Universidad de las Ciencias Informáticas

Facultad 3

Desarrollo de la implementación del subsistema Penal hasta la Sentencia del proyecto Tribunales Populares Cubanos

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

Autores: Félix David Ochoa Martínez

Doannis Delfino Montes de Oca

Tutor: Ing. Darián González Ochoa

Cotutor: Ing. Joan Jon Iglesias

La Habana, Cuba

2012

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 2012.

_____	_____
Félix David Ochoa Martínez	Doannis Delfino Montes de Oca
Autor	Autor
_____	_____
Ing. Darián González Ochoa	Ing. Joan Jon Iglesias
Tutor	Cotutor

Dedicatoria

A mis padres por ser los mejores padres del mundo, por haberse sacrificado todos estos años por mí, por ser mi ejemplo para todo en la vida y porque cada logro mío es un logro de ellos también.

A mi hermanita porque a pesar del tiempo que estamos lejos uno del otro siempre ha sido mejor hermana que se puede tener.

A mi abuelo Ricardo que aunque ya no está con nosotros se que estaría orgulloso de verme graduado y ser el primer ingeniero de la familia.

Doannis

Le dedico esta Tesis a mi Mamá, a mi Papá y a mi abuela los cuales han puesto toda su fe en mí y me han guiado hasta aquí.

Félix

Agradecimientos

A mis padres por apoyarme siempre, por aceptar mis decisiones en todo momento, por confiar en mí y tener la certeza de que yo si podía aun cuando había perdido la confianza en mí mismo, por darme todo el amor que se le puede dar a un hijo y por educarme como lo han hecho.

A mi mamá por ser mi razón de ser, por tolerar mis malos ratos y sobrellevarme siempre.

A mi papá por ser mi ejemplo en la vida, por hacer de mí un mejor revolucionario, por inculcarme que la familia es lo más importante y demostrarme que se pueden cometer errores en la vida pero se puede seguir adelante y ser mejores cada día.

A mi hermanita porque siempre me ha visto como un ejemplo a seguir y he hecho mi mayor esfuerzo para no defraudarla.

A mi familia por ayudarme y apoyarme siempre.

A Ania por ser la mejor amiga que he tenido en la universidad, por demostrarme en todo momento su amistad y cariño incondicional, por ser mi pareja de baile en muchas fiestas, compartir conmigo momentos de estudio y sacrificio, por aconsejarme y transmitirme su optimismo ante la vida.

A Yoslenys por ser mi amigo y hermano todos estos años, por soportarme siempre que no ha sido una tarea fácil.

A mis amigos Lisandra, Rances, Irmel y Reinier García que son parte de mi familia y han estado presentes en los buenos y malos momentos.

A todos mis compañeros de aula con los que he compartido los mejores momentos de la universidad, todos han sido especiales pero no puedo dejar de mencionar a mi gran amigo Iosmany Rodríguez Lorenzo.

Doannis

Agradecimientos

Quiero darle las gracias a mi mama por darme la vida, por luchar por mí, por siempre estar presente en los buenos y malos momentos, por cuidarme, enseñarme y amarme sin condición, por ser la hermosa madre y persona que es, quiero darte las gracias por guiarme a cada paso que doy y por ser el alma y alegría de nuestro hogar.

Quiero darle las gracias también a mi papa que siempre ha sido y será como un faro de luz. Una guía para mi , gracias por tu ayuda ,por tus consejos, por proporcionarme seguridad cuando me hace falta , por enseñarme con tu ejemplo como ser un hombre de bien, trabajador, preocupado, humano, serio ,carismático, estudioso, responsables y un excelente padre. Espero algún día tener algunas de estas cualidades que te definen. Gracias a ustedes dos por traerme al mundo y poder estar hoy aquí, gracias por ser como son y poder decir orgulloso esos son mis padres.

Gracias a mi abuela Concha por cuidarme, atenderme, educarme y quererme como solo ella sabe hacer.

Gracias a mi abuela Carmen por darme todo su amor y hacer lo imposible por hacerme feliz, siempre vivirás en mi corazón.

Gracias a todos los que han hecho mi paso por la UCI una experiencia inolvidable. Gracias a todos mis amigos y compañeros de aula con los que he vivido estos maravillosos 5 años de mi vida. Con los cuales comparto una historia, un recuerdo que quedara siempre grabada en mi corazón.

Gracias Alayna y Monica por soportarme, ayudarme, aconsejarme, compartir fiestas, estudios y preocupaciones. He tenido una gran suerte de contar con ustedes todos estos años. Gracias a mi novia Dailen por su amor, cariño y amistad, por hacer de este último curso un año especial y único. Gracias por dejarme entrar en tu vida y regalarme un pedacito de tu corazón. Gracias por siempre estar dispuesta a perdonar, reír, llorar y amar y demostrarme que personas como tú aun se pueden encontrar.

Espero que el fin de la UCI no sea el fin de nuestra amistad. A todos Gracias.

Félix

RESUMEN

Ante la necesidad de desarrollar un sistema para los tribunales cubanos, en la Universidad de las Ciencias Informáticas (UCI) se está implementando una solución informática para informatizar el sector jurídico en nuestro país. Esto se está llevando a cabo en el proyecto Tribunales Populares Cubanos (TPC), el cual se encarga de la informatización de los diferentes procesos judiciales que se llevan a cabo en los tribunales. La solución cuenta con varios subsistemas o módulos, uno de ellos es el Penal.

En la presente investigación se realiza una descripción de las principales herramientas y tecnologías usadas actualmente para el desarrollo de software, justificando la utilización de cada una de ellas. Se exponen además los artefactos generados durante la implementación del sistema. Al finalizar se muestran los resultados obtenidos al realizar pruebas al sistema desarrollado durante la etapa de implementación del mismo haciendo uso de métricas mundialmente usadas así como pruebas de caja blanca y caja negra para comprobar la calidad del software.

Índice

Índice

Índice

Índice de figuras

Índice

INTRODUCCIÓN

El uso de las Tecnologías de la Información y las Comunicaciones (TIC) forma parte de la vida diaria de la sociedad. El empleo de las computadoras y las nuevas tecnologías se ha hecho indispensable para el desarrollo de muchas actividades cotidianas.

A nivel mundial el empleo de estas tecnologías se ha visto vinculado a todos los sectores para mejorar el desarrollo de la sociedad. El sector judicial es uno de los que se ha visto beneficiado con la informatización de la mayoría de los procesos judiciales. Aplicada al mismo está la informática jurídica que no es más que una ciencia que forma parte de la informática que se aplica sobre el derecho, de manera que se dé el tratamiento lógico y automático de la información legal **(Quintero, 2002)**. Con la informatización de este sector se busca acelerar y agilizar los procesos que se llevan a cabo en estos para poder brindar un servicio con mayor calidad y eficiencia.

A nivel internacional existen sistemas en el ámbito jurídico que digitalizan algunos de los procesos de un tribunal, pero ninguno de ellos satisface las necesidades de los usuarios ni cumplen con la seguridad que este tipo de aplicaciones debe tener. Además no dan respuesta a las necesidades de los tribunales cubanos e incumplen con las leyes de los mismos por lo que en nuestro país, se han hecho varios intentos para informatizar este sector pero no se ha logrado un sistema que cumpla con las condiciones para gestionar toda la información con la que se trabaja en dichos tribunales.

La misión de los tribunales populares cubanos es mantener el respeto y la garantía de los derechos ciudadanos, así como la proporcionalidad entre el delito y la sanción ya que son principios que rigen la administración de justicia en nuestro país. **(Sáez, 2008)**

Ante la necesidad de desarrollar un sistema para los Tribunales Cubanos, en la Universidad de las Ciencias Informáticas (UCI) se está desarrollando una solución informática para informatizar el sector jurídico en nuestro país, tarea que se lleva a cabo en el proyecto (TPC) perteneciente al Centro de Gobierno Electrónico (CEGEL), en el que se vinculan profesores y estudiantes .Este proyecto es el encargado de la informatización los diferentes procesos judiciales que se llevan a cabo en los Tribunales.

Introducción

La solución informática en desarrollo cuenta con varios subsistemas o módulos, uno de ellos es Penal, el cual se encarga de dar solución a los conflictos que se originan por la comisión de delitos y realizar todos los actos procesales, formas y procedimientos que se hacen en el Tribunal para dictar sentencia y ejecutarla conforme dispone la Ley de Procedimiento Penal[1]. Esta investigación se desarrolla para el proceso ordinario de la instancia municipal.

La **problemática** actual en la sección Penal de los Tribunales Populares Cubanos está dada porque todos los procesos que se realizan se hacen de forma manual teniendo provocando que se cometan errores a la hora de elaborar los documentos en formato duro, existiendo gran dependencia del papel, lo cual provoca que en ocasiones la información sea reiterada, al ser registrada en diferentes libros, y que la tramitación de los procesos se dilate. También al existir una forma de trabajo poco centralizada, pueden darse casos de violaciones de los términos fijados por la ley. Actualmente el flujo de documentos e información que se maneja en un tribunal es muy elevado y al estar estos en formato duro se pueden deteriorar y perder con el paso del tiempo por lo que se hace necesario informatizar todo el trabajo en los tribunales.

La introducción de un sistema informático en los Tribunales Populares Cubanos traería como beneficios que se agilicen los trámites y procesos en las diferentes instancias, específicamente en la instancia municipal que es en la que se basa esta investigación. Se evitaría que se cometan errores al registrar los expedientes porque el sistema se encargaría de no repetir los ya existentes y al estar toda la información de forma digital se conserva mejor y no se perdería con el paso del tiempo por problemas de deterioro, además este daría la posibilidad de integrarse en un futuro con otros sistemas legales.

Parte del equipo de desarrollo llevó a cabo el análisis y diseño en los tribunales donde se realizó un levantamiento de requisitos con la ayuda del cliente para luego ser aprobados y definir la arquitectura. Durante la fase de implementación se generan artefactos necesarios que deben cumplir con los requisitos definidos, algunos de estos artefactos son el diagrama de clases, de componentes, de secuencia y de despliegue.

[1] Ley No. 5 de Procedimiento Penal, de fecha 13 de agosto de 1977.

Introducción

Se plantea el siguiente **problema a resolver** ¿Cuál es la especificación de los artefactos necesarios para la informatización de los requisitos identificados para el proceso ordinario de la materia penal?

Se define como **objeto de estudio** de la investigación el proceso de desarrollo de software en los sistemas para la gestión judicial.

Se determina como **objetivo general** realizar el diseño y la implementación de los requisitos identificados para el proceso Ordinario de la materia Penal de los Tribunales Municipales Cubanos para obtener un producto funcional.

Se identifica como **campo de acción** de la investigación la implementación como fase específica dentro del proceso de desarrollo de software en la gestión de la materia Penal en los Tribunales Populares Cubanos.

Se establece la siguiente **idea a defender:** con la especificación de los artefactos necesarios se informatizan los requisitos identificados para el proceso ordinario en la materia penal.

Los **objetivos específicos** que se desarrollan para dar solución a la problemática planteada son:

- ✓ Definir el marco teórico-referencial para fundamentar la investigación realizada.
- ✓ Generar los artefactos necesarios para obtener un producto funcional del subsistema penal del Sistema de Informatización de Tribunales.
- ✓ Realizar la implementación del sistema.
- ✓ Validar la solución desarrollada para garantizar la calidad del sistema.

Para dar cumplimiento a los objetivos propuestos se definen las siguientes **tereas de la investigación:**

1. Análisis de los subprocesos de la Materia Penal para desarrollar la aplicación.
2. Análisis de la arquitectura definida para la implementación de la solución informática.
3. Análisis de los Patrones de Diseño seleccionados para la propuesta de solución.
4. Realización de un análisis sobre las tecnologías y herramientas definidas para el desarrollo de la solución propuesta.

5. Realización de Diagramas de Clases, de Componentes y de Despliegue para la solución.
6. Implementación de los casos de uso necesarios para el desarrollo de la aplicación informática.
7. Validación de la solución haciendo uso de los casos de prueba definidos.

Los métodos científicos son la forma de abordar la realidad, de estudiar la naturaleza, la sociedad y el pensamiento, con el propósito de descubrir su esencia y sus relaciones. Para dar cumplimiento a las tareas se utilizarán los siguientes métodos:

Métodos teóricos: Permiten estudiar las características del objeto de investigación que no son observables directamente.

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

Analítico-sintético: Permite realizar un estudio teórico de la investigación y un análisis previo sobre el funcionamiento de la materia Penal y extraer los elementos más importantes relacionados con el objeto de estudio.

Modelación: Permite la creación de modelos que representan abstracciones con el objetivo de comprender el funcionamiento de la materia Penal que se lleva a cabo en los Tribunales Populares Cubanos.

El presente trabajo consta de Introducción, 3 Capítulos, Conclusiones, Bibliografía y Anexos.
Los capítulos abordan los temas fundamentales distribuidos de la siguiente manera:

Capítulo 1: Fundamentación Teórica. En este capítulo se recoge toda la fundamentación teórica que sustenta el desarrollo de la investigación para el posterior desarrollo del sistema propuesto. Se analiza sobre las diferentes metodologías, las herramientas y los patrones de diseño definidos para el desarrollo del sistema y ver cómo contribuyen al cumplimiento de los requisitos identificados.

Capítulo 2: Propuesta de solución. En este capítulo se fundamenta la utilización de las diferentes herramientas y la metodología seleccionada así como el lenguaje de programación y los patrones de diseño para el desarrollo de la aplicación. También se realiza un análisis de la arquitectura que guiará la construcción del software.

Capítulo 3: Análisis de resultados. En este capítulo se muestran los procedimientos y métodos utilizados para evaluar la calidad de los resultados obtenidos con el desarrollo de este trabajo, haciendo uso de las métricas de medición de la calidad del diseño y la implementación del software. Es de gran importancia el análisis de los resultados obtenidos en cada etapa del proceso de desarrollo de software ya que esto ayuda a minimizar los errores en la construcción de un producto con mayor calidad que cumpla con los intereses del cliente.

CAPÍTULO I FUNDAMENTACIÓN TEÓRICA

Introducción

En este capítulo se fundamenta la utilización de la metodología y las diferentes herramientas utilizadas y que fueron previamente definidas por el equipo de arquitectura. Se realiza un análisis del lenguaje de programación y los patrones de diseño definidos para el desarrollo de la aplicación y se puede ver cómo estas herramientas contribuyen a cumplir con los requisitos identificados por el equipo de analistas.

1.1 Aplicaciones Web

Las aplicaciones web son soluciones informáticas que permiten interactuar con la información y a las cuales se puede acceder a través de una conexión a internet o intranet, sin necesidad de instalarlas previamente en la computadora del lado del cliente; solamente se necesita contar con un navegador web. **(E-lemental, 2010)** Se decidió desarrollar una aplicación web para los tribunales que permitirá alojar un gran número de transacciones y se podrá acceder a ella desde cualquier parte del país sin necesidad de instalar todo el software necesario para su funcionamiento, esto permitirá el ahorro de recursos y se ganará en seguridad.

1.1.2 Ventajas de las aplicaciones web del lado del cliente

- ✓ Al ejecutarse a través de los navegadores, se puede acceder a ellas a través de cualquier computadora en la que se cuente con internet o se encuentre conectada a una intranet.
- ✓ Desde el punto de vista del usuario, no es necesario instalar la aplicación en la computadora para poder trabajar con la misma, por lo que no hay que preocuparse por licencias o actualizaciones.
- ✓ Las actualizaciones las realiza el desarrollador en el servidor y por ende siempre se trabajará con la última versión disponible.
- ✓ No existirán problemas de incompatibilidades con los sistemas operativos porque todo se maneja en el navegador.
- ✓ Consumen pocos recursos de hardware porque las tareas se realizan en el servidor. **(E-lemental, 2010)**

1.2 Metodologías de desarrollo de software

Surgen ante la necesidad de utilizar una serie de procedimientos, técnicas, herramientas y soporte documental a la hora de desarrollar un producto. Una metodología de desarrollo de software es un conjunto de pasos y procedimientos que deben seguirse para desarrollar software. **(García, et al., 2009-2010)**En el ciclo de vida del software se deben completar una serie de tareas para obtener un producto de software y es necesario saber cuándo se puede dar por concluida una tarea, quién debe realizarla, qué tareas preceden o anteceden a una dada y qué documentación se utilizará para llevar a cabo esa tarea.

1.2.1 Metodologías tradicionales

Las metodologías con mayor énfasis en la planificación y control del proyecto, en especificación precisa de requisitos y modelado, reciben el nombre de Metodologías Tradicionales o Pesadas. Se centran en la definición detallada de los procesos y tareas a realizar, herramientas a utilizar, y requiere una extensa documentación, ya que pretende prever todo de antemano **(Pérez, et al., 2008)**

Las metodologías tradicionales se focalizan en documentación, planificación y procesos. Centran su atención en llevar una documentación exhaustiva de todo el proyecto y centran su atención en cumplir con un plan de proyecto, definido todo esto, en la fase inicial del desarrollo del proyecto. **(J. Solís, y otros)**

Imponen una disciplina de trabajo sobre el proceso de desarrollo del software, con el fin de conseguir un software más eficiente. Para ello, se hace énfasis en la planificación total de todo el trabajo a realizar y una vez que está todo detallado, comienza el ciclo de desarrollo del software. Se centran especialmente en el control del proceso, mediante una rigurosa definición de roles, actividades, artefactos, herramientas y notaciones para el modelado y documentación detallada.

Entre las metodologías tradicionales están:

- ✓ RUP (Rational Unified Procces)
- ✓ MSF (Microsoft Solution Framework)
- ✓ Win-Win Spiral Model
- ✓ Iconix **(Acuña, 2009)**

1.2.1.1 Proceso unificado de desarrollo

Rational Unified Procces (RUP por sus siglas en inglés) es un proceso de la ingeniería de software que define quién, cómo, cuándo y qué debe hacerse en el desarrollo de un proyecto. En el ciclo de vida de desarrollo de un proyecto toma en cuenta las siguientes buenas prácticas, utilizándolas en el modelo de desarrollo de software:

- ✓ Desarrollo de software en forma iterativa.
- ✓ Manejo de requerimientos.
- ✓ Modela el software visualmente.
- ✓ Verifica la calidad del software.
- ✓ Controla los cambios.

Su ciclo de vida se caracteriza por:

- ✓ **Dirigido por casos de usos:** Los casos de uso reflejan lo que los usuarios futuros necesitan y desean, lo cual se capta cuando se modela el negocio y se representa a través de los requerimientos. A partir de aquí los casos de uso guían el proceso de desarrollo ya que los modelos que se obtienen, como resultado de los diferentes flujos de trabajo, representan la realización de los casos de uso.
- ✓ **Centrado en la arquitectura:** La arquitectura muestra la visión común del sistema completo en la que el equipo de proyecto y los usuarios deben estar de acuerdo, por lo que describe los elementos del modelo que son más importantes para su construcción, los cimientos del sistema que son necesarios como base para comprenderlo, desarrollarlo y producirlo económicamente.
- ✓ **Iterativo e incremental:** Este modelo plantea la implementación del proyecto a realizar en iteraciones, con lo cual se pueden definir objetivos por cumplir en cada iteración y así poder ir completando todo el proyecto. Una de las ventajas que tiene es que se tienen pequeños avances del proyectos que son entregables al cliente el cual puede probar mientras se está desarrollando otra iteración del proyecto, con lo que el proyecto va creciendo hasta completarlo en su totalidad.

El proceso unificado de desarrollo (RUP) ha sido la metodología definida para el desarrollo de la aplicación web para los Tribunales Populares Cubanos pues con esta metodología se genera

gran cantidad de documentación que es necesaria por la dinámica del proyecto y exigida por el cliente, pues el personal puede variar y se necesita tener documentado lo que se ha hecho anteriormente por otros miembros del proyecto. El equipo de desarrollo está familiarizado con esta metodología, y la introducción de una nueva produciría un atraso del cronograma de ejecución. Además RUP es una metodología altamente configurable que ha dado buenos resultados a lo largo de su utilización, y existe bastante documentación de la misma.

1.3 Paradigmas de programación

Un paradigma de programación es una propuesta tecnológica que trata de resolver uno o varios problemas claramente delimitados. La solución de estos problemas debe suponer consecuentemente un avance significativo en al menos un parámetro que afecte a la ingeniería de software. Un paradigma de programación está delimitado en el tiempo en cuanto a aceptación y uso ya que nuevos paradigmas aportan nuevas o mejores soluciones que la sustituyen parcial o totalmente.

Los paradigmas de programación son modelos que describen cómo diseñar e implementar programas. Diferentes paradigmas dan lugar a diferentes técnicas de programación, lo cual no implica que ellos sean contradictorios, sino que más bien unos tienden a complementar otros, por lo que la programación actual tiende a ser multiparadigma. Los paradigmas se asemejan en que se apoyan en un diseño basado en abstracciones que corresponden a elementos en el problema de programación y que la implementación debe ser una colección de módulos preferentemente reutilizables.

Dentro de los tipos de paradigmas está la Programación Orientada a Objetos (POO) que es la que se utiliza en el desarrollo de la aplicación.

1.3.1 Programación Orientada a Objetos

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

Capítulo I: Fundamentación Teórica

La programación orientada a objeto tiene como objetivos facilitar el desarrollo y comprensión de programas de gran porte y posibilitar la reutilización de código. Su principal preocupación durante el desarrollo de programas es determinar los objetos que representarán, de una forma más adecuada, los elementos presentes en un problema.

La programación orientada a objetos (POO) es una forma especial de programar, más cercana a cómo se expresarían las cosas en la vida real que otros tipos de programación. **(Alvarez, 2001)**

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

Objeto: Es el concepto fundamental de la POO. Está compuesto por estados. Los estados son propiedades del objeto y están representados por variables con valores únicos para cada objeto y que son llamadas variables de instancia. Los objetos representan cosas reales o abstractas del universo del problema a resolver y poseen un nombre que los identifica.

Métodos: Son representaciones de los comportamientos que el objeto es capaz de hacer. Son las funcionalidades asociadas a los objetos. Desde el punto de vista del comportamiento, es lo que el objeto puede hacer. Un método puede producir un cambio en las propiedades del objeto, o la generación de un "evento" con un nuevo mensaje para otro objeto del sistema.

Clases: Son declaraciones de objetos, también se podrían definir como abstracciones de objetos. Esto quiere decir que la definición de un objeto es la clase. Cuando se programa un objeto y se definen sus características y funcionalidades en realidad lo que se está haciendo es programar una clase. **(Álvarez, 2001)**

Encapsulamiento: Significa reunir a todos los elementos que pueden considerarse pertenecientes a una misma entidad, al mismo nivel de abstracción; esto permite aumentar la cohesión de los componentes del sistema. El código o datos de un objeto pueden estar ocultos para cualquier entidad externa a él de manera que sólo se puede cambiar mediante las operaciones definidas para ese objeto.

Herencia: Es la propiedad de crear nuevos objetos a partir de la definición de otros. La herencia permite crear estructuras jerárquicas de clases donde es posible la creación de sub-

clases que incluyan nuevas propiedades y atributos. Estas sub-clases admiten la definición de nuevos atributos, así como crear, modificar o inhabilitar propiedades. Los objetos heredan las propiedades y el comportamiento de todas las clases a las que pertenecen.

Polimorfismo: Se refiere a la capacidad para que varias clases derivadas de una antecesora utilicen un mismo método de forma diferente lo que indica la posibilidad de definir varias operaciones con el mismo nombre, diferenciándolas únicamente en los parámetros de entrada. Dependiendo del objeto que se introduzca como parámetro de entrada, se elegirá automáticamente cual de las operaciones se va a realizar.

Abstracción: El proceso de abstracción permite seleccionar las características relevantes dentro de un conjunto e identificar comportamientos comunes para definir nuevos tipos de entidades en el mundo real. La abstracción es clave en el proceso de análisis y diseño orientado a objetos, ya que mediante ella se puede llegar a armar un conjunto de clases que permitan modelar la realidad o el problema que se quiere atacar. La abstracción permite realizar generalizaciones, simplifica la realidad ignorando los detalles que no tienen relevancia en el universo del problema que se modela y se enfoca en las cosas comunes, pero permitiendo las variaciones.

La POO proporciona las siguientes ventajas:

Uniformidad: Ya que la representación de los objetos implica tanto el análisis como el diseño y la codificación de los mismos.

Comprensión: Tanto los datos que componen los objetos, como los procedimientos que los manipulan, están agrupados en clases, que se corresponden con las estructuras de información que el programa trata.

Flexibilidad: Al tener relacionados los procedimientos con los datos a tratar, cualquier cambio que se realice sobre ellos quedará reflejado automáticamente en cualquier lugar donde estos datos aparezcan.

Estabilidad: Permite un tratamiento diferenciado de aquellos objetos que permanecen constantes en el tiempo sobre aquellos que cambian con frecuencia permite aislar las partes del programa que permanecen inalterables en el tiempo.

Reusabilidad: La noción de objeto permite que programas que traten las mismas estructuras de información reutilicen las definiciones de objetos empleadas en otros programas e incluso los procedimientos que los manipulan. De esta forma, el desarrollo de un programa puede llegar a ser una simple combinación de objetos ya definidos donde estos están relacionados de una manera particular. **(Ceballos)**

1.4 Patrones de diseño

A la hora de desarrollar un software es importante el uso de patrones de diseño. "Una arquitectura orientada a objetos bien estructurada está llena de patrones. La calidad de un sistema orientado a objetos se mide por la atención que los diseñadores han prestado a las colaboraciones entre sus objetos. Los patrones conducen a arquitecturas más pequeñas, más simples y más comprensibles". **(Booch, 2000)**

Un patrón de diseño es una descripción de clases y objetos comunicándose entre sí adaptada para resolver un problema de diseño general en un contexto particular. **(Erich Gamma, 1995)**

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

Los patrones GRASP que se utilizan en la solución son:

- ✓ Bajo Acoplamiento
- ✓ Controlador

El patrón GOF que se utiliza en la solución es:

- ✓ Singleton.

Además otros de los patrones que se emplean son:

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

1.4.1 Patrón Inversión de Control (IoC)

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

1.4.2 Singleton

El patrón de diseño singleton (instancia única) está diseñado para restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto. Su intención consiste en garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella. Se implementa creando en la clase un método que crea una instancia del objeto sólo si todavía no existe alguna. Para asegurar que la clase no puede ser instanciada nuevamente se regula el alcance del constructor (con atributos como protegido o privado).La instrumentación del patrón puede ser delicada en programas con múltiples hilos de ejecución. Si dos hilos de ejecución intentan crear la instancia al mismo tiempo y esta no existe todavía, sólo uno de ellos debe lograr crear el objeto **(Alfonso, 2010)**

1.4.3 Modelo-Vista-Controlador (MVC)

El patrón conocido como Modelo-Vista-Controlador (MVC) separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes:

- ✓ **Modelo:** El modelo administra el comportamiento y los datos del dominio de aplicación, responde a requerimientos de información sobre su estado (usualmente formulados desde la vista) y responde a instrucciones de cambiar el estado (habitualmente desde el controlador).

- ✓ **Vista:** Maneja la visualización de la información y define la interfaz de usuario, HTML+CSS en el navegador, algunas veces es conocida como lógica de presentación
- ✓ **Controlador**: En este se encuentran las funcionalidades las cuales decidirán el comportamiento de la aplicación.

Este patrón desacopla el concepto de interfaz de usuario y lógica de negocio para aumentar la flexibilidad y modularidad del software, permitiendo que el código pueda ser reutilizado. Tanto la vista como el controlador dependen del modelo, el cual no depende de las otras clases. Esta separación permite construir y probar el modelo independientemente de la representación visual. **(Jurado)**

1.4.4 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 propone el diseño de clases que son más independientes, lo que reduce el impacto del cambio y facilita la reutilización en otros sistemas. La idea es tener las clases lo menos ligadas posibles de tal forma que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de clases, potenciando la reutilización, y disminuyendo la dependencia entre las clases **(Alcalá)**

1.4.5 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.

Se encarga de asignar la responsabilidad de controlar el flujo de eventos del sistema a clases específicas. El objeto controlador no será el que realice estas actividades, sino que las delegará en otras clases con las que mantiene un modelo de alta cohesión. **(Alcalá)**

Ventajas del uso de patrones

Proporcionan una estructura conocida por todos los programadores, de manera que la forma de trabajar no resulte distinta entre los mismos. Así la incorporación de un nuevo programador, no requerirá conocimiento de lo realizado anteriormente por otro. Permiten tener una estructura de código común a todos los proyectos que implemente una funcionalidad genérica. La utilización de patrones de diseño, permite ahorrar gran cantidad de tiempo en la construcción de software. El software construido es más fácil de comprender, mantener y extender. Facilitan la reusabilidad, extensibilidad y mantenimiento.

1.5 Métricas de software

Las métricas permiten medir de forma cuantitativa la calidad de los atributos internos del software. Esto permite al ingeniero evaluar la calidad durante el desarrollo del sistema. Las métricas son la maduración de una disciplina, que, según Pressman van a ayudar a la evaluación de los modelos de análisis y de diseño, en donde proporcionarán una indicación de la complejidad de diseños procedimentales y de código fuente, y ayudaran en el diseño de pruebas más efectivas. **(Desarrrollo Web)**

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

Las métricas están diseñadas para evaluar los siguientes atributos de calidad:

Responsabilidad: Consiste en la responsabilidad asignada a una clase en un marco de modelado de un dominio o concepto, de la problemática propuesta.

Complejidad de implementación: Consiste en el grado de dificultad que tiene implementado un diseño de clases determinado.

Reutilización: Consiste en el grado de reutilización presente en una clase o estructura de clase, dentro de un diseño de software.

Acoplamiento: Consiste en el grado de dependencia o interconexión de una clase o estructura de clase, con otras, está muy ligada a la característica de Reutilización.

Complejidad del mantenimiento: Consiste en el grado de esfuerzo necesario a realizar para desarrollar un arreglo, una mejora o una rectificación de algún error de un diseño de software. Puede influir indirecta, pero fuertemente en los costes y la planificación del proyecto.

Cantidad de pruebas: Consiste en el número o el grado de esfuerzo para realizar las pruebas de calidad del producto (componente, módulo, clase, conjunto de clases, etc.) diseñado.

Tamaño operacional de clase (TOC): Está dado por el número de métodos asignados a una clase y evalúa los siguientes atributos de calidad:

Responsabilidad: Un aumento del TOC implica un aumento de la responsabilidad asignada a la clase.

Complejidad de implementación: Un aumento del TOC implica un aumento de la complejidad de implementación de la clase.

Reutilización: Un aumento del TOC implica una disminución del grado de reutilización de la clase.

Relaciones entre clases (RC): Está dado por el número de relaciones de uso de una clase con otra y evalúa los siguientes atributos de calidad:

Acoplamiento: Un aumento del RC implica un aumento del Acoplamiento de la clase.

Complejidad de mantenimiento: Un aumento del RC implica un aumento de la complejidad del mantenimiento de la clase.

Reutilización: Un aumento del RC implica una disminución en el grado de reutilización de la clase.

Cantidad de pruebas: Un aumento del RC implica un aumento de la Cantidad de pruebas de unidad necesarias para probar una clase.

1.6 Pruebas de Caja Blanca o Estructurales

Conociendo el código y siguiendo su estructura lógica, se pueden diseñar pruebas destinadas a comprobar que el código hace correctamente lo que el diseño de bajo nivel indica y otras que demuestren que no se comporta adecuadamente ante determinadas situaciones.

Algunas técnicas de prueba de caja blanca son:

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

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

✓ **Prueba de bucles:** Es una técnica de prueba de caja blanca que se centra exclusivamente en la validez de las construcciones de bucles.

✓ **Prueba del camino básico:** Esta técnica permite obtener una medida de la complejidad lógica de un diseño y usar la misma como guía para la definición de un conjunto básico. La idea es derivar casos de prueba a partir de un conjunto dado de caminos independientes por los cuales puede circular el flujo de control. Para obtener dicho conjunto de caminos independientes se construye el grafo de flujo asociado y se calcula su complejidad ciclomática.

1.7 Pruebas de Caja Negra o Funcionales

Se parte de los requisitos funcionales, a muy alto nivel, para diseñar pruebas que se aplican sobre el sistema sin necesidad de conocer cómo está construido por dentro (caja negra). Las pruebas se aplican sobre el sistema empleando un determinado conjunto de datos de entrada y observando las salidas que se producen para determinar si la función se está desempeñando correctamente por el sistema bajo prueba. Las herramientas básicas son observar la funcionalidad y contrastar con la especificación.

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

Con este tipo de pruebas se intenta encontrar:

✓ Funciones incorrectas o ausentes.

✓ Errores de interfaz.

✓ Errores en estructuras de datos o en accesos a las bases de datos externas.

✓ Errores de rendimiento.

✓ Errores de inicialización y terminación.

1.8 Plataformas de desarrollo

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

ligada a una familia de lenguajes de programación o a una Interfaz de Programación de Aplicaciones (API del inglés Application Programming Interface)

1.9 Lenguajes de programación

Un lenguaje de programación es un idioma artificial diseñado para expresar computaciones que pueden ser llevadas a cabo por máquinas. Pueden usarse para crear programas que controlen el comportamiento físico y lógico de una máquina, para expresar algoritmos con precisión, o como modo de comunicación humana. Permite especificar de manera precisa sobre qué datos debe operar una computadora, cómo deben ser almacenados o transmitidos y qué acciones debe tomar bajo una variada gama de circunstancias. Todo esto, a través de un lenguaje que intenta estar relativamente próximo al lenguaje humano o natural. Una característica relevante de los lenguajes de programación es precisamente que más de un programador pueda usar un conjunto común de instrucciones que sean comprendidas entre ellos para realizar la construcción del programa de forma colaborativa.

1.9.1 Lenguajes del lado del cliente

Son aquellos lenguajes que son asimilados directamente por el navegador y no necesitan pre-tratamiento. Para el desarrollo de la aplicación se utiliza del lado del cliente JavaScript porque es interpretado por cualquier navegador y su uso viene definido por el marco de trabajo Sauxe aunque para un mejor funcionamiento de la aplicación se recomienda el acceso a través de un navegador Internet Explorer 6 o superior y Mozilla Firefox 3.0 o superior.

1.9.1.1 JavaScript

JavaScript es un lenguaje de programación interpretado, dialecto del estándar ECMAScript[2]. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Se utiliza principalmente en su forma del lado del cliente implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas. **(Flanagan, 2002)**

[2] ***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.*

Es un lenguaje de alto nivel, multiplataforma y no necesita compilación. Está basado en objetos y maneja la mayoría de los eventos que se pueden producir sobre la página web. La mayoría de los navegadores en sus últimas versiones interpretan el código JavaScript integrado dentro de las páginas web. **(Mikkonen, et al., 2007)**

1.9.2 Lenguajes del lado del servidor

Son aquellos lenguajes que se ejecutan por el propio servidor y son enviados al cliente en un formato claro para él.

1.9.2.1 PHP 5.3

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

Se usa PHP en el desarrollo de la aplicación para los Tribunales ya que aporta beneficios en cuanto a portabilidad y escalabilidad del sistema. El código fuente escrito en PHP es invisible al navegador y al cliente ya que es el servidor el que se encarga de ejecutar el código y enviar su resultado HTML[3] al navegador. Esto hace que la programación en PHP sea segura y confiable algo necesario para el desarrollo ya que en los Tribunales se trabaja con información confidencial y los datos tienen que viajar de manera segura por la red. Destaca su conectividad con PostgreSQL que es el gestor de base de datos que se utiliza en el desarrollo el cual tiene soporte para grandes volúmenes de datos y velocidad de procesamiento. Es libre, por lo que se presenta como una alternativa de fácil acceso para todos y más para nuestro país con el auge que está teniendo el uso de software libre. Tiene manejo de excepciones (desde PHP5) lo que es importante ya que las excepciones que se produzcan podrán ser tratadas mediante una llamada a un gestor de excepciones que informará al usuario del error a través de la interfaz gráfica y se registrará en un fichero de logs. PHP al ser un lenguaje flexible permite hacer uso del patrón de diseño Modelo Vista Controlador que determina una separación entre la lógica del negocio y la presentación de la aplicación, lo que permite el uso de servicios desde la capa

[3] HyperText Markup Language (lenguaje de marcado de hipertexto)

lógica del negocio ya sea por la capa de presentación del sistema o por otros sistemas que requieran el mismo servicio.

1.10 Entornos de desarrollo integrado

Un IDE o Integrated Development Environmen (IDE por sus siglas en inglés) es un entorno de programación que ha sido empaquetado como un programa de aplicación, o sea, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica. Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes.

Algunas características de los IDE

- ✓ Soporte para diversos lenguajes de programación
- ✓ Integración con Sistemas de Control de Versiones
- ✓ Extensiones y Componentes para el IDE
- ✓ Integración con marcos de trabajo populares
- ✓ Depurador
- ✓ Importar y Exportar proyectos
- ✓ Múltiples idiomas
- ✓ Manual de Usuarios y Ayuda

1.10.1 NetBeans 6.9

Es un programa que sirve como IDE que permite programar en distintos lenguajes además ofrece un excelente entorno para programar en PHP lo que es muy conveniente ya que es el lenguaje en el que se está desarrollando la aplicación. Tiene un excelente balance entre una interfaz con múltiples opciones y un aceptable completamiento de código. Es multiplataforma y se puede integrar a sistemas de control de versiones como es el Subversion[4] utilizado en el desarrollo.

El IDE NetBeans es una herramienta para programadores pensada para escribir, compilar, depurar y ejecutar programas. Está escrito en Java pero puede servir para cualquier otro

[4] Sistema de control de versiones que mantiene los registros de todos los cambios que se han realizado a los archivos de un software, lo que permite el trabajo de distintos desarrolladores en un mismo proyecto.

lenguaje de programación. Es un producto libre y gratuito sin restricciones de uso. **(Corporation, 2012)**

1.11 Servidor de aplicaciones

Proporciona servicios que soportan la ejecución y disponibilidad de las aplicaciones desplegadas. Aloja y ejecuta aplicaciones que responden a peticiones que llegan por la red desde equipos clientes o desde otras aplicaciones. **(Corporation, 2008)**

1.11.1 Servidor Apache 2.2.3

Apache es un servidor web potente que ofrece un servicio estable, es sencillo de mantener y configurar. Es uno de los mayores logros del software libre. **(Foundation, 2011)**
Se utiliza este servidor de aplicaciones para garantizar la alta disponibilidad manteniendo funcionando el sistema las 24 horas del día los 7 días de la semana. Para poder alcanzar esta característica es necesario el uso de técnicas de balanceo de carga y de recuperación ante fallos para evitar la caída del sistema por sobrecargas de datos o por el volumen de usuarios.

Es el servidor HTTP más usado. Apache es uno de los mejores servidores web, por su configurabilidad y robustez. Otra de sus ventajas es que se puede personalizar la respuesta ante los posibles errores que se den en el servidor, por supuesto se puede configurar para que ejecute un determinado script cuando ocurra un error marcado. Permite la creación de ficheros de log a medida del administrador, teniendo así el control máximo sobre lo que pasa en el servidor.

Al usar Apache como servidor de aplicaciones se garantiza la integridad de los datos ya que los cambios y configuraciones se hacen en las máquinas servidoras y las actualizaciones están garantizadas para todos los usuarios. Al usar este servidor se limita el tráfico de la red solamente a la capa de presentación mejorando así el funcionamiento de la aplicación.

1.12 Marco de trabajo

En el desarrollo de software, un marco de trabajo es una estructura conceptual y tecnológica de soporte definida, normalmente con artefactos o módulos de software concretos, en base a la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros programas para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

1.12.1 Mapeador de objeto relacional

Un ORM(Object Relation Mapper por sus siglas en inglés) permite convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional, es decir, las tablas de la base de datos pasan a ser clases y los registros objetos que se pueden manejar con facilidad. **(Mata, 2009)**

Ventajas que facilitan enormemente tareas comunes y de mantenimiento:
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.
Encapsulación: La capa ORM encapsula la lógica de los datos pudiendo hacer cambios que afectan a toda la aplicación únicamente modificando una función.
Portabilidad: Utilizar una capa de abstracción que 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 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 una Inyección SQL
Mantenimiento del código: Gracias al correcto ordenamiento de la capa de datos, modificar y mantener el código es una tarea sencilla **(Mata, 2009)**

1.12.1.1 Doctrine 1.2.1

Es un mapeador de objetos relacional (ORM) escrito en PHP que proporciona persistencia para objetos PHP. Está por encima de la capa de abstracción a la base de datos, una de sus características es la posibilidad de escribir consultas a la base de datos a partir del tratamiento con objetos en PHP llamado Doctrine Query Language (DQL). **(ServerGrove, 2010)**
Doctrine puede generar clases a partir de una base de datos creada, y el programador puede especificar relaciones y agregar funcionalidades comunes para las clases generadas. No hay necesidad de generar o mantener esquemas complejos en XML.

Sauxe utiliza en la capa de acceso a datos el Lenguaje de Consulta de Datos (DQL) que implementa Doctrine. La documentación de éste tiene todas las características necesarias para

ser funcional en casi cualquier proyecto. Entre otros elementos se tiene la posibilidad de exportar una base de datos existente a sus clases correspondientes y también a la inversa, es decir, convertir clases (convenientemente creadas) a tablas de una base de datos. Por otro lado, como la librería es bastante grande tiene un método para ser "compilada" al pasar a producción. **(Baryolo, et al., 2008)**

1.12.2 ExtJS 2.2

ExtJS es una librería JavaScript ligera y de alto rendimiento, compatible con la mayoría de los navegadores que permite crear páginas e interfaces web dinámicas. **(Desarrollo, 2006-2010)**
Permite construir aplicaciones complejas además de flexibilizar el manejo de componentes de la página como el DOM[5], Peticiones Ajax, DHTML[6] tiene la gran funcionalidad de crear interfaces de usuario bastante funcionales.
Esta librería incluye:

- ✓ Componentes UI[7] de alto performance y personalizables.
- ✓ Modelo de componentes extensibles.
- ✓ Un API fácil de usar.
- ✓ Licencias Open Source (GPL)[8]

Este marco de trabajo cuenta con un conjunto de componentes para incluir dentro de una aplicación web como:

- ✓ Cuadros y áreas de texto.
- ✓ Campos para fechas.
- ✓ Campos numéricos.
- ✓ Combos.
- ✓ Radiobuttons y checkboxs.
- ✓ Editor HTML.
- ✓ Elementos de datos (con modos de sólo lectura, datos ordenables, columnas que se pueden bloquear y arrastrar, etc.).
- ✓ Árbol de datos.

[5] Modelo de Objetos del Documento *o* Document Object Model(*DOM)*

[6] HTML Dinámico o DHTML (del inglés Dynamic HTML)

[7] Interfaz de usuario o UI(del inglés U*ser* Interface)

[8] Licencia Pública General de GNU o más conocida por su nombre en inglés GNU General Public License(GPL)

- ✓ Pestañas.
- ✓ Barra de herramientas.
- ✓ Menús al estilo de Windows.
- ✓ Paneles divisibles en secciones.
- ✓ Sliders.

Ventajas

Una de las grandes ventajas de utilizar ExtJS es que permite crear aplicaciones complejas utilizando componentes predefinidos. Evita el problema de tener que validar el código para que funcione bien en cada uno de los navegadores (Firefox, IE, Safari, Opera, etc.). Haciendo uso de sus componentes se puede construir un sistema con interfaz amigable, fácil de utilizar y con un diseño sencillo. El funcionamiento de las ventanas flotantes lo pone por encima de cualquier otro. Permite que la relación entre Cliente-Servidor sea balanceado ya que distribuye la carga de procesamiento permitiendo que el servidor pueda atender más clientes al mismo tiempo.

Sauxe utiliza ExtJS en la capa de presentación, por la gran gama de componentes que se pueden reutilizar para agilizar el proceso de desarrollo y mostrarle al usuario una interfaz más amigable y funcional. **(Baryolo, et al., 2008)**

1.12.3 Marco de trabajo Zend 1.2.7

Es un marco de trabajo de código abierto para desarrollar aplicaciones web y servicios web con PHP5. Zend es una implementación que usa código cien por ciento orientado a objetos. La estructura de los componentes de Zend es algo único; cada componente está construido con una baja dependencia de otros componentes. Esta arquitectura débilmente acoplada permite a los desarrolladores utilizar los componentes por separado. **(Inc, 2005-2010)**

El marco de trabajo Zend se basa en la simplicidad, orientados a objetos las mejores prácticas, licencia amigable corporativa, y una base de código rigurosamente ágil. Zend se centra en la construcción de páginas web más seguras, confiables, modernas. **(Technologies, 2006-2012)**
Zend ofrece un gran rendimiento y una robusta implementación del patrón modelo-vista-controlador, una abstracción de base de datos fácil de usar, y un componente de formularios que implementa la prestación de formularios HTML, validación y filtrado para que los

desarrolladores puedan consolidar todas las operaciones usando de una manera sencilla la interfaz orientada a objetos.

Características fundamentales de Zend:

- ✓ Trabaja con Modelo Vista Controlador (MVC).
- ✓ Zend también incluye objetos de las diferentes bases de datos, por lo que es extremadamente simple para consultar su base de datos, sin tener que escribir ninguna consulta SQL.
- ✓ Una solución para el acceso a base de datos que balancea el Mapeador de Objeto Relacional con eficiencia y simplicidad.
- ✓ Completa documentación
- ✓ Soporte avanzado.
- ✓ Robustas clases para autenticación y filtrado de entrada.
- ✓ Muchas otras clases útiles para hacerlo tan productivo como sea posible. **(Leopoldo, 2007)**

Se usa Zend porque lo necesita el marco de trabajo Sauxe para su funcionamiento además de que con él se puede construir una aplicación más segura y confiable pues así lo requieren los Tribunales Populares Cubanos por el nivel de confidencialidad que tiene la información con la que ahí se trabaja.

Los 51 componentes de Zend conforman un potente y extensible marco de trabajo de aplicaciones web al combinarse entre sí. De todos estos Sauxe utiliza solamente los siguientes:

Componentes
Zend_Cache
Zend_Config
Zend_Controller
Zend_Loader
Zend_Log

Zend_Registry
Zend_Session,
Zend_View

Tabla 1: Componentes de Zend que utiliza Sauxe

A partir de la extensión de algunos componentes de Zend surge ZendExt, desarrollado por el departamento de tecnologías de la UCI en conjunto con la UCID (Unidad de Compatibilización Integración y Desarrollo de Software para la Defensa), con el objetivo de crear un marco de trabajo extensible y configurable centrando el desarrollo de las aplicaciones, en la lógica del negocio, en las interfaces de usuario, alejando a los programadores de los detalles arquitectónicos, con soporte para entornos multi-entidad y para una arquitectura de sistema orientada a componentes. **(Baryolo, Tenrero, & Silega, 2008)**

El marco de trabajo Sauxe surge de la unión de ZendExt, Doctrine y ExtJS.

1.12.4 Sauxe 2.0

Sauxe es un marco de trabajo que contiene un conjunto de componentes reutilizables que provee la estructura genérica y el comportamiento para una familia de abstracciones, logrando una mayor estandarización, flexibilidad, integración y agilidad en el proceso de desarrollo. **(Baryolo, Tenrero, & Silega, 2008)**

Sauxe en su versión 2.0 está formado por los componentes:

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

ZendExt_TransactionManager

Tabla 2: Componentes que forman el marco de trabajo Sauxe

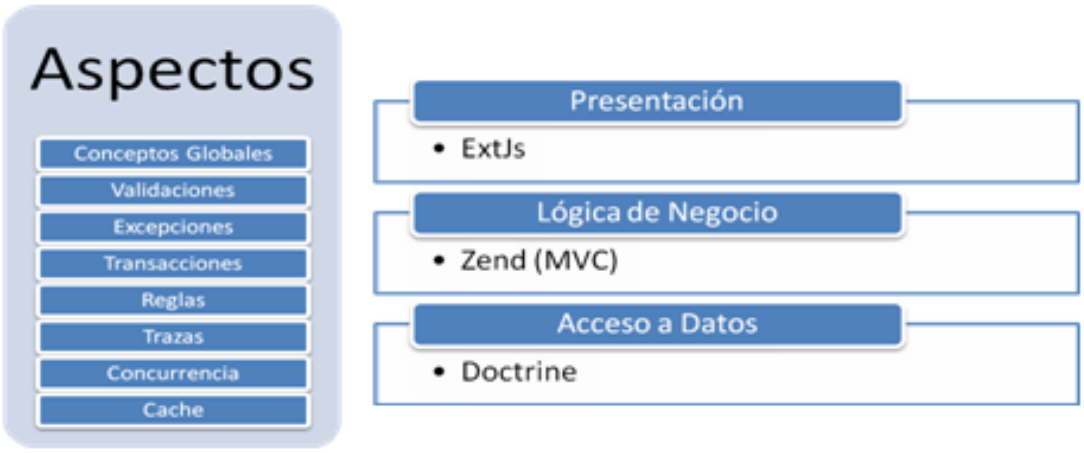

Figura 1: Arquitectura de Sauxe

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

- ✓ **Autenticación:** es el proceso de verificar formalmente la identidad de las entidades participantes en una comunicación o intercambio de información. Por entidad se entiende tanto personas, como procesos o computadoras.
- ✓ **Autorización:** es la parte del sistema que protege los recursos del sistema permitiendo que solo sean usados por aquellos consumidores a los que se les ha concedido autorización para ello. Los recursos incluyen archivos y otros objetos de datos, programas, dispositivos y funcionalidades provistas por aplicaciones.
- ✓ **Administración de perfil:** debe garantizarse la personalización de las aplicaciones de este dominio a nivel de cada usuario, se define como perfil los datos únicos de cada recurso dentro del sistema que define el comportamiento del mismo ante las entradas emitidas por este recurso y las salidas entregadas por el (los) subsistema(s), esto garantizaría un sin número de bondades tanto de usabilidad como de configurabilidad a la solución en cuestión.
- ✓ **Administración de conexiones:** consiste en un grupo de procesos dedicados a la gestión de las conexiones a la base de datos de un sistema determinado ubicado en un

servidor de bases de datos definido también como un parámetro configurable, así como el gestor en uso. Esta solución debe incluir la gestión dinámica de usuarios de bases de datos y permisos sobre ellas.

El mecanismo de seguridad que brinda el marco de trabajo Sauxe es de vital importancia para el desarrollo de la aplicación para los Tribunales Populares Cubanos ya que no todos los usuarios tienen acceso a la misma información y al ser confidencial la misma hay que definir una jerarquía de usuarios para limitar el acceso a la información por parte de los mismos. El marco de trabajo permite establecer la seguridad por roles para tener un mayor control sobre el acceso a la información por parte de los diferentes usuarios de acuerdo al rol que se le asigne dentro del sistema. También Sauxe permite dar seguimiento a todo lo que se haga en la aplicación haciendo uso de un sistema de trazas.

1.13 Sistema gestor de base de datos

El propósito general de los sistemas de gestión de bases de datos es el de manejar de manera clara, sencilla y ordenada un conjunto de datos que posteriormente se convertirán en información relevante para una organización.

Los sistemas gestores de base de datos permiten:

Abstracción de la información: ahorran a los usuarios detalles acerca del almacenamiento físico de los datos. Da lo mismo si una base de datos ocupa uno o varios archivos. Así, se definen varios niveles de abstracción.

Independencia: la independencia de los datos consiste en la capacidad de modificar el esquema (físico o lógico) de una base de datos sin tener que realizar cambios en las aplicaciones que se sirven de ella.

Consistencia: en aquellos casos en los que no se ha logrado eliminar la redundancia, será necesario vigilar que aquella información que aparece repetida se actualice de forma coherente, es decir, que todos los datos repetidos se actualicen de forma simultánea.

Seguridad: la información almacenada en una base de datos puede llegar a tener un gran valor. Los SGBD[9] deben garantizar que esta información se encuentra segura de permisos a usuarios y grupos de usuarios, que permiten otorgar diversas categorías de permisos.

Manejo de transacciones: una transacción es un programa que se ejecuta como una sola operación. Esto quiere decir que luego de una ejecución en la que se produce una falla es el mismo que se obtendría si el programa no se hubiera ejecutado. Los SGBD proveen mecanismos para programar las modificaciones de los datos de una forma mucho más simple que si no se dispusiera de ellos.

Tiempo de respuesta: lógicamente, es deseable minimizar el tiempo que el SGBD demora en proporcionar la información solicitada y en almacenar los cambios realizados.

1.13.1 PostgreSQL 8.4

Es un sistema de gestión de bases de datos objeto-relacional (ORDBMS)[10]

Principales características de este gestor de bases de datos:

- ✓ Implementación del estándar SQL92/SQL99.
- ✓ 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, etc.
- ✓ Incorpora funciones de diversa índole: manejo de fechas, geométricas, orientadas a operaciones con redes, etc.
- ✓ Permite la declaración de funciones propias, así como la definición de disparadores.
- ✓ Soporta el uso de índices, reglas y vistas.
- ✓ Incluye herencia entre tablas (aunque no entre objetos, ya que no existen), por lo que a este gestor de bases de datos se le incluye entre los gestores objeto-relacionales.
- ✓ Permite la gestión de diferentes usuarios, como también los permisos asignados a cada uno de ellos.

Se utiliza este gestor de base de datos porque se ejecuta en los principales sistemas operativos incluyendo Linux que es sobre el cual se desarrolla la aplicación. Tiene un buen tiempo de respuesta antes las solicitudes de información. Permite definir varios niveles de

[9] Sistemas gestores de base de datos

[10] Object-Relational Database Management System

abstracción ya que la base de datos del proyecto no tiene un solo archivo sino uno para cada subsistema aunque esto es transparente al usuario.

1.14 Herramientas CASE

Las herramientas CASE[11] son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero. Estas herramientas pueden ayudar en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de realizar un diseño del proyecto, cálculo de costes, implementación de parte del código automáticamente con el diseño dado, documentación o detección de errores entre otras.

La tecnología CASE supone la automatización del desarrollo del software, contribuyendo a mejorar la calidad y la productividad en el desarrollo de sistemas de información. Para mejorar la calidad y la productividad de los sistemas de información a la hora de construir software se plantean los siguientes objetivos:

- ✓ Permitir la aplicación práctica de metodologías estructuradas, las cuales al ser realizadas con una herramienta se consigue agilizar el trabajo.
- ✓ Facilitar la realización de prototipos y el desarrollo conjunto de aplicaciones.
- ✓ Simplificar el mantenimiento de los programas.
- ✓ Mejorar y estandarizar la documentación.
- ✓ Aumentar la portabilidad de las aplicaciones.
- ✓ Facilitar la reutilización de componentes de software.
- ✓ Permitir un desarrollo y un refinamiento visual de las aplicaciones, mediante la utilización de gráficos.

1.14.1 Visual Paradigm 6.4

Propicia un conjunto de ayudas para el desarrollo de programas informáticos, desde la planificación, pasando por el análisis y el diseño, hasta la generación del código fuente de los programas y la documentación.

[11] ***C**omputer **A**ided **S**oftware **E**ngineering*, Ingeniería de Software Asistida por Computadora.

Visual Paradigm es una herramienta CASE que utiliza como lenguaje UML[12] para el modelado, soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El lenguaje de modelado UML ayuda a una construcción más rápida de aplicaciones de calidad, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. **(Paradigm, 2011)**

Principales características

- ✓ Disponibilidad en múltiples plataformas (Windows, Linux).
- ✓ Diseño centrado en casos de uso y enfocado al negocio que generan un software de mayor calidad.
- ✓ Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.
- ✓ Capacidades de ingeniería directa e inversa.
- ✓ Modelo y código que permanece sincronizado en todo el ciclo de desarrollo
- ✓ Disponibilidad de múltiples versiones, para cada necesidad.
- ✓ Soporta aplicaciones Web.
- ✓ Varios idiomas.
- ✓ Exportación como HTML.
- ✓ Fácil de instalar y actualizar.
- ✓ Soporte de UML versión 2.1.
- ✓ Modelado colaborativo con Subversión.
- ✓ Interoperabilidad con modelos UML2 (metamodelos UML 2.x para plataforma Eclipse) a través de XMI[13] .
- ✓ Generación de código - modelo a código, diagrama a código.
- ✓ Soporte ORM - generación de objetos Java desde la base de datos.
- ✓ Generación de bases de datos - Transformación de diagramas de Entidad-Relación en tablas de base de datos.
- ✓ Ingeniería inversa de bases de datos - desde sistemas gestores de bases de datos (DBMS)[14] existentes a diagramas de Entidad-Relación.

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

[13] XML Metadata Interchange o XML de Intercambio de Metadatos

- ✓ Generador de informes.
- ✓ Distribución automática de diagramas - Reorganización de las figuras y conectores de los diagramas UML.
- ✓ Importación y exportación de ficheros XMI.
- ✓ Editor de figuras. **(Pressman, 2002)**

Por las características que posee Visual Paradigm se utiliza en el desarrollo de la aplicación para la parte del modelado ya que posee alta capacidad de integración con el lenguaje orientado a procesos BPMN[15] y UML que son los utilizados para el desarrollo del sistema además de que la aplicación se está implementando en PHP y Visual Paradigm se integra fácilmente con este lenguaje.

1.15 Arquitectura de software

La arquitectura de software es un conjunto de patrones que proporcionan un marco de referencia necesario para guiar la construcción de un software, permitiendo a los programadores, analistas y todo el conjunto de desarrolladores compartir una misma línea de trabajo y cubrir todos los objetivos y restricciones de la aplicación. Es considerada el nivel más alto en el diseño de la arquitectura de un sistema puesto que establecen la estructura, funcionamiento e interacción entre las partes del software.

1.15.1 Estilos arquitectónicos

Se identifican los estilos arquitectónicos como un "conjunto de reglas de diseño que identifica las clases de componentes y conectores que se pueden utilizar para componer un sistema o subsistema, junto con las restricciones locales o globales de la forma en que se lleva a cabo la composición. Es en gran medida la interacción entre los componentes, mediados por conectores, lo que confiere a los distintos estilos sus características distintivas" **(Carlos Reynoso, 2004).**

1.15.1.1 Estilos de Llamada y Retorno

[14] Sistemas de gestión de bases de datos (en inglés Database Management System, abreviado DBMS*)*

[15] Business Process Modeling Notation (Notación de Modelado de Procesos del Negocio)

Esta familia de estilos enfatiza la modificabilidad y la escalabilidad. Son los estilos más generalizados en sistemas en gran escala. Miembros de la familia son las arquitecturas de programa principal y subrutina, los sistemas basados en llamadas a procedimientos remotos, los sistemas orientados a objeto y los sistemas jerárquicos en capas.

1.15.2 Arquitectura en capas

En el año 2000 se acordó definir oficialmente la Arquitectura de Software según figura en el documento IEEE Std 1471-2000: “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”

La arquitectura en capas define cómo organizar el modelo de diseño en capas, que pueden estar físicamente distribuidas, lo cual quiere decir que los componentes de una capa sólo pueden hacer referencia a componentes en capas inmediatamente inferiores. La aplicación se divide en tres capas lógicas distintas, cada una de ellas con un grupo de interfaces perfectamente definido. Esta separación entre la lógica de aplicación de la interfaz de usuario añade una enorme flexibilidad al diseño de la aplicación. Pueden construirse y desplegarse múltiples interfaces de usuario sin cambiar en absoluto la lógica de aplicación siempre que está presente una interfaz claramente definida a la capa de presentación.

1.16 Conclusiones

Después del estudio realizado como parte de la fundamentación teórica de la investigación, se han reflejado los conceptos más importantes a tener en cuenta para seguir con el desarrollo de la investigación. También se fundamentaron las principales características de la metodología, lenguaje de programación y herramientas definidas para la implementación, se analizaron las métricas que permitirán medir la calidad del software, además de las pruebas que tributan al cumplimiento de los objetivos propuestos en el desarrollo de este capítulo.

CAPÍTULO II PROPUESTA DE SOLUCIÓN

Introducción

En este capítulo se propondrá una solución técnica de la investigación. Teniendo en cuenta la arquitectura del sistema a desarrollar, sus capas y sus componentes. Se presentarán los distintos patrones de diseño utilizados en el desarrollo de la aplicación con el fin de dotar al sistema con una mayor robustez, flexibilidad y seguridad. Se realizará el modelo de diseño, el cual se encuentra conformado por los diagramas de clases y de secuencia. El modelo de implementación con los diagramas de componentes y el diagrama de despliegue previamente diseñados.

La solución a desarrollar consta de 16 casos de uso, la descripción de los mismos se encuentra en el Modelo del sistema v2.0 del subsistema Penal hasta la Sentencia. Los casos de uso a desarrollar son los siguientes:

- ✓ Registrar solicitud del fiscal
- ✓ Registrar acusado
- ✓ Registrar pruebas
- ✓ Registrar Perito
- ✓ Entregar EFP por devolución
- ✓ Admitir personería de abogado
- ✓ Registrar prestación de fianza
- ✓ Disponer orden de arresto
- ✓ Dar por presentado el EFP
- ✓ Comprobar piezas de convicción
- ✓ Disponer devolución de EFP
- ✓ Tramitar apertura
- ✓ Designar abogado de oficio
- ✓ Dejar sin efecto orden de arresto
- ✓ Tramitar sin efecto de orden de arresto
- ✓ Registrar Notificación a Acusado

Capítulo II: Propuesta de Solución

2.1 Arquitectura del sistema

La arquitectura definida para el desarrollo de la aplicación es la arquitectura en tres capas la cual está formada por la Capa de presentación, Capa de negocio y la Capa de acceso a datos.

2.1.1 Capa de presentación

En esta capa se emplea las facilidades que brinda el marco de trabajo ExtJS para la construcción de interfaces entendibles y fáciles de usar para el usuario. ExtJS centra su desarrollo en tres componentes fundamentales JavaScript, CSS y AJAX. **(Andux, 2010)**

Es la que se encarga de que el sistema interactúe con el usuario y viceversa, muestra el sistema al usuario, le presenta la información y obtiene la información del usuario. En el mundo de la informática la capa de presentación es conocida como interfaz gráfica. Esta capa se comunica únicamente con la capa de negocio.

2.1.2 Capa de negocio

Es donde residen las funciones que se ejecutan, se reciben las peticiones del usuario, se procesa la información y se envían las respuestas tras el proceso. Se denomina capa de negocio o capa de lógica del negocio, porque es aquí donde se establecen todas las reglas que deben cumplirse. Esta capa se comunica con la de presentación, para recibir las solicitudes y presentar los resultados, y con la capa de acceso a datos, para obtener o enviar información al gestor de base de datos.

2.1.3 Capa de acceso a datos

En esta capa estará presente el mapeador de objeto relacional Doctrine, como persistidor para la comunicación con el servidor de datos mediante el protocolo PDO, también estará un persistidor de configuración que es el encargado de comunicarse vía XML con los ficheros de configuración del sistema denominado FastResponse. **(Andux, 2010)**

2.2 Estructura Organizativa

Existen dos paquetes fundamentales, “apps” y “web” donde se encuentran principalmente las clases necesarias para el desarrollo de la aplicación. Dentro de cada uno de estos hay un paquete para cada módulo para organizar las clases correspondientes a cada uno de ellos, ya sean clases de presentación, de negocio o clase de datos. Dentro del paquete “web” están

todas las clases de presentación o sea las clases del tipo js, y dentro del paquete “apps” están todas las clases del negocio y las clases de datos. Dentro del paquete “apps”, en el paquete “Controllers” están las clases controladoras. Paralelo al paquete “Controllers” está el paquete “Models” quien tiene todas las clases que se generan por cada tabla de la base de datos. **(Urrutia, 2010)**

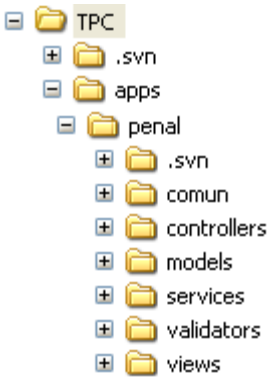

Figura 2: Estructura organizativa del marco de trabajo para el paquete apps

Controllers: en esta carpeta se almacenan todas las clases controladoras del módulo. La estructura del nombre de la clase será: el nombre propio de la misma en mayúsculas seguido por la palabra Controller. Las funciones que sean programadas dentro de esta clase tendrá al final la palabra Action.

Models: contiene las carpetas “domain” y “business”. En los ficheros del “domain” se programan las consultas y en el “business” las modificaciones como son: los métodos invocados desde la funcionalidad de la clase controladora.

Services: contiene los servicios que utiliza por ejemplo, servicios web, servicios de seguridad, etc.

Validators: esta carpeta contiene las clases de tipo PHP que van a realizar acciones de validación en el componente, como por ejemplo las precondiciones que se deben cumplir antes de que un determinado método sea ejecutado.

Views: en esta carpeta se recopilan los ficheros que van a gestionar la capa de presentación, estos son idioma y scripts. Dentro de la carpeta idioma existirán dos subcarpetas, una “es” donde se almacenan los archivos que gestionan una presentación en español como por ejemplo ficheros de tipo json que recopilan etiquetas para mostrar mensajes en el idioma

correspondiente, y otra carpeta llamada “en” donde se gestiona la presentación en inglés. Dentro de la carpeta scripts se incluyen todas las vistas, para ello se crea una carpeta para cada clase controladora y dentro se incluye la vista o script. Estos no son más que archivos de extensión phtml donde se especifica el título de la página que se gestiona y se carga el archivo js que mostrará la presentación como tal.

Figura 3: Estructura organizativa del marco de trabajo para el paquete web

Views: en esta carpeta se recopilan los ficheros que van a gestionar la capa de presentación, en este caso dentro de esta estarán css y js. Dentro de la carpeta css se encuentran las plantillas para el diseño del módulo, por lo general no se utiliza ninguna plantilla ya que con el marco de trabajo ExtJS se está resolviendo todo el tema de la presentación. Un fichero js no es más que un documento con la extensión .js donde se escribirá el código correspondiente a la capa de presentación, aquí es donde se carga el diseño de la aplicación, para cada clase controladora se crea una carpeta que tendrá incluido el fichero js correspondiente a dicha clase control.

2.3 Patrones de diseño utilizados

Modelo Vista Controlador

Este patrón de diseño viene definido por el marco de trabajo Sauxe. Está estructurado para crear las clases por separadas, las clases del modelo representan la información en la cual la aplicación opera, están en el paquete **app\penal\Model**. Las clases de la vista, que son las que renderizan el modelo dentro de una página web apropiada para que el usuario pueda interactuar, se encuentran en el paquete **app\penal\Views** y las clases controladoras encargadas de responder a las acciones del usuario e invocar cambios en el modelo o generar la vista apropiada, se encuentran en el paquete **app\penal\Controllers**.

Este patrón separa la lógica de negocio (model) y la presentación (view), el resultado es en un código mantenible y organizado.

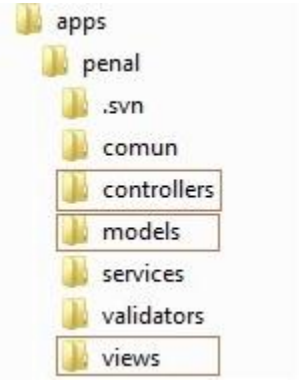

Figura 4: Estructura del paquete App aplicando el patrón MVC

Ioc

Este patrón es utilizado por el marco de trabajo para el trabajo con las dependencias que se hacen mediante el uso de servicios. Cada módulo del proyecto cuenta con un esquema propio en la base de datos. Cuando un módulo necesita acceder al esquema de otro, lo que se hace es crear y brindar un servicio para que el que lo necesite lo consuma.

El marco de trabajo a través del patrón IoC, define la creación de clases, para de ellas brindar los servicios necesarios.

A continuación se muestra un ejemplo donde se crea la función ObtenerProcedimientos.

```
class ProcedimientosService {

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

        function ProcedimientoExpediente($idexpediente) {
            return DexpedientepreparatorioExt::procedimientoExpediente($idexpediente);
        }

}
```

Figura 5: Representación de la clase ProcedimientosService, ubicada en el paquete apps\penal\services, donde se crea la función ObtenerProcedimientos.

Cada módulo cuenta con un fichero ioc-general.xml en el cual se brindan los servicios deseados, para que otros lo consuman.

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

Figura 6: Fichero ioc-general.xml en al paquete apps\penal\común\recursos\xml, donde se crea el servicio ObtenerProcedimientos.

A continuación se muestra cómo se consume desde una clase controladora del módulo Común el servicio brindado por el módulo penal

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

Figura 7: Representación de la función cargarComboProcesosAction en la clase InicioController.php del módulo Común, la cual consume el servicio ObtenerProcedimientos brindado por el módulo Penal.

Singleton

En el sistema propuesto se utiliza este patrón con el objetivo que de una clase solo exista una sola instancia y proporcionar un punto de acceso global a ella y se fuerza a que no se puedan crear objetos de forma directa mediante el operador "new". En el ejemplo se muestra cómo se crea una instancia de ZendExt_Event.

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

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

        return self :: $_instance;
    }
```

Figura 8: Representación del patrón Singleton en la clase ZendExt_Event.

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.

Bajo acoplamiento

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

```php
<?php
class PnlDabogado extends BasePnlDabogado
{
    public function setUp()
    {
        parent::setUp();

        $this->HasMany('PnlDescritopersoneria', array('local'=>'idabogado','foreign'=>'idabogado'));
        $this->HasMany('PnlDacusadoabogado', array('local'=>'idabogado','foreign'=>'idabogado'));
         $this->hasOne('Dabogado', array('local'=>'idabogado','foreign'=>'idpersona'));
    }

}
?>
```

Figura 9:Clase del modelo que solo depende su clase base

Como se aprecia en la figura anterior la clase del modelo “PnlDabogado” solo depende de su clase Base y no de ninguna clase de la vista o la capa del controlador.

Controlador

En la estructura que propone Sauxe este patrón se evidencia dentro de la carpeta controllers que se encuentra en “TPC/app/modulo/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 caso de uso determinado, además son las que reciben los datos del usuario y los utilizan de acuerdo a la acción solicitada.

```
<?php

...

class RegnotificacionaacusadoController

    function init() {...}

    function regnotificacionaacusadoAction() {...}


 function listarExpAction()
 {...}

    function listarAcusadosAction()
    {...}

   function insertarFechaAction()
        {...}


}
?>
```

Figura 10: Clase RegistrarnotificacionaacusadoController encargada de implementar las funcionalidades del caso de uso registrar notificación a acusado

2.4. Diseño e implementación

El propósito del diseño es especificar una solución que pueda ser fácilmente convertida en código fuente y construir una arquitectura simple y extensible. El diseño tiene en cuenta los requisitos funcionales, por lo que se centra fundamentalmente en cómo se cumplen los objetivos del sistema. El modelo de implementación describe cómo los elementos del modelo del diseño se implementan en términos de componentes y cómo estos se organizan de acuerdo a los nodos específicos en el modelo de despliegue.

2.4.1 Modelo de diseño

En el modelo de diseño se describe la realización de los casos de uso. Este modelo incluye los diagramas de interacción y de clases.

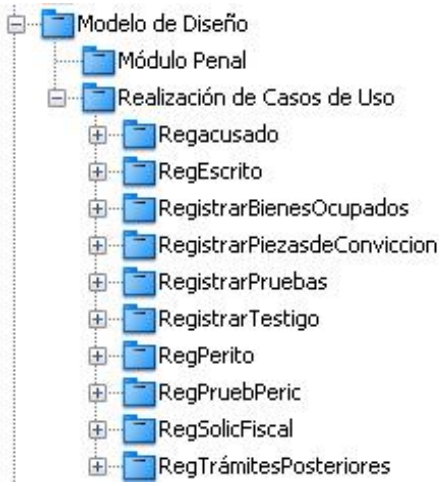

Figura 11: Estructura interna de un módulo dentro del paquete "Modelo de Diseño"

2.4.1.1 Diagrama de clases

Los diagramas de clases representan la estructura estática del sistema, mostrando las clases, sus atributos y las relaciones entre las mismas. Un diagrama de clases del diseño está formado por: clases, asociaciones y atributos, interfaces, con sus operaciones y constantes, métodos, información sobre los tipos de atributos, navegabilidad y dependencias.

A continuación se muestra la descripción de un caso de alta complejidad con el diagrama de clases correspondiente. La descripción de todos los casos de uso de la solución se encuentran en el modelo del sistema v2.0 del subsistema Penal hasta la Sentencia.

Caso de uso Registrar Acusado

Caso de uso:	Registrar acusado.
Actores:	Fiscal.
Resumen:	El caso de uso se inicia cuando el Fiscal necesita registrar un acusado. Consiste en que el Fiscal selecciona la opción registrar acusado y llena los campos requeridos para el registro. El caso de uso termina con el acusado registrado.
Precondiciones	

Referencias	RF.04, RF.83, RF.84, RF.85
Prioridad	Crítico.

Tabla 3: Descripción del Caso de Uso Registrar Acusado

El siguiente diagrama está compuesto por un número de clases las cuales se encuentran separadas según propone el patrón MVC. En la vista se encuentra regacusado.js en la cual se implementada la interfaz gráfica que será mostrada al usuario y esta usa las librerías ExtJS para lograr su correcto funcionamiento, en esta misma capa se encuentra regacudado.phtml, en este están referenciados todos los .js que se van a usar en el caso de uso en este caso regacusado.js. En la capa controladora se implementa la clase RegacusadoController en la que se encuentra todas las funcionalidades definidas y es la encargada de enviar y obtener información tanto de la capa de la vista como de la del modelo, y en la capa del modelo se encuentra PnlRegacusadoModel en la que se encuentran implementado un gestionar acusado y el resto de las clases como Pnldelito y BasePnldelito. En total en este diagrama intervienen un número de 21 clases.

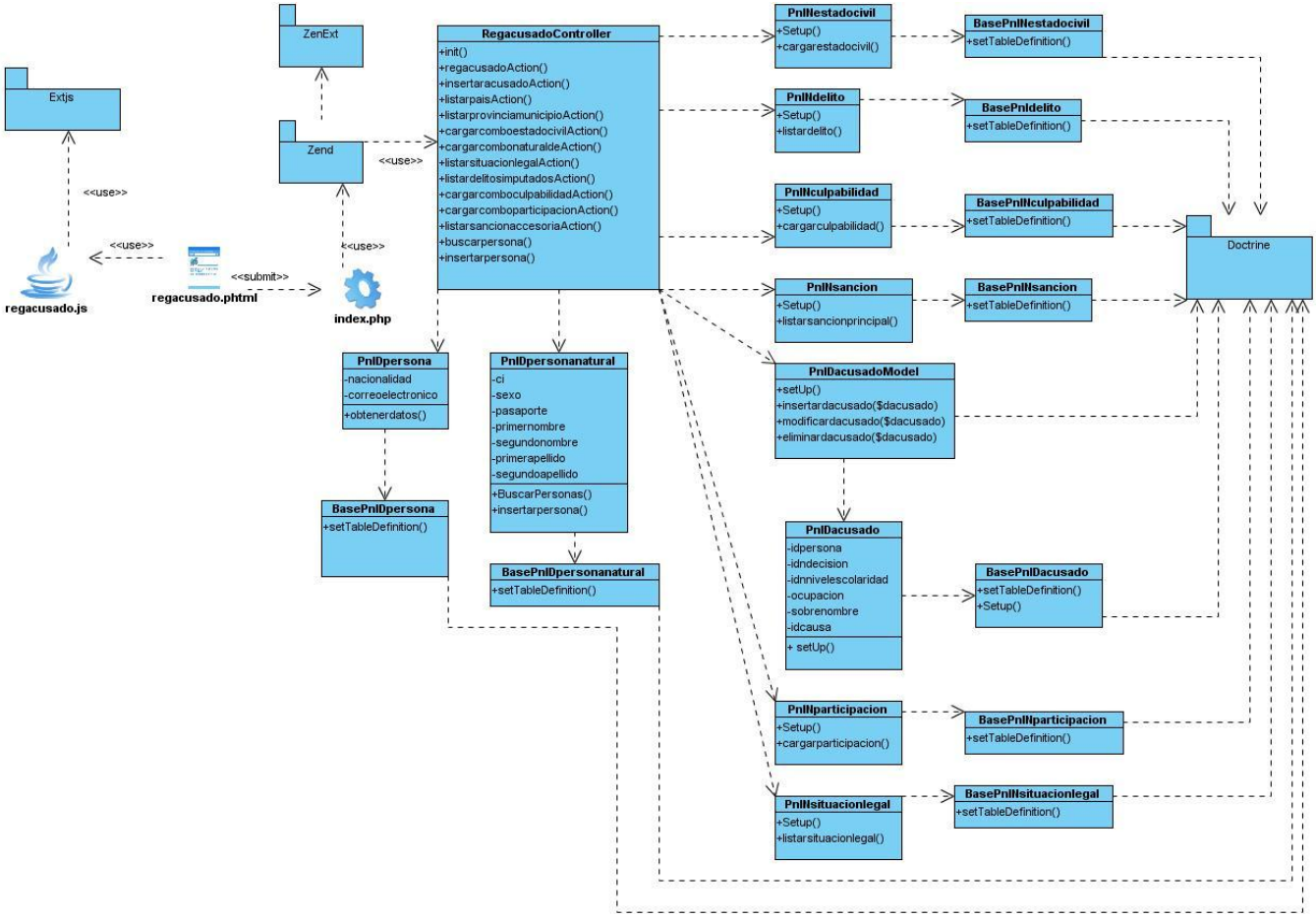

Figura 12: Diagrama de Clases: Registrar Acusado

2.4.1.2 Diagrama de interacción

Los diagramas de interacción son diagramas que describen cómo grupos de objetos colaboran para conseguir algún fin. Capturan el comportamiento de los casos de uso.

Un diagrama de secuencia muestra la interacción de un conjunto de objetos en una aplicación a través del tiempo y se modela para cada caso de uso. El diagrama de secuencia contiene detalles de implementación del escenario, incluyendo los objetos y clases que se usan para implementar el escenario, y mensajes intercambiados entre los objetos.

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

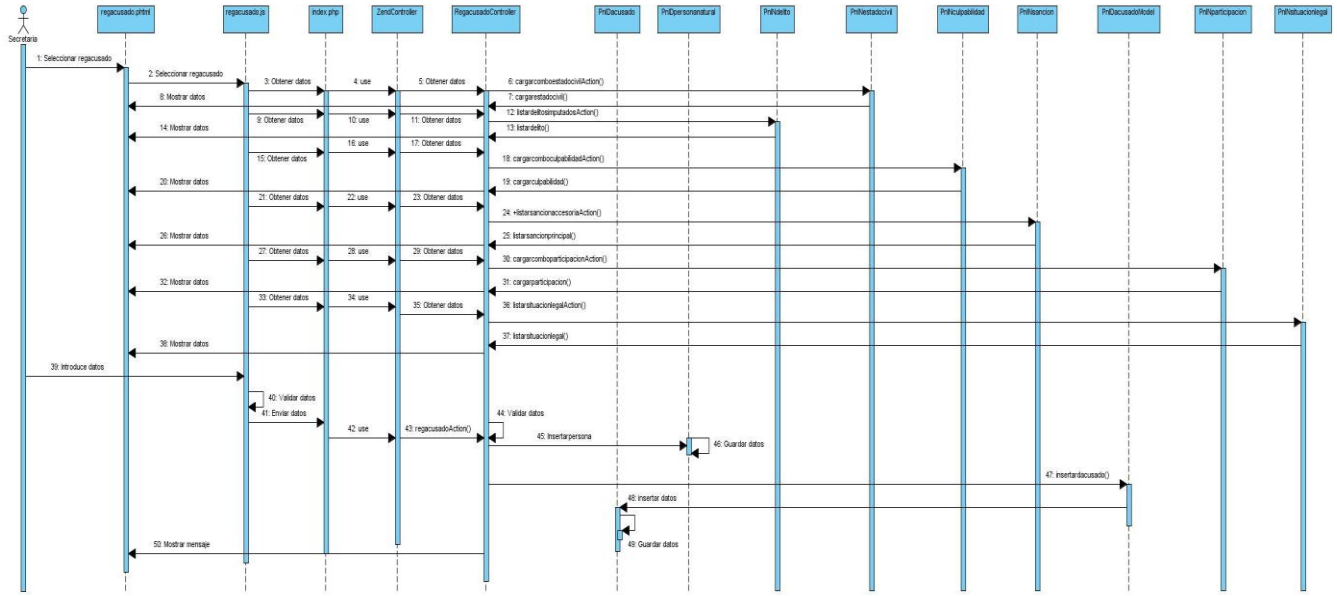

Figura 13: Diagrama de secuencia: Registrar Acusado

2.4.2 Modelo de implementación

Un modelo de implementación muestra las dependencias entre las partes de código del sistema y la estructura del sistema en ejecución. Como parte del modelo de implementación se encuentran los diagramas de componentes y los diagramas de despliegue.

2.4.2.1 Diagrama de componentes

Los diagramas de componentes modelan la vista estática de un sistema. Se representa como un grafo de componentes software unidos por medio de relaciones de dependencia (compilación, ejecución), pudiendo mostrarse las interfaces que estos soporten. No es necesario que un diagrama incluya todos los componentes del sistema, normalmente se realizan por partes, por lo cual cada diagrama describe un apartado del sistema. En un diagrama de componentes se tienen en consideración los requisitos relacionados con la facilidad de desarrollo, la gestión del software, la reutilización y las restricciones impuestas por los lenguajes de programación y las herramientas utilizadas en el desarrollo.**(Aprendizaje, 2012)**

Seguidamente se muestra el diagrama de componentes del sistema:

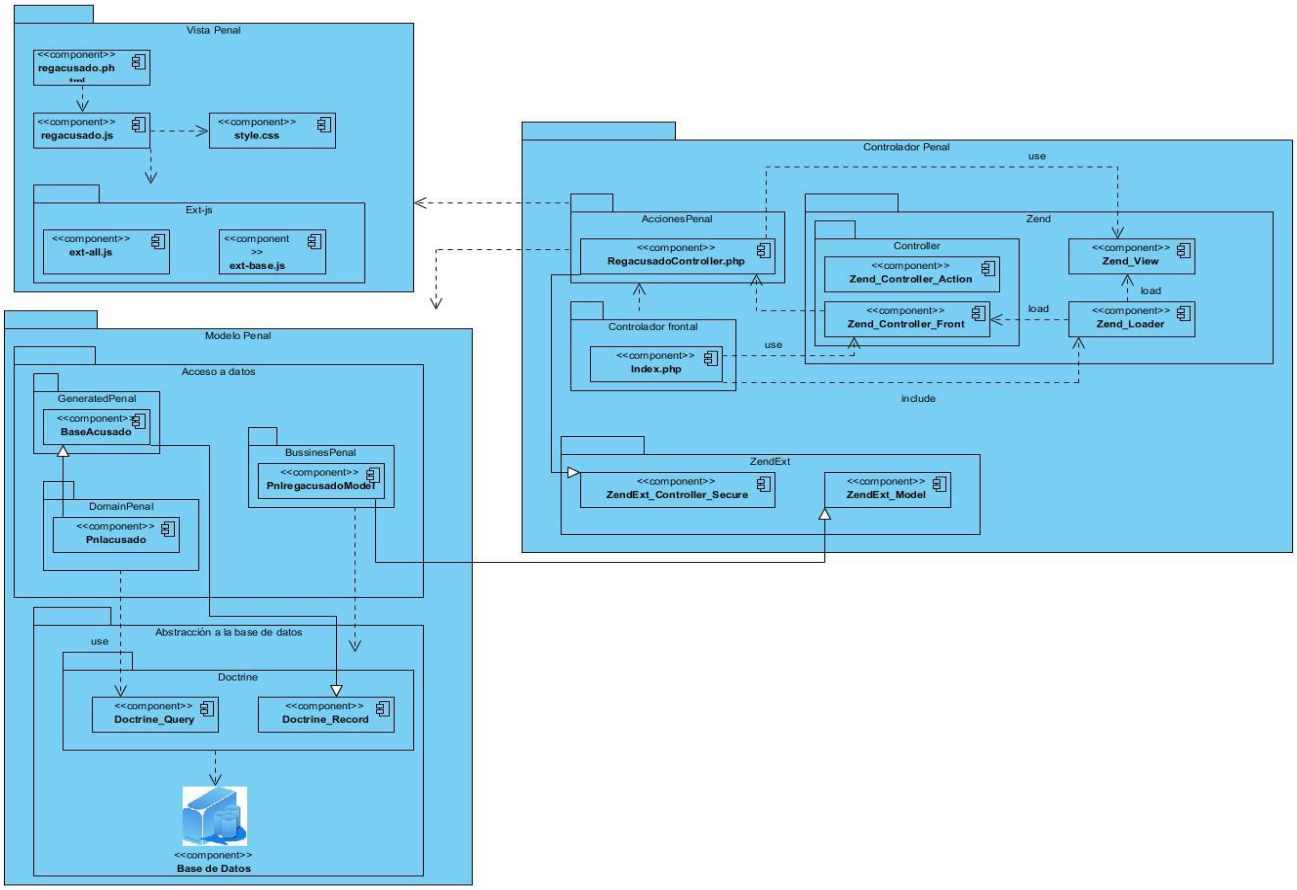

Figura 14: Diagrama de componentes del caso de uso Registrar Acusado

Este diagrama de componentes queda separado en tres paquetes con el fin de mostrar con una mayor claridad el MVC dentro de los cuales están definidos los componente que intervienen en cada capa y se muestran los componentes que interactúan con el fin de lograr la comunicación entre dichas capas.

2.4.2.2 Diagrama de despliegue

Un Diagrama de Despliegue muestra las relaciones físicas de los distintos nodos que componen un sistema y el reparto de los componentes sobre dichos nodos. La vista de despliegue representa la disposición de las instancias de componentes de ejecución en instancias de nodos conectados por enlaces de comunicación. Estos diagramas describen la arquitectura física del sistema durante la ejecución en términos de procesadores, dispositivos y componentes de software **(Alva, 2012)**

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

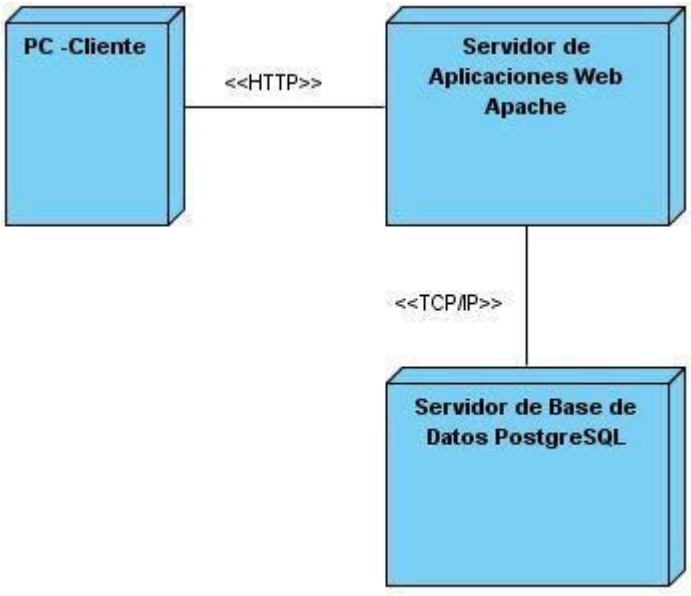

Figura 15: Diagrama de despliegue de la solución

2.5 Estándar de codificación

Los estilos de codificación son pautas de programación que no están enfocadas a la lógica del programa, sino a su estructura y apariencia física para facilitar la lectura, comprensión y mantenimiento del código. **(Pupo, 2010)**

Un estándar de codificación completo comprende todos los aspectos de la generación de código. El mejor método para asegurarse de que un equipo de programadores mantenga un código de calidad es establecer un estándar de codificación sobre el que se efectuarán luego revisiones del código de rutinas. Usar técnicas de codificación sólidas y realizar buenas prácticas de programación con vistas a generar un código de alta calidad es de gran importancia para la calidad del software y para obtener un buen rendimiento y mantenimiento del software.

La adopción de un estándar de codificación sólo es viable si se sigue desde el principio hasta el final del proyecto de software. No es práctico, ni prudente, imponer un estándar de codificación una vez iniciado el trabajo. En el caso del proyecto Tribunales Populares Cubanos es necesario aclarar que tanto el diseño para PHP como para JavaScript se debe hacer orientado a objetos

a fin de potenciar todas las ventajas que este paradigma implica, el estándar de codificación fue definido por la dirección del proyecto en sus inicios y es basado en los estándares de codificación para PHP **(Rodríguez, 2010)**

A continuación se muestran algunos de los estándares para la codificación que se encuentran en el documento *Pautas a seguir para la codificación del proyecto Tribunales Populares Cubanos en el lenguaje PHP* y que fueron aplicados a la hora de desarrollar la solución.

- ✓ Cabecera del archivo.

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

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

- ✓ Comentarios en las funciones.

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

- ✓ Clases.

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

- ✓ Nombres de variables.

No se debe adoptar la notación húngara[16] en el código. Muchos proyectos afirman que es una de las técnicas de ofuscación de código más ampliamente usadas en la actualidad.

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

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

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

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

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

- ✓ Precedencia de operadores

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

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

2.6 Conclusiones

En este capítulo se ha logrado llevar a la práctica lo estudiado en el anterior, mostrándose y justificándose los elementos que componen la arquitectura definida en TPC. A partir de las especificaciones de los requisitos funcionales de los caso de usos que pertenecen a dicho módulo se obtuvieron los artefactos correspondientes al modelo de diseño y al modelo de despliegue entre los que se encuentran, diagrama de clases, diagramas de secuencia, diagrama de componentes y diagrama de despliegue, en los cuales se evidencia el uso de los patrones de diseño usados para la obtención de un producto funcional del subsistema penal del Sistema de Informatización de Tribunales.

CAPÍTULO III: ANÁLISIS DE RESULTADOS

3.1 Introducción

En este capítulo se muestran los procedimientos y métodos utilizados para evaluar la calidad de los resultados obtenidos con el desarrollo de este trabajo, haciendo uso de las métricas de medición de la calidad del diseño y la implementación del software. Es de gran importancia el análisis de los resultados obtenidos en cada etapa del proceso de desarrollo de software ya que esto ayuda a minimizar los errores en la construcción de un producto con mayor calidad que cumpla con los intereses del cliente.

3.2 Pruebas de software

Las pruebas son una actividad en la cual un sistema o componente es ejecutado bajo condiciones o requerimientos especificados. Dentro de cada una de las etapas de desarrollo de un software las pruebas son fundamentales ya que a partir de ellas es posible controlar que los productos cumplan requisitos mínimos de operabilidad además de garantizar la calidad de estos productos. **(Pressman, 2002)**

Las pruebas constituyen una etapa imprescindible durante el proceso de desarrollo del software ya que permiten detectar y corregir el máximo de errores posibles antes de la entrega al cliente del software desarrollado, por lo que el éxito de las mismas mejora la percepción de calidad del usuario final. La calidad de un sistema está determinada, entre otras cosas, por el cumplimiento de los requisitos establecidos. Para comprobar el grado de cumplimiento de estos requisitos se usan las pruebas del sistema para eliminar posibles defectos que pueda tener el mismo.

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

La métrica se aplicó a 38 clases que forman parte de la solución. (Ver Anexo 1)

El resultado de la aplicación de esta métrica está dado por el número de métodos asignados a una clase y evalúa atributos de calidad como responsabilidad, complejidad de implementación y reutilización. Mientras menor es el TOC menor es la responsabilidad asignada a las clases y la complejidad de implementación no siendo así con la reutilización que aumenta mientras menor es el TOC.

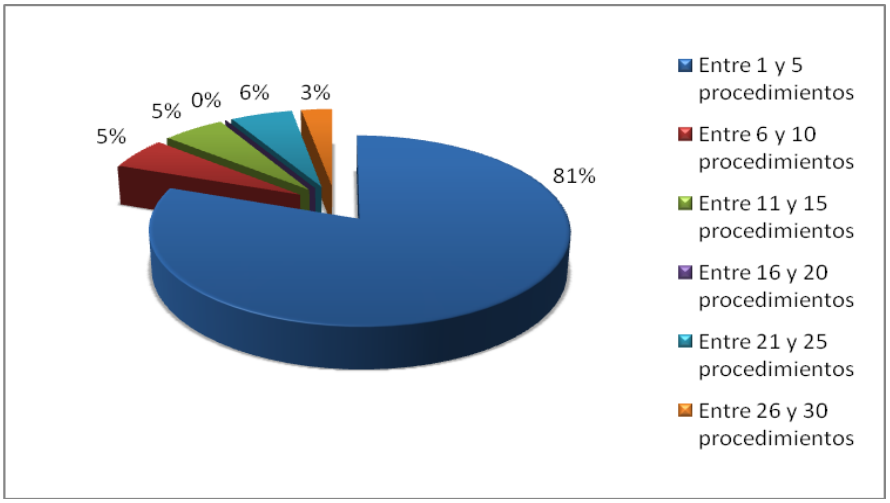

Figura 16: Representación en % de los resultados obtenidos en el instrumento agrupados en los intervalos definidos.

La gráfica anterior muestra el porciento de las clases que pertenecen a cada intervalo definido de acuerdo a la cantidad de procedimientos que poseen obteniéndose como resultado que un 81 % de las clases de la solución que tienen entre 1 y 5 procedimientos.

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

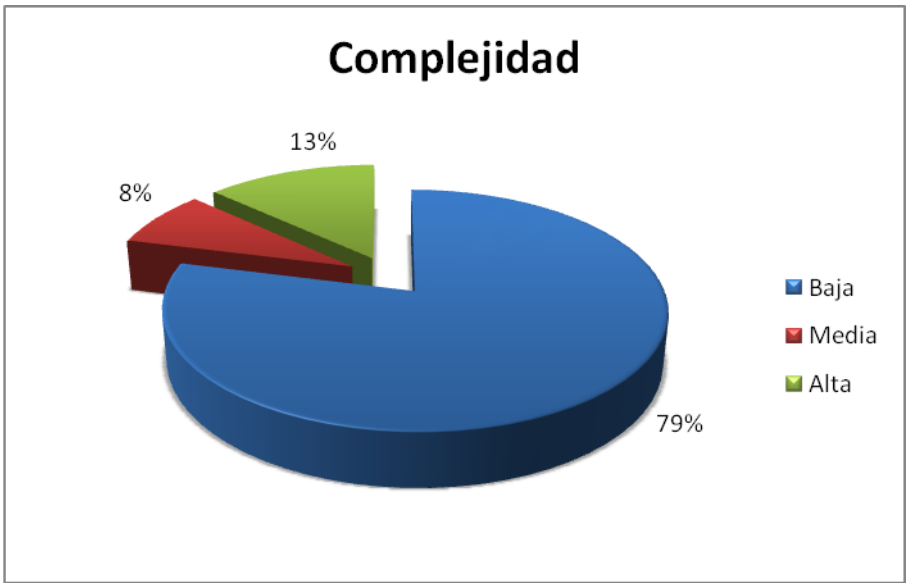

Figura 18: Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Complejidad de Implementación

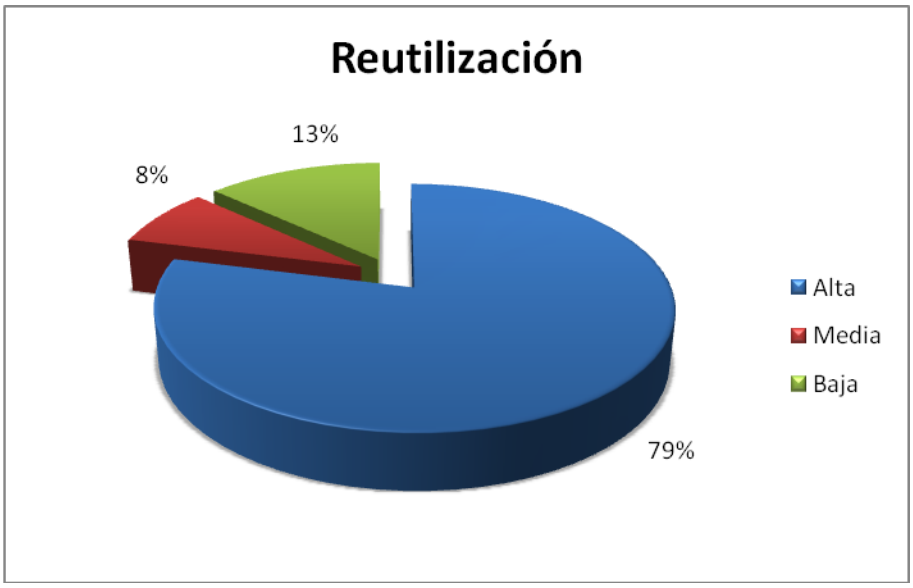

Figura 19: Representación de la incidencia de los resultados de la evaluación de la métrica TOC en el atributo Reutilización.

Haciendo un análisis de los resultados obtenidos en la evaluación del instrumento de medición de la métrica TOC, se puede concluir que el diseño del proceso penal del Sistema de

Informatización de Tribunales tiene una buena calidad ya que los resultados son satisfactorios y se incluyen en los rangos de calidad aceptables dentro del desarrollo de software. De acuerdo con los resultados el 87% tiene buen índice de responsabilidad siendo esto positivo a la hora de realizar cambios en una clase ya que no afecta a la demás. Se obtuvo una solución eficiente con niveles aceptables de reutilización (87%) lo que garantiza que estas clases se puedan utilizar en otros módulos del proyecto. También se obtuvo índices aceptables de complejidad de implementación (87%) y responsabilidad (87%) demostrando así que se le asignaron correctamente las responsabilidades a las clases involucradas en la solución lo cual garantiza una solidez en el diseño del sistema.

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

A continuación se muestran los resultados obtenidos al aplicar la métrica Relación entre Clases (RC). Está métrica se aplicó a 11 clases de la solución. (Ver Anexo 2)

Los resultados de la aplicación de esta métrica están dados por la cantidad de relaciones de uso de una clase con otras. Dicha métrica permite evaluar atributos como el acoplamiento, complejidad de mantenimiento, reutilización y la cantidad de pruebas. Mientras mayor sea las relaciones de uso entre clases mayor será el acoplamiento, la complejidad de mantenimiento y la cantidad de pruebas, mientras que la reutilización disminuye. Después de aplicar dicha métrica se obtuvieron los siguientes resultados:

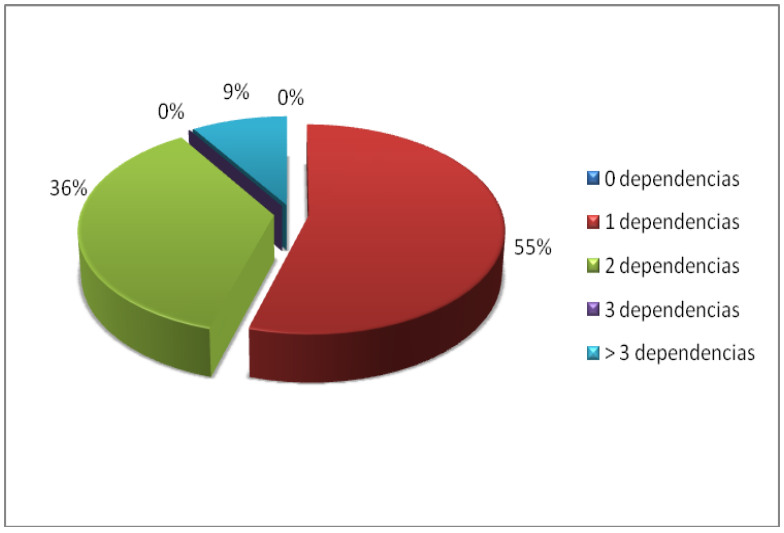

Figura 20: Representación en % de los resultados obtenidos en el instrumento agrupados en los intervalos definidos.

La gráfica anterior muestra los porcientos de las clases en los intervalos definidos de acuerdo a la cantidad de dependencias que tienen las mismas. Se obtuvo que un 91% de las clases poseen entre 1 y 2 dependencias de otras clases lo que favorece el hecho de que al ocurrir un cambio en alguna de las clases, la afectación en las restantes sea de poca importancia.

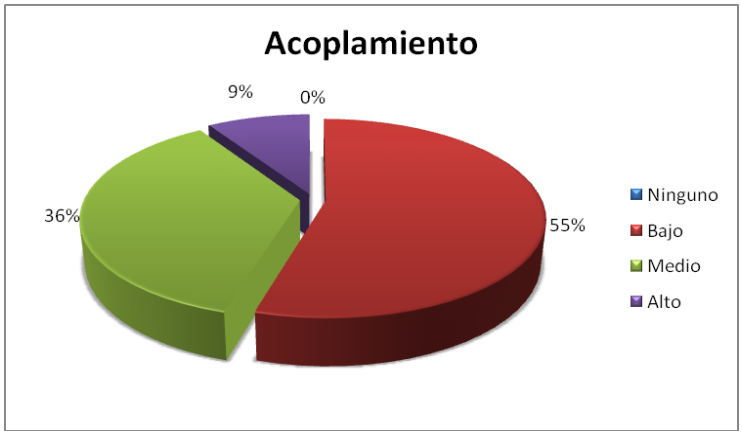

Figura 21: Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Acoplamiento.

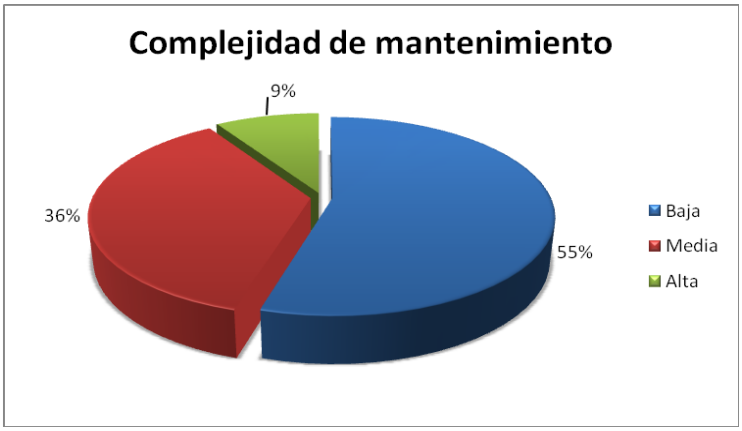

Figura 22: Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Complejidad de Mantenimiento

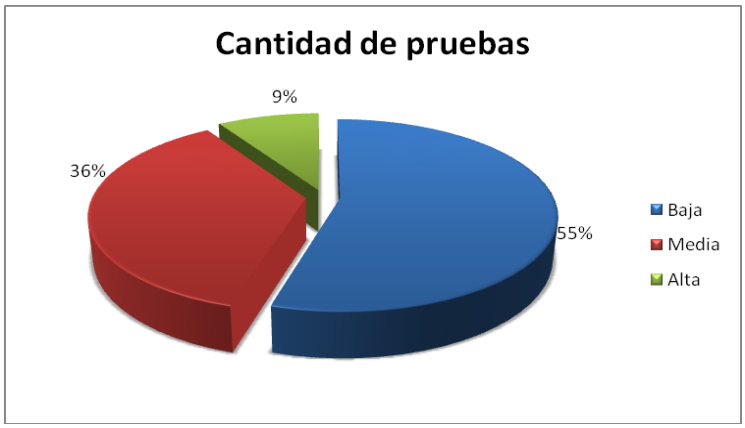

Figura 23 Representación de la incidencia de los resultados de la evaluación de la métrica RC en el atributo Cantidad de Pruebas

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

Haciendo un análisis de los resultados obtenidos en la evaluación del instrumento de medición de la métrica RC, se puede concluir que el diseño del proceso Penal del Sistema de Informatización de Tribunales tiene una buena calidad. El 55% de las clases posee bajos índices en cuanto a acoplamiento y el 36% posee índices medios para ese atributo de calidad, esto posibilita que se realicen modificaciones en el sistema teniendo poco impacto en el mismo y que entre las clases exista un bajo acoplamiento. Por su parte los atributos de calidad complejidad de mantenimiento y cantidad de pruebas se comportan satisfactoriamente en un 91% de las clases que tienen índices aceptables lo que facilita las tareas de corrección, modificación y mantenimiento de la solución. También se obtuvo índices aceptables (91%) de reutilización lo que permite el reuso de las clases y facilita el mantenimiento del sistema.

3.5 Casos de prueba

Con el objetivo de verificar el grado de satisfacción de los requerimientos de software se llevó a cabo el diseño de diferentes casos de prueba para probar el software. Éstos casos de prueba contienen los juegos de datos a usar que son los válidos o esperados y los no válidos o no esperados por el programa. El objetivo de los casos de prueba es forzar al máximo el sistema en los puntos críticos para encontrar fallos y detectar defectos. Las pruebas se deben aplicar durante todo el ciclo de vida del software. El diseño de cada caso de prueba debe ir acompañado del resultado que ha de producir el software al ejecutar dicho caso para detectar un posible fallo en el programa. Los casos de prueba determinan un conjunto de entradas, condiciones de ejecución y resultados esperados para un objetivo particular.

3.6 Pruebas de Caja Blanca

Para realizar la prueba se usó la técnica del camino básico, ya que permite obtener una medida de la complejidad lógica del diseño y usar la misma como guía para la definición de un conjunto de caminos básicos, garantizando que durante la prueba se ejecute al menos una vez cada sentencia del programa.

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

A continuación se analizan y enumeran las sentencias de código de uno de los métodos contenidos en la clase RegacusadoController, específicamente: listarcusadoAction, este por su parte se encarga de cargar en un gridpanel los datos de los acusados que hay guardados en la base de datos. Solo se muestra el resultado de una de las pruebas realizadas a un método en específico, con el objetivo de mostrar cómo es que se realiza este tipo de procedimiento.

```
function listaracusadoAction()
{
if (!isset($_SESSION['Adddelitos']))//1
    {
     $_SESSION['Adddelitos'] = array();//2
    }
$result = array('cantidad_filas' => count($_SESSION['Adddelitos']), 'datos' => $_SESSION['Adddelitos']);//3
$cant=$result['cantidad_filas'];//3
$delitos=$result['datos'][0]['delitos'];//3
for($a=0; $a<$cant; $a++)//4
{
$delitos=$delitos.' '.$result['datos'][$a]['Articulo'];//5
$delitos=$delitos.' '.$result['datos'][$a]['Apartado'];//5
$delitos=$delitos.' '.$result['datos'][$a]['Inciso'];//5
}
$delitos2['delitos'] = $delitos;//6
 array_push($_SESSION['Adddelitos'], $delitos2);//6
 $deli = array('cantidad_filas' => 1, 'datos' => $_SESSION['Adddelitos']);//6
 echo json_encode($deli);//6
 return;//6
}
```

Figura 25: Método listarcusadoAction en el caso de uso Registrar Acusado

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

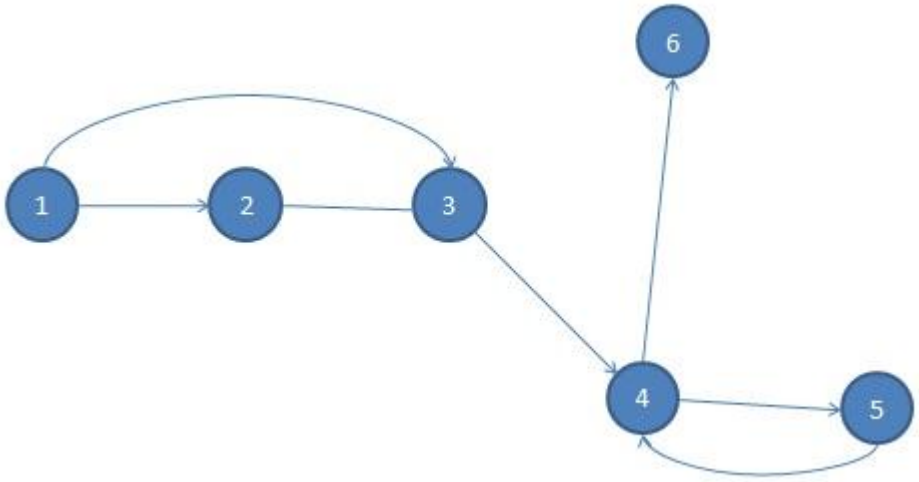

Figura 26: Grafo de flujo asociado al algoritmo listarcusadoAction

Una vez construido el grafo de flujo asociado al procedimiento anterior se determina la complejidad ciclomática, el cálculo es necesario efectuarlo mediante tres vías o fórmulas de manera tal que quede justificado el resultado, siendo el mismo en cada caso:

1. V (G) = (A – N) + 2

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

V (G) = (7 -6) + 2

V (G) = 3.

2. V (G) = P + 1

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

V (G) = 2 + 1

V (G) = 3.

3. V (G) = R

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

V (G) = 3

El cálculo efectuado mediante las fórmulas antes presentadas muestran una complejidad ciclomática de valor 3, de manera que existen 3 posibles caminos por donde el flujo puede circular, este valor representa el número mínimo de casos de pruebas para el procedimiento tratado.

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

Camino básico #1: 1 -- 3 – 4 – 6

Camino básico #2: 1 -- 2 – 3 – 4 – 6

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

Capítulo III: Análisis de resultados

Se procede a ejecutar los casos de pruebas para cada uno de los caminos básicos determinados en el grafo de flujo. Para demostrar cómo se hace el procedimiento se muestra las pruebas hechas a uno de los caminos básicos. Para definir los casos de prueba es necesario tener en cuenta:

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

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

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

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

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

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

Descripción: Se deben cargar en el gridpanel los datos de los acusados que hay guardados en la base de datos.

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

Entrada: El objeto SESSION está creado

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

El resultado obtenido fue correcto ya que al ejecutarse la función y de acuerdo a la condición de ejecución y la estrada de la misma el gridpanel salió sin datos.

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

Condición de ejecución: Para el algoritmo es necesario que en el objeto SESSION en la posición “Adddelitos” existan datos.

Entrada: El objeto SESSION no está creado y no hay datos en la posición “Adddelitos”.

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

El resultado obtenido fue correcto ya que al ejecutarse la función y de acuerdo a la condición de ejecución y la estrada de la misma el gridpanel salió sin datos.

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

Condición de ejecución: Para el algoritmo es necesario que en el objeto SESSION en la posición “Adddelitos” existan datos.

Entrada: El objeto SESSION no está creado y hay datos en la posición “Adddelitos”.

Resultados esperados: Teniendo en cuenta los datos pasados por parámetro se espera que en el gridpanel salgan los datos de los acusados.

El resultado obtenido fue correcto ya que al ejecutarse la función y de acuerdo a la condición de ejecución y la estrada de la misma el gridpanel mostró los datos de los acusados.

3.7 Pruebas de Caja Negra

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

Para la realización de las pruebas de caja negra se empleó la técnica partición de equivalencia que representa una de las más efectivas pues permite examinar los valores válidos e inválidos de las entradas existentes en el software.

Con estas pruebas se descubre de forma inmediata errores que de otro modo requerirían la ejecución de muchos casos antes de detectar el error genérico. Dirige la definición de casos de pruebas que descubran clases de errores, reduciendo el número de casos de prueba que hay que desarrollar. **(Pressman, 2002)**

Se hizo una primera iteración de las pruebas de caja negra donde se detectaron 24 no conformidades que fueron corregidas en su mayoria; al hacer una segunda iteración se detectaron 10 no conformidades que fueron corregidas en su totalidad ya que al hacer una tercera iteración no se detecto ninguna.

Figura 27: Resultados de las pruebas de caja negra

En la gráfica anterior se muestran los resultados obtenidos durante la realización de las pruebas de caja negra, estas arrojaron resultados satisfactorios a los efectos de la implementación del sistema.

Caso de prueba de Caja Negra para validar el caso de uso Dar por Presentado el EFP.

Descripción General

El caso de uso se inicia cuando la Secretaria necesita dar por presentado el EFP. Consiste en que la Secretaria selecciona el trámite dar por presentado el EFP y llena los campos requeridos para la presentación. El caso de uso termina con la presentación del EFP.

Condiciones de Ejecución

Que se haya registrado en el sistema un EFP

Secciones a probar en el Caso de Uso

Nombre de la sección	Escenarios de la sección	Descripción de la

		funcionalidad
SC 1: Dar por presentado el EFP.	EC 1.1: Dar por presentado el EFP exitosamente.	Consiste en que la Secretaria selecciona el trámite dar por presentado el EFP y llena los campos requeridos para la presentación.
	EC 1.2: Datos incorrectos	No selecciona un EFP. Muestra un mensaje informando que debe seleccionar un EFP.
	EC 1.3: Cancelar la operación.	Cancela la operación. Cierra la interfaz. Terminando así el caso de uso.

Descripción de variable

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	Número EFP	Campo de texto	No	Se inserta el número de EFP para filtrar una búsqueda.

SC 1 Dar por presentado el EFP

Escenario	Número EFP	Respuesta Esperada	Resultado de la Prueba	Flujo Central
EC 1.1: Dar por presentado el EFP exitosamente.	20	El sistema filtra una búsqueda entre todos los números EFP y muestra si existe la pieza igual a la escrita en el campo de texto.	Satisfactorio	1. Inicio. 2. Penal. 3. Dar por presentado el

Escenario	Número EFP	Respuesta Esperada	Resultado de la Prueba	Flujo Central
	10	El sistema filtra una búsqueda entre todos los números EFP y muestra si existe la pieza igual a la escrita en el campo de texto. En este caso no la muestra porque no existe.	Satisfactorio	EFP.
EC 1.2: Datos incorrectos	-	Al poner el signo menos el sistema muestra todos los datos existentes en la base de datos. Esto no debería suceder.	No Satisfactorio	1. Inicio. 2. Penal. 3. Dar por presentado el EFP.
EC 1.3: Cancelar la operación.	NA	Se cancela la operación.	Satisfactorio	1. Inicio. 2. Penal. 3. Dar por presentado el EFP.

3.8 Conclusiones

En el presente capítulo se evaluó el cumplimiento de las funcionalidades del sistema de acuerdo al análisis de las métricas, pruebas de caja negra y pruebas de caja blanca que fueron aplicadas con el objetivo de alcanzar la calidad requerida. El análisis de los diferentes resultados fue documentado y el mismo evidencia la erradicación de las no conformidades e irregularidades que se encontraron.

CONCLUSIONES

Con el desarrollo del trabajo de diploma y el tiempo dedicado a la investigación, implementación y prueba de la solución propuesta, se llegó a las siguientes conclusiones:

- Se realizó un estudio de los conceptos fundamentales que se manejan en torno al subsistema penal hasta la sentencia para lograr entender los procesos que se realizan.
- Se realizó un estudio sobre la metodología de desarrollo de software, lenguajes de programación y herramientas definidas por el equipo de arquitectura, teniéndose presente las necesidades de los clientes y usuarios finales permitiendo justificar la utilización de las mismas en el desarrollo de este trabajo.
- Se obtuvieron los artefactos correspondientes definidos en el modelo de diseño y en el modelo de despliegue, obteniéndose como resultado, los diagramas clases, diagramas se secuencia, los diagramas de componentes y el diagrama de despliegue.
- Se probaron las funcionalidades a través de las métricas, pruebas de caja blanca y pruebas de caja negra, con el objetivo verificar la calidad del software.

RECOMENDACIONES

Con la realización de este trabajo se recomienda:

- ✓ Continuar la implementación del subsistema Penal hasta la Sentencia.
- ✓ Comenzar la implementación del subsistema Penal después de la Sentencia.

BIBLIOGRAFÍA

1. **Alcalá, Universidad de.** Patrones de diseño: GRASP. [Online] ftp://www.cc.uah.es/pub/Alumnos/I.Informatica/.../PatDisGRASP.pdf.
2. **Alfonso, Jorge. 2010.** Tratando de enterderlo:Patrones de Diseño. [Online] Enero 26, 2010. http://tratandodeentenderlo.blogspot.com/2010/01/.
3. **Alva, Eduardo Rivera. 2012.** Scribd. *Arquitectura de Software II. Diagrama de Componentes y Despliegue.* [En línea] 2012. http://www.scribd.com/doc/7884665/Arquitectura-de-Software-II-Diagrama-de-Componentes-y-Despliegue.
4. **Alvarez, Miguel Angel. 2001.** Desarrollo Web. [Online] Julio 24, 2001. http://www.desarrolloweb.com.
5. **Andux, Yadira Piñera. 2010.** Formalización y estandarización de la documentación técnica de la arquitectura tecnológica del Marco de Trabajo Sauxe versión 2.0. 2010.
6. **Aprendizaje, Entorno Virtual de. 2012.** Ingenieria de Software II. *Conferencia 6: Fase de Construcción. FT Implementación. Modelo de implementación.* [Online] 2012. http://eva.uci.cu/file.php/160/Curso_2010-2011/Semana_8/Conferencia_6/Materiales_Basicos/Implementacion.pdf.
7. **Baryolo, Gomez Oiner, Tenrero, Cabrera Mariela and Silega, Martinez Nemuris. 2008.** *Plantilla Registro de la propiedad intelectual(Sauxe).* 2008.
8. **Booch, Grady. 2000.** *UML-El lenguaje de Modelado,Manual de Referencia.* s.l. : Madrid Addison Wesley, 2000. ISBN 84-7829-036-2.
9. **Budd, Timothy. 1991.** *An introduction to Object-Oriented Programming.* 1991. ISBN: 0-201-54709-0.
10. **Corporation, Oracle. 2012.** NetBeans. [Online] 2012. http://netbeans.org/.
11. **Ceballos, Alejandra Duque.** Algunos Fundamentos Teóricos de la Programación de Computadores. [En línea] http://kiragymcode.googlecode.com/files/articulo.pdf.
12. **Corporation, Microsoft. 2008.** Microsoft. [Online] Marzo 7, 2008. http://www.microsoft.com/spain/windowsserver2008/web/web_as.mspx#E5.

Bibliografía

13. **Desarrollo, Comunidad de. 2006-2010.** ExtJs en Español. [Online] 2006-2010. http://www.extjs.es/.

14. **Desarrrollo Web.** Metricas de software. [Online] [Cited: Marzo 31, 2012.] http://www.desarrolloweb.com/articulos-copyleft/articulo-metricas-de-software.html

15. **E-lemental. 2010.** [Online] 2010. http://www.e-lemental.com.pe/que-es-una-aplicacion-web/.

16. **Flanagan, David. 2002.** *JavaScript: The Definitive Guide (4 ª Edición edición).* 2002. ISBN 0-596-00048-0.

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

18. **García, Félix Óscar Rubio and Santos, Crescencio Bravo. 2009-2010.** Escuela de Ingenieria Civil Informática. [Online] 2009-2010. http://www.eici.ucm.cl/Academicos/R_Villarroel/descargas/ing_sw_1/Metodologias.pdf.

19. **Gamma, Erich, et al. 1995.** *Design Patterns: Elements of Reusable Object-Oriented Software.* 1995. ISBN 0-201-63361-2.

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

21. **Inc, Zend Technologies. 2005-2010.** Manual de Zend Framework en español. [Online] 2005-2010. http://www.manual.zfdes.com/es/.

22. **J. Solís, Camilo, A. Cabrera, Armando y G. Figueroa, Roberth.** Entorno Virtual de Aprendizaje. *METODOLOGÍAS TRADICIONALES VS. METODOLOGÍAS ÁGILES.* [En línea] http://eva.uci.cu/file.php/161/Documentos/Materiales_complementarios/UD_1_Procesos/Metodologias/METODOLOGIAS_TRADICIONALES_VS._METODOLOGIAS_AGILES.pdf.

23. **Jurado, Jose Luis.** PATRON DE DISEÑO MVC (Modelo Vista Controlador). *Comunidad Virtual del Departamento de Sistemas. Universidad del Cauca - Popayán - Colombia.* [Online] pis.unicauca.edu.co/moodle/file.php/291/Patron_Diseno_MVC.pdf.

24. **Leopoldo, Carlos. 2007.** ZendFramework, introducción. [Online] 2007. http://techtastico.com/post/zend-framework-una-introduccion/.

25. **Mata, Manel Pérez. 2009.** ¿Qué es Doctrine ORM? . [Online] Julio 3, 2009. http://www.tecnoretales.com/programacion/que-es-doctrine-orm.

Bibliografía

26. **Mikkonen, Tommi and Taivalsaari, Antero. 2007.** *Using JavaScript as a Real.* 2007.

27. **Paradigm, Visual. 2011.** Visual Paradigm. [Online] 2011. http://www.visual-paradigm.com.

28. **Pérez, Isaías Carrillo, González, Rodrigo Pérez and Martín, Aureliano David Rodríguez. 2008.** Metología de desarrollo de software. [Online] 2008. https://code.google.com/p/solusoft-g11/downloads/list.

29. **PHP. 2001-2011.** Hypertext Preprocessor. [Online] 2001-2011. http://www.php.net/.

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

31. **Pressman, Roger S. 2002.** *Ingenieria de Software,un enfoque práctico.* s.l. : McGraw-Hill Companies, 2002. ISBN:8448132149.

32. **Pupo, Yanisleydi Cañete. 2010.** *Libro de Ayuda del Marco de Trabajo Sauxe 2.0.* Ciudad de la Habana : s.n., 2010.

33. **Quintero, Héctor Ramón Peñaranda. 2002.** Observatorio para la CiberSociedad. *La informática jurídica: mecanismo de gestión de la información jurídica.* [En línea] 2002. http://www.cibersociedad.net/congreso/comms/g13penaranda2.pdf.

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

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

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

37. **ServerGrove. 2010.** Doctrine. [Online] 2010. http://www.doctrine-project.org/.

38. **Technologies, Zend. 2006-2012.** Zend Framework. [Online] 2006-2012. http://www.framework.zend.com.

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