirando el de ellos, así la aplicación será más legible, uniforme y fácil de mantener. Adoptar un estándar de codificación para la construcción de un sistema sólo es viable si se sigue desde el principio hasta el final. No es factible adoptarlo comenzado el proyecto ni que sea cambiado, es por ello que en el caso del PTPC fase 1 el equipo de arquitectura en conjunto con la dirección del proyecto definieron el estándar[10] de codificación en sus inicios. Es importante destacar que tanto el diseño para PHP como para JavaScript se debe hacer orientado a objetos a fin de potenciar todas las ventajas que este paradigma implica, además de saber que el definido se apoya en el establecido para el lenguaje PHP.

- **Nomenclatura General**
  - Se exceptúan el uso de las tildes y la letra ñ, la que será sustituida por nn.
  - En todo momento se utilizarán nombres que sean claros, concretos y libres de ambigüedades. Ejemplo: "fechaTurnado" y no solamente "fecha",
  - El nombre de todas las variables y métodos comenzarán con letra minúscula y si este está compuesto por varias palabras se utilizará el estilo de escritura lowerCamelCase, que dicta que para un nombre compuesto por varias palabras comenzará con minúscula pero todas las palabras internas que lo componen comienzan con mayúscula.
- **Identación**
  - En el contenido siempre se identará este con tabs, nunca utilizando espacios en blanco.
- **Clases**
  - El nombre de las clases controladoras estará en correspondencia con la funcionalidad que represente y seguido de la palabra Controller.

[10] Disponible en: http://alfresco.cegel.prod.uci.cu/alfresco/faces/jsp/browse/browse.jsp

- El nombre de las clases comenzará con mayúscula y solo la letra inicial de la palabra Controller también con mayúscula.

Ejemplo:

```
class ConfigurarcriteriosturnadoController extends ZendExt_Controller_Secure
{
```

Figura 17 Ejemplo de cómo se deben nombrar las clases

- Intentar mantener los nombres de las clases simples y descriptivas. Usar palabras completas, evitar acrónimos y abreviaturas (a no ser que la abreviatura sea mucho más conocida que el nombre completo, como URL o HTML).
- El número de métodos de una clase debe ser inferior a 40.

- **Rutinas y Métodos**
  - Un buen nombre para una función o método es aquel que describe todo lo que la rutina hace.
  - Es recomendable que los nombres de los métodos comiencen con un verbo, seguido del objeto al que afecta. Por ejemplo: verExpediente, turnarJuez.
  - Los nombres de los métodos de las clases controladoras incluirán luego del nombre de la funcionalidad la palabra "Action". Ejemplo: function turnarAction(), function cargarJuecesDadoSalaAction().
- **Nombre de variables**
  - No se utilizarán nombres de variables que puedan ser ambiguos. Por ejemplo Col es un nombre ambiguo ya que puede ser columna o color.
  - Se procurará evitar dar nombres sin sentido a variables temporales. Por ejemplo: temp, i, tmp.
  - Las variables booleanas deben tener nombres que sugieran respuestas o contenidos de tipo S/N, por ejemplo: Trabaja, Correcto, Realizado.
  - Los nombres de las variables booleanas deben ser positivos, por ejemplo: encontrado en lugar de noEncontrado.

Así queda descrito parte importante del estándar de codificación definido para el PTPC fase 1, para una mejor referencia y estudio del mismo consultar el documento "Estilos de Codificación" del PTPC fase 1[11].

[11] Es uno de los documentos entregados en la investigación.

2.9 Descripción de clases y operaciones

Documentar el código de una aplicación es añadir suficiente información como para explicar lo que hace detalladamente, de forma que no sólo los ordenadores sepan qué hacer, sino que además los humanos entiendan qué están haciendo y por qué. La descripción de las clases y sus operaciones constituye una de las tareas dentro de la documentación del código, además constituye una necesidad que sólo se aprecia en su debida magnitud cuando hay errores que reparar o hay que extender el programa con nuevas capacidades o adaptarlo a un nuevo escenario.

Por una u otra razón, todo componente o programa que tenga éxito será modificado en el futuro, bien por el programador original, bien por otro programador que le sustituya. Pensando en esta revisión de código es por lo que es importante que el programa se entienda: para poder repararlo y modificarlo. A continuación se pone un ejemplo de cómo se describen las clases y sus operaciones en el documento[12] de descripción de clases del Módulo Común, para esto se tuvo en cuenta el componente “Configurar Criterios de Turnado”.

2.9.1 Clases controladoras

Nombre: ConfigurarcriteriosturnadoController.php	
Tipo de clase: Controladora	
Nombre de función	**Descripción**
configurarcriteriosturnadoAction	Acción que manda ejecutar al controlador visual la interfaz correspondiente a esta clase controladora.
cargarGridDelitosAction	Acción que carga los jueces que turnan por un delito determinado en el tribunal o la sala por la que se ha autenticado el usuario.
cargarGridProcedimientosAction	Acción que carga los jueces que turnan por un procedimiento determinado en el tribunal o la sala por la que se ha autenticado el usuario.
cargarGridTerritoriosAction	Acción que carga los jueces que turnan por un territorio especifico en el tribunal o la sala por la que se ha

[12] Ver en los artefactos entregados como resultado de la investigación.

	autenticado el usuario.
turnarAction	Acción que guarda todas las configuraciones de turnado que haya realizado el usuario.
cargarComboDelitosAction	Acción que carga todos los delitos correspondientes a la instancia del tribunal por el que se ha iniciado sesión.
cargarJuecesdadoSalaAction	Acción que carga todos los jueces pertenecientes a la sala por la que se ha iniciado sesión.
cargarTerritorioDadoTribunalAction	Acción que carga todos los territorios correspondientes al tribunal por el que se ha iniciado sesión.

Tabla 2 Descripción de la clase controladora del componente CCT

2.9.2 Clases del modelo

Nombre: DconfturnadoExt.php	
Tipo de clase: Modelo	
Nombre de función	**Descripción**
buscarConfiguracion($idTribunal,$idCriterio)	Devuelve la configuración correspondiente a los parámetros entrados.
getIdConfTurnado($idTribunal,$idCriterio)	Devuelve el identificador de la configuración de turnado que contenga los parámetros especificados.
removeJuecesDelitos($idconfturnado)	Función que elimina todas tuplas existentes en la tabla “Dconfdelitojuez” correspondientes al parámetro especificado.
removeJuecesTerritorios($idconfturnado)	Función que elimina todas las tuplas existentes en la tabla “Dconfterritoriojuez” correspondientes al parámetro dado.
buscarConfTurnado($idTribunal,$idnTipoTurnado)	Devuelve la configuración correspondiente a los parámetros

	entrados.
ObtenerJuecesDadoDelito($idconfturnado,$iddelito)	Devuelve todos los jueces que pertenecen a la configuración y el delito que se especifica por parámetros.
ObtenerJuecesDadoTerritorio($idconfturnado,$iddpa)	Devuelve todos los jueces que pertenecen a la configuración y el territorio que se especifica por parámetros.
buscarConfiguracionTurnadoSalas($idTribunal,$idCriterio)	Devuelve la configuración de turnado por salas correspondiente a los parámetros especificados.
getIdConfTurnadoSalas($idTribunal,$idCriterio)	Devuelve el identificador de una configuración de turnado por salas correspondiente a los parámetros especificados.
removeSalaDelitos($idconfturnado)	Función que elimina todas las tuplas existentes en la tabla “Dconfdelitosala” correspondientes al parámetro especificado.
removeSalasTerritorios($idconfturnado)	Función que elimina todas las tuplas existentes en la tabla “Dconfterritoriosalas” correspondientes al parámetro especificado.
ObtenerSalasDadoDelito($idconfturnado,$iddelito)	Devuelve todas las salas que turnan por el delito especificado y que pertenecen a la configuración pasada por parámetro.
ObtenerSalasDadoTerritorio($idconfturnado,$iddpa)	Devuelve las salas que turnan por el territorio especificado y correspondiente a la configuración pasada por parámetro.

Tabla 3 Descripción de la clase del modelo del componente CCT

2.9.3 Clases y ficheros de la presentación

Nombre del componente: Configurar turnado de expedientes	
Tipo de clase: Presentación	
Nombre de fichero	**Descripción**
configurarcriteriosturnado.phtml	Archivo que incluye los ficheros JavaScript y CSS para la presentación.
RowExpander.js	Archivo JavaScript que contiene la implementación del plugin “Expander” que se emplea en la interfaz gráfica de las tablas.
DDView.js	Archivo JavaScript que contiene la implementación de una función que permite arrastrar objetos en la interfaz gráfica de la configuración de turnado.
MultiSelect.js	Archivo que contiene la implementación de la clase “MultiSelect” la cual permite la selección múltiple de elementos en la interfaz gráfica del componente “Configurar Criterios de Turnado”.
ItemSelector.js	Archivo que contiene la implementación de la clase “ItemSelector”, la cual posibilita selección de elementos entre dos listas desplegadas en la interfaz gráfica de la configuración de turnado.
Stores.js	Archivo que contiene la implementación donde se crean las estructura que almacenan los datos que se muestran en la interfaz gráfica del componente “Configurar Criterios de Turnado”.
configurarcriteriosturnado.js	Archivo que contiene la implementación de todos los componentes visuales que contiene la interfaz gráfica del componente “Configurar Criterios de Turnado”.

Tabla 4 Descripción de los ficheros de la capa de presentación del componente CCT

2.10 Conclusiones

Con la realización de este capítulo, se evidenció que el diseño propuesto para la implementación de los componentes posibilitó la realización satisfactoria de los mismos, ya que el diseño tuvo en cuenta

las características del marco de trabajo empleado, con el objetivo de lograr un mayor aprovechamiento de estas para el funcionamiento de los componentes. Se definieron además los estándares de codificación que se asumieron para la implementación del PCS para obtener un código más legible y entendible, ayudando a la comunicación entre desarrolladores.

Se obtuvo una propuesta de solución que respondió a la implementación del conjunto de actos procesales comunes para cada una de las materias en el PTPC fase 1, brindando de esta manera una ayuda a los restantes subsistemas que solo tendrán que utilizar estos componentes.

Capítulo 3

Validación de la Solución Propuesta

3.1 Introducción

En el proceso de desarrollo de software la aparición de errores es frecuente debido a que en muchas ocasiones cuando los desarrolladores programan rutinas o funciones obvian algunas posibilidades que pueden variar el resultado esperado durante la ejecución del código, por lo que de alguna forma hay que tratar de asegurar que esto no suceda, y es entonces cuando entran las pruebas y validaciones de software a jugar su papel fundamental. Los errores pueden suceder a causa del mal uso de estructuras de datos, errores de integración de componentes, errores lógicos, entre otras causas.

En el presente capítulo se evidencia el resultado de las pruebas y validaciones realizadas a los diferentes componentes, que permiten garantizar el correcto funcionamiento y totalidad de las funcionalidades de los mismos.

3.2 Pruebas de software

Según la IEEE una prueba es: “Una actividad en la cual un sistema o uno de sus componentes se ejecuta en circunstancias previamente especificadas, los resultados se observan y registran y se realiza una evaluación de algún aspecto*”*. El objetivo fundamental de las mismas es medir el grado en que el software cumple con los requisitos.

3.2.1 Objetivos de las pruebas

- La prueba es el proceso de ejecución de un programa con la intención de descubrir un error.
- Un buen caso de prueba es aquel que tiene una alta probabilidad de mostrar un error no descubierto hasta entonces.
- Una prueba tiene éxito si descubre un error no detectado hasta entonces (30 pág. 304).

3.3 Evaluación de la calidad utilizando métricas

Las métricas de software constituyen los elementos que permiten evaluar la calidad de una determinada característica o artefacto que se genere en un proyecto de software. En este epígrafe se validará la solución propuesta mediante la aplicación de métricas que permitirán medir el estado de algunos atributos de calidad que facilitarán expresar una opinión valorativa sobre el diseño y la calidad de los componentes obtenidos.

Con la aplicación de estas métricas se abarcan los siguientes atributos de calidad:

- **Cantidad de pruebas**: consiste en el número o el grado de esfuerzo para realizar las pruebas de calidad del producto (clase, conjunto de clases, componente, módulo, entre otros.) diseñado.
- **Responsabilidad o Cohesión**: consiste en la responsabilidad asignada a una clase modelada 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.
- **Complejidad del mantenimiento**: consiste en el grado de esfuerzo necesario a realizar para desarrollar una reparación, una mejora o una corrección de algún error de un diseño de software. Influye fuertemente en los costes y la planificación del proyecto.
- **Reutilización**: consiste en el grado de reutilización presente en una clase o estructura de clase, dentro de un diseño de software.
- **Acoplamiento**: consiste en el grado de dependencia o interconexión de una clase o estructura de clase con otras, está muy atada a la característica de Reutilización.

3.3.1 Tamaño Operacional de las Clases

El tamaño general de una clase OO[13] se ajusta a realización de un cálculo de sus atributos u operaciones totales.

- El total de operaciones (operaciones tanto heredadas como privadas de la instancia), que se encapsulan dentro de la clase.
- El número de atributos (atributos tanto heredados como privados de la instancia), encapsulados por la clase.

Los valores grandes para la métrica TOC indican que la clase debe tener bastante responsabilidad, lo que propicia un bajo nivel de reutilización de la clase y complicará la implementación y las pruebas. En general, operaciones y atributos heredados o públicos deben ser ponderados con mayor

[13] Orientada a Objetos

importancia cuando se determina el tamaño de clase, pues las operaciones y atributos privados, permiten la especialización.

También se pueden calcular los promedios para el número de atributos y operaciones de cada clase. Cuanto más pequeño sea el valor promedio para el tamaño, será más viable para que las clases dentro del sistema puedan reutilizarse. Durante la realización de este trabajo se decidió aplicar la métrica para el tamaño de clases en función de sus operaciones (**TOC**[14]). Luego de aplicar dicha métrica a las clases de la solución la misma arrojó los siguientes resultados:

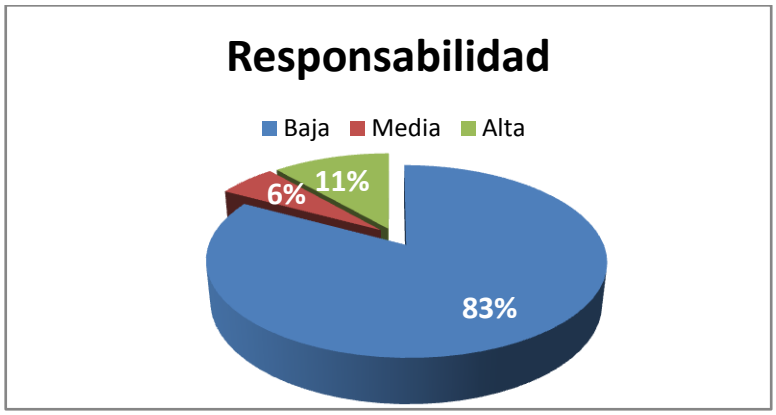


Figura 18 Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Responsabilidad

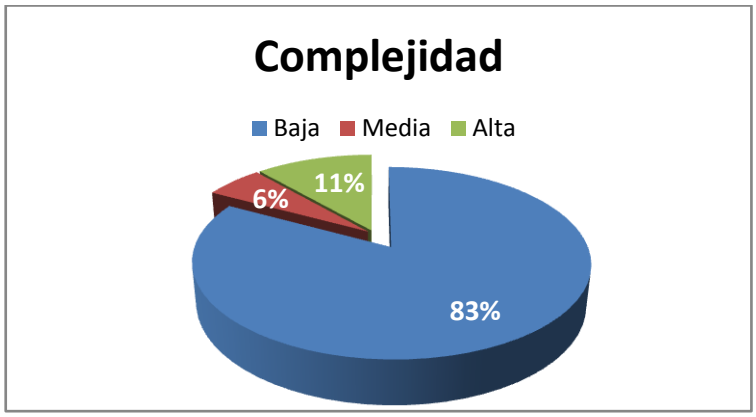


Figura 19 Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Complejidad de implementación

[14] Tamaño Operacional de Clases

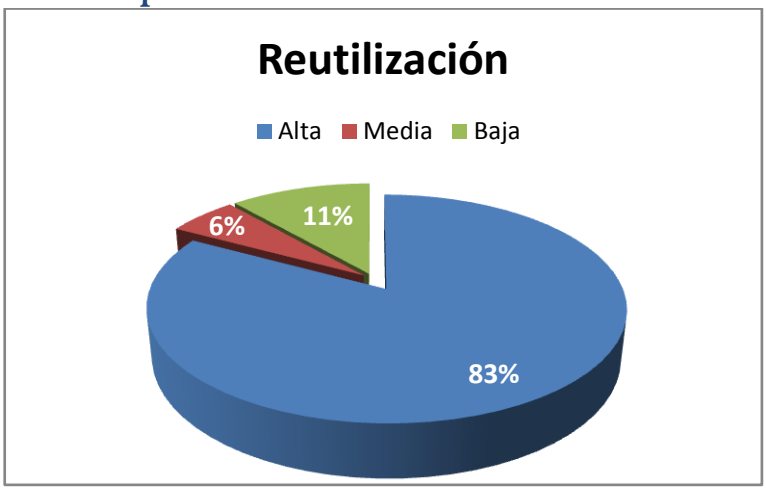


Figura 20 Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Reutilización

Después de aplicada la métrica se puede concluir que la implementación del PCS tiene una buena calidad, pues tuvo resultados satisfactorios en los diferentes atributos de calidad medidos, por lo que se puede afirmar que se obtuvo una solución eficiente con altos niveles de reutilización (83%) lo que garantiza que los componentes elaborados puedan servir de ayuda a otro proyectos de software. También se evidencia que el diseño realizado fue correcto ya que se obtuvieron bajos niveles de responsabilidad (83%) y complejidad de implementación (83%), pues se le asignaron correctamente las responsabilidades a las clases involucradas en la solución.

3.3.2 Relaciones entre clases

El resultado de la aplicación de la métrica **RC** está dado por el número de relaciones de uso de una clase con otras. La aplicación de dicha métrica permite evaluar atributos como el Acoplamiento, la Complejidad de mantenimiento, la Reutilización y la Cantidad de pruebas. De forma tal que mientras mayor sea las relaciones de uso entre clases mayor será el Acoplamiento, la Complejidad de Mantenimiento y la Cantidad de pruebas, mientras que su Reutilización disminuye.
Después de aplicar dicha métrica se obtuvieron los siguientes resultados:

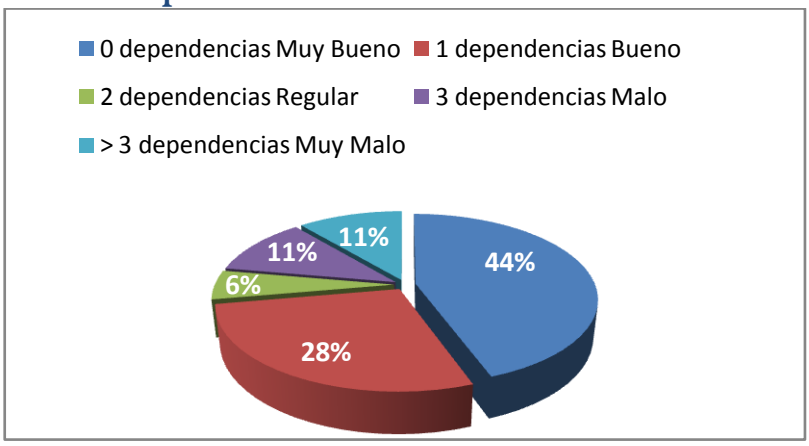


Figura 21 Representación en % de los resultados obtenidos al analizar las dependencias entre clases

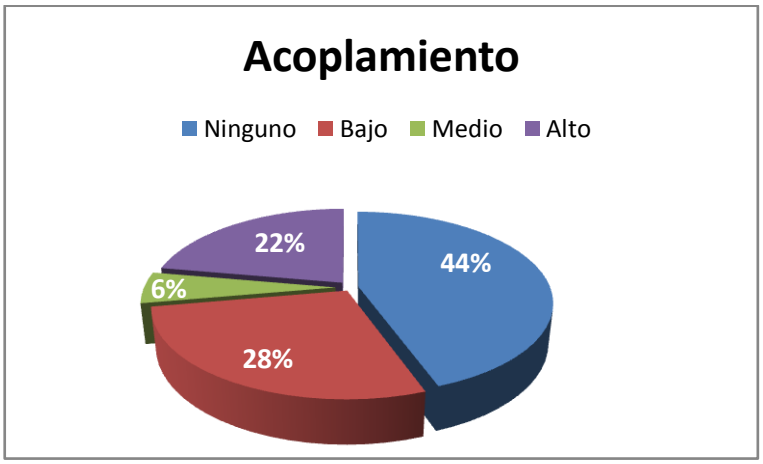


Figura 22 Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Acoplamiento

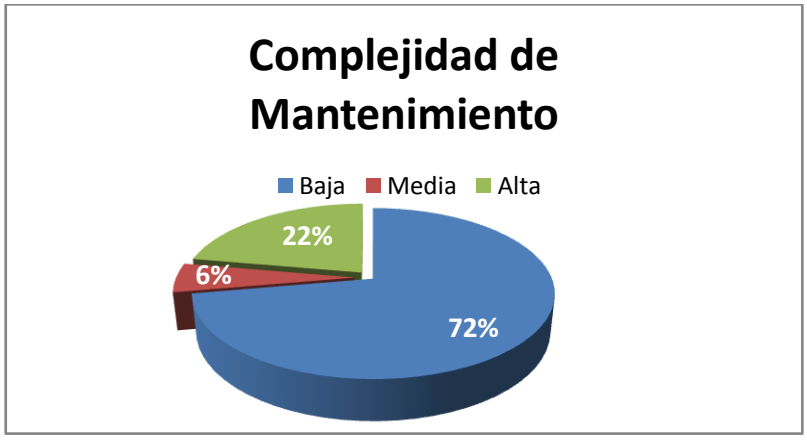


Figura 23 Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Complejidad de Mantenimiento

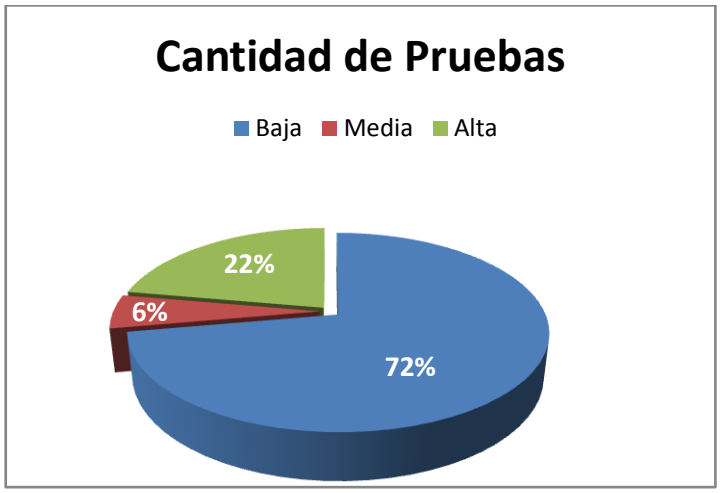


Figura 24 Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Cantidad de Pruebas

La aplicación de esta métrica de software demostró que existe una baja dependencia entre las diferentes clases de la solución propuesta con un 44% de ella sin dependencias y un 28% con una sola dependencia, además se demostró que dichas clases posee un bajo acoplamiento lo que posibilita una fácil desarticulación da las partes de los componentes desarrollados a la hora de reutilizar los mismos. También arrojó resultados satisfactorios en el atributo Complejidad de mantenimiento lo que facilita las tareas de corrección, modificación y mantenimiento de los componentes.

3.4 Pruebas de caja blanca

Las pruebas de caja blanca son las que se realizan sobre las funciones internas de un módulo, es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para obtener los casos de prueba. Mediante los métodos de prueba de caja blanca se puede obtener casos de prueba que:

- Garanticen que se ejercita por lo menos una vez todos los caminos independientes de cada módulo.
- Ejerciten todas las decisiones lógicas en sus vertientes verdadera y falsa.
- Ejecuten todos los bucles en sus límites y con sus límites operacionales.

- Ejerciten las estructuras internas de datos para asegurar su validez.

Algunas técnicas de prueba de Caja Blanca son:

- Prueba de condición: ejercita las condiciones lógicas contenidas en el módulo de un programa.
- Prueba del flujo de datos: 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: se centra exclusivamente en la validez de las construcciones de bucles.
- La prueba del camino básico: permite obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida corno guía para la definición de un conjunto básico de caminos de ejecución. La idea es realizar un grupo de casos de prueba que garanticen que se pueda probar por lo menos una vez cada una de las sentencias del programa (30).

En la validación de la solución del módulo, se realizó la prueba del camino básico a las funcionalidades de mayor complejidad, esta permite definir los diferentes caminos independientes de la función y probar su funcionamiento al menos una vez. A continuación se muestra el proceso desarrollado al método "*cargarComboPonenteGridAction*" de la clase "*ReturnarexpedienteController*".

Para ello se comienza por analizar el código y enumerar las instrucciones

```
function cargarComboPonenteGridAction(){
    $sala = $_SESSION['UCID_Cedrux_UCI']['entidad']->sala;//1
    $idjuez = $this->_request->getPost('idjuez');//1
    $integrator = ZendExt_IoC::getInstance();//1
    $juecesProfesionales =
        $integrator->administracion_gobierno->DevolverDatosJuecesProfesionales($sala);//1
    $presidente = $integrator->administracion_gobierno->DevolverPresidenteSala($sala);//1
    $juecesProfesionales[count($juecesProfesionales)]=array('id'=>$presidente[0]['id']
        , 'nombrecompleto' => $presidente[0]['nombrecompleto']);//1
    for ($i = 0; $i < count($juecesProfesionales); $i++){//2
        if($juecesProfesionales[$i]['id'] == $idjuez){//3
            unset($juecesProfesionales[$i]);//4
            $juecesProfesionales = array_values($juecesProfesionales);//4
            $juecesProfesionales[count($juecesProfesionales)] =
                array('id' => 'Seleccione', 'nombrecompleto' => 'Seleccione');//4
            $i = count($juecesProfesionales);//4
        }//4
    }//5
    echo json_encode($juecesProfesionales);//6
    return;//6
}
```

Figura 25 Función a la que se le aplica el método del Camino Básico

Seguidamente es necesario representar el grafo de flujo (o grafo del programa), que va a representar el flujo de control lógico del código anterior, a través de nodos, aristas y regiones. Quedando como muestra la Figura.

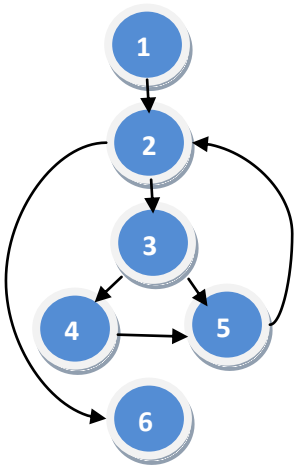


Figura 26 Grafo de flujo del código de la función de la figura anterior

Luego de realizado el grafo es necesario conocer la cantidad de caminos independientes que se deben buscar para probar; y para esto se calcula la complejidad ciclomática, que se puede calcular de tres formas:

1. El número de regiones del grafo de flujo coincide con la complejidad ciclomática. Por lo que V(G) = **3**
2. V (G) = (A – N) + 2 donde "**A"** es el número de aristas del grafo de flujo y "**N**" es el número de nodos del mismo.
   V (G) = 7 – 6 +2 = **3**
3. V (G) =P+ 1 donde "**P**" es el número de nodos predicado contenidos en el grafo.
   Los nodos 2 y 3 son nodos predicados. V (G) = 2+1 = **3**

Como se muestra en cualquiera de las tres formas anteriores la complejidad ciclomática es **3**, por lo que la cantidad de caminos básicos que puede tomar el algoritmo durante su ejecución es **3** y quedan definidos así:

Camino básico #1: 1 – 2 – 6

Camino básico #2: 1 – 2 – 3 – 5 – 2 – 6

Camino básico #3: 1 – 2 – 3 – 4 – 5 – 2 – 6

Ya seguidamente se preparan los casos de pruebas que forzarán la ejecución de cada camino del conjunto básico, quedando de la siguiente forma:

Descripción: Se deben cargar en el combo los datos que envía la función "cargarComboPonenteGridAction**".**

Esta descripción es común para todos los casos de prueba que se realizarán.

Caso de prueba para el camino básico # 1.

Condición de ejecución: Para esta prueba es necesario que la variable "$juecesProfesionales" no reciba datos, se encuentre vacía.

Entrada: La variable "$juecesProfesionales" vacía.

Resultados esperados: Teniendo en cuenta la condición de ejecución se espera que devuelva un arreglo vacío, el combo se queda sin datos.

El resultado obtenido fue correcto.

Caso de prueba para el camino básico # 2.

Condición de ejecución: Para esta prueba es necesario que la variable "$juecesProfesionales" reciba un grupo de datos (Jueces Profesionales), pero que no esté incluido el juez profesional que se está returnando.

Entrada: La variable "$juecesProfesionales recibe un arreglo de datos (Los jueces profesionales de la sala excluyendo el juez que se está returnando).

Resultados esperados: Teniendo en cuenta la condición de ejecución se espera que devuelva un arreglo de datos, en el combo se muestren los jueces profesionales.

El resultado obtenido fue correcto.

Caso de prueba para el camino básico # 3.

Condición de ejecución: Para esta prueba es necesario que la variable "$juecesProfesionales" reciba un arreglo de datos (Los jueces profesionales de la sala), donde esté incluido el juez que se está returnando.

Entrada: La variable "$juecesProfesionales recibe un arreglo de datos (Los jueces profesionales de la sala).

Resultados esperados: Teniendo en cuenta la condición de ejecución se espera que devuelva un arreglo de datos (Jueces profesionales de la sala), y que el juez que se está returnando haya sido eliminado, en el combo se muestren los jueces profesionales.

El resultado obtenido fue correcto.

3.5 Pruebas de unidad

Otro de los métodos de prueba aplicado a la solución fue el de las pruebas unitarias, el cual se centra el proceso de verificación en la menor unidad del diseño del software: el componente software o módulo.

Las pruebas que se dan como parte de las pruebas de unidad son:

- Estructuras de datos locales.
- Condiciones límite.
- Caminos independientes.
- Caminos de manejo de errores.

Las mismas se aplican a los módulos o componentes mediante la ejecución de los casos de pruebas definidos. Se prueba la interfaz del módulo para asegurar que la información fluye de forma adecuada hacia y desde la unidad de programa que está siendo probada y se examinan las estructuras de datos locales para asegurar que los datos que se mantienen temporalmente conservan su integridad durante todos los pasos de ejecución del algoritmo (30).

Durante la realización del proceso de pruebas se decidió aplicar pruebas unitarias a los algoritmos que presentan una complejidad considerable dentro de cada uno de los componentes. A continuación se presenta un ejemplo de la aplicación de dichas pruebas al algoritmo **buildDocument($texto,$valor)** del componente **Visor de documentos**. Para la ejecución de estas pruebas se utilizó el siguiente caso de prueba:

#	$texto	$valor	Respuesta esperada	Respuesta obtenida
1	Null	Null	False	False
2	Null	array()	False	False
3	‘Texto de Prueba’	Null	False	False
4	‘Texto de Prueba’	array(‘Antonio’,’Pepe’)	False	False
5	‘Texto de Prueba’	array()	True	True
6	‘Este texto #!valor!# es para hacer una prueba unitaria’	array(‘presidente del tribunal’)	True	True

Tabla 5 Caso de prueba aplicado al algoritmo buildDocument del componente Visor de documentos

En las figuras 27 y 28 se muestra la aplicación de estas pruebas a dicho algoritmo con los juegos de datos de la tabla anterior.

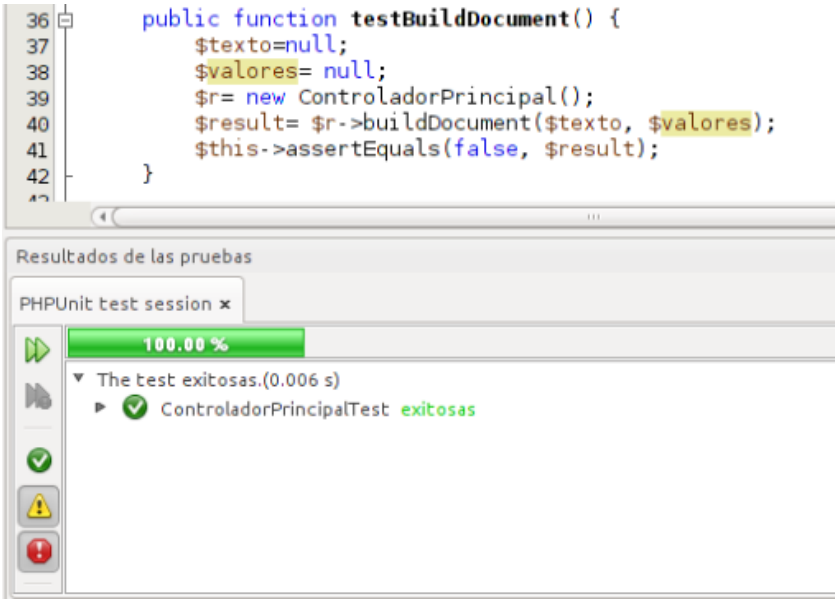


Figura 27 Resultado de la aplicación del juego de datos #1 del caso de prueba anterior

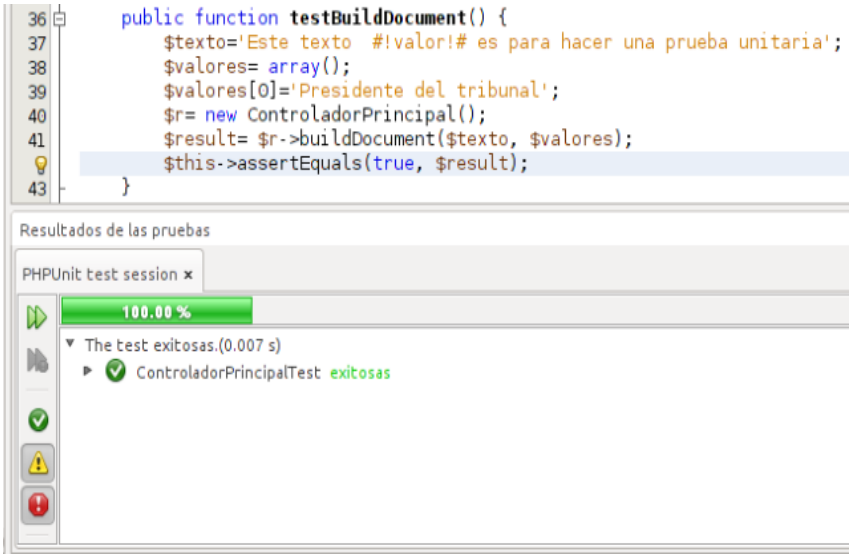


Figura 28 Resultado de la aplicación del juego de datos #6 del caso de prueba anterior

Como resultado de la aplicación de todos los casos de pruebas se detectaron un total de tres errores (igualdad esperada cuando los errores de precisión la hacen poco probable y dos condiciones

límites). Por lo que fue necesario corregir dichos errores y realizar una segunda iteración de pruebas, en la cual se evidenció el correcto funcionamiento de los algoritmos que estaban implicados en dichos errores como se puede apreciar en la figura 29.

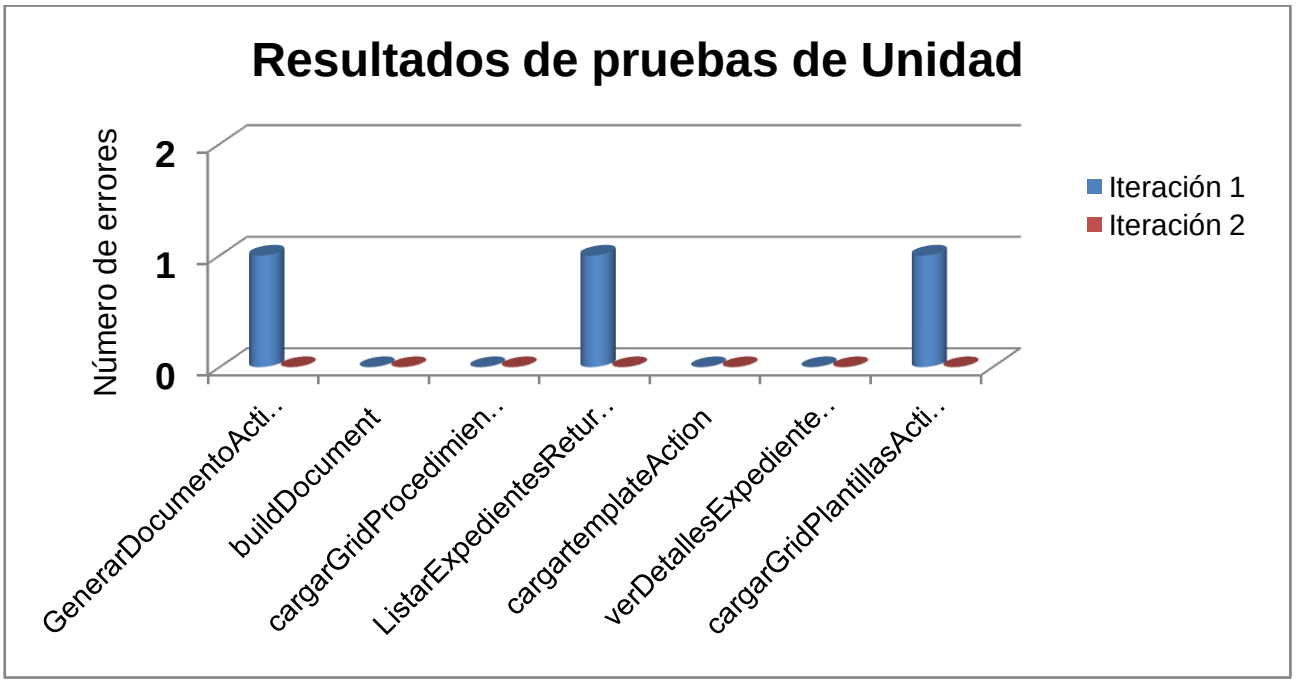


Figura 29 Resultado de la aplicación de las pruebas unitarias

3.6 Pruebas de caja negra

Las pruebas de caja negra también conocidas con sus varios nombres como pruebas funcionales, pruebas de caja opaca o pruebas de entrada/salida se llevan a cabo sobre la interfaz del software, obviando el comportamiento interno y la estructura del programa y su objetivo principal es la detección de errores mediante el uso de casos de prueba. Las pruebas de caja negra pretenden encontrar estos tipos de errores:

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

3.6.1 Ejecución de las pruebas de caja negra

Entre los diferentes métodos de pruebas de caja negra que existen se empleó la prueba de Partición de Equivalencia. La ***partición equivalente*** es un método de prueba de caja negra que divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba (30 pág. 296).

La partición equivalente se dirige a la definición de casos de prueba que descubran clases de errores, reduciendo así el número total de casos de prueba que hay que desarrollar. Una ***clase de equivalencia*** representa un conjunto de estados válidos o no válidos para condiciones de entrada. El diseño de los casos de prueba realizados se basa en una evaluación de las clases de equivalencia para una condición de entrada. Para el diseño de las clases de equivalencia se siguieron las siguientes directrices:

- Si una condición de entrada especifica un ***rango,*** y existen motivos para pensar que pueden surgir errores se define una clase de equivalencia válida y dos no válidas.
- Si una condición de entrada requiere un valor específico, se define una clase de equivalencia válida y dos no válidas.
- Si una condición de entrada es lógica, se define una clase de equivalencia válida y una no válida.
- Si una condición de entrada especifica un miembro de un conjunto, y existen motivos para pensar que el programa puede tratar diferente a los distintos miembros del conjunto entonces se crea una clase de equivalencia válida y una no válida (30 pág. 297).

Después de tener las clases válidas y no válidas identificadas se procedió a la elaboración de los casos de pruebas para cada uno de los componentes. Para la elaboración de los mismos se tuvieron en cuenta los siguientes aspectos:

- Construir un nuevo caso de prueba que cubra el mayor número de clases de equivalencias válidas como sea posible hasta que todas las clases de equivalencia válidas sean cubiertas por los casos de prueba.
- Elaborar un nuevo caso de prueba que tenga en cuenta una única clase de equivalencia inválida hasta que todas las clases de equivalencia inválidas sean cubiertas por casos de prueba.

Los datos de los casos de prueba correspondientes a cada uno de los componentes se entregan como uno de los artefactos generados de la presente investigación. Las pruebas fueron aplicadas por el equipo de calidad del PTPC fase 1 y durante la ejecución de las mismas se realizaron tres

iteraciones a cada uno de los componentes. Entre los errores detectados se encuentran errores ortográficos, de validación de interfaces y de carga de datos, los cuales fueron corregidos y en una tercera iteración no se detectó ninguno. Los resultados de cada una de estas iteraciones se aprecian en la siguiente gráfica.

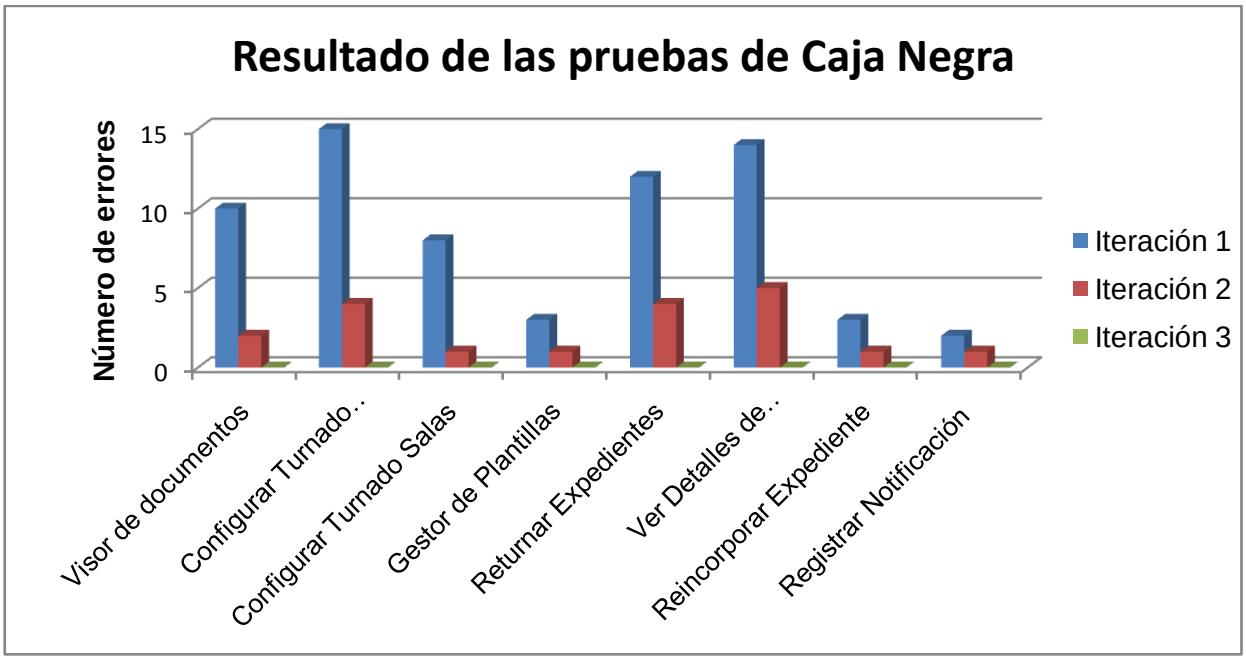


Figura 30 Cantidad de no conformidades detectadas a cada componente por iteración

3.7 Pruebas de Integración

En muchas ocasiones después de realizar las pruebas a los componentes, es válido preguntarse: “si todos los componentes funcionan bien independientemente, ¿Por qué no funcionan todos juntos?”. Pues la principal causa es agruparlos todos de una vez (interacción), debido a que muchas veces se pueden perder los datos de una interfaz, un componente puede tener un efecto adverso sobre los demás, las funciones internas de un componente pueden modificar datos globales que necesiten otros, en fin es muy grande la lista de riesgos que puede surgir al integrar un conjunto de componentes simultáneamente. De aquí la necesidad realizar una integración incremental, probando pequeños segmentos del programa en los que los errores son más fáciles de probar y corregir, además es importante utilizar una correcta estrategia para llevar a cabo la integración de los componentes de un sistema.

A través de los años han surgido varias estrategias para la integración de componentes con el objetivo de erradicar la mayor parte de los riesgos que pueden surgir durante el ensamblaje de

componentes a un sistema de software. Algunas de las estrategias más conocidas son la Integración descendente (Top-Down) y la Integración ascendente (Bottom-Up) (30 pág. 312). Para la integración de los componentes en el PTPC fase 1 se empleó la estrategia Ascendente, esta empieza las pruebas desde los componentes atómicos, es decir, probando los componentes de los niveles más bajos en la estructura del SIT. Al integrarse los componentes de abajo hacia arriba, el proceso requerido de los componentes subordinados siempre está disponible y se elimina la necesidad de resguardos.

Para lograr la integración ascendente de los componentes desarrollados se siguieron los siguientes pasos:

- Se combinan los componentes de bajo nivel en grupos (a veces denominados construcciones) que realicen una sub-función específica del software.
- Se escribe un controlador (un programa de control de la prueba) para coordinar la entrada y la salida de los casos de prueba.
- Se prueba el grupo.
- Se eliminan los controladores y se combinan los grupos moviéndose hacia arriba escalando la estructura del programa.

La integración sigue el esquema de la figura 31:

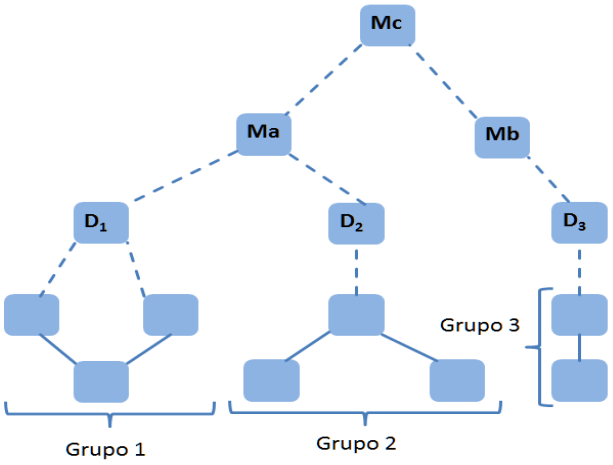


Figura 31 Integración ascendente

Se combinan los componentes para formar los grupos 1,2 y 3. Cada uno de los grupos se somete a pruebas mediante un controlador. Los componentes de los grupos 1 y 2 son subordinados de **Ma**, los controladores **D_1** y **D_2** se eliminan y los grupos interaccionan directamente con **Ma.** De forma similar se procede con la rama derecha del árbol**,** posteriormente se integra **Ma** y **Mb** con **Mc** y así

sucesivamente. A medida que la integración avanza hacia arriba, va disminuyendo la necesidad de controladores de prueba diferentes (30 pág. 314).

3.8 Conclusiones

Con la realización de este capítulo se evidenció la importancia y la necesidad de la realización de diferentes tipos de pruebas de software, pues ayudaron a descubrir errores una vez que se terminó el desarrollo de los componentes. Se hicieron pruebas de caja negra, las cuales arrojaron resultados satisfactorios, al encontrar un total de 7 no conformidades entre errores de validación de campos de entrada y errores ortográficos en la interfaz de algunos componentes, los cuales fueron corregidos eficazmente para lograr la posterior integración de los componentes a cada uno de los módulos del SIT. También se aplicaron métricas de calidad a las diferentes clases permitiendo evaluar algunos atributos de calidad como Reutilización, Acoplamiento, Complejidad de mantenimiento, entre otros, arrojando resultados satisfactorios en cada uno de los atributos medidos. Además dichas métricas contribuyeron a la validación del diseño realizado para la construcción de cada uno de los componentes del PCS. Durante la construcción de los componentes también se fueron realizando pruebas de integración, vitales para garantizar el correcto funcionamiento de cada uno de ellos al ser integrados a cada uno de los subsistemas y además se realizaron pruebas de caja blanca a algunos algoritmos implícitos en los componentes.

Conclusiones generales

La realización de la presente investigación propició un estudio de los diversos modelos de componentes que existen en el mundo, arrojando como resultado que ninguno satisface las necesidades existentes para el desarrollo del PCS necesario en el PTPC, lo que trajo consigo a la utilización de un modelo de desarrollo de componentes sustentado por el marco de trabajo Sauxe en conjunto al paradigma de Programación Orientado a Objetos y las ventajas que ofrece el lenguaje de programación PHP.

El empleo del modelo anteriormente mencionado, evidenció la completa realización de cada uno de los componentes necesarios (Visualizar documentos, Configurar turnado de expedientes, Configurar turnado de expedientes por salas, Gestionar plantillas, Ver detalles de expedientes, Returnar expedientes, Registrar notificación y Reincorporar expedientes), logrando alcanzar así el objetivo general, brindando de esta manera una alternativa eficiente para economizar el empleo de recursos humanos y materiales, pues la utilización de este paquete de componentes posibilita la reutilización de la implementación del funcionamiento de todos los actos procesales identificados como comunes en el PTPC.

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

Recomendaciones

Una vez terminado el paquete de componentes de software se recomienda:

- Realizar mejoras a los componentes en función de las nuevas versiones de las tecnologías empleadas, pues en ocasiones se producen nuevas versiones de las herramientas con el objetivo de eliminar brechas de seguridad existentes en versiones anteriores.
- Incluir el paquete de componentes de software en un repositorio de componentes del centro CEGEL con el objetivo de garantizar su posterior utilización en otros proyectos y que sean optimizados por la comunidad de desarrolladores de la universidad.
- Realizar pruebas unitarias a todos los algoritmos correspondientes a los componentes.

Bibliografía

1. *Derecho informático 2a. ed.* **Téllez, Julio.** México : Mc. Graw-Hill, 1996.

2. **Microsoft.** *The Component Object Model Specification,Report Vol. 99.* s.l. : Redmond, 1996.

3. **S. Pressman, Roger.** *Ingeniería de Software. Un enfoque práctico.* España : Mc Graw Hill, 2005.

4. **Brown, A.W., y K.C. Wallnau.** *Engineering of Component Based Systems, Component-Based Software.* s.l. : IEEE Computer Society Press, 1996.

5. **Szyperski, C.** *Component Software—Beyond Object-Oriented Programming.* s.l. : Addison-Wesley, 1998.

6. **D'Souza, D and A. C. Wills.** *Objects, Components and Frameworks: The Catalysis.* s.l. : Addison-Wesley, 1998.

7. **Larsson, Ivica Crnkovic and Magnus.** *Building Reliable Component-Based.* Boston - London : ARTECH HOUSE, INC, 2002. 1-58053-327-2.

8. **Sun Microsystems,JavaBeans 1.01 Specification.** http://java.sun.com. [En línea] http://java.sun.com/beans.

9. **Santana de Almeida, Eduardo, y otros.** *Component Reuse in Software Engineering.* s.l. : CESAR E-Books.

10. **Eddon, G y Eddon, H. .** *Inside COM+ Base Services.* s.l. : Microsoft Press, 2000.

11. **Oberleitner, J y Gschwind, T.** *The Vienna Component Framework - Enabling Composition Across Component Models. In: 25th International Conference on Software Engineering (ICSE'03).* s.l. : IEEE Press, Mayo, 2003.

12. **Sun Microsystems.** *Enterprise JavaBeans Specification, Version 2.0.* Abril, 2001.

13. **Microsoft Corporation.** *The Component Object Model Specification.* 1995.

14. *Software Engineering with Reusable Components.* **Sametinger, J.** s.l. : Springer-Verlag, 1997.

15. *Distributed Systems: Principles and Paradigms.* **Tanenbaum, A.S.** s.l. : Prentice Hall, 2002.

16. **Alvaro, A, Almeida, E.S y Meira.** *Software Component Certification: A Survey.In: 31st IEEE EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), Component-Based Software Engineering Track. Portugal.* Portugal : IEEE Press, 2005.

17. *Recommended Practice for Software Architecture Descriptions of Software Intensive Systems.* **IEEE.** s.l. : IEEE Presss, 2000.

18. **Symfony.** Open-Source PHP Web marco de trabajo. [En línea] [Citado el: 15 de 05 de 2012.] http://www.symfony-project.org/.

19. **Ing. Oiner Gómez Baryolo, Ing. Yoandry Morejón Borbón , Ing. Darien García Tejo.** *ARQUITECTURA TECNOLÓGICA PARA EL DESARROLLO DE SOFTWARE.* La Habana : s.n., 2008.

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

21. **Frederick, S., Ramsay, Colin y Blades, Steve 'Cutter'.** *Learning Ext JS.* 2008.

22. **Smith, L.** *Introduction to the Doctrine Object Relational Mapper.* Abril,2009.

23. **Larman, Carig.** *UML y Patrones.Introducción al análisis y diseño orientado a objetos.* s.l. : Prentice Hall.

24. **Pilgrim, Mark.** *HTML5 Up and Running.* s.l. : O'Reilly, August 2010.

25. **PostgreSQL, E.e.d.d.d.** *Manual del usuario de PostgreSQL. .* 2008.

26. *Programación web II.* **Santiago.** s.l. : I.S.C.B.M., noviembre 2009.

27. **RapidSVN.** RapidSVN. *RapidSVN.* [En línea] 9 de 11 de 2011. [Citado el: 12 de 05 de 2012.] http://www.rapidsvn.org/.

28. **Morín, Arley Triana.** DesarrolladorSenior. [En línea] 2009. [Citado el: 29 de marzo de 2012.] http://desarrolladorsenior.blogspot.com/2009/09/el-patron-de-diseno-singleton-esta.html.

29. *Desarrollo orientado a objetos con UML.* **Xavier Ferré Grau, María Isabel Sánchez Segura (Facultad de Informática–UPM).** 2004.

30. **S. Pressman, Roger.** *Ingeniería de Software. Un enfoque práctico.* España : Mc Graw Hill, 2005.

31. **Alvaro, A, Almeida, E.S y Meira.** *Towards a Software Component Quality Model In: The 31st IEEE EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA).* Portugal : s.n., 2005.

32. **Alvaro, A, Almeida, E.S y Meira, S.R.L.** *A Component Quality Model for Component Quality Assurance.* Brazil : s.n., 2005.

33. **Gutiérrez, Alex Gómez.** Diseño e Implementación de una solución informática para el proceso Administrativo en los Tribunales Provinciales Cubanos. La Habana : s.n., 2011.