Universidad de las Ciencias Informáticas

Trabajo de diploma para optar por el título de
Ingeniero en Ciencias Informáticas

Título: Diseño e implementación de una solución informática para los procesos de Prueba Pericial y de Presunción del Sistema de Informatización para la Gestión de los Tribunales Populares Cubanos

Autores: Alain Reyes Salas
Yoannys Gustavo Dueñas Pérez

Tutor: Ing. Arianna Leyva Campos
Co-tutor: Ing. Reinier Fernández Coello

La Habana, Cuba
Curso: 2013-2014

“El trabajo va a ocupar gran parte de tu vida, y la única forma de estar realmente satisfecho es hacer lo que consideren un trabajo extraordinario. Y el único camino para lograrlo es amando lo que hacen. Si no lo han descubierto aún, sigan intentando. No se conformen”.

Steve Jobs

Declaración de autoría

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

Autores:

_____ _____

Alain Reyes Salas Yoannys Gustavo Dueñas Pérez

Tutor: Co-tutor:

_____ _____

Ing. Arianna Leyva Campos Ing. Reinier Fernández Coello

Datos de contacto

Tutor: Ing. Arianna Leyva Campos

Co-tutor: Ing. Reinier Fernández Coello

Síntesis del Tutor: (alcampos@uci.cu) Graduada de Ingeniero en Ciencias Informáticas en el año 2010 en la Universidad de las Ciencias Informáticas (UCI). Especialista General interno. Ha desempeñado el rol de programadora en la actividad de desarrollo en el proyecto Tribunales. Ha cursado varios diplomados entre los que destacan el de Gobierno Electrónico. Tiene una experiencia acumulada en el área de desarrollo de aproximadamente 3 años.

Síntesis del Co-tutor: (rfcoello@uci.cu) Graduado de Ingeniero en Ciencias Informáticas en el año 2012 en la Universidad de las Ciencias Informáticas (UCI). Recién Graduado en Adiestramiento con 4 años de experiencia en la producción, desempeñándose como programador en el proyecto Tribunales desde sus inicios. Ha cursado un total de 8 cursos de postgrados en su etapa de adiestramiento entre ellos se encuentra el postgrado de Informática Jurídica.

Resumen

La tramitación de los procesos de Prueba Pericial y de Presunción en los Tribunales Populares Cubanos presenta limitaciones que los dilatan y afectan su consecutividad, atentando contra un mejor desarrollo y conclusión de los mismos. El presente trabajo describe el diseño e implementación de una solución informática que contribuye a su agilización y garantiza su consecutividad.

El estudio basó su fundamentación teórica en la caracterización de las principales tecnologías, tendencias de desarrollo de software y herramientas que se emplearon en el desarrollo de la solución informática. La evaluación de los resultados obtenidos se realizó a partir del empleo de métricas y de pruebas de software.

Los impactos más relevantes de la solución informática que se obtuvo como resultado del presente trabajo fueron: control, cumplimiento y alerta del vencimiento de términos, informatización integral de la actividad jurisdiccional para los procesos de Prueba Pericial y de Presunción, celeridad en la tramitación de dichos procesos, y el apoyo a la decisión de los jueces en cuanto a los trámites que deben priorizar y en la selección de las fechas para la celebración de los actos procesales perteneciente a los mismos.

Índice de contenido

Introducción

Hoy en día los avances tecnológicos influyen significativamente en la construcción de la sociedad del conocimiento, revolucionando de esta manera la forma de vivir y comunicarse, así como las formas de resolver problemas y satisfacer necesidades en el ámbito empresarial y social. La informática desempeña un papel fundamental en el tratamiento automatizado de la información, favoreciendo la disminución de los costos, la agilización de la toma de decisiones y el logro de una cultura organizacional más eficiente.

La rama del Derecho es parte interesada en la aplicación de los beneficios que supone la informatización de la sociedad, es por esto que se desarrolla la Informática Jurídica, como técnica interdisciplinaria que tiene por objeto el estudio los conocimientos de la informática aplicables al tratamiento de información jurídica.

En Cuba urge la aplicación de la Informática Jurídica en los TPC. Por ello se desarrolla el Sistema de Informatización para la Gestión de los Tribunales Populares Cubanos (SITPC) en el Centro de Gobierno Electrónico (CEGEL) de la Universidad de las Ciencias Informáticas (UCI).

Este sistema tiene como objetivo la informatización de la actividad jurisdiccional en el marco de los TPC, lo que traería consigo impactos positivos para el trabajo en los tribunales y el sector público, tales como: el aumento de la calidad de la tramitación de los procesos, reportes estadísticos en tiempo real, almacenamiento seguro y organizado de la información, facilidad de acceso o consulta por los abogados y el tribunal actuante en un proceso determinado, esto último se restringe según los niveles de acceso definidos por el cliente, garantizando la seguridad.

El SITPC se divide en los subsistemas Administración y Gobierno, Común, Penal, Económico, Administrativo, Laboral y Civil. El subsistema Común incluye los procesos de negocio correspondientes a todos los procedimientos que son comunes en el resto de los subsistemas.

Entre estos procesos se encuentra el proceso de Pruebas que es el más complejo y extenso, y se implementa a partir de los diferentes tipos de pruebas que se realizan en los tribunales. Dicho proceso está dividido en varios subprocesos cuyos nombres hacen referencia a los tipos de pruebas, entre ellos se encuentran los subprocesos de Prueba Pericial y de Presunción.

Las Pruebas Periciales se basan en el dictamen pericial que emiten una o varias personas expertas en materias no jurídicas que permite al juez valorar el conocimiento

y apreciación de los hechos. En cambio las Pruebas de Presunción consisten en deducir, dado un hecho base, un hecho consecuencia, es decir, la averiguación de un hecho desconocido deduciéndolo de otro conocido.

Para estar al tanto de cada una de las resoluciones que se van dictando durante la realización de los procesos de Prueba Pericial y de Presunción y en consecuencia actuar, los abogados deben consultar frecuentemente el expediente[1] sobre el que están trabajando. La manera en la que se realiza esta actividad es desfavorable por dos cuestiones fundamentales, una es la existencia de un ente intermediario, en este caso el secretario que es el encargado de buscar y dar la información que se le solicita. Esto aumenta su carga de trabajo lo que se agrava con el número de procesos tramitándose paralelamente. El segundo inconveniente se debe a que los abogados tengan que trasladarse hasta el tribunal, lo que implica que inviertan horas laborales en esta tarea; ocupando el 60% del tiempo hábil de la realización de un proceso y dilatándose así su conclusión.

Los actos procesales son hechos, y a veces también omisiones, que influyen en la relación procesal. Esta última puede ser dividida o descompuesta en los distintos actos que la constituyen. (Levene, 1993)

En la LPCALE[2] se establecen términos procesales que fijan el plazo de tiempo para la realización de un acto procesal. Los términos procesales fueron establecidos con el objetivo de garantizar el derecho a la no dilación indebida del proceso, mediante la delimitación del tiempo máximo para la ejecución de un acto procesal.

Durante la tramitación de estas pruebas el secretario es el encargado de controlar el vencimiento de los términos procesales para la actuación de los abogados y del tribunal actuante. Las limitaciones de la concepción de esta actividad se deben principalmente al volumen de información que debe consultar el secretario para calcular las fechas de vencimiento de los términos para cada acto procesal y el número elevado de estos últimos llevándose a la par en el tribunal.

Lo explicado anteriormente condiciona que se desaprovechen los recursos humanos y que el enfoque de la realización de esta actividad se centre solamente en el cálculo de la fecha de vencimiento para cada acto procesal en lugar de enfocarse en su seguimiento individual. Esto priva a los secretarios, los abogados y al tribunal actuante de alertas diarias que le notifiquen el tiempo del que van disponiendo para realizar sus actuaciones, lo que influye negativamente en el cumplimiento de los términos procesales.

[1] Registro formal de documentos que es la memoria procesal escrita y común para todos los actores en el proceso judicial.

[2] Ley de Procedimiento Civil, Administrativo, Laboral y Económico.

El cúmulo de información que se genera durante los procesos de Prueba Pericial y de Presunción y el número de casos que se llevan paralelamente dificulta la búsqueda y consulta de los documentos generados (actas, providencias, resoluciones, escritos, etc.), así como su análisis estadístico. Estos documentos se realizan de forma manual, provocando que estén expuestos a la introducción de errores de repetición en el número de los asientos, tachaduras, saltos en los espacios de las anotaciones y borraduras. En ocasiones existen inconsistencias de la información entre documentos de un mismo caso, por ejemplo: números de expedientes, datos de los involucrados, elemento negativo en la emisión de documentos oficiales.

Actualmente los abogados pueden presentar en cualquier momento del proceso el escrito de solicitud de aclaración del dictamen entregado por los peritos. Esto provoca que el juez deba disponer sobre escritos que por extemporáneos no debieron presentarse violándose el principio de consecutividad del proceso y generando trabajo innecesario para el tribunal actuante del expediente, lo que influye negativamente en la agilización del proceso. De igual forma ocurre en la presentación del escrito de ampliación o adición de la prueba pericial por los abogados y en la entrega de los informes periciales y de ampliación de dictamen por los peritos.

Teniendo en cuenta los aspectos planteados anteriormente y partiendo de que ya han sido definidos y especificados los requisitos del subsistema Común surge como **problema a resolver**: ¿Cómo contribuir a la agilización y consecutividad de los procesos de Prueba Pericial y de Presunción en el Sistema de Informatización para la Gestión de los Tribunales Populares Cubanos?

Teniendo, para ello, como **objeto de estudio**: La Informática Jurídica de Gestión. De esta forma, se determina como **objetivo general**: Desarrollar el diseño e implementación de una solución informática para los procesos de Prueba Pericial y de Presunción del SITPC que contribuya a la agilización y consecutividad de los mismos.

Se identifica como **campo de acción**: El diseño e implementación de procesos jurisdiccionales: Prueba Pericial y de Presunción.

Teniendo como referencia el problema a resolver y el objetivo general se plantea la siguiente **idea a defender:** con el desarrollo del diseño e implementación de una solución informática para los procesos de Prueba Pericial y de Presunción del Sistema de Informatización para la Gestión de los Tribunales Populares Cubanos se contribuirá a la agilización y consecutividad de los mismos.

Para dar complimiento al objetivo se definen las siguientes **tareas de investigación**:

- Caracterización del proceso de Pruebas, específicamente los procesos de Prueba Pericial y de Presunción.

- Caracterización de la metodología de desarrollo de software, los lenguajes de programación y las herramientas de desarrollo vinculado al desarrollo web.
- Análisis de los paradigmas de Programación Orientado Objeto y Estructurados para dar respuesta a la propuesta de solución.
- Análisis de la arquitectura definida por el proyecto y su aplicación en la propuesta de solución.
- Análisis y selección de los Patrones de diseño más factibles para esta propuesta de solución.
- Elaboración de los diagramas, como producto de la metodología de desarrollo de software, en las etapas de Diseño e Implementación que se utilizan en la propuesta de solución.
- Desarrollo de la implementación al diagrama de clases que describe la propuesta de solución.
- Selección y aplicación de las métricas para la validación del diseño.
- Diseño de Casos de Prueba para la validación de a propuesta de solución.
- Realización de las pruebas de software a la propuesta de solución.
- Resolución de las No conformidades en las diferentes iteraciones de pruebas a la solución informática.
- Análisis de la validación de las pruebas de software aplicadas a la propuesta de solución.

Para llevar a cabo las tareas de la investigación se emplearon los siguientes métodos de **Investigación + Desarrollo.**

Métodos lógicos: son todos aquellos que se basan en la utilización del pensamiento en sus funciones de deducción, análisis y síntesis. Se clasifican para su estudio en Métodos lógicos formales y en Métodos lógicos de soporte.

Los **métodos lógicos de soporte** utilizados fueron:

- **Modelación:** Se utiliza para la creación de los siguientes diagramas y modelos: diagrama de procesos de Prueba Pericial y de Presunción, diagramas de clases del diseño, diagramas de paquetes, diagramas de secuencia, modelo de datos y diagrama de despliegue.

Métodos empíricos: son los que se aproximan al conocimiento del objeto mediante su conocimiento directo y el uso de la experiencia. Se clasifican para su estudio en Métodos cuantitativos y Métodos cualitativos.

Los **métodos empíricos cuantitativos** se clasifican en Métodos experimentales controlados y en Métodos observacionales. Estos últimos recogen los datos relevantes a medida que se desarrolla el proyecto, de ellos se utilizó el siguiente:

- **Estudio de campo**: Se emplea al realizar una visita al Tribunal Provincial Popular de la Habana con el objetivo de observar y analizar cómo son llevados a cabo los procesos de Prueba Pericial y Presunción y determinar así las limitaciones que estos presentan.

Los **métodos empíricos cualitativos** utilizados fueron:

- **Investigación – Acción**: Incluye el diagnóstico del problema, intervención de acción y aprendizaje reflexivo. Se usa al analizar detalladamente con los jueces los problemas que puede presentar el Sistema de Informatización para la Gestión de los Tribunales Populares Cubanos (SITPC), en cuanto a la tecnología utilizada en los tribunales y a problemas que ya se han presentado, con el objetivo de mitigar dichos problemas en la solución a desarrollar.
- **Etnografía**: Se aplica al realizar un estudio de las personas a las que va dirigido la solución informática, con el objetivo de adecuarlo a sus necesidades y características particulares.

La presente investigación consta de 3 capítulos que abordan los temas fundamentales distribuidos de la siguiente manera:

Capítulo 1: Fundamentación Teórica. En este capítulo se recoge la fundamentación teórica que sustenta el progreso de la investigación para el desarrollo de la solución propuesta. En el mismo se plantean un conjunto de definiciones relacionadas con el objeto de estudio y el campo de acción y se describen la metodología de desarrollo de software y tecnologías definidas por el equipo de arquitectura del proyecto.

Capítulo 2: Solución Propuesta. En este capítulo se propone la solución técnica de la investigación. Se presenta el modelo de diseño, conformado por los diagramas de clases y los diagramas de secuencia y además el modelo de implementación con el diagrama de despliegue. Se exponen además los patrones de diseño utilizados en la solución propuesta.

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

Capítulo 1: Fundamentación teórica

1.1 Introducción.

El presente capítulo contiene una descripción de la Informática Jurídica como objeto de estudio, específicamente la Informática Jurídica de Gestión, aplicadas a los procesos de Prueba Pericial y de Presunción. Además se caracteriza la metodología de desarrollo de software, así como los lenguajes de programación y las herramientas de desarrollo vinculadas a la propuesta de solución. De igual forma aborda los patrones de diseños que podrían ser utilizados en la investigación, así como un análisis de los paradigmas de programación y de la arquitectura definida por el proyecto y su aplicación en este trabajo. Por último se muestra una caracterización de las métricas de diseño y las pruebas de software a tener en cuenta en la propuesta de solución.

1.2 Informática Jurídica.

La Informática Jurídica consiste en una ciencia que forma parte de la Informática, es la especie en el género, y se aplica sobre el Derecho; de manera que, se dé el tratamiento lógico y automático de la información legal.

Es una ciencia que estudia la utilización de aparatos o elementos físicos electrónicos, como la computadora, en el Derecho; es decir, la ayuda que este uso presta al desarrollo y aplicación del Derecho. En otras palabras, es ver el aspecto instrumental dado a raíz de la Informática en el Derecho. Descubriendo así las técnicas y conocimientos para la investigación y desarrollo de los conocimientos de la Informática para la expansión del Derecho, a través de la recuperación jurídica, como también la elaboración de material lingüístico legal, instrumentos de análisis, y en general el tratamiento de la información jurídica.

La Informática Jurídica como disciplina dentro de la Iuscibernética -que constituye el marco mediato entre la relación Derecho e Informática, y que la misma forma parte de la Cibernética como ciencia general-, han hecho posible el desarrollo de ciencias que al mezclarse posibilitan un mejor desarrollo y tratamiento de la comunicación de las mismas, como se refleja en esta relación entre el Derecho e Informática de las cuales se desprenden disciplinas como lo son la Informática Jurídica, el Derecho Informático, la Jurimetría y la Modelística Jurídica. (Raibal, 2011)

1.2.1 Clasificaciones de la Informática Jurídica.

- Informática Jurídica Documental, que consiste en la creación y recuperación de información jurídica como leyes y doctrinas.
- Informática Jurídica de Control y Gestión, que describe el desarrollo de actividades jurídico-adjetiva de las cuales se pudiesen obtener actos jurídicos como contratos, certificaciones y mandatos judiciales.
- Informática Jurídica Metadocumental, a través de la cual se ayuda o apoya en la toma de decisiones, en la educación, investigación, redacción y previsión del Derecho. (Téllez Valdes, 1987)

Teniendo en cuenta que la presente investigación tiene como objeto de estudio la Informática Jurídica de Gestión, a continuación se muestra una caracterización de dicho objeto de estudio, lo que es fundamental para el desarrollo de la propuesta de solución.

1.2.2 Informática Jurídica de Gestión.

La Informática Jurídica de Gestión está encaminada a organizar y controlar la información jurídica de documentos, expedientes y libros, mediante la aplicación de programas de administración que permitan crear identificadores y descriptores para la clasificación de dicha información. Se utiliza mayormente para llevar el seguimiento de trámites y procesos con el objetivo de mantener actualizada la información y llevar un control eficiente de la misma. Su uso en las oficinas relacionadas con el derecho a nivel público (tribunales que administran justicia), permite que un sistema informático efectúe el control de los trámites de actuación repetitiva, así como de los expedientes allí radicados y de los pasos obligados en las diferentes etapas del proceso, con la emisión de los documentos correspondientes en cada caso.

Una vez teniendo conocimiento del objetivo de la Informática Jurídica de Gestión, a continuación se abordará su aplicación en órganos jurisdiccionales.

1.2.3 Informática Jurídica de Gestión en órganos jurisdiccionales.

El uso de la Informática Jurídica de Gestión en órganos jurisdiccionales ha impulsado su desarrollo durante las últimas décadas. Las actividades automatizadas a nivel de la judicatura son numerosas y variadas: desde la formulación agendaria de jueces y magistrados hasta la redacción automática de textos jurídicos a manera de sentencias. Hay una enorme cantidad de acciones desarrolladas en juzgados, tribunales y cortes, que han sido objeto de estudio, análisis y automatización; un ejemplo lo constituye la aceptación, registro e indicación de competencia y seguimiento de los expedientes. Por otra parte, las diferentes fases del proceso pueden ser conocidas en cualquier

momento, permitiendo conocer el estado del juicio, así como el lugar donde se encuentra el expediente. (Téllez Valdes, 1987)

Partiendo del campo de acción de la presente investigación, a continuación se abordan los procesos jurídicos de Prueba Pericial y de Presunción, que mediante la propuesta de solución se les pretende aplicar la Informática Jurídica de Gestión.

1.2.3.1 Proceso de Prueba Pericial.

A continuación se define el concepto de Prueba Pericial mediante la opinión de dos autores:

Prueba Pericial: La prueba pericial es aquella que aporta al proceso, mediante el dictamen pericial, conocimientos científicos, artísticos, técnicos o prácticos que permiten al juez valorar la existencia de hechos, la manera de ser de éstos, o que le permite conocer el contenido o sentido de otras pruebas practicadas en el seno del procedimiento judicial. (…)

La pericia es, ciertamente, una actividad plenamente procesal, en virtud de la cual una o varias personas expertas en materias no jurídicas, elaboran y transmiten al tribunal información especializada dirigida a permitir a éste el conocimiento y apreciación de hechos y circunstancias fácticas relevantes y controvertidas en el proceso. (Martínez.-Ltdo., 2008)

Como se plantea anteriormente el proceso de Prueba Pericial se basa en el dictamen pericial que emiten una o varias personas expertas en materias no jurídicas que permite al juez valorar el conocimiento y apreciación de los hechos.

Actualmente en los TPC este proceso comienza con la presentación del escrito de solicitud de pruebas que contiene la especialidad, el propósito y los peritos a practicarla, este último es opcional. A continuación el juez debe pronunciarse en el término correspondiente sobre la solicitud de prueba. En el caso de subsanación explicará el motivo y fijará un plazo para que las partes corrijan los elementos señalados mientras que en el caso de admisión deberá registrar los peritos conocidos o librar oficio solicitando peritos si no han sido propuestos aún y señalar la fecha de comparecencia en el tribunal. Finalmente se generan las citaciones a los involucrados.

Luego se celebra el acta de juramento de cargo en el que los peritos deben comprometerse a ser imparciales durante el proceso y además se fija el plazo para que estos entreguen el dictamen pericial. Más adelante cuando los peritos hacen entrega del dictamen al tribunal el secretario se encarga de registrar el informe pericial y de notificar al juez.

A continuación el juez da por recibido el informe y los abogados pueden presentar un escrito de solicitud de aclaración, sobre el que el juez dispondrá más adelante,

siempre en el término establecido y teniendo en cuenta el calendario de días no hábiles establecido en el tribunal. Si los abogados no registraran un escrito de solicitud de aclaración se da por practicada la prueba. En caso de ser admitido el escrito de solicitud de aclaración el próximo paso sería celebrar el acta de ampliación de dictamen que tiene como objetivo que los peritos aclaren elementos técnicos del dictamen entregado anteriormente. Una vez celebrado el acto se da por practicada la prueba.

1.2.3.2 Proceso de Prueba de Presunción.

Además de practicar Pruebas Periciales el tribunal actuante puede practicar Pruebas de Presunción, a continuación se define el concepto de estas:

Prueba de Presunción: Las presunciones son una prueba indirecta, que consiste en deducir, partiendo de un hecho base, un hecho consecuencia. Puede por ello, ser definida, en términos generales, como la averiguación de un hecho desconocido, deduciéndolo de otro conocido.

En el caso de estas pruebas, actualmente en los TPC, una vez presentado el escrito de solicitud el juez dispone sobre el mismo teniendo en cuenta el término establecido, finalizando así el subproceso.

Para un mayor entendimiento de la descripción de los procesos antes mencionados, a continuación se muestra en la figura 1 el diagrama de ambos procesos.

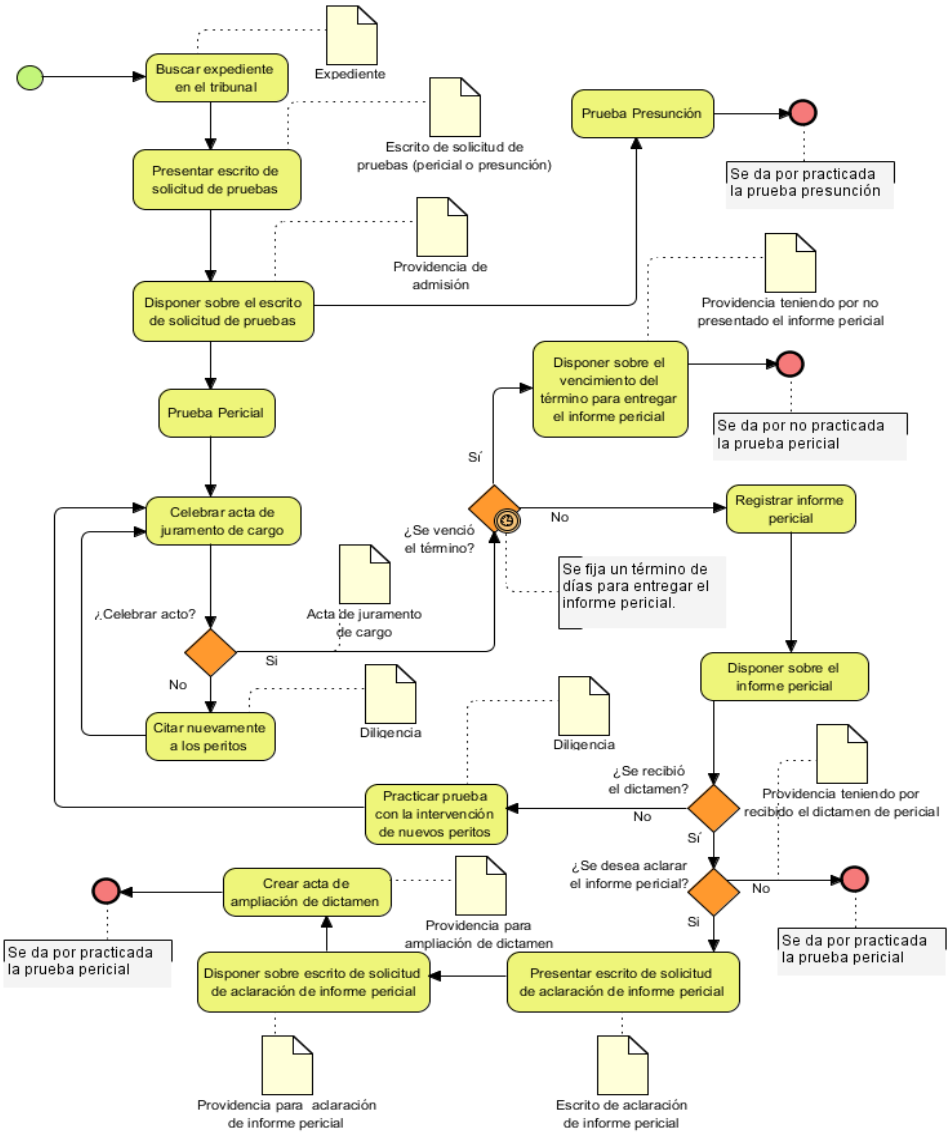

Figura 1: Diagrama de procesos de prueba pericial y presunción.

Se concluye que la Informática Jurídica abre una serie de facilidades en el control y gestión de los asuntos judiciales, como por ejemplo, el tratamiento automático de la información a nivel judicial que podría enfocarse desde muchos puntos de vistas, tales como: el registro de documentos, tramitación de expedientes, estadísticas, control de recursos humanos, pero se presenta también la posibilidad de establecer sistemas de redes que unan a todo este cuerpo judicial, de manera tal que cuando se hable de la informatización de un ente jurisdiccional, no se tome a este en su singular sentido, sino en el de estimular la creación de una estructura judicial entendida como un todo.

Por ejemplo, cuando se habla acerca de la búsqueda y consulta de información jurídica, esta conlleva a tomar en cuenta distancias y tiempo, las cuales se podrían sustituir por el flujo electrónico, en el sentido de que esos procedimientos eviten el traslado por carro y/o avión, ya que quedarían las posibilidades de transmitir esa

información de un juzgado a otro de manera inmediata, a través de redes de información, ahorrando tiempo, dinero y recursos humanos.

Una vez abordados los conceptos y definiciones implicadas en el objeto de estudio y en el campo de acción, es necesario tener en cuenta la metodología que guiará el proceso de desarrollo de software de la propuesta de solución. Es por esto que en el siguiente epígrafe se caracteriza la metodología seleccionada por el equipo de arquitectura del proyecto al que pertenece la presente investigación.

1.3 Metodología de desarrollo.

Las metodologías de desarrollo de software "son todas las actividades necesarias para transformar los requisitos de un usuario en un sistema de software". (Jacobson, y otros, 2000)

Brindan una guía para el proceso de desarrollo de software, haciendo dicho proceso más disciplinario, predecible y eficiente. Su objetivo principal es desarrollar un producto con mayor calidad. Las metodologías deben ser adaptadas a las características de cada proyecto, por eso, como se mencionó anteriormente, a continuación se caracteriza la metodología seleccionada por el equipo de arquitectura así como su aporte y aplicación en la presente investigación.

1.3.1 Proceso Unificado de Rational (RUP[3]).

RUP es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas de software, para diferentes áreas de aplicación, diferentes tipos de organizaciones, diferentes niveles de aptitud y diferentes tamaños de proyecto. (Sommerville, 2005)

Utiliza UML[4] para modelar todos los esquemas necesarios para el desarrollo de un sistema. Los autores de esta metodología destacan que el proceso de software propuesto tiene tres características esenciales:

- Dirigido por casos de uso: según El Proceso Unificado de Desarrollo de Software[5], los casos de uso "(...) no solo son herramientas para especificar los requisitos de un sistema. También guían su diseño, implementación y prueba" (Jacobson, y otros, 2000). En otras palabras, los casos de uso no solo inician el proceso sino que proporcionan un hilo conductor permitiendo establecer trazabilidad entre los artefactos generados.

[3] Siglas en inglés de Rational Unified Process (Proceso Unificado de Rational).
[4] Siglas en inglés de Unified Modeling Language (Lenguaje de Modelado Unificado).
[5] El Proceso Unificado de Desarrollo de Software es una guía del Proceso Unificado escrita por Ivar Jacobson, Grady Booch y James Rumbaugh

- Centrado en la arquitectura: la arquitectura se ve influenciada por la plataforma de desarrollo, sistema operativo, gestor de bases de datos, protocolos, consideraciones de desarrollo, etc. Además su definición debe tomar en consideración elementos de calidad, rendimiento, reutilización y capacidad de evolución por lo que debe ser flexible durante todo el proceso. “El concepto de arquitectura del software incluye los aspectos estáticos y dinámicos más significativos de un sistema” (Jacobson, y otros, 2000). La arquitectura engloba todos los elementos del desarrollo plasmados en las vistas arquitectónicas que integran los componentes del sistema.
- Iterativo e incremental: el trabajo se divide en iteraciones. Una iteración es un recorrido más o menos completo a lo largo de los flujos de trabajo fundamentales, del cual se obtiene un incremento que produce un crecimiento en el producto.
- RUP se repite a través de ciclos de vida, donde cada uno representa una versión del producto. Estos ciclos constan de cuatro fases: Inicio, Elaboración, Construcción y Transición.

Un proceso de desarrollo de software define quién hace qué, cómo y cuándo. Para ello RUP define cuatro elementos: Rol, Actividad, Artefacto y Disciplina o flujo de trabajo.
Esta metodología define nueve disciplinas, de las cuales las primeras seis se conocen como flujos de la ingeniería y las restantes como flujos de apoyo. La siguiente figura muestra cómo está distribuido el trabajo según fases y disciplinas.

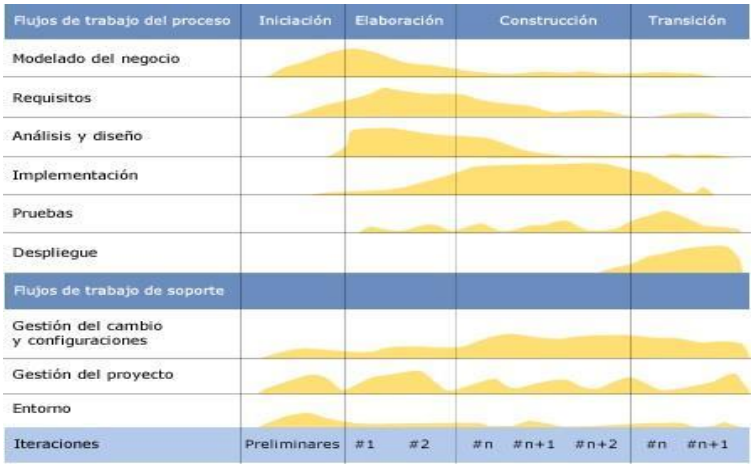

Figura 2: Distribución del esfuerzo en cada disciplina y fase según RUP.

En esta investigación se adopta RUP ya que es una metodología robusta e idónea para proyectos grandes, complejos y de larga duración como es el caso de SITPC. Además genera una gran cantidad de documentación necesaria para un proyecto que tiene constantes cambios en el equipo de desarrollo y con un cliente que no siempre está disponible para ver el avance del proceso.

Este trabajo se centra en las fases de elaboración y construcción. Como parte de la primera se realiza el diseño, perfeccionamiento del análisis realizado que traduce a términos más cercanos a la programación los requisitos y casos de uso identificados, garantizando la estructuración eficiente de la aplicación. Adentrándose en la fase de construcción se observa el flujo de trabajo más importante de la misma, la implementación, que define la codificación de clases y objetos en componentes del sistema. Este proceso está compuesto por un ciclo de varias iteraciones en las cuales se van incorporando sucesivamente los casos de uso, de acuerdo a los factores de riesgo del proyecto y en cada cual se realiza una medición de la calidad de los artefactos obtenidos. En cada una de estas iteraciones se realizan pruebas para ir garantizando la calidad del sistema.

Como parte del proceso para alcanzar el nivel 2 de madurez de CMMI[6], se realiza en la Universidad de las Ciencias Informáticas la implementación del Programa de mejoras definido por el Centro de Calidad para Soluciones Informáticas (CALISOFT) para garantizar la calidad de los artefactos durante el proceso de desarrollo así como del producto de forma general. La metodología determina los artefactos a construir en el proceso, sin embargo para certificar el nivel que se desea alcanzar, el Programa de mejoras indica cómo se deben construir cada uno de ellos garantizando que los mismos cumplan con los estándares de calidad correspondientes.

Por esta razón y en aras de garantizar tiempo y calidad de los artefactos, la presente investigación basará su solución, en el caso de la fase de elaboración, en los diagramas de clases que describen la realización física de los casos de uso, centrándose en cómo los requisitos funcionales junto con otras restricciones relacionadas con el entorno de implementación tienen impacto en la propuesta de investigación. Los casos de uso son representados por las clases de diseño y sus relaciones. Además se encuentran los diagramas de secuencias con el objetivo de describir el comportamiento dinámico del sistema de información, haciendo énfasis en la secuencia de los mensajes intercambiados por los objetos. Por último, en esta fase, se encuentra el diagrama de paquete, utilizado con el objetivo de obtener una visión más clara del sistema de información orientado a objetos, organizándolo en subsistemas, agrupando los elementos del diseño y detallando las relaciones de dependencia entre ellos, además para un mejor entendimiento del diagrama de clases. En el caso de la fase de construcción se encuentra el diagrama de despliegue, estos utilizados para mostrar la disposición de las particiones físicas del sistema de

[6] Siglas en inglés de Capability Maturity Model Integration (Integración de modelos de madurez de capacidades).

información y la asignación de los componentes software a estas particiones, es decir, las relaciones físicas entre los componentes software y hardware en el desarrollo de la propuesta de solución.

El proceso de desarrollo de software además de su componente metodológica está sustentado en el uso de herramientas. A continuación se hace una caracterización de las utilizadas en el propósito de esta propuesta de solución, las cuales fueron seleccionadas previamente por el equipo de arquitectura del proyecto al que pertenece la presente investigación.

1.4 Herramientas y tecnologías.

Para el desarrollo de la propuesta de solución se hizo necesario utilizar determinadas herramientas para facilitar el trabajo y la modelación de los procesos de Prueba Pericial y de Presunción. A continuación se detallan las características significativas que definieron su uso y selección.

1.4.1 Herramienta CASE[7] para el modelado.

Una herramienta CASE es una herramienta que ayuda al ingeniero de software a desarrollar y mantener software. A continuación se presentan algunas definiciones dadas para el término CASE.

- Herramientas individuales para ayudar al desarrollador de software o administrador de proyecto durante una o más fases del desarrollo de software (o mantenimiento). (Terminology for Software Engineering Environment (SEE) and Computer-Aided Software Engineering (CASE), 1990)
- Una combinación de herramientas de software y metodologías de desarrollo. (The CASE Experience, 1989)

La pieza fundamental, y más importante avance tecnológico asociado a una herramienta CASE, es su repositorio integrado. En el repositorio se almacena toda la información de uno o varios sistemas de información, por ejemplo, datos acerca de:

- El dominio (problema) de los sistemas desarrollados o en desarrollo.
- Modelos de solución e implementación.
- Información de la metodología que está siendo usada.
- Historia de los proyectos, recursos y presupuestos.
- Contexto organizacional: organigramas, planes estratégicos y factores críticos de éxito.

[7] Siglas en inglés de Computer-Aided Software Engineering (Ingeniería de Software Asistida por Computadora).

1.4.1.1 Visual Paradigma 8.0.

Visual Paradigm for UML es una herramienta de la Ingeniería de Software Asistida por Computadora que permite el modelado mediante UML y proporciona asistencia a los analistas, ingenieros y desarrolladores de software, durante el ciclo de vida de desarrollo de un Software.

Las funcionalidades que proporciona Visual Paradimg for UML son:

- Dibujo. Facilita el modelado de UML, al proporcionar herramientas específicas para ello, permitiendo la estandarización de la documentación, ajustándose al estándar soportado por la herramienta.
- Corrección sintáctica. Verifica que el modelado con UML sea correcto.
- Coherencia entre diagramas. Al disponer de un repositorio común, posibilita visualizar el mismo elemento en varios diagramas, evitando la ocurrencia de duplicidades.
- Integración con otras aplicaciones. Permite integrarse con otras aplicaciones, como herramientas ofimáticas, lo cual aumenta la productividad.
- Trabajo multiusuario. Permite el trabajo en equipo, proporcionando herramientas de compartición de trabajo.
- Reutilización. Brinda una gran cobertura a la reutilización, ya que constituye una herramienta centralizada donde se encuentran los modelos utilizados para otros proyectos.
- Generación de código. Permite generar código de forma automática, reduciendo el tiempo de desarrollo del software y evitando errores de codificación.
- Generación de informes. Permite generar detallados informes a partir de la información introducida en la herramienta.

La UCI presenta licencia para la utilización de esta herramienta y la misma cumple con las políticas de migración a software libre que se está llevando a cabo en Cuba, al ser una herramienta multiplataforma que se puede utilizar tanto en Linux como en Windows.

1.4.2 Servidor web.

Un servidor web es un programa que transfiere datos en forma de páginas web, hipertexto o páginas HTML[8]: textos complejos, botones, imágenes, animaciones,

[8] Siglas en inglés de HyperText Markup Language (Lenguaje de marcado de hipertexto), lenguaje de marcado predominante para la elaboración de páginas web.

reproductores de sonido y otros. La transferencia de estos datos se hace a través del protocolo HTTP[9] que es el que garantiza la comunicación entre el cliente y el servidor.
Existen varios servidores web como son: Microsoft IIS, Ngnix, Lighttp y Apache, entre otros. El servidor elegido para que la aplicación pueda ejecutarse en las estaciones de trabajo es Apache versión 2.2.

1.4.2.1 Servidor HTTP Apache v2.2.

Apache es uno de los servidores más usados en todo el mundo por sus características; este servidor es rápido y muy estable, lo que ofrece mucha confianza a sus clientes. Por otro lado, este servidor se configura fácilmente. Cuenta con un gran centro de ayuda, herramientas extras y mucha información en Internet, de manera que se pueden buscar respuestas a las preguntas que puedan surgir sobre su uso.
Esta versión del Servidor Apache permite instalar fácilmente PHP, MySQL y otros componentes que se quieran usar para las páginas web. Además, incluye una gran cantidad de módulos que permiten acceder a sus funciones ampliadas; funciones como soporte SSL, mod_rewrite, autenticación, cache, compresión, FastCGI y Gzip. (Segura, 2014)
Existen directivas que permiten un control específico sobre características de seguridad en el servidor web Apache. Normalmente se suelen aplicar para proteger la infraestructura del servidor web y las aplicaciones que en él se ubican. Tomando en cuenta la seguridad con que debe publicarse la aplicación SITPC estas son por consecuencia de gran utilidad. Algunas de estas directivas son:

- Ocultación del Server Header: Permite ocultar cualquier tipo de información sensitiva sobre el servidor o sus aplicaciones.
- Especificar usuarios con privilegios limitados: Permite especificar usuarios con privilegios limitados a directorios alojados en el servidor web.
- Desactivación de módulos innecesarios: Permite tener disponibles solo los módulos que se van a necesitar para las aplicaciones web ubicadas en el mismo, lo cual cumple permite cumplir con la idea de que la simplicidad es una característica propia de los sistemas con buenos niveles de seguridad.
- Limitación en el tamaño de las peticiones: El uso de la directiva LimitRequestBody permite limitar el tamaño máximo de las peticiones al servidor lo que protege de un posible ataque de denegación de servicio.
- Limitación del número de usuarios concurrentes: Da la posibilidad de limitar el número de usuarios que hacen peticiones al servidor de manera concurrente

[9] Siglas en inglés de Hypertext Transfer Protocol (Protocolo de Transferencia de Hipertexto)

mediante mecanismos de control de hilos. (The Apache Software Foundation, 2014)

1.4.3 Sistema Gestor de Base de Datos.

Para definir, crear y mantener las bases de datos, así como proporcionar y controlar el acceso a estas, son usados los Sistemas Gestores de Bases de Datos (SGBD). Varios autores, han planteado diferentes definiciones acerca de estas aplicaciones, por ejemplo, Sara Álvarez, expresa que: "un sistema gestor de base de datos se define como el conjunto de programas que administran y gestionan la información contenida en una base de datos" (Álvarez, 2010). Los SGBD permiten la utilización y la actualización de los datos almacenados en una o varias bases de datos por uno o varios usuarios desde diferentes puntos de vista. Además permiten almacenar y acceder a los datos de las bases de datos de forma rápida y estructurada. Sirven de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan. (Matos García, 2005)

Los SGBD permiten definir los datos a distintos niveles de abstracción y manipularlos, garantizando la seguridad e integridad de los mismos. Para esta investigación se escogió el SGBD PostgreSQL v9.1 y su fundamentación se plantea seguidamente.

1.4.3.1 PostgreSQL v9.1.

PostgreSQL es un sistema gestor de base de datos objeto-relacional avanzado con un gran historial de desarrollo. Se puede utilizar en una amplia variedad de plataformas, así como en el más pequeño de los sistemas integrados y hasta en enormes sistemas de terabytes.

Uno de los mejores beneficios que presenta PostgreSQL es que es de código abierto, lo que significa que brinda una licencia para instalar, usar y distribuir PostgreSQL sin pagar regalías. Presenta características de la orientación a objetos, como puede ser la herencia, tipos de datos, funciones, restricciones, disparadores, reglas e integridad transaccional.

Implementa la utilización de subconsultas y transacciones, haciendo su funcionamiento mucho más eficaz y soporta varios tipos de datos: además de los tipos base, también soporta datos de tipo fecha, monetarios, elementos gráficos, datos sobre redes, cadenas de bits, etc. Brinda la opción de crear tipos propios. Finalmente y como una de sus ventajas principales es que mediante un sistema denominado MVCC[10] PostgreSQL permite que mientras un proceso escribe sobre una tabla de la base de datos, otros accedan a la misma tabla sin necesidad de bloqueos y de esta

[10] Siglas en inglés de Multiversion Concurrency Control (Acceso concurrente multiversión).

forma cada usuario obtiene una visión consistente de la última versión de datos existentes en el servidor. (PostgreSQL Global Development Group, 2014)
Además de sus ofertas de soporte usa como cliente PgAdmin III que es distribuido junto a él y que también es libre.

1.4.4 Marcos de trabajo.

El concepto de “Marco de trabajo” se utiliza en muchos ámbitos del desarrollo de sistemas software. En general, el término marco de trabajo es conocido como framework (según su significado en inglés) y se refiere a una estructura de software compuesta por componentes personalizables e intercambiables para el desarrollo de una aplicación. Sus objetivos principales son: acelerar el proceso de desarrollo de software, reutilizar códigos ya existentes y promover buenas prácticas de desarrollo, tales como el uso de patrones de arquitecturas. De esta forma un marco de trabajo para el desarrollo de aplicaciones web, se define como un conjunto de componentes que integran un diseño reutilizable que facilita y agiliza el desarrollo de aplicaciones web.

1.4.4.1 Symfony 2.1.

Symfony es un completo marco de trabajo diseñado principalmente para optimizar el desarrollo de las aplicaciones web, separando la lógica de negocio, la lógica de servidor y la presentación de la aplicación web. Brinda varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja. Además, automatiza las tareas más comunes, permitiendo al desarrollador centrarse en los aspectos específicos de cada aplicación.
Symfony está desarrollado completamente con PHP 5. Ha sido probado en numerosos proyectos reales y se utiliza en sitios web de comercio electrónico de primer nivel. Es compatible con la mayoría de los sistemas gestores de bases de datos, como MySQL, PostgreSQL, Oracle y SQL Server de Microsoft. Se puede ejecutar tanto en plataformas Unix, como en plataformas Windows. Está muy bien optimizado y puede manejar los sitios web con mucho tráfico sin problemas
Puede ser completamente personalizado para cumplir con los requisitos de las empresas que disponen de sus propias políticas y reglas para la gestión de proyectos y la programación de aplicaciones. Por defecto incorpora varios entornos de desarrollo diferentes e incluye varias herramientas que permiten automatizar las tareas más comunes de la ingeniería del software.
A continuación se muestran algunas características adicionales:

- Es fácil de instalar y configurar en la mayoría de las plataformas (y con la garantía de que funciona correctamente en los sistemas Windows y Unix estándares).
- Es independiente del sistema gestor de bases de datos.
- Es sencillo de usar en la mayoría de casos, pero lo suficientemente flexible como para adaptarse a los casos más complejos.
- Sigue la mayoría de mejores prácticas y patrones de diseño para la web.
- Está preparado para aplicaciones empresariales y es adaptable a las políticas y arquitecturas propias de cada empresa, además de ser lo suficientemente estable como para desarrollar aplicaciones a largo plazo.
- Código fácil de leer que incluye comentarios de phpDocumentor y que permite un mantenimiento sencillo.
- Fácil de extender, lo que permite su integración con librerías desarrolladas por terceros. (Eguiluz, 2013)

1.4.4.2 Twitter Bootstrap 1.0.

Bootstrap es un pequeño pero potente marco de trabajo que combina HTML5, CSS y JavaScript para crear formularios, botones, tablas, barras de navegación y demás componentes que son visibles comúnmente en cualquier sitio Web, de forma más amigable para el usuario. Su principal objetivo es simplificar el proceso de diseño web sobre todo en conceptos delicados como compatibilidad con navegadores y su mayor valor es la capacidad para diseño fluido (responsive design). Mediante el diseño fluido de una web se logra que un sitio web se adapte según el dispositivo donde se visualiza, o sea, que la web se adapte según la resolución del dispositivo o ventana del navegador.

Lo que hace que las aplicaciones web desarrolladas con Bootstrap se vean más agradables son los estilos que provee el marco de trabajo. Simplemente agregando algunas clases y la etiqueta indicada se pueden lograr casi sin esfuerzo grupos de botones, barras de navegación y ventanas flotantes. (Rodríguez, 2012)

1.4.4.3 Doctrine 2.0.

Doctrine es un potente y completo sistema ORM para PHP 5.2 o superior, con una capa de abstracción de la base de datos DBAL (Database Abstraction Layer por sus siglas en inglés) incorporada, donde su principal tarea es traducir de forma transparente los objetos PHP en filas de una base de datos relacional y viceversa, por tal motivo es muy recomendable para aplicaciones que utilicen el paradigma de la programación orientada a objetos. Una de sus características más importantes es la

posibilidad de acceder a la base de datos a través de un lenguaje integrado llamado Doctrine Query Language (DQL). Es el ORM que trae integrado el marco de trabajo Symfony y contiene una documentación muy completa.

La utilización de Doctrine tiene un conjunto de ventajas que facilitan las tareas comunes y de mantenimiento que se realizan durante el desarrollo de una aplicación web:

- Reutilización: La cual permite llamar a los métodos asociados a un objeto de datos desde distintas partes del código de la aplicación.
- Portabilidad: Utilizando la capa de abstracción de los datos, Doctrine permite que si se necesite cambiar el gestor de base de datos, y el impacto sería mínimo. Esto se debe a que no utiliza sintaxis específicas de lenguajes de bases de datos para acceder al modelo, sino que utiliza una sintaxis propia que es capaz de traducir los diferentes tipos de bases de datos.
- Seguridad: Implementa mecanismos de seguridad que protegen la aplicación de los ataques más comunes que se realizan a las bases de datos, como inyecciones SQL.
- Mantenimiento del código: Gracias a la correcta ordenación de la capa de datos, modificar y mantener el código de una aplicación se convierte en una tarea sencilla para los desarrolladores. (Kahwe Smith, 2009)

1.4.5 JQuery.

JQuery es una librería de JavaScript que facilita el acceso a los elementos del DOM, los efectos, interactuar con los documentos HTML, desarrollar animaciones y agregar interacción con la tecnología AJAX a páginas web.

La característica principal de JQuery es que permite cambiar el contenido de una página web sin necesidad de recargarla. Este propósito se puede lograr mediante la manipulación del árbol DOM y peticiones AJAX.

JQuery ofrece una infraestructura con la que el desarrollador tiene mayor facilidad para la creación de aplicaciones complejas del lado del cliente. Es una de las mejores opciones para utilizar dentro del desarrollo de aplicaciones web. La agregación de plugins se realiza de forma muy simple, lo que supone un ahorro substancial de tiempo y esfuerzo para el programador.

La licencia open source de JQuery permite que la librería siempre cuente con soporte constante y rápido, publicándose actualizaciones de manera muy frecuentes.

1.4.6 Patrones de diseño.

Los diseñadores expertos no resuelven los problemas desde sus principios; reutilizan soluciones que han funcionado en el pasado. Se encuentran patrones de clases y objetos de comunicación recurrentes en muchos sistemas orientados a objetos. Estos patrones resuelven problemas de diseño específicos y hacen el diseño flexible y reusable. 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. (Gamma, y otros, 1995)

Los patrones de diseño constituyen la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces. Proporcionan una estructura conocida por todos los desarrolladores, de forma que la manera de trabajar no resulte distinta entre ellos.

La utilización de patrones de diseño, permite ahorrar tiempo en la construcción de software. El producto obtenido es más fácil de comprender, mantener y extender. Existen versiones ya implementadas de estas funcionalidades comunes (marcos de trabajo) que permiten centrarse en desarrollar sólo la funcionalidad específica requerida por cada aplicación y que además, dan mejor imagen de profesionalidad y calidad. Los patrones de diseño se agrupan dos grandes categorías: GRASP[11] y GOF[12].

Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos. Este grupo contiene patrones como: Experto, Creador, Alta cohesión, Bajo acoplamiento, Controlador y No hables con extraños (Ley de Demeter).

Los patrones de diseño GOF son 23 y se clasifican según su propósito en Creacionales, Estructurales y de Comportamiento; y según su ámbito en Objeto y de Clase. Estos 23 patrones fueron recopilados y documentados en el libro "*Design Patterns: Elements of Reusable Object Oriented Software*" presentado por primera vez en 1944 por los Gang of Four.

1.4.7 Paradigmas de programación.

Un paradigma de programación es una propuesta tecnológica que representa un enfoque particular para el desarrollo de software, utilizando directrices especificadas.

[11] Siglas en inglés de General Responsability Assigment Software Patterns (Patrones Generales de Software para Asignación de Responsabilidades).

[12] Gang of Four (GoF, que en español es la pandilla de los cuatro) formada por Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides. Ellos recopilaron y documentaron 23 patrones de diseño aplicados usualmente por expertos diseñadores de software orientado a objetos.

Este modelo permite determinar la estructura correcta de los programas y controla el modo en que se piensa y formula la solución.

Durante el proceso de desarrollo de sistemas la selección del paradigma de programación debe facilitar la elaboración del producto final, debe ser la propuesta tecnológica más cómoda para formular la solución. Debido a las características que debe presentar el SITPC el paradigma base utilizado es el paradigma de Programación Orientado a Objetos (POO).

1.4.7.1 Paradigma de Programación Orientado a Objetos.

Este es un paradigma que simula el comportamiento de objetos reales para diseñar aplicaciones informáticas, definiendo tipos o clases de objetos que cumplen determinadas propiedades y que puedan llevar a cabo una determinada funcionalidad dentro de una aplicación. Los detalles internos de cómo se logra esa funcionalidad deben quedar ocultos (encapsulados). De este modo los diferentes niveles de complejidad de un problema irán quedando ocultos en los objetos que se describen para resolverlo, permitiendo así acomodarse al modo de pensar del hombre y no al de la máquina, esto facilita la implementación, ya que estos objetos pasan a ser el concepto fundamental de la POO, ellos van a poseer atributos que representan sus características o propiedades y métodos que representan su comportamiento o acciones que realizan. Este paradigma está basado en varias técnicas, como son: abstracción, encapsulamiento y herencia.

- Abstracción: Posibilita modelar la realidad a través de objetos en la programación, simplifica la realidad ignorando los detalles que no tienen relevancia en el problema que se modela y se enfoca en las cosas comunes, pero permitiendo las variaciones.
- Encapsulamiento: Es la propiedad que permite que los objetos presenten la misma interfaz pero oculten información de su funcionamiento, es decir, ocultan sus datos de manera que sólo se puede cambiar mediante las operaciones definidas para ese objeto.
- Herencia: Es el mecanismo fundamental para implementar la reutilización y extensibilidad del software. Esta facilita la clasificación que se hace a distintos objetos, permitiendo construir nuevas clases partiendo de una jerarquía ya existente, ya que permite crear objetos que guarden propiedades y formas de interactuar comunes, permitiendo que un nuevo objeto tome las mismas características que su ancestro en la jerarquía.

1.4.8 Lenguajes de programación.

Los lenguajes de programación son un conjunto de reglas, herramientas y condiciones que permiten la creación de programas o aplicaciones dentro de una computadora.
Están divididos en dos partes: sintáctica y semántica. Poseen reglas acerca de cómo se deben escribir las sentencias. (Bonanata, 2008)
En otras palabras, es un idioma artificial diseñado para expresar computaciones que pueden ser llevadas a cabo por máquinas como las computadoras. Pueden usarse para crear programas que controlen el comportamiento físico y lógico de una máquina.
Para el desarrollo de una aplicación web, existen lenguajes de programación que responden del lado del cliente y otros del lado del servidor.

1.4.8.1 Lenguajes del lado del cliente.

Son aquellos lenguajes de programación capaces de ser interpretados directamente por el navegador sin necesidad de un pre-tratamiento. Para el desarrollo de la presente investigación, teniendo en cuenta la selección del equipo de arquitectura, se utilizaron los siguientes lenguajes de programación:

1.4.8.1.1 JavaScript.

Es un lenguaje de programación interpretado que maneja objetos dentro de la página web que facilitan la programación de páginas interactivas, a la vez que se evita la posibilidad de ejecutar comandos que puedan ser peligrosos para la máquina del usuario, tales como el formateo de unidades y la modificación de archivos. Es un lenguaje dinámico que responde a eventos en tiempo real como presionar un botón, pasar el puntero del mouse sobre un determinado texto o el simple hecho de cargar la página o caducar un tiempo. Se utiliza esencialmente del lado del cliente, implementado como parte del navegador web permitiendo mejoras significativas en la seguridad puesto que es muy eficiente en los temas de validaciones. (Bradenbaugh, 2000)

1.4.8.1.2 HTML.

El HTML es el lenguaje con el que se escriben las páginas web. Es un lenguaje de hipertexto que permite escribir texto de forma estructurada y que está compuesto por etiquetas, que marcan el inicio y el fin de cada elemento del documento. Un documento de hipertexto no sólo se compone de texto, puede contener imagen, sonido, video y otros, por lo que el resultado puede considerarse como un documento multimedia. (Musciano, y otros, 1999)

1.4.8.2 Lenguajes del lado del servidor.

Los lenguajes de lado del servidor son aquellos reconocidos, ejecutados e interpretados por el propio servidor y que se envían al cliente en un formato comprensible para él. Para los efectos de esta investigación se tuvo en cuenta PHP 5.3.

1.4.8.2.1 PHP 5.3.

PHP (Hypertext Pre-Processor por sus siglas en inglés) es un lenguaje de programación interpretado, diseñado originalmente para la creación de páginas web dinámicas. Es usado principalmente en la 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 aplicaciones incluyendo aplicaciones con interfaz gráfica.
Dentro de sus características se pueden enunciar:

- Es un lenguaje orientado al desarrollo de aplicaciones web dinámicas con acceso a información almacenada en una base de datos.
- Es un lenguaje fácil de aprender, ya que en su desarrollo se simplificaron distintas especificaciones, como es el caso de la definición de las variables primitivas.
- Su código fuente es invisible al navegador web y al cliente, ya que el servidor es el encargado de ejecutar el código y enviar su resultado HTML al navegador.
- Posee capacidad de conexión con el gestor de base de datos PostgreSQL el cual se usará en la investigación.
- Cuenta con una gran documentación en su sitio web oficial publicado en Internet y contiene en un único archivo de ayuda las explicaciones y ejemplificaciones de todas las funciones predefinidas del sistema.
- Permite aplicar técnicas de programación orientada a objetos. (The PHP Group, 2014)

1.4.9 Entorno de Desarrollo Integrado (IDE).

Un IDE[13] es un entorno de programación que consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica. Ofrece un marco de trabajo amigable para la mayoría de los lenguajes de programación tales como C++, Python, Java, C#, Delphi y Visual Basic.

[13] Siglas en inglés de Integrated Development Environment (Entorno de Desarrollo Integrado).

1.4.9.1.1 NetBeans 7.4.

Netbeans es una IDE sumamente completo, fácil de usar, cómodo y de excelente calidad; y es completamente gratis. A continuación se listan un conjunto de características que hicieron que NetBeans fuera elegido por el equipo de arquitectura:

- Soporta los populares marcos de trabajo: ZendFramework, Symfony1 y Symfony2.
- Soporta a Doctrine 2 que es el marco de trabajo ORM (Object Relational Mapping) que se usará en el desarrollo de la solución.
- Proporciona un soporte para las plantillas Twig (que están estrechamente relacionados con Symfony). Proporciona completamiento de código para todos los elementos de Twig. Incluye etiquetas, filtros, funciones y operadores, todos ellos documentados.
- Gestión fácil y eficiente de proyectos: Permite mantener una visión clara de las aplicaciones grandes, con miles de carpetas y archivos, y millones de líneas de código, lo cual es una tarea de enormes proporciones y necesaria en el caso de la gestión del SITPC. Para lograr esto NetBeans proporciona diferentes vistas de los datos. Provee además desde múltiples ventanas de proyectos hasta herramientas útiles para la creación de aplicaciones de manera eficiente, lo que le permite profundizar en los datos de forma rápida y sencilla. También proporciona herramientas para el control de versiones como por ejemplo: Subversion, Mercurial y Git.
- Análisis estático del código: El editor de NetBeans tiene la característica de poder analizar el código estáticamente, o sea, que puede encontrar posibles problemas y detectar inconsistencias en el código fuente antes de ser ejecutado. Esto es posible gracias a la integración con la herramienta FindBugs. (Sun MicroSystems, 2013)

1.4.10 Arquitectura de software.

“La arquitectura de un sistema es la estructura del sistema, la cual comprende los componentes del software, las propiedades de esos componentes visibles externamente y las relaciones entre ellos”. (Pressman, 2002)

Representa la base para todo el funcionamiento de la aplicación; es el pilar principal del producto que se quiere construir. En su forma más simple, es la estructura u organización de los componentes del programa (módulos), la manera en que estos componentes interactúan y la estructura de datos que utilizan.

La solución informática que se pretende obtener como resultado de la investigación no está aislada sino que forma parte del módulo Común del Sistema de Informatización para la Gestión de los Tribunales Populares Cubanos. Por lo que debe seguir la arquitectura definida por el equipo de arquitectura.

En el desarrollo del SITPC se emplea una arquitectura multicapas (n-layers), la cual está compuesta por las capas: Vista, Controlador, Modelo y Datos. El uso de esta arquitectura permite dividir los problemas a resolver y que cada capa contenga solo las funcionalidades relacionadas con sus tareas, proporcionándose una alta reutilización del código. Se hace uso del patrón arquitectónico Modelo Vista Controlador (MVC), el cual permite la independencia entre las capas, así como la posibilidad de realizar cambios en las mismas sin tener que modificar las otras capas, además facilita la estandarización, la utilización eficiente de los recursos y la administración. El uso de esta arquitectura sirvió como norma para el desarrollo de la solución informática propuesta para informatizar los procesos de Prueba Pericial y de Presunción.

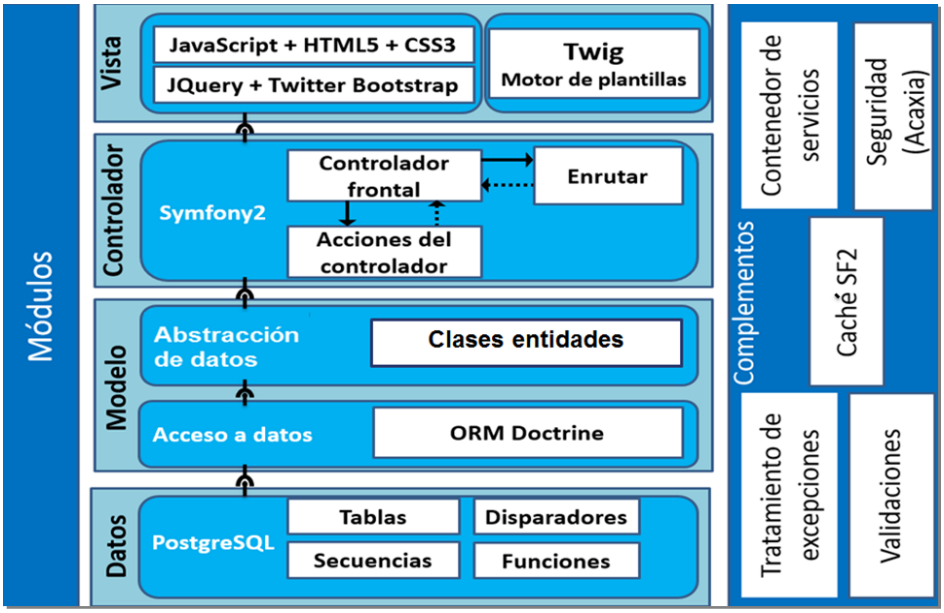

Figura 3: Arquitectura del sistema.

1.4.10.1 Vista.

En esta capa se encuentra todo lo que se refiere a la visualización de la información, el diseño, colores, estilos y la estructura visual de las páginas. En el desarrollo de la solución propuesta y del SITPC en general, queda evidenciada esta capa a través del uso de los archivos JavaScript, HTML y CSS, que en conjunto con la librería de JavaScript JQuery, el marco de trabajo Bootstrap y el motor de plantillas Twig, son los encargados de crear las páginas de la aplicación.

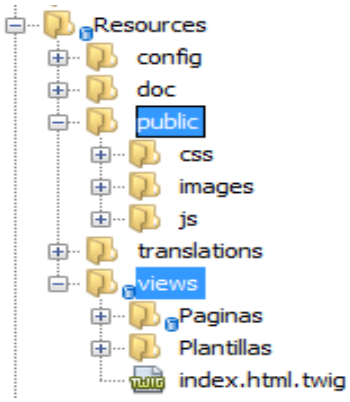

Figura 4: Estructura de la carpeta Resources, donde se encuentran las carpetas public y views

1.4.10.2 Controlador.

Esta capa es la responsable de procesar y mandar a mostrar los datos obtenidos en la capa de acceso a datos, es decir, trabaja de intermediaria entre la capa de la vista y la capa de acceso a datos.

Symfony2 define para esta capa un controlador frontal (app.php) para el entorno de producción que es el que se despliega y otro para el entorno de desarrollo (app_dev.php). Esta separación se realiza por medidas de seguridad, de modo que todas las peticiones que se realizan a la aplicación entran por uno de estos puntos de entrada. Las peticiones son hechas al componente de ruteo el cual devuelve las peticiones en forma de rutas y valida algunas acciones de seguridad, por ejemplo, si se tienen o no permisos para acceder a esas rutas, las cuales contienen las acciones de los controladores que deben ejecutarse, que son los controladores de la aplicación.

En la solución informática propuesta el uso de esta capa se observa a través de las clases controladoras que son las encargadas de realizar el procesamiento de los datos y lógica de negocio asociada a las peticiones realizadas desde la capa de la vista.

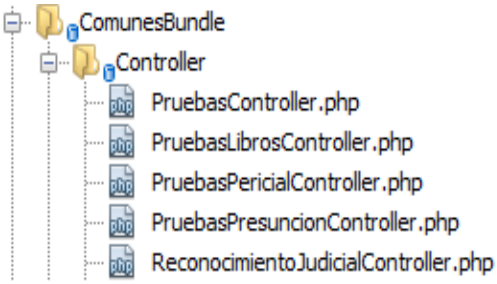

Figura 5: Estructura de la carpeta Controller.

1.4.10.3 Modelo.

En la arquitectura definida para el desarrollo del SITPC esta capa está dividida en dos subcapas; la de acceso a datos y la de abstracción de datos.

1.4.10.3.1 Capa de acceso a datos.

Esta capa se encarga del manejo de la lógica de negocio, además sirve como intermediaria entre la capa controladora y el sistema gestor de base de datos. Las clases que componen esta capa son las gestoras y las repositorios, ubicadas en las carpetas Negocio y Repository respectivamente. Las clases gestoras reciben la información obtenida de la vista por medio de las controladoras y posteriormente se encargan de tratar dicha información en conjunto con las clases repositorios. Las clases repositorios contienen un conjunto de consultas para acceder y realizar acciones sobre los datos de tablas específicas de la base de datos. En esta capa también se encuentra el ORM Doctrine que posibilita la separación de la aplicación respecto al sistema gestor de base de datos mediante su lenguaje propio de consultas DQL.

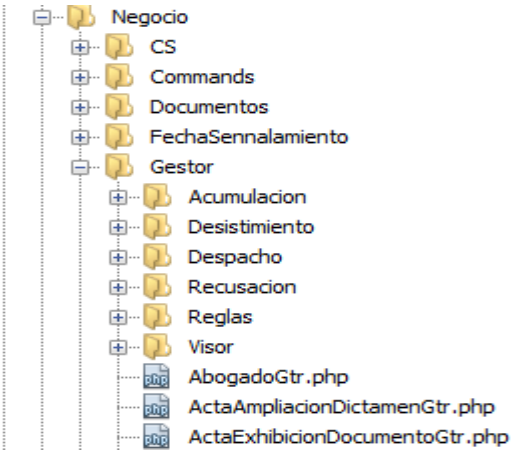

Figura 6: Estructura de la carpeta Negocio.

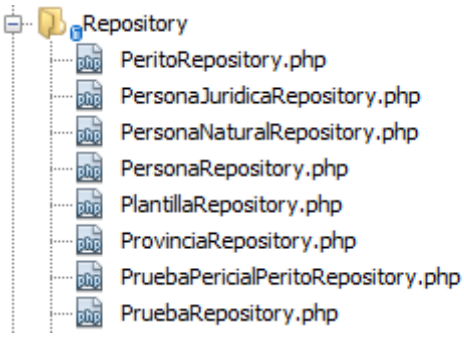

Figura 7: Estructura de la carpeta Repository.

1.4.10.3.2 Capa de abstracción de datos.

Esta capa contiene las clases entidades, las cuales son una representación de las tablas de la base de datos mapeadas por el ORM Doctrine, de ahí que constituyen una abstracción de los datos reales almacenados en la base de datos. Las clases entidades se encuentran en la carpeta Entity del bundle ComunBundle.

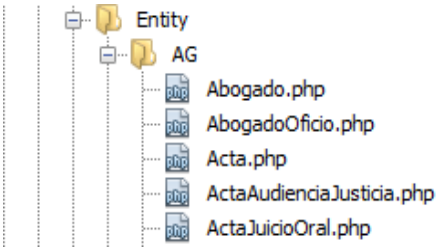

Figura 8: Estructura de la carpeta Entity.

1.4.10.4 Datos.

En esta capa se maneja la información de la base de datos por medio de una plataforma potente, el sistema gestor de base de datos PostgreSQL.

Luego de la caracterización de las herramientas, tecnologías y la arquitectura a utilizar en la presente investigación, se tiene como base su funcionamiento para una futura aplicación en el desarrollo de la propuesta de solución, aprovechando los beneficios que ofrecen. Teniendo presente estos elementos se hace necesario en un segundo momento la caracterización y selección de las métricas para la medición del diseño del software que aseguren la calidad del diseño de la propuesta de solución. Es por esto que se aborda el siguiente estudio en el próximo epígrafe.

1.5 Métricas para la medición del diseño de software.

Cuando se construye un sistema informático es importante tener en cuenta todos los aspectos necesarios para obtener un producto de software con la mayor calidad posible. Según la terminología de la IEEE[14], la calidad de un sistema, componente o proceso de desarrollo de software, se obtiene en función del cumplimiento de los requerimientos iniciales especificados por el cliente o el usuario final. (IEEE, 1990)

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

Las métricas son la maduración de una disciplina que va 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 por último ayudarán en el diseño de pruebas más efectivas. Es por eso que propone un proceso de medición que se caracteriza por cinco actividades:

[14] Siglas en inglés de Insitute of Electrical and Electronics Engineers (Instituto de Ingenieros Eléctricos y Electrónicos): Asociación técnico-profesional mundial dedicada a la estandarización.

- Formulación: la obtención de medidas y métricas apropiadas para la representación de software en cuestión.
- Colección: el mecanismo empleado para acumular datos necesarios para obtener las métricas formuladas.
- Análisis: el cálculo de las métricas y la aplicación de herramientas matemáticas.
- Interpretación: la evaluación de los resultados de las métricas en un esfuerzo por conseguir una visión interna de la calidad de la representación.
- Realimentación: recomendaciones obtenidas de la interpretación de métricas técnicas trasmitidas al equipo de software.

Los sistemas de software orientados a objetos como el propuesto en esta investigación, poseen características especiales que lo diferencian de otros sistemas de software construidos. Es por ello que la medición de los artefactos del diseño se vuelve tarea importante para lograr una buena implementación con un código reutilizable, ágil y de calidad garantizada.

1.5.1 Métricas orientadas a clases.

Teniendo en cuenta que la clase es la unidad fundamental de un sistema orientado a objetos, muchas métricas la usan como la unidad básica medible. Las métricas aplicadas a clases, sus jerarquías y colaboraciones son un recurso de suma importancia para la estimación de la calidad del sistema. Las métricas que se han definido a nivel de clases se agrupan en familias como sigue a continuación:

- Familia de métricas CK (propuestas por Chidamber y Kemerer). Entre estas métricas propuestas de diseño basadas en clases a las cuales suelen aludirse con el nombre de conjunto de métricas CK se encuentran: Árbol de profundidad de herencia (APH), Relaciones entre clases (RC) y Carencia de cohesión en los métodos (CCM).
- La serie de métricas LK (propuestas por Lorenz y Kidd). Estas métricas se dividen en cuatro categorías: tamaño, herencia, valores internos y valores externos. Las métricas orientadas a tamaño se centran en el cálculo de atributos y operaciones de una clase de manera individual; las basadas en herencia se enfocan en la reutilización de las operaciones en una jerarquía de clases; las basadas en valores internos examinan la cohesión y otros aspectos relacionados con el código; y las métricas orientadas a valores externos se centran en el acoplamiento y la reutilización.

En este grupo de métricas se encuentran: Tamaño operacional de las clases (TOC), Número de operaciones redefinidas por una clase (NOR) y Número de operaciones añadidas por una clase (NOA).

Existen además otras familias de métricas que se pueden aplicar para determinar la calidad del diseño en un sistema, entre estos grupos se encuentran:

- Métricas de Li y Henry que son una modificación de las métricas CK más cuatro nuevas métricas.
- Métricas de Hendersson-Sellers que están relacionadas fundamentalmente con el acoplamiento y la cohesión del diseño.

Con el objetivo de medir el nivel de relaciones entre las clases del sistema y el grado de complejidad de implementación de las clases se decide utilizar las siguientes mediciones: Relaciones entre clases y Tamaño operacional de clase. Con las cuáles se pretende validar la calidad requerida para el diseño de la propuesta de solución.

El desarrollo de un software es algo complejo y son innumerables las posibilidades de errores por lo que este ha de ir acompañado de alguna actividad que garantice la calidad, es por esto que es necesario, a parte de la aplicación de las métricas, el uso de las pruebas de software, dado estas circunstancias a continuación se realiza una caracterización y selección de estas pruebas con el objetivo de obtener una solución con el grado máximo de calidad.

1.6 Pruebas de software.

Las pruebas de software, al contrario de lo que normalmente se considera, tienen como objetivo detectar errores no encontrados hasta el momento en la aplicación. Se puede hablar entonces del éxito de las pruebas siempre y cuando se hallen errores en el software. Con las pruebas se puede además observar hasta qué punto el software parece funcionar en concordancia con los requisitos descritos por el cliente; aunque no pueden asegurar la ausencia de defectos, sólo puede mostrar que existen. (Pressman, 2002)

Existen diferentes tipos de prueba:

- El enfoque funcional o de caja negra: que realiza pruebas sobre la interfaz del programa a probar, entendiendo por interfaz las entradas y salidas de dicho programa. No es necesario conocer la lógica del programa, únicamente la funcionalidad que debe realizar.
- El enfoque estructural o de caja blanca: que se basa en un minucioso examen de los detalles procedimentales del código a evaluar, por lo que es necesario conocer la lógica del programa.

1.6.1 Diseño de casos de pruebas.

El diseño de casos de prueba puede requerir tanto esfuerzo como el propio diseño inicial del producto, se deben diseñar pruebas que tengan la mayor probabilidad de encontrar el mayor número de errores con la menor cantidad de esfuerzo y tiempo posible. Un caso de prueba se puede definir como un conjunto de entradas de prueba, condiciones de ejecución y resultados esperados. Los casos de prueba son desarrollados para cumplir un objetivo en particular o una función esperada. Tienen como misión verificar:

- Si el producto satisface los requerimientos del usuario, tal y como se describe en la especificación de los requisitos.
- Si el producto se comporta como se desea, tal y como se describe en las especificaciones funcionales del diseño.

Para la validación de la propuesta solución se decide realizar pruebas de caja blanca con la utilización de la técnica de camino básico que permite al diseñador de casos de prueba obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución.

Para complementar estas pruebas se aplicarán además la ejecución de las pruebas de caja negra, ya que las mismas se centran en los requisitos funcionales que debe cumplir el software, asegurando de esa manera el correcto funcionamiento de la aplicación mediante la realización de casos de pruebas, permitiendo que el sistema se ejecute en todas sus variantes. Las mismas serán aplicadas por un equipo de calidad interna del proyecto de SITPC.

1.7 Conclusiones parciales.

Se realizó un análisis de la metodología y las diferentes herramientas y tecnologías informáticas que se utilizarán en el desarrollo del sistema que fueron previamente definidas en el Documento de Arquitectura del proyecto SITPC (Proyecto SITPC, 2014), permitiendo conocer las características más relevantes de las mismas. También permitió sentar las bases necesarias para su óptimo uso y aplicación y en consecuencia lograr un correcto diseño e implementación de los requisitos identificados para los procesos de Prueba Pericial y de Presunción del SITPC y luego su satisfactoria integración con el resto de la solución.

Para la validación del diseño se utilizarán las siguientes métricas: relaciones entre clases y tamaño de clase. Las pruebas de software se harán mediante las pruebas de caja blanca con la utilización de la técnica de camino básico además de las pruebas de caja negra.

Capítulo 2: Propuesta de solución

2.1 Introducción.

En este capítulo se presenta una propuesta de solución a la investigación, la cual está constituida por el modelo de diseño además del modelo de implementación. El modelo de diseño está conformado por los diagramas de clases, de paquetes y de secuencia, y el modelo de implementación por el diagrama de despliegue. Se presentan también los patrones de diseño utilizados en la solución así como los estándares de codificación definidos para la implementación.

2.2 Diseño e implementación.

El diseño es el centro de atención al final de la fase de elaboración y el comienzo de las iteraciones de construcción según la metodología RUP. Esto contribuye a una arquitectura estable y sólida, y crea un plano del modelo de implementación. Durante la fase de construcción, cuando la arquitectura es estable y los requisitos están bien entendidos el centro de atención se desplaza a la implementación.

La implementación es el centro durante las iteraciones de construcción, aunque también se lleva a cabo durante la fase de elaboración, para crear la línea base ejecutable de la arquitectura, y durante la fase de transición, para tratar defectos tardíos como los encontrados con distribuciones betas del sistema. Durante la implementación el programador escribe el código fuente, reutiliza código, compila e implementa los elementos del modelo de diseño. Si se encuentran defectos en el diseño, se regresa a la fase de análisis y diseño y se corrigen, también se arreglan defectos del código.

La entrada principal de las actividades de diseño fueron los casos de usos correspondientes a los procesos de Prueba Pericial y de Presunción del subsistema Común, que ya habían sido previamente definidos y especificados. Estos se listan a continuación:

- Registrar escrito de ampliación o adición de prueba pericial.
- Dictar providencia admitiendo la ampliación o adición de prueba pericial.
- Gestionar prueba pericial.
- Gestionar perito.
- Disponer sobre prueba pericial.
- Crear acta de Juramento de Cargo.

- Crear acta de ampliación de dictamen.
- Registrar informe pericial.
- Disponer sobre el informe pericial.
- Disponer sobre el vencimiento del término para entregar informe pericial.
- Registrar escrito de solicitud de aclaración de informe pericial.
- Disponer sobre escrito solicitando aclaración de informe pericial.
- Registrar escrito de ampliación de dictamen.
- Crear resolución teniendo por recibido el informe de ampliación de dictamen.
- Registrar escrito proponiendo peritos.
- Disponer sobre el vencimiento del término para proponer perito.
- Disponer sobre el escrito proponiendo peritos.

2.2.1 Modelo de datos.

El modelo de datos se utiliza para describir la estructura lógica y física de la información persistente gestionada por el sistema. Representa de forma abstracta los datos y las relaciones entre ellos. El modelo de datos correspondiente a los procesos de Prueba Pericial y de Presunción consta de un total de 88 tablas, de ellas 26 nomencladores. Para su construcción se tomaron en cuenta la información que se debía persistir de cada uno de los casos de usos listados anteriormente. A continuación se muestra un fragmento del modelo general de los datos.

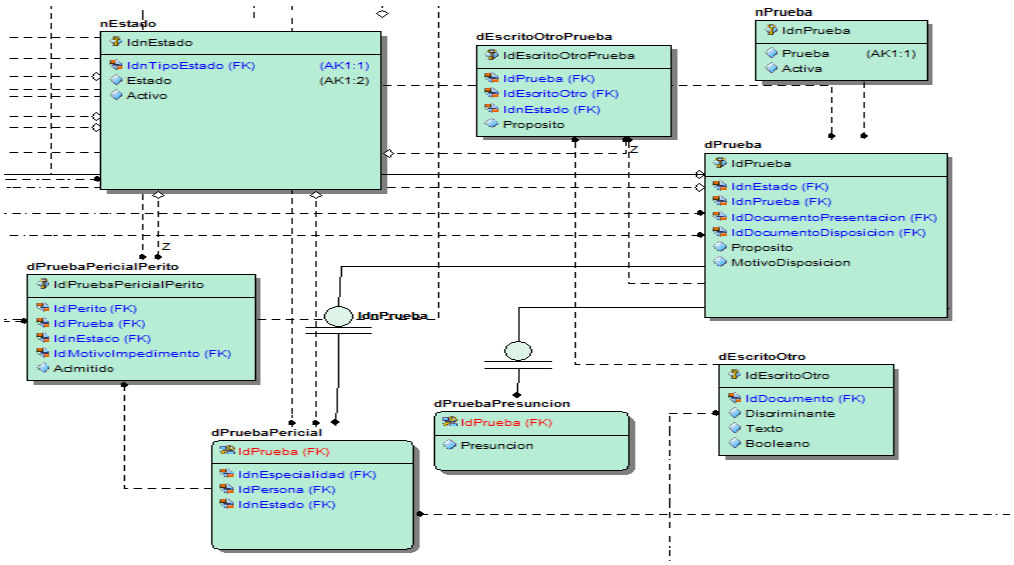

Figura 9: Fragmento del modelo de datos de los procesos de Prueba Pericial y de Presunción.

El nombre de las entidades se establecen siguiendo un conjunto reglas definidas por el equipo de arquitectura del proyecto: las tablas nomencladoras comienzan con la letra

(n), el resto de las tablas comienzan con la letra (d) y las especializaciones con el nombre de la tabla padre más nombre de la tabla hija.

2.2.2 Modelo de diseño.

El modelo de diseño es un modelo de objeto que describe la realización física de los casos de uso centrándose en como los requisitos funcionales y no funcionales, junto con las restricciones relacionadas con el entorno de implementación, tienen impacto en el sistema a considerar. Sirve de abstracción de la implementación y se utiliza como la entrada fundamental de las actividades de implementación.

2.2.2.1 Diagrama de clases del diseño.

Los diagramas de clases presentan las clases con sus relaciones estructurales y de herencia. Para la realización de los mismos se tuvieron en cuenta el modelo de datos y los principios planteados por los patrones de diseño. Estuvo condicionado además, por la arquitectura definida para el SITPC.

A continuación se presenta el diagrama de clases perteneciente al caso de uso Gestionar Prueba de Presunción.

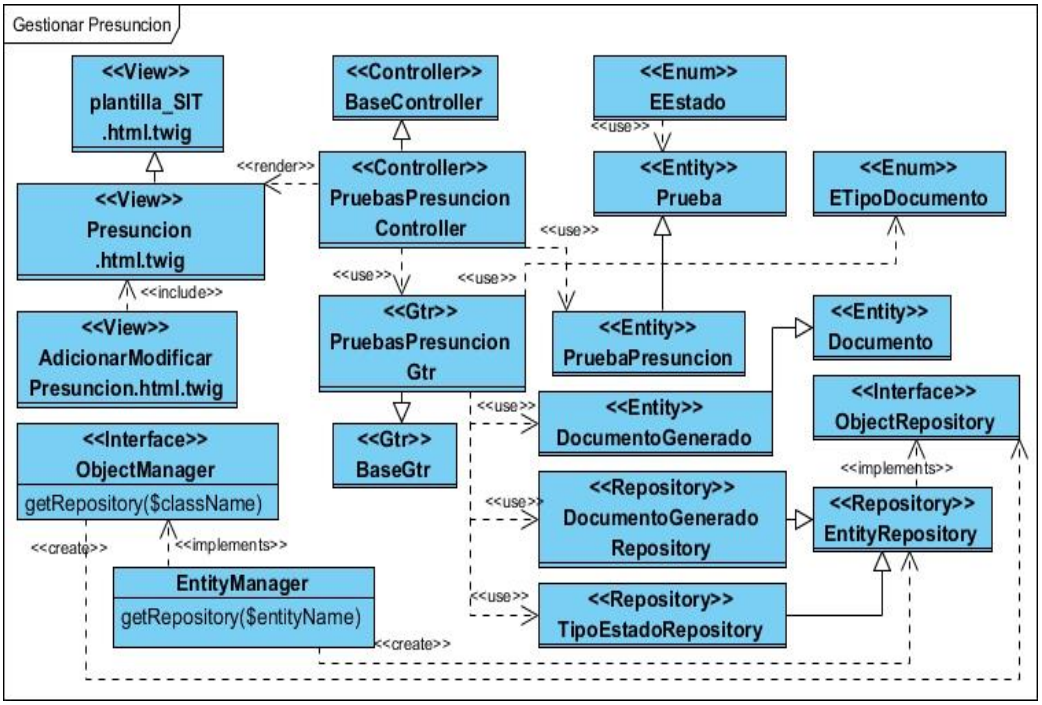

Figura 10: Diagrama de clases del diseño del CU Gestionar Prueba de Presunción.

2.2.2.2 Diagrama de paquetes.

Un diagrama de paquetes ayuda a obtener una visión más clara del sistema de información orientado a objetos, agrupando los elementos del análisis, diseño o construcción y detallando las relaciones de dependencia entre ellos.

El siguiente diagrama constituye una simplificación del diagrama de clases anteriormente presentado del caso de uso Gestionar Prueba de Presunción en el que se observa la influencia del estilo arquitectónico previamente definido.

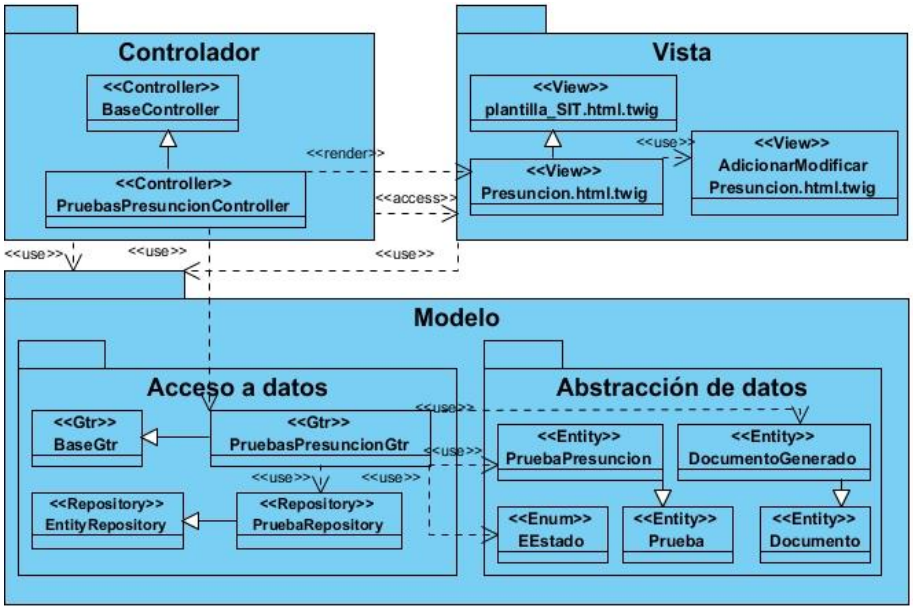

Figura 11: Diagrama de paquetes del CU Gestionar Prueba de Presunción.

2.2.2.3 Diagrama de secuencia.

El diagrama de secuencia representa y describe gráficamente la interacción de un conjunto de objetos a través del tiempo. Esta descripción explica detalladamente los casos de uso, aclarándolos al nivel de mensajes de los objetos existentes, es decir, proporciona la interacción entre los objetos en un escenario específico durante la ejecución del sistema. Gráficamente un diagrama de secuencia es una tabla con dos ejes: el eje horizontal muestra el conjunto de objetos y el eje vertical muestra el conjunto de mensajes.

Como parte de la solución se obtuvieron los diagramas de secuencia para los escenarios de los casos de usos identificativos de los procesos de Prueba Pericial y de Presunción. A continuación se presenta el diagrama de secuencia correspondiente al caso de uso Gestionar Prueba de Presunción.

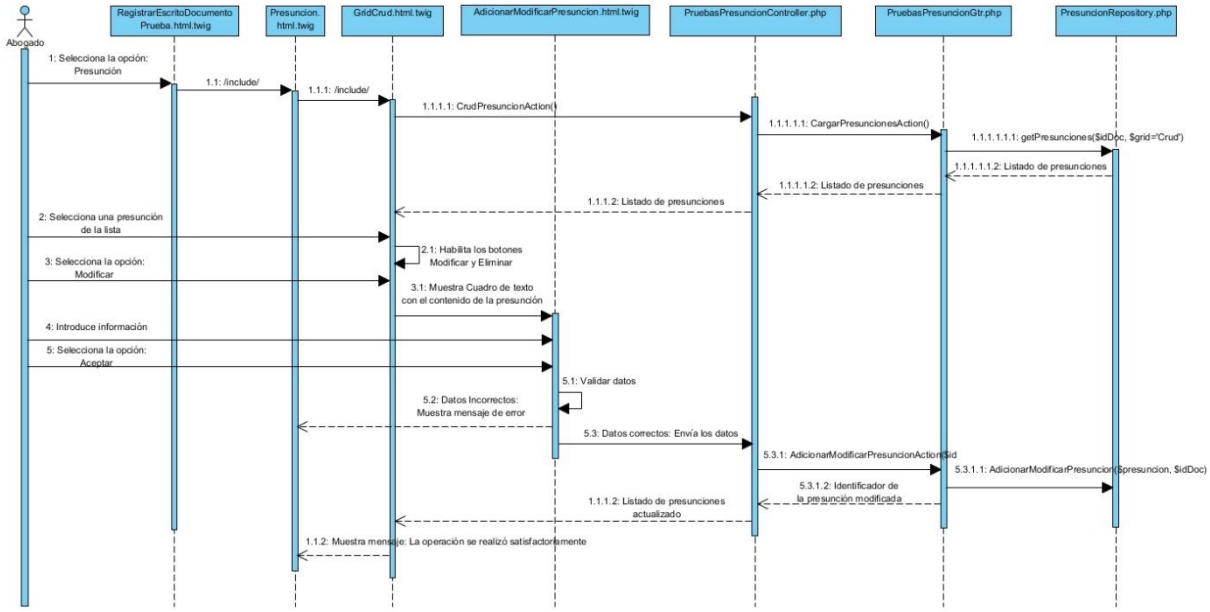

Figura 12: Diagrama de secuencia del escenario Modificar del CU Gestionar Prueba de Presunción.

2.2.3 Modelo de implementación.

Este modelo describe como los elementos del modelo de diseño, como las clases, se implementan en términos de componentes, ficheros de código fuente y ejecutables. El modelo de implementación describe también como se organizan los componentes de acuerdo con los mecanismos de estructuración y modularización disponibles en el entorno de implementación y en el lenguaje o lenguajes de programación utilizados.

2.2.3.1 Diagrama de despliegue.

Los diagramas de despliegue describen las relaciones físicas entre los componentes hardware y software que componen el sistema final. La vista de despliegue representa la disposición de las instancias de componentes de ejecución, en instancias de nodos conectados por enlaces de comunicación. Un nodo es un recurso de ejecución tal como una computadora, un dispositivo o memoria y son conectados por asociaciones de comunicación tales como enlaces de red y conexiones TCP/IP.

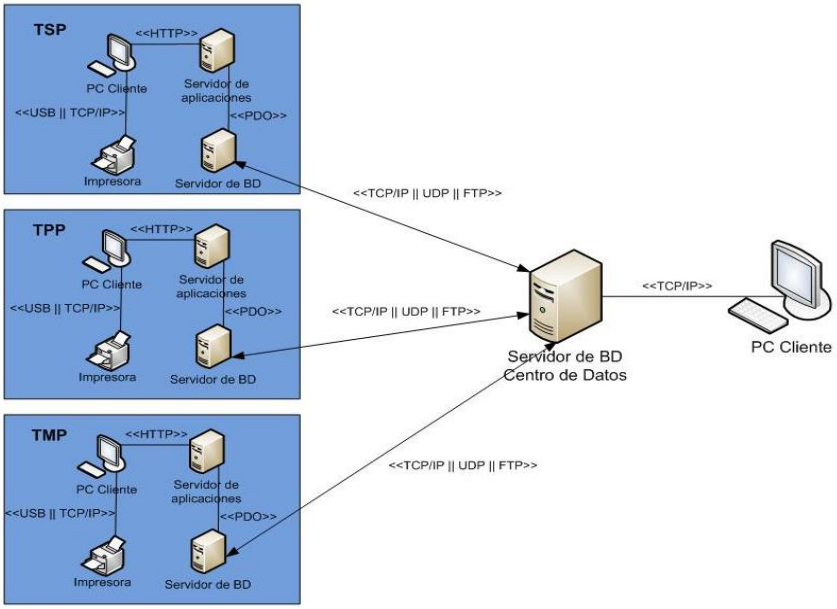

Figura 13: Diagrama de despliegue.

2.2.3.1.1 Descripción de los nodos físicos.

Los nodos pertenecientes a la figura 13 se definen a continuación:

- **Nodo PC Cliente:** Computadora con el sistema operativo Linux y el navegador Mozilla Firefox, mediante el cual los usuarios podrán acceder al sistema y hacer uso del mismo.
- **Nodo Servidor de Base de Datos (Centro de Datos):** Servidor en que se encontrará la base de datos que contendrá la información de todas las instancias de los tribunales, es decir, TSP15, TPP16 y TMP17.
- **Nodo Servidor de bases de datos (Tribunal Supremo Popular):** Servidor en que se encontrará la base de datos que contendrá toda la información del TSP. Este servidor se sincronizará con el Centro de Datos para enviar la información del día en horario no laboral y una vez al día, específicamente de 12:00am a 6:00am.
- **Nodo Servidor de bases de datos (Tribunales Populares Provinciales):** Servidor en que se encontrará la base de datos que contendrá toda la información de los TPP, al igual que el TSP cada uno se sincronizará con el Centro de Datos en horario no laboral para enviar su información diaria.

[15] Tribunal Supremo Popular.
[16] Tribunal Provincial Popular.
[17] Tribunal Municipal Popular.

- **Nodo Servidor de aplicaciones:** Servidor que contendrá todo lo referente al sistema, se incluirán los archivos necesarios para que el usuario pueda tener acceso, manejará las informaciones específicas de los registros, así como sus clases; almacenará la configuración general del sistema, las clases y librerías externas además de los plugins de instalación de la aplicación.
- **Nodo Impresora:** Dispositivo de impresión necesario para llevar a formato duro los diferentes documentos necesarios para el funcionamiento de los tribunales.

2.3 Patrones de diseño.

Un patrón de diseño es una solución simple, elegante e iterativa a un problema específico. Esta solución no es un diseño terminado que puede traducirse directamente a código, sino una descripción sobre cómo resolver el problema planteado, la cual puede ser utilizada en diversas situaciones. Los patrones documentan y explican problemas de diseño, y luego discuten una buena solución a dicho problema. Con el tiempo, los patrones comienzan a incorporarse al conocimiento y experiencia colectiva de la industria del software, lo que demuestra que el origen de los mismos radica en la práctica misma más que en la teoría. A continuación se enuncian los patrones que fueron utilizados en el desarrollo de la solución.

2.3.1 Patrones Gang of Four (GOF).

De este grupo de patrones los utilizados son el Factory Method (Método de fabricación) y el Decorator (Decorador).

2.3.1.1 Método de fabricación (Factory Method).

El patrón de diseño Factory Method consiste en utilizar una clase constructora abstracta con varios métodos definidos y otros abstractos. En esencia permite que una clase delegue en sus subclases la creación de objetos. Todos los objetos instanciados normalmente provienen de una misma clase padre, siendo su comportamiento el que los diferencia.

En el desarrollo de la solución de esta investigación, el patrón Factory Method es utilizado para crear y retornar instancias de los repositorios de las entidades y funciona de la siguiente manera: se realiza una petición al contenedor de servicios en la que se le pasa por parámetros el nombre de un servicio, tal y como se muestra a continuación.

```php
public function peritosPorPruebaPericial($idPruebaPericial){
    $peritos = $this->get('comun.servicios.Perito')->obtenerXPruebaPericial($idPruebaPericial);
    return $peritos;
}
```

Figura 14: Ejemplo de petición al contenedor de servicios.

En este servicio está definida la clase repositorios, el Factory Method y la entidad a la que se refiere el servicio:

```
comun.servicios.Perito:
    class: Base\ComunBundle\Repository\PeritoRepository
    factory_service: doctrine.orm.default_entity_manager
    factory_method: getRepository
    arguments: [Base\ComunBundle\Entity\AG\Perito]
```

Figura 15: Servicio de la clase service.yml

Luego internamente se realiza una llamada a la clase EntityManager.php, pasándole por parámetro la entidad identificada en el servicio. La clase EntityManager.php es la encargada de implementar el patrón Factory Method, lo que se hace específicamente en el método getRepository($entityName).

```php
public function getRepository($entityName)
{
    $entityName = ltrim($entityName, '\\');

    if (isset($this->repositories[$entityName])) {
        return $this->repositories[$entityName];
    }

    $metadata = $this->getClassMetadata($entityName);
    $repositoryClassName = $metadata->customRepositoryClassName;

    if ($repositoryClassName === null) {
        $repositoryClassName = $this->config->getDefaultRepositoryClassName
    }

    $repository = new $repositoryClassName($this, $metadata);

    $this->repositories[$entityName] = $repository;

    return $repository;
}
```

Figura 16: Ejemplo de método que implementa el patrón Factory Method.

2.3.1.2 Decorador (Decorator).

Este patrón se especializa en añadir responsabilidades a objetos concretos de manera dinámica sin afectar a otros objetos, proporcionando más flexibilidad que la herencia

estática y evitando así que las clases más altas en la jerarquía estén demasiado cargadas de funcionalidad.

En la implementación de la solución este patrón se evidencia cuando las clases de la vista hacen una inclusión en tiempo de ejecución de información existente en otras clases vistas mediante la cláusula 'include' que define el motor de plantillas Twig. A continuación se muestra una página decorada con el uso de este patrón.

```
<div id="id_comparecencia2" class="accordion-body collapse">
    <div class="accordion-inner">
        <div id="fecha_hora_comparecencia">
            {% include "ComunesBundle:Paginas/FechaSennalamiento:
                        FechaSennalamiento.html.twig" %}
        </div>
    </div>
</div>
```

Figura 17: Ejemplo de página decorada mediante el uso del patrón Decorador.

2.3.2 Patrones GRASP.

Los patrones GRASP tienen como objetivo principal la asignación de responsabilidades a objetos o clases de objetos. Los patrones GRASP utilizados en la construcción de la solución fueron Método de fabricación, Decorador, Creador, Experto, Controlador e Inyección de dependencias.

2.3.2.1 Creador.

Este patrón es el encargado de asignar responsabilidades relacionadas con la creación de instancias y su principal objetivo es que el creador de la instancia sea capaz de conectar con la instancia producida en cualquier evento. De modo que responde a la siguiente interrogante: ¿Quién es responsable de crear una nueva instancia de alguna clase?

El uso de este patrón se evidencia en las clases gestoras (Gtr) en el momento de crear nuevos documentos generados, estos documentos se generan cuando se crea un escrito de prueba y cuando el juez o el abogado disponen sobre este. Las clases gestoras al recibir toda la información referente a estos documentos son las responsables de crear los mismos.

A continuación se muestra un ejemplo de lo explicado anteriormente.

```
public function getDocumentoGenerado($tramiteActual, $documentoFisico, $documentoDigi
    $documentoAdjuntoGtr = $this->get('comunes.DocumentoAdjuntoGtr');
    $documentosAdjuntos = $documentoAdjuntoGtr->crearDocumentosAdjuntos($documentoFis
    return $this->getGestorEscrito()->crearDocumentosGenerado($documentosAdjuntos, $t
}
```

Figura 18: Uso del patrón creador en la clase EscritoAclaracionInformePericialController.

También en las clases controladoras se evidencia este patrón, que son las responsables de la creación de los formularios que se mostrarán en las vistas.

```
public function principalAction() {
    #Modelo y Ruta del grid de prueba pericial
    $modeloPericial = $this->get('comunes.gridpericial.tm');
    $rutaPericial = new RutasGrid();
    $rutaPericial->setRutaDatosAjax('comunes_gridpericial_ajax');
    $frmPericial = $this->createForm(new PericialType());
```

Figura 19: Uso del patrón creador en la clase PericialController.

2.3.2.2 Experto.

El patrón Experto manifiesta que la clase que debe tener la responsabilidad es la experta en información, o sea, aquella clase que contiene toda la información para cumplir con la responsabilidad. De esta forma el patrón responde a la siguiente interrogante: ¿A qué clase es más conveniente asignarle una responsabilidad en un diseño orientado a objetos?

Este patrón se observa en las clases controladoras, que son expertas en la información requerida para crear los formularios que se muestran en las interfaces de usuario.

```
$form = $this->createForm(new PresuncionType(), $presuncion);
if ('POST' == $request->getMethod()) {
    $form->bindRequest($request);
    if ($form->isValid()) {
        $documento = $this->getGestor()->AdicionarModificarPresuncion($pre
        $this->publicar('idDocPresntacion', $documento);
            return new Response(1);
    }
}else return $this->render('ComunesBundle:Paginas\Pruebas\PruebasPresuncio
```

Figura 20: Uso del patrón Experto en la clase PruebasPresuncionController.

2.3.2.3 Controlador.

El patrón controlador es un patrón que sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma la clase que lo evidencia es la que recibe los datos del usuario y la que los envía a las distintas clases según el método llamado. Sugiere que la lógica de negocios debe estar separada de la capa de presentación, para aumentar la reutilización del código y la vez tener un mayor control.

En la solución informática propuesta se evidencia el uso de este patrón cuando se crea para cada caso de uso una clase controladora que se encarga del manejo de la lógica del negocio correspondiente. En la siguiente figura se muestra un ejemplo de

una clase controladora correspondiente a un caso de uso del proceso de Prueba Pericial.

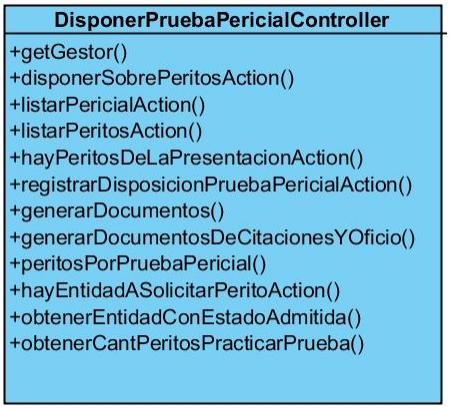

Figura 21: Clase controladora del caso de uso Disponer sobre Prueba Pericial.

Un defecto frecuente al diseñar controladores consiste en asignarles demasiada responsabilidad. Normalmente un controlador debería delegar a otros objetos el trabajo que ha de realizarse mientras coordina la actividad. En la solución esto se evidencia cuando las clases controladoras (Controller) delegan responsabilidades a las clases gestoras (Gtr) en el momento de la realización de acciones que repercuten en los datos de la base de datos.

2.3.3 Inyección de dependencias.

La inyección de dependencias consiste en pasar (inyectar) a las clases todos los objetos que necesitan (dependencias) ya creados y configurados. Estas dependencias se pueden inyectar mediante el constructor (“constructor injection”), que es el caso más común, mediante el uso de los métodos setter de una clase (“setter injection”), y directamente a través de las propiedades de las clases (“property injection”).

Para utilizar este patrón se debe recordar siempre antes de utilizar una clase, crear y configurar correctamente todas sus dependencias algo que puede obviarse si ocurre algún descuido. La solución a este problema la tiene el llamado contenedor de inyección de dependencias, que es un objeto que sabe cómo crear los objetos de una aplicación, evitando manejar todas las dependencias a mano. Para ello, este contendor conoce todas las relaciones entre las clases y las configuraciones necesarias para instanciar correctamente cada clase.

La utilización de un contenedor simplifica el uso de la inyección de dependencias en una aplicación, pero crear uno con cientos de dependencias es una tarea demasiado complicada por lo que el marco de trabajo Symfony2 incluye un completo contenedor de inyección de dependencias listo para usarse. En este ámbito aparece el término

servicio[18], el cual hace referencia a cualquier clase/objeto manejada por el contenedor de inyección de dependencias.

En Symfony2 los servicios se obtienen a través del contenedor de inyección de dependencias. Para lograr el objetivo de que un servicio pueda ser accedido desde cualquiera clase controladora, en el marco de trabajo se crea una clase abstracta ContainerAware la cual tiene el contenedor de servicios. De esta clase hereda la clase Controller que constituye la clase padre de todas las clases controladoras de la aplicación.

En la figura se muestra un ejemplo de la utilización de este contenedor en una de las clases controladoras correspondiente a un caso de uso del proceso de Prueba Pericial.

```
class DisponerPruebaPericialController extends BaseController {

    /**...5 lines */
    private function getGestor() {
        if (!$this->container->has('comunes.gtr.disponerPruebaPericial')) {

        return $this->container->get('comunes.gtr.disponerPruebaPericial');
    }
```

Figura 22: Uso del contenedor de servicios en la clase DisponerPruebaPericialController.

El uso del contenedor de servicios en las clases controladores permite acceder a los métodos de las clases gestoras a través de los servicios publicados. Esto aumenta la flexibilidad y las posibilidades de mantenimiento ya que la clase que implementa el servicio puede ser incluso sustituida por otra sin necesidad de cambiar la clase dependiente.

2.4 Estándares de codificación.

Los estándares de codificación comprenden todos los aspectos relacionados con la generación de código. Un código fuente completo debe reflejar un estilo armonioso, como si un único programador hubiera escrito todo el código de una sola vez. Al comenzar el desarrollo de un proyecto de software, se debe establecer un estándar de codificación para asegurarse de que todos los programadores del proyecto trabajen de forma coordinada. De manera que cuando el proyecto de software incorpore código fuente previo, o cuando realice el mantenimiento de un sistema de software creado anteriormente, el estándar de codificación debería establecer cómo operar con la base de código existente.

[18] Los servicios son clases PHP que realizan cualquier tarea global dentro de la aplicación.

Identación.

La identación se realizará mediante **tabulaciones**, en lugar de insertar espacios. El tamaño de las tabulaciones es de cuatro espacios:

```
61      private function getGestor(){
62          if (!$this->container->has('comunes.RegSolicitudAclaracionIP.gtr')) {
63              throw new \LogicException('Este servicio no esta registrado en la aplicacion');
64          }
65          return $this->container->get('comunes.RegSolicitudAclaracionIP.gtr');
66      }
```

Figura 23: Ejemplo del uso de identación.

Cabecera de archivo.

Todos los archivos o clases con la extensión ".php" inician con una cabecera específica, la cual indica la información de la versión, autor de los últimos cambios y fecha de creación, tal y como se muestra a continuación:

```
<?php

/**
 * Descripcion de la clase EscritoAclaracionInformePericialController
 * @modified 21/02/2014
 * @author Alain
 */
```

Figura 24: Ejemplo del uso de cabecera de archivo.

Comentarios en las funciones.

Estos tipos de comentarios se escriben en el código para aclarar el funcionamiento de las funciones. El comentario en cuestión se realiza en la línea inmediatamente superior a la línea de código fuente que se desea aclarar y se utilizan los comentarios de doble barra (//) de la siguiente forma:

```
305     // Obtengo el tramite que dio lugar al informe
306     public function getTramiteExpedienteOrigenXInforme($idInforme){
307         $tramite = $this->getTramiteOrigenXInforme($idInforme);
308         $tramiteExp = $tramite->getTramiteExpediente();
309         return $tramiteExp;
310     }
```

Figura 25: Ejemplo de comentario en las funciones.

En caso de que el comentario sea bastante largo, se usan los comentarios multilínea con apertura (/*) y cierre (*/). El uso de asteriscos en cada inicio de línea es opcional aunque recomendable teniendo en cuenta que muchos editores los añaden automáticamente:

```
349     /**
350      *  Se registra un escrito y se Inserta en EscritoOtroPrueba para
351      *  guardar su relación con la prueba y se pone en estado SolicitudAclaracion.
352      *  Este escrito se relaciona con el Inf pericial al q se le va a pedir ampliación.
353      *  Relacion: EscritoOtro-DocumentoGenerado-Informe
354      *
355      * @return \Base\ComunBundle\Entity\AG\DocumentoGenerado
356      */
357     public function registrarEscritoAclaInforPericial($idInforme, $idExpediente, $docume
```

Figura 26: Ejemplo de comentario en las funciones.

De esta forma se logra que el programador que desee conocer la utilidad de una función determinada lo haga sin necesidad de estar analizando su código. Tanto el nombre como el comentario que acompañe a la función deben bastar para ello.

Ubicación de los archivos.

Los archivos están ubicados según convenciones establecidas por Symfony2, estructurados por directorios llamados bundles, donde cada uno de ellos implementa una sola característica dentro de la aplicación. Cada directorio contiene todo lo relacionado con esa característica, incluyendo archivos PHP, plantillas, hojas de estilo, archivos JavaScript y cualquier otra cosa necesaria.

La estructura de directorios de un bundle es simple y flexible. Por defecto el sistema de bundles sigue una serie de convenciones que ayudan a mantener el código consistente entre todos los bundles.

Denominación de los archivos.

Para la denominación de los archivos se siguen las convenciones establecidas por el equipo de arquitectura del proyecto. Ejemplo:

Las clases de gestión del negocio usan el sufijo Gtr:

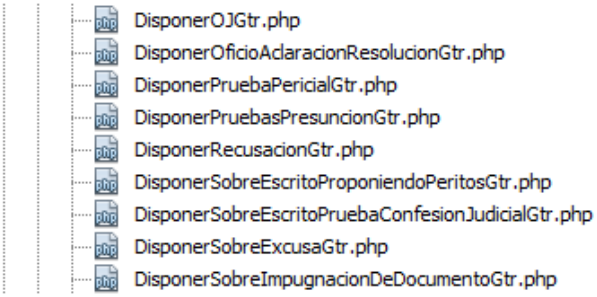

Figura 27: Ejemplo de las entidades de gestión del negocio.

Los modelos para las tablas definidos para los Grid usan el sufijo TableModel:

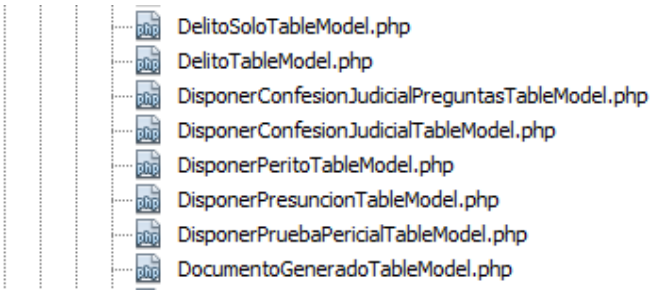

Figura 28: Ejemplo de las entidades TableModel.

Las entidades de presentación usan el sufijo EP:

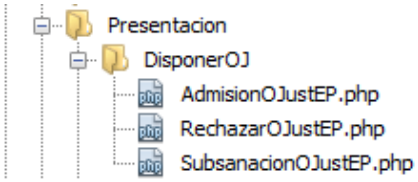

Figura 29: Ejemplo de las entidades Presentación.

En la parte de las vistas, los nombres de las páginas o plantillas html.twig siguen el estándar de la denominación de las clases:

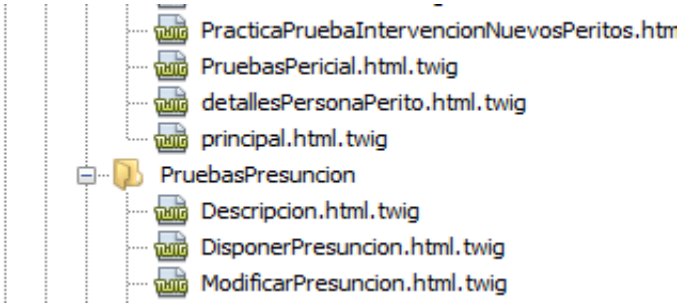

Figura 30: Ejemplo de las páginas y plantillas.

Los servicios se denominaron siguiendo los siguientes elementos:

- El nombre de las rutas será todo en minúscula utilizando como separador el caracter guión bajo (_).
- El nombre de los parámetros y servicios será todo en minúscula utilizando como separador el caracter punto (.) y en caso de ser necesario el caracter guión bajo (_).

```
comunes_presuncion_disponer:
    pattern:  /presuncion/disponer
    defaults: { _controller: ComunesBundle:PruebasPresuncion:CrudDisponerPresuncion  }
comunes_presuncion_disponer_ajax:
    pattern:  /presuncion/disponer/ajax
    defaults: { _controller: ComunesBundle:PruebasPresuncion:CargarPresuncionesDisponer }
```

Figura 31: Ejemplo de rutas.

```
# Disponer Pruebas de presuncion
    comunes.disponer_prueba_presuncion_gtr.class: SIT\SRV\ComunesBundle\Negocio\Gestor\Dispo
    comunes.disponerPruebaPresuncion.tm.class: SIT\SRV\ComunesBundle\Negocio\TableModel\Disp
```

```
comunes.pruebasPresuncionGtr:
    class: %comunes.pruebasPresuncionGtr.class%
    parent: arquitectura.basegtr
```

Figura 32: Ejemplo de parámetros y servicios.

Declaración de las clases.

Las clases están declaradas en archivos ".php". La denominación del archivo coincidirá con el de la clase. Las clases siguen las mismas reglas de las funciones, por tanto, antes de la declaración se coloca un comentario explicando su utilidad. Los nombres de las clases inician con letra mayúscula, en caso de que un nombre contenga más de una palabra, la primera letra de cada nueva palabra también se escribe en mayúscula.

```php
<?php

/**
 * Descripcion de la clase EscritoAclaracionInformePericialController
 * @modified 21/02/2014
 * @author Alain
 */
namespace SIT\SRV\ComunesBundle\Controller;

use Base\ArquitecturaBundle\Controller\BaseController;
use Base\ComunBundle\Entity\AG\DocumentoGenerado;
use Base\ComunBundle\Entity\AG\Expediente;
use Base\ComunBundle\Entity\AG\Litigante;

class EscritoAclaracionInformePericialController extends BaseController {

```

Figura 33: Ejemplo de una Clase controladora.

Estilo y reglas de escritura de código PHP.

La utilización de estilos de codificación, así como la aplicación de reglas de escritura de código que incluye una gran gama de aspectos, permite aportar un alto nivel de eficiencia al proceso de desarrollo, logrando que la aplicación sea más robusta y comprensible. De esta forma y como parte fundamental dentro de la propuesta de solución técnica de esta investigación, se decidió adoptar de forma consistente determinados estilos y reglas de manera que contribuya a facilitar la reutilización de código y la detección de errores.

Nombre de las variables.

Los nombres son descriptivos y concisos, por lo que sirven de identificación de las variables. No se usan grandes frases ni pequeñas abreviaciones de modo que es muy

sencillo saber qué hace cada variable con sólo conocer su nombre. Esto se aplica para los nombres de variables, funciones, argumentos de funciones y clases.

```
177     public function registrarEscrito($documentoGenerado,
178         $plantilla = $this->DocumentoCargarPlantilla($no
179         $html = $this->DocumentoGenerarHtml($plantilla,
180         $documentoGenerado->getHtmlPdf()->setPdf($this->
181         $documentoGenerado->getHtmlPdf()->setHtml(null);
182         $escritoOtro = new EscritoOtro();
```

Figura 34: Ejemplo de nombres de variables.

Definición de funciones.

Los nombres de la funciones están compuestos por una combinación de caracteres alfanuméricos y siempre empiezan con letra minúscula. Cuando un nombre de la función consiste de más de una palabra, la primera letra de cada nueva palabra se escribe con mayúscula.

```
public function getFechaProvidenciaDictada($idInforme){
    $tramiteOrigen = $this->getTramiteOrigenXInforme($idInforme);
    $fecha = $tramiteOrigen->getFechaCreacion();
    return $fecha;
}
```

Figura 35: Ejemplo de función con más de una palabra en el nombre.

Llamadas a funciones.

Las funciones se llaman a través de la siguiente estructura: sin la ocurrencia de espacios entre el nombre de la función, el paréntesis de la apertura, y el primer parámetro; los espacios se incluyen solo entre las comas y cada parámetro, y finalmente sin ningún espacial entre el último parámetro, el paréntesis del cierre, y el punto y coma que culmina la sentencia del código.

```
$idsPerito = $this->adicionarModificarProponerPerito($idPrueba, $datos
$this->getGestor()->InsertarPeritosPrueba($idsPerito, $idPrueba);
```

Figura 36: Ejemplo de llamadas a funciones.

Siempre incluir llaves.

Siempre a la hora de codificar un bloque de instrucciones, éste estará encerrado entre llaves, aun cuando conste de una sola línea.

```
    private function getGestorPericial() {
        if (!$this->container->has('comunes.pericial')) {
            throw new \LogicException('Este servicio no esta registrado en la aplicacion');
        }
        return $this->container->get('comunes.pericial');
    }
```

Figura 37: Ejemplo de usos de llaves.

Espacios entre signos.

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

```
    //El escrito Solicitud de Aclaracion de Inf. Per. se registra con estado Solicitud
    if($tipoEscrito == ETipoEscrito::Solicitud_Aclaracion_Informe_Pericial){
            $tipoEstado = $this->getGestorEscrito()->BuscarTipoEstado(EEstado::Informe_
            $escrito->setTipoEstado($tipoEstado);
    }
    $this->getGestor()->salvarModificarEntidad($escrito);
```

Figura 38: Ejemplo de usos de espacios entre signos.

Precedencia de operadores.

Para estar siempre seguro de la procedencia de los operadores se hace uso de paréntesis. Básicamente, lo principal es no codificar operaciones complejas y estar seguros que nuestros compañeros de equipo con menos “habilidad” comprendan todo sin problemas.

```
$abogado = $this->getGestor()->BuscarAbogadoXPersonaNatural($idPresentadoPor);
if(isset($abogado) && ($expediente == 10001389 || $expediente == 10001390)){
    $bufeteAbogado = $this->get('ag.usuariogtr')->cargarBufeteActivoAbogado($abogado->
}else{
    throw new \LogicException('Tiene que seleccionarse obligatoriamente un abogado');
}
```

Figura 39: Ejemplo de usos de procedencia de operadores.

2.5 Elementos relevantes de la solución.

Para la construcción de la solución se tuvieron en cuenta algunos elementos relevantes. A continuación se muestran las más importantes.

- Captación de los datos en el lugar de origen: La solución permite captar los datos de los peritos en el lugar de origen, ayuda a evitar errores en la trascripción de documentos.

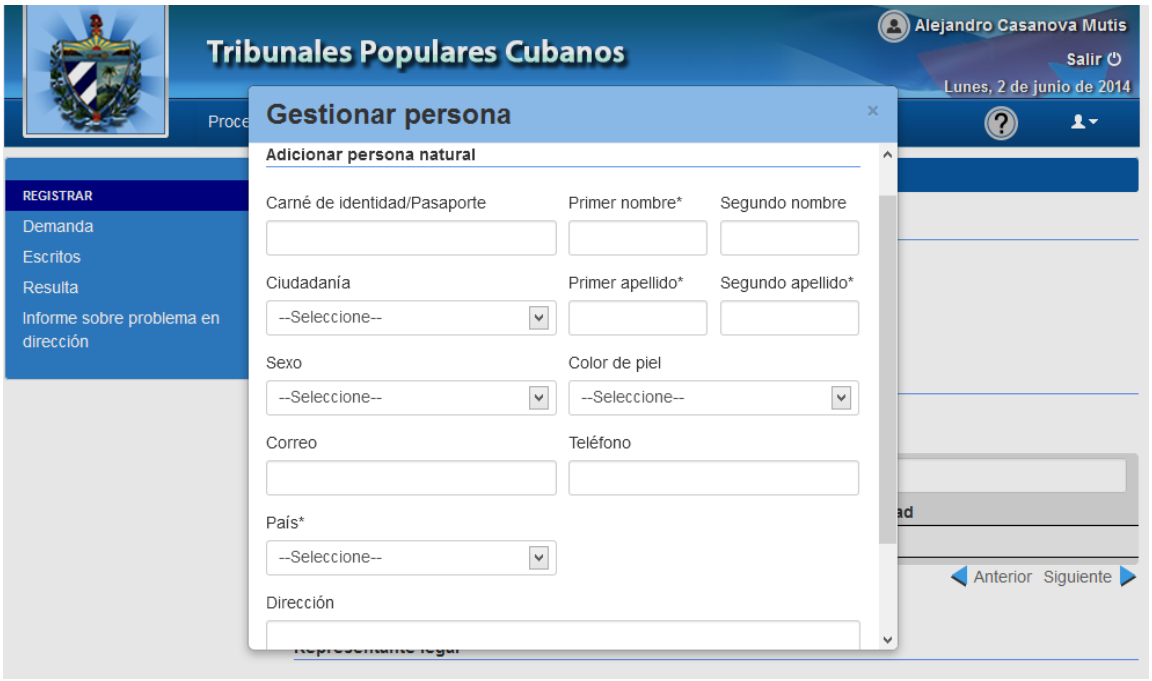

Figura 40: Imagen representativa de la captación de los datos en el lugar de origen.

- Alerta del vencimiento de los términos: Una funcionalidad importante es la alerta del vencimiento de los términos, que ayuda, guía y alerta al juez u otro usuario sobre el vencimiento de los términos de un expediente, orientando al juez diariamente en el trabajo a realizar y ayuda a evitar el incumplimiento en términos de los actos procesales.

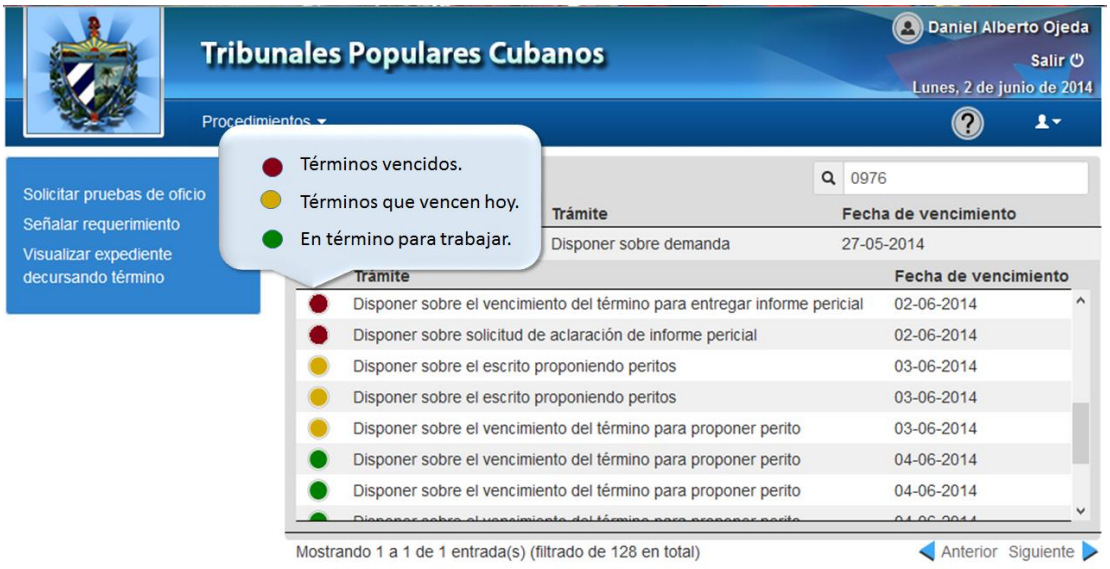

Figura 41: Imagen representativa de la alerta del vencimiento de los términos.

- Generación dinámica de documentos dentro del sistema, mediante la utilización del componente común denominado “Motor de plantillas”, lo cual contribuye a la estandarización de los mismos.

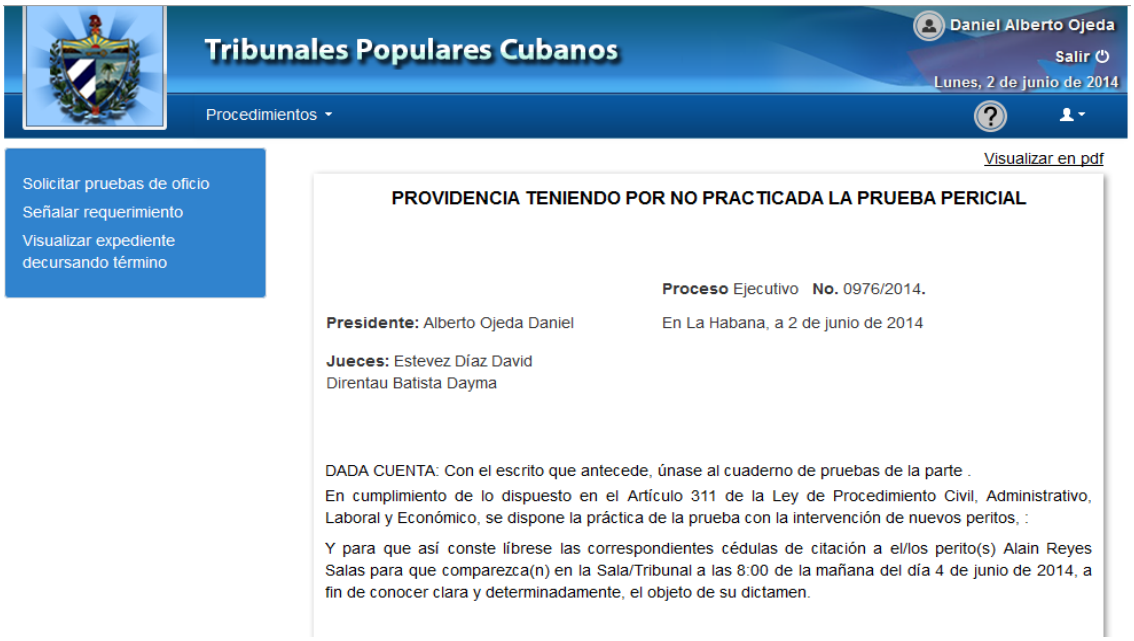

Figura 42: Imagen representativa de la generación automática de documentos.

2.6 Conclusiones parciales.

El diseño realizado para los requisitos identificados en los procesos de Prueba Pericial y de Presunción durante el presente capítulo estuvo condicionado por la metodología, arquitectura, lenguajes de programación, marcos de trabajo y herramientas definidas por el equipo de arquitectura del Sistema de Informatización para la Gestión de los Tribunales Populares Cubanos. Con la realización del diseño se obtuvieron los artefactos necesarios: modelo de datos, diagrama de clases y de secuencia para realizar posteriormente la implementación cuyo resultado fueron el diagrama de despliegue y el código fuente de la solución.

Como resultado final se obtuvo una propuesta de solución que responde al diseño e implementación de los procesos de Prueba Pericial y de Presunción y que puede ser integrada con cada uno de los subsistemas de SITPC. Los resultados obtenidos en el capítulo sentaron las bases para poder realizar la validación de la solución, tanto en términos de diseño como en términos funcionales.

Capítulo 3: Validación de la Solución

3.1 Introducción.

La validación de sistemas informáticos se realiza para medir hasta qué punto es correcta la implementación y que se cumple con las necesidades y los requisitos del usuario. (Vicente Moret Bonillo, 2012)

En este capítulo se aplican las métricas predeterminadas para medir la calidad de los artefactos del diseño y se elaboran y ejecutan los casos de prueba para verificar el correcto funcionamiento de las funcionalidades implementadas. Se hace un análisis de los resultados obtenidos en la aplicación de las métricas y en la ejecución de las pruebas de caja blanca y negra.

3.2 Validación del Diseño

En la ingeniería de software se ha hecho recurrente la necesidad de comprobar si los artefactos que se generan durante el diseño de un software son factibles. Para ello es necesario aplicar técnicas que posibiliten obtener resultados cuantitativos que evidencien si el diseño se ha realizado de forma correcta. Estos resultados posibilitan obtener respuestas efectivas que permiten determinar si se debe rediseñar un diagrama de clases, si existe una correcta relación entre las clases que los conforman, entre otros aspectos los cuales hacen de las métricas una poderosa herramienta para medir la calidad del diseño de un software.

Para validar el diseño obtenido de la solución informática propuesta se utilizaron métricas de software, con la finalidad de medir el nivel de relaciones que existen entre las clases, la complejidad de implementación y mantenimiento, así como el nivel de reutilización y asignación de responsabilidades de las mismas, para luego realizar un análisis de los resultados obtenidos.

3.3 Validación del diseño utilizando métricas.

En el capítulo 1 del presente trabajo se hace referencia a dos conjuntos de métricas de software orientado a objetos: el propuesto por Lorenz y Kidd y el propuesto por Chidamber y Kemerer.

Estas métricas de diseño basadas en clases, a las cuales suele aludirse con el nombre de conjunto de métricas LK y CK respectivamente son las empleadas para validar los artefactos de diseño generados. Específicamente se aplican dos de ellas: Tamaño operacional de las clases y Relaciones entre clases, las cuales permiten medir los siguientes atributos:

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

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

Complejidad del mantenimiento: Consiste en el grado de esfuerzo necesario a realizar para desarrollar una reparación, una mejora o una corrección de algún error de un diseño de software. Influye fuertemente en los costes y la planificación del proyecto.

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

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

3.3.1 Métrica Tamaño operacional de las clases (TOC).

El tamaño general de una clase orientada a objetos se obtiene al realizar un cálculo de sus atributos u operaciones.

Los valores grandes para la métrica TOC indican que la clase tiene mucha responsabilidad, lo que propicia un bajo nivel de reutilización de la clase y complicará la implementación y las pruebas.

Atributos	Métrica TOC
Responsabilidad	Un aumento del valor de la métrica TOC implica un aumento de la responsabilidad asignada a la clase.
Complejidad de implementación	Un aumento del valor de la métrica TOC implica un aumento de la complejidad de implementación de la clase.
Reutilización	Un aumento del valor de la métrica TOC implica una disminución del grado de reutilización de la clase.

Cálculo de atributos de la métrica TOC:

Atributo	Categoría	Criterio
Responsabilidad	Baja	TOC < = Promedio
	Media	Promedio < TOC <= 2*Promedio
	Alta	TOC > 2* Promedio
Complejidad implementación	Baja	TOC < = Promedio
	Media	Promedio < TOC <= 2*Promedio
	Alta	TOC > 2* Promedio

Reutilización	Baja	TOC > 2*Promedio
	Media	Promedio <= TOC < 2*Promedio
	Alta	TOC <= Promedio

TOC: Cantidad de operaciones por clase.

Promedio: Media Aritmética de la cantidad de operaciones por clase.

En este trabajo se aplica la métrica para el tamaño operacional de las clases en función de sus operaciones. La aplicación de dicha métrica a las clases arrojó los siguientes resultados:

Figura 43: Evaluación de la métrica TOC en el atributo responsabilidad.

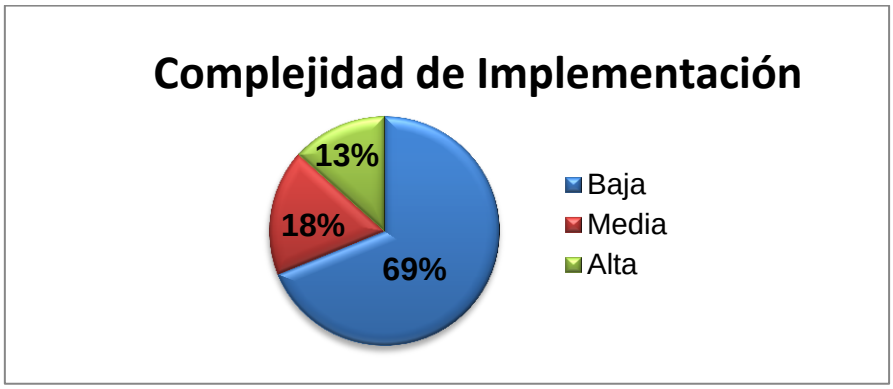

Figura 44: Evaluación de la métrica TOC en el atributo complejidad de implementación.

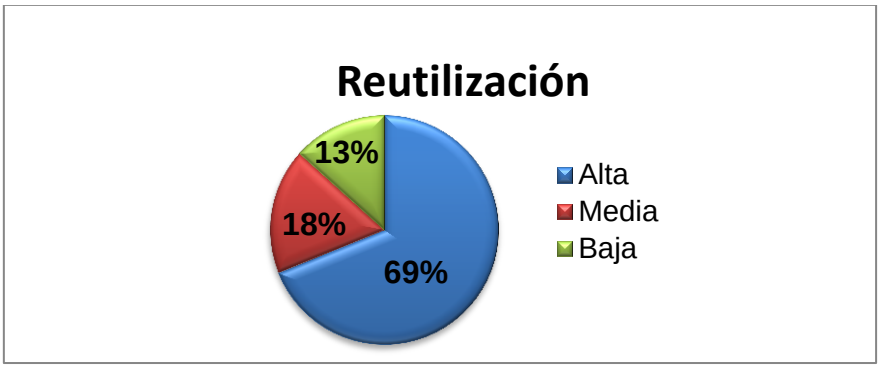

Figura 45: Evaluación de la métrica TOC en el atributo reutilización.

Luego de haber aplicado la métrica TOC y haber analizado los resultados obtenidos en los atributos anteriormente mencionados, se concluye que la realización del modelo de diseño posee la calidad requerida, propiciando que se facilite la implementación de las clases diseñadas. Los resultados obtenidos fueron satisfactorios en los diferentes atributos de calidad medidos, por lo que se puede afirmar que se obtuvo una solución eficiente con altos niveles de reutilización (69% de las clases), lo que garantiza que las funcionalidades elaboradas puedan ser reutilizadas en otros subsistemas, aspecto que tiene gran importancia dentro del marco del subsistema Común.

Además se evidencia que el proceso de asignación de responsabilidades a las clases involucradas en la solución se realizó de forma correcta (69% con alta reutilización), haciendo que el diseño cumpla con los principios del paradigma orientado a objetos. Finalmente se demuestra el bajo grado de dificultad que presenta el diseño, debido a que el 69% de las clases ostentan un bajo nivel de complejidad de implementación, lo que significa que para implementar las clases se necesite poco esfuerzo humano, aspecto que garantiza la agilización y rapidez de dicha implementación. En forma general esta métrica avala con buenos resultados el modelo del diseño de clases realizado.

3.3.2 Relaciones entre clases (RC).

La métrica Relaciones entre las clases tiene como resultado el número de relaciones de uso de una clase con otras. Esta métrica evalúa los atributos acoplamiento, complejidad de mantenimiento, reutilización y la cantidad de pruebas. De manera tal que mientras mayor sea la cantidad de relaciones de uso entre clases mayor será el acoplamiento, la complejidad de mantenimiento y la cantidad de pruebas, mientras que por otro lado su reutilización disminuye.

Atributos	**RC**
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.

Cálculo de atributos de la métrica RC:

Atributo	Categoría	Criterio
Complejidad de mantenimiento	Baja	<= Promedio
	Media	Entre Promedio y 2* Promedio
	Alta	> 2* Promedio.

Cantidad de pruebas	Baja	<= Promedio
	Media	Entre Promedio y 2* Promedio
	Alta	> 2* Promedio

Después de aplicar dicha métrica se obtuvieron los siguientes resultados:

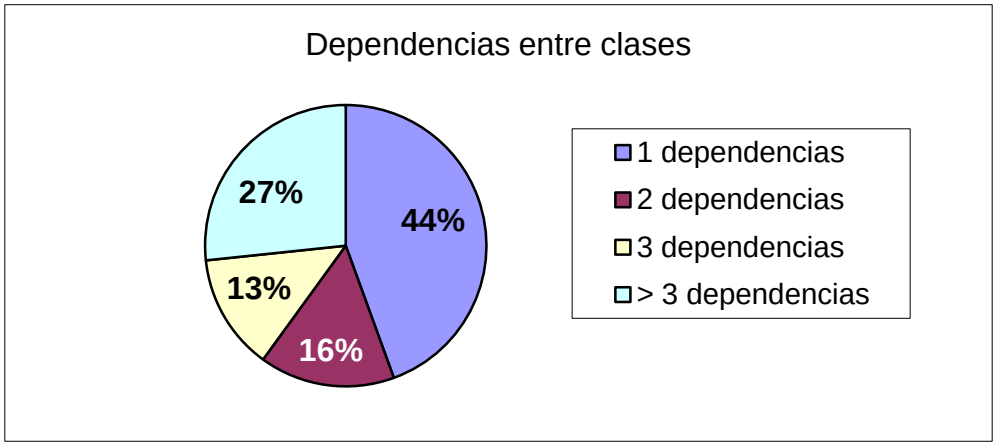

Figura 46: Representación en % de los resultados obtenidos al analizar las dependencias entre clases.

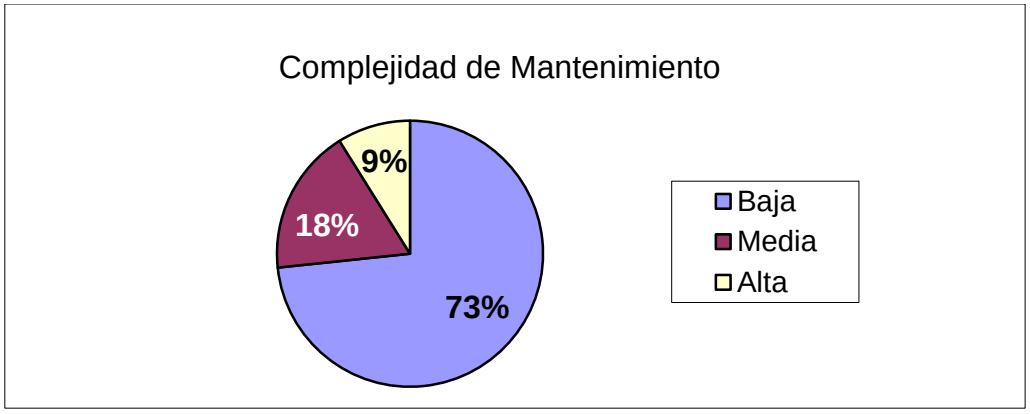

Figura 47: Representación de la métrica RC en el atributo complejidad de mantenimiento.

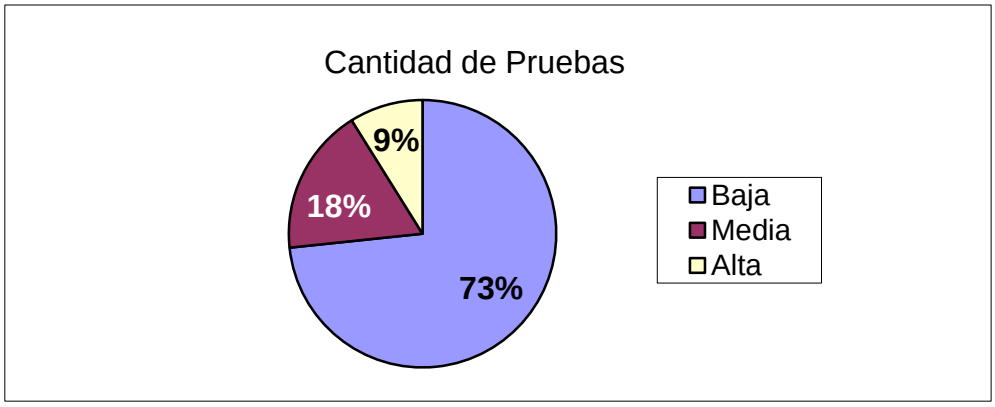

Figura 48: Representación de la métrica RC en el atributo cantidad de pruebas.

La aplicación de esta métrica de software demostró que existe una baja dependencia entre las diferentes clases de la solución propuesta, con un 44% del total de las clases con una sola dependencia, 16% con dos dependencias, 13% con tres y un 27% con más de tres dependencias. También arrojó resultados satisfactorios en los atributos complejidad de mantenimiento (73% de las clases con baja complejidad) y cantidad de pruebas (73% de las clases con bajo grado de esfuerzo para realizarles pruebas), lo que facilita las tareas de corrección, modificación, mantenimiento y pruebas a las funcionalidades implementadas.

3.4 Pruebas de software.

3.4.1 Pruebas de caja blanca.

A través de técnicas de prueba de caja blanca se obtuvieron casos de prueba que:

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

La técnica del camino básico permite al diseñador de casos de prueba obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida corno guía para la definición de un conjunto básico de caminos de ejecución. Los casos de prueba obtenidos del conjunto básico garantizan que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa. (Pressman, 2002)

Para la validación de la solución se realizó la prueba del camino básico a las funcionalidades de mayor complejidad, de manera que permitió definir los diferentes caminos independientes de las funciones y probar sus funcionamientos al menos una vez. A continuación se muestra el desarrollado del método camino básico para la funcionalidad "RegistrarDisposicionPruebaPericialAction" de la clase "DisponerPruebaPericialController".

Para comenzar primero se analiza el código y se enumeran las instrucciones:

```
public function getDatos($idExpediente, $idPresentadoPor, $peritos, $tipoEscrito){
    $datos = array();//1
    $expediente = $this->getGestorEscrito()->BuscarExpediente($idExpediente);//1
    $abogado = $this->getGestor()->BuscarAbogadoXPersonaNatural($idPresentadoPor);//1
    $bufeteAbogado = $this->get('ag.usuariogtr')->cargarBufeteActivoAbogado($abogado->getId());//1
    $gestor = $this->getGestorEscrito();//1
    $idTipoEscrito = ETipoEscrito::Solicitud_Aclaracion_Informe_Pericial;//1
    $documentoClasificacion = $gestor->BuscarTipoDocumentoClasificacion($idTipoEscrito);//1
    if($tipoEscrito == ETipoEscrito::Solicitud_Aclaracion_Informe_Pericial){//2
        $datos['nombreResolucion'] = $documentoClasificacion->getNombreCompleto();//3
        $datos['fechaProvidenciaDictada'] = $this->recuperar('_fecha');//3
        $datos['peritos'] = $peritos;//3
    }else{//4
        $peritosListos = array();//4
        foreach($peritos as $perito){//5
            $array = array();//6
            $array['nombrePerito'] = $perito[2];//6
            $array['direccionPerito'] = $perito[4];//6
            $peritosListos[] = $array;//6
        }//7
        $datos['peritos'] = $peritosListos;//8
    }//9
    $datos['nombreProceso'] = $this->getProceso();//9
    $datos['anno'] = $expediente->getFechaRadicacion()->format('Y');//9
    $datos['nombreAbogado'] = $abogado->getPersonaNatural()->getNombreCompleto();//9
    $datos['bufeteAbogado'] = $bufeteAbogado->getEstructura()->getDenominacion();//9
    $datos['litigante'] = $this->getGestor()->getLitigantes($idExpediente, $idPresentadoPor);//9
    return $datos;//9
}
```

Figura 49: Función a la que se aplica el camino básico.

Luego se realiza el grafo que va a representar el flujo de control lógico del código anterior, a través de nodos, aristas y regiones. Quedando como muestra la siguiente figura.

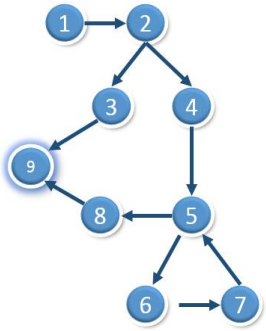

Figura 50: Grafo de flujo del código de la función de la figura anterior.

Después de haber realizado el grafo se procede a conocer la cantidad de caminos independientes que se deben buscar para probar, para ello se calcula la complejidad ciclomática, la cual está basada en la teoría de grafos. Según Pressman la complejidad se puede calcular de tres formas:

1. El número de regiones del grafo de flujo coincide con la complejidad ciclomática.
2. La complejidad ciclomática, V(G), de un grafo de flujo G se define como V(G)=A-N+2 donde A es el número de aristas del grafo de flujo y N es el número de nodos del mismo.
3. La complejidad ciclomática, V(G), de un grafo de flujo G también se define como V(G) = P + 1 donde P es el número de nodos predicado contenidos en el grafo de flujo G.

A continuación se muestran los cálculos realizados para obtener la complejidad por las tres formas mencionadas anteriormente propuestas por Pressman, las cuales deben llegan al mismo resultado.

1. Número de regiones del grafo G de la figura 50: 3.
   Entonces V(G) = 3.
2. Cantidad de aristas del grafo G de la figura 50: A = 10.
   Cantidad de nodos del grafo G de la figura 50: N = 9.
   Entonces V(G) = (A - N) + 2 = (10 - 9) + 2 = 3.
3. Cantidad de nodos predicados del grafo G: P = 2.
   Entonces V(G) = P + 1 = 3.

El resultado anterior indica que la cantidad de caminos básicos que puede tomar el algoritmo durante su ejecución es 3 y quedan definidos de la siguiente manera:

1. camino básico #1: 1 – 2 – 3 – 9.
2. camino básico #2: 1 – 2 – 4 – 5 – 8 – 9.
3. camino básico #3: 1 – 2 – 4 – 5 – 6 – 7 – 5 – 8 – 9.

Posteriormente se preparan los casos de pruebas que forzarán la ejecución de cada camino del conjunto básico.

Descripción de la función:

Esta función devuelve un arreglo de datos, los cuales se utilizan para llenar un documento predefinido que existe en forma de plantilla en la base de datos de la aplicación y para posteriormente pasarlo a definitivo, el cual será utilizado como documento oficial dentro de los procesos jurídicos que se llevan a cabo en los tribunales.

La descripción anterior es común para los tres casos de pruebas que se especifican a continuación, debido a que en los tres casos la función debe devolver un arreglo con datos para llenar la plantilla.

Caso de pruebas del camino básico #1:

Condiciones de ejecución:

Para esta prueba es necesario que las variables “$idExpediente”, “$idPresentadoPor” contengan datos, que la variable “$peritos” contenga los datos de uno o varios peritos y que “$tipoEscrito” tenga el valor “Solicitud_Aclaracion_Informe_Pericial” del enum “ETipoEscrito”.

Entrada:

Las variables “$idExpediente”, “$idPresentadoPor” contienen valores numéricos, la variable “$peritos” contiene datos de uno o varios peritos, los datos de estas variables son obtenidos de la base de datos y “$tipoEscrito” tiene como valor “Solicitud_Aclaracion_Informe_Pericial” del enum “ETipoEscrito”.

Salida esperada:

Partiendo de los valores de entrada se espera que la función devuelva un arreglo con los siguientes datos: nombre de la resolución, fecha de la providencia dictada, un solo perito, nombre del proceso, año, nombre del abogado, bufete y litigante.

Salida real:

El resultado obtenido coincide con la salida esperada.

Caso de pruebas del camino básico #2:

Condiciones de ejecución:

Para esta prueba las variables “$idExpediente”, “$idPresentadoPor” deben contener datos, la variable “$peritos” debe estar vacía y “$tipoEscrito” debe tener un valor del enum “ETipoEscrito” distinto de “Solicitud_Aclaracion_Informe_Pericial”.

Entrada:

Las variables “$idExpediente”, “$idPresentadoPor” con valores numéricos obtenidos de la base de datos, la variable “$peritos” está vacía y “$tipoEscrito” con un valor del enum “ETipoEscrito” distinto de “Solicitud_Aclaracion_Informe_Pericial”.

Salida esperada:

Teniendo en cuenta los valores de entrada se espera que la función devuelva un arreglo con los siguientes datos: nombre del proceso, año, nombre del abogado, bufete y litigante.

Salida real:

El resultado obtenido coincide con la salida esperada.

Caso de pruebas del camino básico #3:

Condiciones de ejecución:

Para que se ejecute esta prueba las variables “$idExpediente”, “$idPresentadoPor” deben contener datos, la variable “$peritos” debe tener los datos de uno o varios peritos y la variable “$tipoEscrito” debe estar inicializada con un valor del enum “ETipoEscrito” diferente de “Solicitud_Aclaracion_Informe_Pericial”.

Entrada:

Las variables “$idExpediente”, “$idPresentadoPor” con valores numéricos obtenidos de la base de datos, la variable “$peritos” con los datos de uno o varios peritos y “$tipoEscrito” con un valor del enum “ETipoEscrito” diferente de “Solicitud_Aclaracion_Informe_Pericial”.

Salida esperada:

Según los valores de entrada se espera que la función devuelva un arreglo con los siguientes datos: datos de uno o varios peritos listos para mostrar en la plantilla, nombre del proceso, año, nombre del abogado, bufete y litigante.

Salida real:

El resultado obtenido coincide con la salida esperada.

Estas pruebas se le aplicaron a funcionalidades críticas de los casos de usos más ilustrativos de la solución. Una vez ejecutados todos los casos de pruebas se detectaron errores internos los cuales fueron corregidos. Finalmente se comprobó que el resultado para cada uno de los flujos más relevantes de la solución es el esperado.

3.4.2 Pruebas de caja negra.

Las pruebas de caja negra se llevan a cabo sobre la interfaz del software, sin tener en cuenta el comportamiento interno y la estructura del programa.

Dentro de los métodos de las pruebas de caja negra se utilizó el de partición de equivalencia, el cual divide el dominio de entrada de un software en clases de datos, numéricos y lógicos, de los que se derivan casos de prueba.

Estos casos de prueba pretenden demostrar que:

- las funciones del software son operativas,
- la entrada se acepta de forma correcta,
- se produce una salida correcta,
- la integridad de la información externa se mantiene.

Y permiten encontrar estos tipos de errores:

- funciones incorrectas o ausentes,
- errores en la interfaz,
- errores en estructuras de datos o en accesos a bases de datos externas,
- errores de rendimiento,
- errores de inicialización y de terminación.

La partición equivalente se dirige a la definición de casos de prueba que descubran la mayor cantidad posible de clases de errores. El diseño de estos casos de prueba se basa en una evaluación de las clases de equivalencia, las cuales representan un conjunto de estados válidos o no válidos para cada condición de entrada. Permitiendo obtener un conjunto de condiciones de entrada que ejerciten completamente todos los requisitos funcionales del software.

Escenario	**Descripción**	**Nombre(s) y apellidos de los peritos**	**Fecha de acta de juramento de cargo**	**Documentos que acompañan**	**Presentado por**	**Tipo de escrito**	**Respuesta del sistema**	**Flujo central**
EC 1.1 Registrar informe pericial	El sistema permitirá registrar un nuevo informe pericial entregado por el/los perito(s).	V Paola Guillén Pacheco	V 14/02/2014	V Ver DCP Registrar Documentos que acompañan	V Ver variable	V Informe pericial	Registra el informe pericial	1. Selecciona en el menú izquierdo la opción Registrar escrito 2.Llena los datos 3. Presiona el botón Registrar.
EC 1.2 La operación es cancelada	La operación es cancelada.	N/A					Cierra la interfaz y no guarda los cambios.	1. Selecciona en el menú izquierdo la opción Registrar

				escrito 2.Llena los datos 3. Presiona el botón Cancelar.

Las pruebas fueron realizadas por un equipo de calidad del proyecto al cual pertenece este trabajo, se realizaron tres iteraciones del flujo completo de las funcionalidades de los procesos de Prueba Pericial y de Presunción, durante la ejecución de las mismas se encontraron mayormente errores de ortografía, de validaciones en las vistas y en la carga de datos de la base de datos, estos errores fueron resueltos en el tiempo establecido para ello, de manera que en la tercera iteración no se detectaron errores:

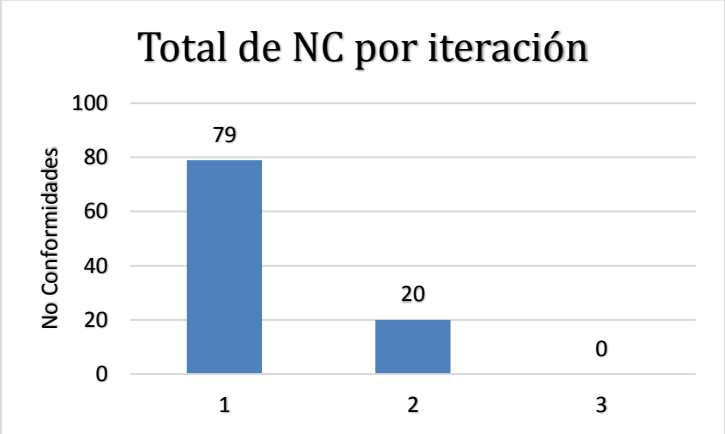

Figura 51: Continuación de la función a la que se aplica el camino básico.

3.5 Conclusiones parciales.

Durante este capítulo se abordaron las pruebas de caja blanca y las pruebas de caja negra que se le realizaron al software. Para lograr aumentar la detección de No Conformidades se conformaron los casos de prueba a ejecutar. Se mostraron los resultados obtenidos siendo posible comprobar que el software posee una calidad aceptable. La aplicación de métricas permitió determinar que el diseño fomentaba la reutilización de las clases y que no es difícil de mantener.

Conclusiones generales

La realización del marco teórico de la investigación posibilitó caracterizar la metodología de desarrollo, la arquitectura del sistema, las herramientas, tecnologías y patrones de diseño y de arquitectura, permitiendo la adquisición de los conocimientos necesarios para el diseño e implementación de la solución informática propuesta.

La construcción del modelo del diseño permitió generar los elementos fundamentales para facilitar y agilizar la implementación de la solución.

Teniendo como base el modelo de diseño, se implementó una solución informática funcional que contribuye a la agilización y consecutividad de los procesos de Prueba Pericial y de Presunción del SITPC.

La validación de la solución informática mediante la aplicación de métricas y pruebas de software, permitió avalar con buenos resultados la calidad del diseño y de las funcionalidades implementadas.

Recomendaciones

Luego de haber cumplido los objetivos de la investigación se recomienda:

Realizar las pruebas de aceptación y encuestas periódicas sobre la efectividad del sistema, para medir el grado de satisfacción de los usuarios, así como funcionalidades que se le pueden incorporar. Todo esto previendo un futuro perfeccionamiento del sistema.

Tomar esta investigación como material de estudio para la implementación de otros procesos de pruebas similares.

Bibliografía referenciada

Abogados en Madrid y Granada. 2014. Diccionario jurídico. Jurisconsultas. [En línea] 2014. [Citado el: 20 de 05 de 2014.] http://www.ic-abogados.com/diccionario-juridico/presunciones/33.

Álvarez, Sara. 2010. Desarrollo Web. desarrolloweb.com. [En línea] 2010. http://www.desarrolloweb.com/articulos/sistemas-gestores-bases-datos.html.

Bonanata, Maximiliano. 2008. Programación y Algoritmos. 2008. ISBN: 9875261564 .

Bradenbaugh, Jerry. 2000. Aplicaciones JavaScript. [ed.] España : Anaya 2000 Madrid. Madrid : Anaya Multimedia, 2000. pág. 540.

Eguiluz, Javier. 2013. Desarrollo web ágil con Symfony2. s.l. : easybook versión 5.0-DEV, 2013.

Gamma, E., y otros. 1995. Desing Patterns. s.l. : Addison Wesley, 1995.

IEEE. 1990. Computer dictionary. s.l. : IEEE Computer Society, 1990. 610.

Jacobson, Ivar, Booch, G. y Rumbaugh, J. 2000. El Proceso Unificado de Desarrollo de Software. s.l. : Madrid: Addison Wesley, 2000.

Kahwe Smith, Lukas. Abril, 2009. Introduction to the Doctrine Object Relational Mapper. Abril, 2009.

Levene, Ricardo. 1993. Manual de Derecho Procesal Penal. Buenos Aires : Depalma, 1993.

Martínez.-Ltdo., Don Ferran González i. 2008. Ferran Abogados & Asociados. [En línea] 2008. http://www.ferranabogados.com.

Matos García, Rosa María. 2005. Sistemas de Bases de Datos. 2005.

Muñóz, D.C. 2008. MÉTRICAS DE LA CALIDAD DEL SOFTWARE. s.l. : La Mancha, 2008.

Musciano, C. y Kennedy, B. 1999. HTML, La Guía Completa. s.l. : O'Reilly, 1999.

PostgreSQL Global Development Group. 2014. PostgreSQL 9.1.13 Documentation. PostgreSQL. [En línea] 2014. [Citado el: 10 de 06 de 2014.] http://www.postgresql.org/docs/9.1/static/intro-whatis.html.

Pressman, Roger S. 2002. Ingeniería del Software, un enfoque práctico. s.l. : McGraw-Hill, 2002. 601.

Proyecto SITPC. 2014. CEGEL_SITPC_0120_6 Arquitectura Vista de Entorno de Desarrollo Tecnológico.doc. 2014.

Raibal, Isabel Martínez. 2011. La informática jurídica: mecanismo de gestión de la información jurídica. España : s.n., 2011.

Rodríguez, Txema. 2012. Genbeta Dev. [En línea] 2012. [Citado el: 19 de 04 de 2014.] http://www.genbetadev.com/frameworks/bootstrap.

Segura, Benjamín. 2014. Servidor Apache gratis. Portal programas. [En línea] 2014. [Citado el: 10 de 06 de 2014.] http://gratis.portalprogramas.com/Servidor-Apache.html.

Sommerville, Ian. 2005. Ingeniería del Software. Madrid : Pearson Educación, S.A., 2005. ISBN: 84-7829-074-5.

Sun MicroSystems. 2013. NetBeans. NetBeans. [En línea] 2013. https://netbeans.org/index_es.html.

Téllez Valdes, Julio Alejandro. 1987. Informática Jurídica: Documentaria, Control y Gestión y Metadocumentaria. [aut. libro] Julio Téllez Valdes. Derecho Informático. s.l. : Universidad Nacional Autónoma de México, 1987.

Terminology for Software Engineering Environment (SEE) and Computer-Aided Software Engineering (CASE). Logee, D. y Terry, B. Abril 1990. Abril 1990.

The Apache Software Foundation. 2014. Índice de Directivas. Apache HTTP Server Project. [En línea] 2014.[Citado el: 10 de Junio de 2014.]http://httpd.eu.apache.org/docs/trunk/es/mod/directives.html.

—. McClure, Carma. Abril 1989. Abril 1989.

The PHP Group. 2014. PHP. PHP. [En línea] 2014. [Citado el: 12 de Mayo de 2014.] http://www.php.net/.

Vicente Moret Bonillo, Elena Ernández Pereira, Eduardo Mosqueira. 2012. Validación y Usabilidad de sistemas informáticos. 2012.

Bibliografía consultada

Álvarez, Alejandro Loredo. Biblioteca Jurídica Virtual. [En línea] biblio.juridicas.unam.mx/libros/6/2940/32.pdf.

Coello, Rolando Alfredo Hernández León y Sayda. 2011. El proceso de investigación científica. Ciudad de La Habana : Editorial Universitaria, 2011.

IEEE. 1993. Computer dictionary. s.l. : IEEE Standars Collection, 1993. 610.

Oracle Corporation. NetBeans. NetBeans. [En línea] Oracle Corporation. [Citado el: 19 de 04 de 2014.] https://netbeans.org/index.html.

Métodos "I + D" de la Informática. Barchini, Graciela Elisa. 2005. Santiago del Estero, Argentina : s.n., 2005, Revista de Informática Educativa y Medios Audiovisuales, Vol. II, págs. 16 - 24. ISBN/1667-8338.

The CASE Experience. 1989. 4, McGraw-Hill, Inc. Hightstown, NJ, USA : Journal, Abril de 1989, Vol. 14, págs. 235-244.

Norambuena, Daniel Antonio Ortega. 2012. Tesis electrónicas de la Universidad de Chile. [En línea] 2012. [Citado el: 20 de 01 de 2014.] www.tesis.uchile.cl/bitstream/handle/2250/111915/cf-ortega_dn.pdf?sequence=1.

Enseñanza de Derecho y Nuevas Tecnologías, Informática jurídica. Peláez, Carlos. 2001. 40, Bolivia : Alfa Redi, 2001.

Peña, Juan Manuel Fernández. 2010. Universidad Venezuela. [En línea] 07 de 2010. [Citado el: 05 de 06 de 2014.] http://www.uv.mx/personal/jfernandez/files/2010/07/Pruebas-de-Sistema.pdf.

Rolando Alfredo Hernández León, Sayda Coello González. 2011. El proceso de investigación científica. La Habana : Editorial Universitaria, 2011.

Santiago., Lic. Mónica Flores López y Mtra. María de Lourdes. [En línea] http://www.utvm.edu.mx/OrganoInformativo/orgJul07/RUP.htm.

Universidad Politécnica de Valencia. Departamento de Sistemas Informáticos y Computación. Universidad Politécnica de Valencia. [En línea] [Citado el: 10 de 06 de 2014.] www.dsic.upv.es/asignaturas/eui/mtp/doc-practicas/intro_case_SA.pdf.

Visual Paradigm para UML . www.software.com.ar. [En línea] [Citado el: 10 de 06 de 2014.] http://www.software.com.ar/visual-paradigm-para-uml.html.