Universidad de las Ciencias Informáticas
Facultad 3

Informatización de los procesos Súplica y Desistimiento del Proyecto de Informatización para la Gestión de los Tribunales Populares Cubanos.

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

Autores: Osvaldo Santos Acosta
Samuel Adrián Cruz Echevarria

Tutores: Ing. Yulia Fustiel Alvarez
Ing. Mailen Edith Escobar Pompa

La Habana, Junio de 2014

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 firmo la presente a los ____ días del mes de _____ del año _____.

_____
Osvaldo Santos Acosta
Firma del Autor

_____
Samuel Adrián Cruz Echevarria
Firma del Autor

_____
Ing. Yulia Fustiel Alvarez
Firma del Tutor

_____
Ing. Mailen Edith Escobar Pompa
Firma del Tutor

DATOS DE CONTACTO

Tutor: Ing. Yulia Fustiel Alvarez

Correo: yfustiel@uci.cu

Tutor: Ing. Mailen Edith Escobar Pompa

Correo: meescobar@uci.cu

Autor: Osvaldo Santos Acosta

Correo: osantos@estudiantes.uci.cu

Autor: Samuel Adrián Cruz Echevarria

Correo: sacruz@estudiantes.uci.cu

AGRADECIMIENTOS

A mi mamá por su amor y apoyo incondicional.

A Jorge por ser más que un padre para mí, por apoyarme y ayudarme a ser quien soy hoy.

A monseñor Héctor Luis Peña por ayudarme a ser mejor persona y ser un ejemplo a seguir.

A mis abuelos, mi tía y mi prima por su preocupación y sus consejos.

A May por el apoyo constante y por ayudarme a recorrer este camino.

A Yulia y Yosvany por sus revisiones, sugerencias y regaños.

Al equipo de Tribunales por estar siempre ahí para responderme las dudas, en especial a Yosvany, Yosviel, Pompa, Ivet, Yamilé, Isabel, Pedro Frank y Franklin. A Hardam por su ayuda.

A Jess, Ángel y Gelson por la ayuda brindada y el tiempo dedicado.

A Naivy, Knito, Titica, Dagmar, Orlando, Félix y Darian por los buenos momentos que pasamos juntos.

A mi compañero de tesis por el esfuerzo realizado para salir adelante frente a las dificultades.

Al tribunal por todo el tiempo dedicado, por sus acertados consejos y sugerencias.

A todos los amigos que me hicieron compañía durante la carrera.

A todos los profesores de la carrera por su formación.

Osvaldo

Quisiera empezar agradeciendo a mi familia, que en mi opinión son imprescindibles para poder cumplimentar una carrera universitaria.

A mi madre por los momentos en que tuvo que contar hasta 10 para no darme una mala contesta y por los muchos consejos que me dio no solo en los 5 años de la carrera sino en toda mi vida.

A mi padre que durante los 5 años de la carrera me complacía con mis comidas favoritas, por lo mucho que disfruto al comer de su sazón. A su esposa Regina que me ha ayudado mucho y me quiere como el hijo varón que siempre quiso.

A mis tíos Maritza y Ramón que están conmigo semana tras semana dándome apoyo no solo en mi vida como estudiante universitario sino desde que nací.

A mis tíos Aldo y Raysa que aunque están lejos juegan un papel importantísimo en mi formación, por todo el apoyo que me han dado en mi vida como persona.

A mis primas que quiero como hermana porque así nos han criado, Winie por ser mi compañera desde que éramos unos bebitos, Indira por el cariño que siempre me trasmites y Zuzet por ser hermana mayor y ejemplo a seguir.

A mi hermana de sangre Olivia que aunque se encuentra lejos representa mucho para mí.

A mi novia Zoemi que tanto me ha enseñado en el tiempo que llevamos de relación, por su paciencia y dedicación. A su familia que me han aceptado como soy y me hacen sentir como uno de ellos.

A mis compañeros de la universidad que me han enseñado la gran diversidad de pensamientos que existen y la mejor forma de lidiar con eso.

A mis compañeros del grupo tanto del original, que quedamos tan poquitos, como los que adquirí cuando nos unieron, que sepan que los considero a todos como amigos.

A mi compañero de tesis que me ha demostrado de lo que puede ser capaz uno cuando se quiere.

AGRADECIMIENTOS

A mis tutoras Yulia y Mailen que nos brindaron su apoyo y su conocimiento en todo momento.

A los miembros del tribunal por ser tan educativos en todo momento.

Gracias a todos por estar aquí en este día tan especial.

Samuel

DEDICATORIA

A mi mamá, Jorge y mi hermanito, porque son el tesoro más grande que tengo.

A mis abuelos, mi tía y mi prima, porque a pesar de la distancia siempre han estado pendientes de mi.

A Mimí, por su amor y por estar siempre en los buenos y malos momentos.

A todos mis amigos.

Osvaldo

Dedicar este logro a mi madre, que durante toda mi vida estudiantil ha jugado un papel definitorio en mi formación y finalmente como ingeniero el cual era un sueño de ambos.

Samuel

RESUMEN

Como parte del proceso de informatización que se ha estado realizando en Cuba y en aras de lograr mejoras en la Administración de Justicia, los Tribunales Populares de Cuba emprendieron en el año 2009 el desafío de informatizar sus procesos. Para cumplir esta tarea se concibió la creación del Proyecto de Informatización para la Gestión de los Tribunales Populares Cubanos, en colaboración con la Universidad de las Ciencias Informáticas.

La presente investigación surge debido a la necesidad de informatizar los procesos de Súplica y Desistimiento de los Tribunales Populares Cubanos. Para ello se analizó el marco teórico, profundizándose en los principales conceptos y escogiéndose las herramientas y tecnologías a utilizar. Haciendo uso de la metodología seleccionada, se presentan los principales artefactos generados en los flujos de trabajo de diseño e implementación de los componentes.

Se validó la solución propuesta mediante métricas para el diseño y pruebas de verificación, utilizándose los métodos de caja blanca y caja negra. Finalmente se analiza el cumplimiento de la idea a defender planteada, demostrándose que con el sistema se contribuye a a elevar la disponibilidad y control de la información en los procesos de Súplica y Desistimiento.

Palabras clave: proceso Desistimiento, proceso Súplica, Tribunales Populares Cubanos.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

En la actualidad, ha tenido lugar un desarrollo progresivo de las Tecnologías de la Información y las Comunicaciones (TIC) a nivel mundial, las cuales están presentes en gran parte de las actividades humanas. Este desarrollo ha sido esencial para mejorar la productividad, calidad y competitividad de las empresas y organizaciones, siendo muchas veces su utilización la que brinda una ventaja frente a la competencia.

Entre las distintas áreas que las entidades se centran en mejorar se encuentra la gestión de la información que se genera en sus procesos. La gestión de la información es "*todo lo relacionado con la obtención de la información adecuada, en la forma correcta, para la persona indicada, al costo adecuado, en el tiempo oportuno, en el lugar apropiado, para tomar una decisión correcta*"(Ponjuán, 1998). Mejorarla va a permitir a las organizaciones aumentar la calidad de sus productos y la eficiencia con la que brindan sus servicios, fundamentado en la realización de una toma de decisiones acertada.

En Cuba se han realizado esfuerzos por insertar las TIC dentro de las organizaciones. Desde el triunfo de la revolución se han llevado a cabo acciones para la informatización de los distintos sectores de la economía, la salud, la educación y de ramas como la Informática Jurídica.

La Universidad de las Ciencias Informáticas (UCI) es una de las principales instituciones cubanas dedicadas al desarrollo de software y tiene como misión formar profesionales comprometidos con su patria y altamente calificados en la rama de la informática, capaces de producir aplicaciones y brindar servicios informáticos de alto valor agregado a partir de la vinculación estudio-trabajo como modelo de formación. Para poner en práctica este modelo se crearon en la UCI varios centros de producción de software, entre los que se encuentra el Centro de Gobierno Electrónico (CEGEL), que tiene como objetivo satisfacer las necesidades de clientes gubernamentales mediante el desarrollo de productos, servicios y soluciones integrales de alta confiabilidad, calidad, competitividad, fidelidad y eficiencia (GESPRO, 2013). Entre los clientes del centro CEGEL están los Tribunales Populares de Cuba (TPC) que, al igual que otras entidades existentes en el país, buscan informatizar sus procesos para mejorar la gestión de su información.

En los TPC la creación y control de la documentación se realiza de forma manual. Esto trae consigo la introducción de posibles errores, tachaduras, borrones y que en ocasiones existan números de

expedientes duplicados. El almacenamiento de los expedientes se realiza en estantes lo que provoca su deterioro debido a la humedad, insectos y roedores. Durante la ejecución de los procesos se generan documentos como escritos, providencias y autos, en los cuales no se reutiliza la información que es común para todos ellos, teniendo que escribirse la misma repetidas veces. Los problemas antes mencionados influyen negativamente en la rapidez del proceso y en ocasiones el tiempo de tramitación es elevado, debido a que los documentos que se necesitan aún no han sido generados.

El centro CEGEL se encuentra enfrascado en el desarrollo de un producto de software a solicitud de los TPC para darle solución al problema que presenta este organismo en la gestión de la información. En este contexto se crea el proyecto de Informatización para la Gestión de los Tribunales Populares Cubanos (SITPC).

Una vez estudiados los flujos de trabajo, procedimientos y la documentación referente a las leyes, se identificaron procesos como Notificación, Acumulación, Súplica y Desistimiento, que se usan de igual manera dentro de las materias Penal, Laboral, Civil-Administrativo y Económico. Estos procesos fueron organizados en el módulo Común.

De acuerdo a lo planteado en la Ley de Procedimiento Civil, Administrativo, Laboral y Económico (Amaro Salup, 1996), el proceso Súplica tiene como objetivo manifestar inconformidad con resoluciones que no son recurribles directamente en apelación o casación. Por otra parte el Desistimiento es uno de los modos de extinción del proceso, pues permite poner fin a la solicitud realizada por parte del demandante.

Actualmente en los tribunales para la ejecución de los procesos Súplica y Desistimiento se carece de retroalimentación estadística rápida, debido a que los datos son almacenados en estantes, provocando demoras en las búsquedas y obtención de la información.

El acceso simultáneo por el personal autorizado a los expedientes de cada uno de los procesos legales que se realizan en los tribunales, se dificulta, puesto que se realiza a través de la secretaria del Tribunal, conllevando a demoras en su respuesta por el esfuerzo humano requerido, devenido en la organización actual de los documentos.

Los expedientes se almacenan en formato duro en los archivos que se encuentran en la secretaría del Tribunal, y aunque están ubicados en estantes agrupados por años, el acceso a cualquier expediente se torna complejo, por el gran volumen de información. Los jueces y los fiscales necesitan tener a su alcance tanto los expedientes que se encuentran abiertos, como otros que pueden estar cerrados para realizar su

consulta, estudio y análisis. Además esta actividad se puede ver afectada en el transcurso del tiempo por el inevitable deterioro.

Según lo planteado por (Amaro Salup, 1996), los trámites tienen un término o plazo para realizarse y una vez que se vencen, la secretaria tiene que dar cuenta del vencimiento de este término al juez para que dicte una resolución. Actualmente por el alto contenido de trabajo de los jueces y secretarias, se vencen los términos sin que el proceso sea resuelto, provocando así la existencia de información inconsistente.

Por lo antes expuesto se define como **problema a resolver**: ¿Cómo gestionar la información asociada a los procesos Súplica y Desistimiento de manera que contribuya a elevar la disponibilidad y control de la información?

Se define como **objeto de estudio**: el desarrollo de software en la informática jurídica de gestión.

Para darle solución al problema planteado se define como **objetivo general**: Informatizar los procesos Súplica y Desistimiento, de manera que se contribuya a elevar la disponibilidad y control de la información en los mismos.

La investigación está enmarcada en el **campo de acción**: la informatización de los procesos Súplica y Desistimiento perteneciente al módulo Común del Proyecto SITPC.

Para dar cumplimiento al objetivo general, se plantean los siguientes **objetivos específicos**:

- Definir el marco teórico de la investigación mediante el estudio y el análisis de los principales referentes teóricos en los que se sustenta el desarrollo de software en la informática jurídica de gestión.
- Diseñar una propuesta de solución de los requisitos pactados con el cliente para los procesos de Súplica y Desistimiento.
- Implementar la solución para obtener los componentes de software de los procesos de Súplica y Desistimiento en los TPC.
- Valorar la efectividad de la solución propuesta mediante la realización de pruebas de software.

Para lograr los objetivos específicos se definieron las siguientes **tareas de la investigación**:

1. Análisis del negocio de los procesos Súplica y Desistimiento de los TPC.

2. Análisis de sistemas informáticos existentes a nivel internacional y nacional.
3. Caracterización de la metodología, las tecnologías y las herramientas a utilizar para el desarrollo de la solución.
4. Diseño de la estructura de los componentes Súplica y Desistimiento.
5. Diseño del comportamiento de los componentes Súplica y Desistimiento.
6. Diseño del modelo de datos de los componentes Súplica y Desistimiento.
7. Validación del diseño propuesto.
8. Implementación de los componentes Súplica y Desistimiento.
9. Diseño del diagrama de despliegue del sistema.
10. Diseño de casos de prueba de los componentes Súplica y Desistimiento.
11. Validación de la implementación realizada.
12. Validación de las variables de la investigación.

Como **idea a defender** se plantea: El desarrollo de los componentes Súplica y Desistimiento del proyecto SITPC contribuirá a elevar la disponibilidad y control de la información de estos procesos en los TPC.

Para el desarrollo de la investigación se emplearon los siguientes **métodos científicos**:

- El método analítico-sintético es utilizado para revisar la bibliografía existente acerca de los sistemas de gestión jurídica existentes a nivel nacional e internacional. Además se analizan los distintos patrones de diseño, herramientas a utilizar, estándares de codificación y metodología de desarrollo de software.
- El método modelación fue utilizado para representar la estructura y comportamiento de las clases diseñadas, mediante los diagramas de clases y de secuencia en la fase de diseño.
- El método medición se evidencia mediante la utilización de las métricas para la validación del diseño.

La tesis está estructurada en tres capítulos cuyo contenido se describe a continuación:

Capítulo 1 Fundamentación teórica: En este capítulo está recogida toda la fundamentación teórica de la investigación. Se realiza un estudio de los procesos Súplica y Desistimiento, así como de la metodología, herramientas y patrones de diseño para el posterior diseño e implementación de los componentes.

Capítulo 2 Descripción de la solución propuesta: En este capítulo se encuentran recogidos todos los artefactos generados durante el proceso de diseño del software. Se explica la utilización de los patrones de diseño seleccionados, validándose los artefactos mediante la aplicación de métricas.

Capítulo 3 Implementación y pruebas: En este capítulo se encuentra toda la información referente al modelo de datos del sistema, los artefactos generados durante la implementación, el uso de los estándares de codificación seleccionados y los resultados obtenidos después de la realización de pruebas al software.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Introducción

En este capítulo se abordan los principales conceptos para la comprensión de los procesos Súplica y Desistimiento. Se realiza un análisis de los sistemas existentes a nivel internacional y nacional que realizan la gestión jurídica. Se caracterizan la metodología, herramientas, lenguajes de programación y patrones de diseño existentes así como la fundamentación de las escogidas para dar solución al problema antes expuesto.

1.1 Descripción de los procesos Súplica y Desistimiento

En Cuba el proceso de Súplica está descrito en (Amaro Salup, 1996) en los artículos 615, 616 y 617 del Consejo de Gobierno del Tribunal Supremo Popular, donde, según el artículo 615 (Amaro Salup, 1996), se define que el proceso Súplica no es más que un recurso que se autoriza contra las providencias, dentro de los tres días siguientes a su notificación, y contra los autos no recurribles directamente en apelación o casación, dentro de los cinco días siguientes en que éstos fueren notificados.

El artículo 616 (Amaro Salup, 1996) plantea que admitido el recurso, se dará traslado, con entrega de copia, a las demás partes por el plazo común de tres días, a fin de que puedan exponer lo que a sus derechos convenga; y transcurrido, el Tribunal resolverá lo que sea procedente dentro del tercer día.

Mientras que el artículo 617 (Amaro Salup, 1996) dispone que contra resoluciones que resuelvan la Súplica no cabe recurso alguno. Se exceptúan aquellas cuestiones que por su índole sean susceptibles del recurso de casación, las cuales podrán reproducirse como motivos de aquél si se interpusiere contra la resolución que ponga fin a la instancia, y siempre a condición de que el motivo se prepare con previa protesta consignada dentro del segundo día.

Para el caso de que se ordene la subsanación del escrito interponiendo el recurso, y éste no se subsana, se dictará una providencia teniendo por no establecido el recurso. El recurso se resuelve mediante auto. En este auto debe resolverse también la resolución que fue suplicada. Luego de denegada la Súplica se puede formular la protesta (Barrios, 2013).

El Desistimiento está regido por el artículo 652 de (Amaro Salup, 1996), y es el proceso mediante el cual el demandante expresa su deseo de no continuar con el proceso comenzado y que hará innecesaria la

sentencia. Para ello es necesario que se haga constar mediante una declaración formulada por el demandante.

Del escrito en que se formule se dará traslado a las demás partes por término de cinco días a fin de que muestren su conformidad o no con él, y en su vista, el Tribunal resolverá lo que corresponda sobre la continuación o no del proceso.

Aunque todas las partes hayan mostrado su conformidad con el Desistimiento, el Tribunal, no obstante, previo traslado al Fiscal por tres días, podrá disponer que el proceso continúe hasta su terminación con arreglo a derecho siempre que el Desistimiento sea contrario al interés social o a los derechos de terceros protegidos por la ley. En este caso el Fiscal asumirá la representación de dichos intereses o derechos.

La aprobación del Desistimiento indica ponerle fin al proceso, y se realiza mediante un auto definitivo. Sin embargo, el Desistimiento por sí solo producirá todos sus efectos si se formula antes de la contestación a la demanda o de haber transcurrido el término para contestarla, sin que se hubiere efectuado, teniendo el demandante cinco días hábiles para subsanar los errores encontrados en la declaración formulada para la petición del Desistimiento.

1.2 Sistemas de Gestión de Tribunales

Con el avance de las tecnologías ha aumentado la creación de sistemas informáticos para agilizar y facilitar los procesos jurídicos que se llevan a cabo en distintas entidades. Entre los sistemas más destacados se encuentran IURIX creado en Argentina e Infolex, Lexnet, GEDEX, Minerva-NOJ y GIAJ-TRAMIX creados en España.

Cuba a pesar de ser un país subdesarrollado ha realizado intentos para sumarse a este desarrollo creando sistemas como SISPROP y SISECO.

A continuación se hace una descripción de los sistemas mencionados anteriormente.

IURIX (Software de Gestión Judicial)

Es un sistema privativo creado en Argentina por Unitech y utilizado para la gestión judicial de ese país. Las ventajas fundamentales que posee este sistema son: agiliza los trámites de los expedientes, permite la comunicación electrónica segura entre las oficinas judiciales y permite el diligenciamiento electrónico de notificaciones y cédulas. Está basado en un desarrollo Web Enabled orientado a la construcción del expediente electrónico y al acceso a través de redes públicas y privadas (Unitech, 2011).

Infolex (Software de gestión jurídica)

Es un sistema privativo creado en España por Jurisoft que actualmente se encuentra integrada con la multinacional Thomson Reuters. Esta empresa se encarga de la creación de herramientas y servicios para mejorar la eficiencia del trabajo diario de los operadores jurídicos. Infolex comprende varios módulos entre los que se encuentran: Minutación y Factura, Tributación, Expedientes Mercantiles y Gestión de Expedientes Administrativos (Jurisoft, 2012). La herramienta es compatible con todos los sistemas operativos Microsoft Windows, fue desarrollada con Visual Studio .NET 2008 y utiliza como sistema gestor de base de datos Microsoft SQL Server.

Lexnet (Sistema de gestión de notificaciones telemáticas)

Lexnet es una plataforma de intercambio seguro de información entre los órganos judiciales y los operadores jurídicos que, en su trabajo diario, necesitan intercambiar documentos judiciales (notificaciones, escritos y demandas) (Justica, 2013). Esta plataforma fue creada en España y se encuentra regulada por el Real Decreto 84/2007 de ese país. Su funcionamiento está basado en un sistema de correo electrónico seguro, con firma electrónica, a través del cual el usuario recibe las notificaciones emitidas por el juzgado y presenta los escritos por vía telemática. Entre las funcionalidades que posee se encuentran: el envío de notificaciones por parte del órgano judicial, emisión de acuses de recibo y la presentación de documentos y adjuntos al órgano judicial (Atrio, 2011).

GEDEX (Software jurídico)

Este software fue creado en España y es uno de los más utilizados en ese país y también en Latinoamérica. Sus inicios se remontan a 1996 y solamente es compatible con sistemas operativos Microsoft Windows (Brindys, 2013a). Esta aplicación ofrece un sistema para la gestión de expedientes jurídicos y contactos, tanto en despachos de abogados con un único ordenador como en bufetes con redes de ordenadores (Brindys, 2013b). Entre las funcionalidades de GEDEX se encuentran (Brindys, 2013a):

- Ofrece soporte para despachos que dispongan de una red local, permitiendo compartir toda la información y escritos en distintos ordenadores y delegaciones.
- Clasificación de expedientes en función de múltiples aspectos.
- Ofrece un avanzado sistema de contraseñas, con el que puede limitar el acceso a la información, ocultar expedientes y contactos.

- Se adquiere de forma gratuita.

Minerva-NOJ (Software de gestión judicial)

Minerva-NOJ es una aplicación de gestión procesal que soporta la tramitación de la información relativa a los procedimientos judiciales, de forma que cualquier órgano judicial implicado en la tramitación de un determinado procedimiento, pueda acceder a la información asociada con las garantías de reserva, control y confidencialidad requeridas (Isdefe, 2013).

El sistema permite la gestión de usuarios donde los secretarios judiciales definen el perfil de trabajo de cada persona, además de la tramitación guiada donde se le propone al usuario una vez realizado un trámite cual es el siguiente o la opción de escoger entre varios posibles. Minerva-NOJ posee un conjunto de utilidades informáticas para controlar los plazos de las actuaciones, los presos preventivos, las medidas cautelares, la Súplica, el Desistimiento, entre otras (Isdefe, 2013).

El sistema usa para los servidores base de datos Oracle Enterprise, Linux Red Hat Enterprise, Samba y plataforma Uniface. Para el correcto uso del sistema por los usuarios es necesaria la utilización de la plataforma Uniface, el sistema operativo Window XP y el navegador web Internet Explorer (Isdefe, 2013).

GIAJ-TRAMIX (Software jurídico)

El sistema GIAJ-TRAMIX está planteado con arquitectura cliente/servidor para los usuarios internos del Poder Judicial, y arquitectura Web Enabled con motor Oracle, para los usuarios externos (abogados y personas naturales). Entre las funcionalidades que posee están: la posibilidad de ingresar a los expedientes y efectuar el control de las actuaciones, notificar y también incorporar escritos de forma remota, es decir a través de Internet, todo mediante el uso de cualquier dispositivo que permita esta navegación (Judicial, 2009).

Este sistema genera importantes ventajas, que se traducen en avances relacionados con la agilización de los procesos, incremento de la tarea de oficio favorecida por la incorporación de procesos automatizados y el descongestionamiento de los circuitos administrativos, factores que permitirían proporcionar una operatoria más eficiente y eficaz, disminuyendo también los costos (Judicial, 2009).

SISPROP (Software jurídico)

SISPROP es un sistema creado en Cuba por la Universidad Central Marta Abreu de Las Villas para abarcar los procesos penales del Tribunal Supremo y los Tribunales Populares Provinciales. El software

no tuvo una buena aceptación por los clientes, presentando algunas no conformidades como la falta de validaciones en los datos suministrados por los usuarios. Además fue concebido en el lenguaje de programación Visual Pascal, con la herramienta de desarrollo Delphi usando a SQL Server para su base de datos, por lo que lo hace incompatible con el software libre. Esta aplicación, al no registrar ningún dato de la tramitación, no emite informes estadísticos y no elimina las barreras del papel, quedando por debajo de las expectativas de los clientes. Además SISPROP solamente realiza los procesos de la materia Penal

(González and González, 2008).

SISECO (Software jurídico)

Sistema utilizado en la Sala de lo Económico del Tribunal Popular Provincial de La Habana desarrollado en el 2002 para el área de la estadística. Esta aplicación fue desarrollada en Delphi 6 y con Microsoft SQL Server 2000 como servidor de base de datos. El sistema le dio solución a los problemas estadísticos pero desde el 2010 viene presentando dificultades con el aumento del tamaño de la base de datos, razón por la cual el sistema demora un poco en iniciar. Los expedientes nada más pueden ser radicados hasta el año 2010, porque el campo “año” fue llenado manualmente y no se puede corregir este error debido a que no se posee el código fuente de la aplicación. Estos inconvenientes provocan que el Tribunal Supremo realice actualmente los procesos estadísticos de forma manual (Estévez and Romero, 2012).

En la Tabla 1 se muestra una comparación entre los distintos sistemas teniendo como indicadores: la realización de los procesos Súplica y Desistimiento, si es software privativo, el uso de tecnologías libres en su creación y si cumplen con las políticas de desarrollo del proyecto SITPC.

Tabla 1 Comparación entre los software jurídicos estudiados.

Sistema	Súplica	Desistimiento	Privativo	Creado con tecnologías libres	Políticas de desarrollo SITPC
IURIX	No	No	Sí	Sí	No
Infolex	No	No	Sí	No	No
Lexnet	No	No	Sí	Sí	No
GEDEX	No	No	No	Sí	No
MINERVA-NOJ	Sí	Sí	Sí	No	No

GIAJ-TRAMIX	No	No	Sí	No	No
SISPROP	No	No	No	No	No
SISECO	No	No	No	No	No

De forma general, los sistemas estudiados anteriormente no satisfacen las necesidades que presentan los TPC, debido a que no realizan todos los procesos necesarios para la correcta gestión de la información.

De ellos, solamente el sistema Minerva-NOJ realiza los procesos de Súplica y Desistimiento. Sin embargo esta herramienta es privativa, y fue desarrollada con tecnologías propietarias que no cumplen con las políticas de desarrollo del proyecto SITPC y atentan contra el proceso de migración a software libre que se está llevando a cabo en el país.

De lo planteado anteriormente surge la necesidad de realizar el diseño e implementación de los componentes Súplica y Desistimiento.

1.3 Metodología de desarrollo de software

Es una realidad que con el avance de las tecnologías ha aumentado la producción de software, pero para ello es necesario realizar un correcto control del proceso a realizar. Un estudio realizado por (Garzás, 2010) plantea que en el año 2009 sólo un 32% de los software desarrollados lograron ser exitosos, un 44% se consideraron como deficientes o sea no poseían la calidad requerida por los clientes y el 24% se declararon como fracasados, siendo una de las principales causas de fracasos el uso incorrecto o la carencia de una metodología de desarrollo de software.

Las metodologías de desarrollo de software tienen como objetivo proporcionarle al equipo de desarrollo de software una guía para ordenar las actividades de un equipo, dirigir las tareas de cada desarrollador por separado y del equipo como un todo, especificar los artefactos que deben desarrollarse y ofrecer criterios para el control y la medición de los productos y actividades del proyecto (Jacobson *et al.*, 2000) . A continuación se describe la metodología a utilizar para el desarrollo de la solución.

Proceso Unificado de Desarrollo (RUP)

RUP posee tres aspectos que definen a la metodología (Jacobson *et al.*, 2000):

- Dirigido por casos de uso

  Un caso de uso es un fragmento de funcionalidad del sistema que proporciona al usuario un resultado importante. Los casos de uso representan los requisitos funcionales. Todos los casos de uso juntos constituyen el modelo de casos de uso, el cual describe la funcionalidad total del sistema.
- Centrado en la arquitectura

  Los arquitectos deben diseñar el sistema de forma tal que evolucione, no solo durante la etapa inicial sino también en las generaciones venideras. Para esto deben trabajar sobre los casos de uso significativos los cuales constituyen las funciones fundamentales del software.
- Iterativo e incremental

  En los sistemas grandes es práctico dividir el trabajo en partes más pequeñas o miniproyectos, donde cada miniproyecto es una iteración que posteriormente se convierte en un incremento o crecimiento del producto.

RUP define nueve disciplinas (flujos) a realizar en cada fase del proyecto: Modelado del negocio, análisis de requisitos, análisis y diseño, implementación, pruebas, distribución (despliegue), gestión de configuración y cambios, gestión del proyecto y gestión del entorno.

RUP se divide en cuatro fases (Jacobson *et al.*, 2000):

- Inicio
- Elaboración
- Construcción
- Transición

En la Figura 1 se muestra la relación entre las fases y las disciplinas mencionadas anteriormente.

RUP hace uso de buenas prácticas de la industria del software (de Sousa´s, 2009):

- Desarrollo iterativo
- Administración de requisitos
- Uso de componentes
- Modelo visual
- Verificación de calidad

- Control de cambios

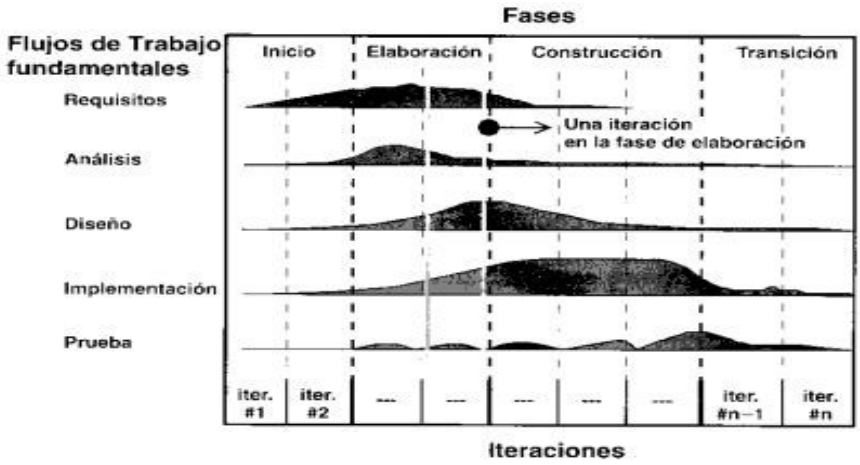

Figura 1 Fases y disciplinas de RUP (Pressman, 2010)

La selección de una determinada metodología en un proyecto se encuentra en dependencia de la organización y necesidades del equipo de proyecto, la comunicación existente entre los involucrados, el plazo de entrega del producto, las restricciones y riesgos.

En el SITPC por las características que presenta se ha seleccionado RUP para guiar el proceso de desarrollo, tomándose la decisión basada en varios aspectos, principalmente por ser una metodología robusta que se adapta a las características y complejidad del sistema, genera gran cantidad de documentación, lo que se ajusta perfectamente al proyecto, el cual requiere una especificación en detalle de todo el trabajo realizado anteriormente, por su larga duración y el constante cambio del personal de desarrollo, centra el proceso de desarrollo en la arquitectura definiendo para ella actividades y artefactos, provee calidad, reutilización de los componentes, seguridad y mantenimiento del software mediante una gestión sistemática de los riesgos.

1.4 Lenguaje de modelado UML

El Proceso Unificado utiliza el Lenguaje Unificado de Modelado (UML) para preparar todos los esquemas de un sistema de software. De hecho, UML es una parte esencial del Proceso Unificado siendo desarrollados de forma paralela (Jacobson *et al.*, 2000).

UML es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Se usa para entender, diseñar, hojear, configurar, mantener, y controlar la información sobre estos sistemas (Rumbaugh *et al.*, 2000).

Algunas de las propiedades de UML como lenguaje de modelado estándar son (Rumbaugh *et al.*, 2000):

- Concurrencia, es un lenguaje distribuido y adecuado a las necesidades de conectividad actuales y futuras.
- Ampliamente utilizado por la industria desde su adopción por el Grupo de Gestión de Objetos.
- Modela estructuras complejas.
- Las estructuras más importantes que soportan tienen su fundamento en las tecnologías orientadas a objetos.
- Modela el comportamiento del sistema.

1.5 Herramienta CASE

Se puede definir a las herramientas CASE (Ingeniería de Software Asistida por Computadora) como un conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del ciclo de vida de desarrollo de un software (INEI, 1999).

La realización de un nuevo software requiere que las tareas sean organizadas y completadas en forma correcta y eficiente. Las herramientas CASE fueron desarrolladas para automatizar esos procesos y facilitar las tareas de coordinación de los eventos que necesitan ser mejorados en el ciclo de desarrollo de software (INEI, 1999).

A continuación se describen las características de la herramienta CASE a utilizar.

Visual Paradigm para UML v 8.0

Visual Paradigm para UML es una herramienta de diseño para UML creada para auxiliar el proceso de desarrollo de software. Soporta los principales estándares de modelado como UML, SoaML, ERD y BPMN. Esta herramienta ayuda al equipo de desarrollo de software en las fases de captura de requisitos, planificación, el modelado de clases y el modelado de datos. Entre las principales características que presenta esta herramienta se encuentran (Paradigm, 2013):

- Es compatible con plataformas Windows y Linux.
- Permite la creación de los diagramas necesarios durante el proceso de desarrollo de software.

- Permite la generación de diagramas en formato PDF, HTML y Microsoft Word de forma completamente configurable.
- Soporta las tres fases del modelado de datos con los Diagramas Entidad-Relación, además de permitir la generación de la base de datos a partir de estos diagramas.
- Permite la generación de código fuente a partir de los diagramas UML en distintos lenguajes de programación como Java, PHP, C++, Python entre otros.

1.6 Paradigma de programación

Un paradigma de programación provee (y determina) la visión y métodos de un programador en la construcción de un programa o subprograma. Diferentes paradigmas resultan en diferentes estilos de programación y en diferentes formas de pensar la solución de problemas (con la solución de múltiples "problemas" se construye una aplicación) (Alegsa, 2006).

Al estudio de los lenguajes en cuanto al enfoque del proceso de programación se le denomina paradigmas de la programación, entendiéndose el término paradigma como la forma de ver y hacer los programas. Bajo este enfoque se tienen cuatro paradigmas los cuales son (Gabbrielli and Martini, 2006):

- paradigma por procedimientos o paradigma imperativo
- paradigma declarativo
- paradigma funcional
- paradigma orientado a objetos

Programación Orientada a Objetos

Los inicios de este paradigma tienen sus raíces con el sistema SIMULA en 1967 pero no fue completamente desarrollado hasta 1970 con Smalltalk. Smalltalk es considerado como el modelo base para un lenguaje de programación orientado a objeto (Sebasta, 2012).

Los conceptos claves de un lenguaje orientado a objetos son:

- Objeto: Un objeto es un conjunto de operaciones sobre algún dato oculto. Es una forma de encapsular cualquier combinación de datos y funcionalidades. Un objeto puede ser tan pequeño como un integer o tan grande como una base de datos. Todas las interacciones que se realizan

sobre un objeto se producen por medio de mensajes o llamadas a las funciones miembros (métodos) (C. Mitchell, 2004).

- Clases: Constituye un modelo para un conjunto de objetos. Establece cuales son los datos que poseerán un determinado objeto, la visibilidad que poseerán y la implementación de cada uno de los métodos de la clase (Gabbrielli and Martini, 2006).
- Herencia: Es un mecanismo creado en la programación orientada a objetos que ayuda a aumentar la productividad del programador. La herencia está basada en que una subclase comparte todos los métodos de la superclase a menos que la subclase los redefina (Watt, 2004).
- Polimorfismo: Esta característica de los lenguajes orientados a objetos le permite a una subclase ser tratada como un objeto de la clase que hereda, de esta forma se puede tener una colección de objetos diferentes que serán tratados como una superclase común (Watt, 2004).

1.7 HTML v 5

El lenguaje de marcado de hipertexto (HTML por sus siglas en inglés), es un lenguaje semántico para el marcado de texto como su nombre indica. El marcado provee una descripción del contenido que los navegadores web usan para construir la página web correspondiente. HTML5 permite la utilización de enlaces que permite tener referencias en un documento hacia otros sin importar donde estén localizados físicamente haciendo uso del Localizador Uniforme de Recursos (URL). La Web es huésped de muchos protocolos pero HTML es la base, proporcionando el lenguaje básico para el mercado de texto contenido dentro de un documento estructurado que describe los roles y atributos de los elementos que la componen (Aronson, 2011).

1.8 Motor de plantillas Twig v 1.9

Twig es un motor de plantillas que brinda un ambiente amigable para el diseñador y desarrollador apegado a los principios de PHP, añadiendo útil funcionalidad a los entornos de plantillas. Como características claves se puede mencionar que es rápido, compila las plantillas hasta código PHP optimizado, posee un modo de recinto de seguridad para evaluar el código de plantilla que no es confiable, permitiendo utilizar Twig como un lenguaje de plantillas para aplicaciones donde los usuarios pueden modificar el diseño de la plantilla; es soportado por flexibles analizadores léxico y sintáctico, brindándole al desarrollador la opción de definir sus propias etiquetas, filtros personalizados, y crear su propio Lenguaje de Dominio Específico (Potencier, 2011).

1.9 Lenguaje de programación

Un lenguaje de programación es un mecanismo de abstracción que le permite a un programador especificar una operación abstracta y dejar que el programa (usualmente llamado ensamblador, compilador o intérprete) implemente la especificación de una forma detallada para la ejecución de esa operación en la computadora o sea que la computadora entienda lo que tiene que hacer para realizar esa operación (Ben-Ari, 1996).

JavaScript v 1.8

JavaScript es un lenguaje de programación que se ejecuta dentro de un navegador web y se encarga de manipular los elementos de las páginas HTML en respuesta de las acciones del usuario y otros eventos. Existen otros lenguajes de script como ASP, Phyton y .NET pero la sintaxis y términos de JavaScript son usados en las especificaciones de HTML5. El código en este lenguaje de programación puede estar embebido dentro del la página web o puede ser importada desde un archivo separado (Aronson, 2011).

Los últimos avances en JavaScript y la proliferación de tecnologías como DHTML, Java y Macromedia Flash, han colocado a JavaScript en una posición ventajosa para aprovechar estas herramientas en la creación de grandes soluciones para la Web. Entre las principales ventajas que posee este lenguaje de programación se encuentran (Brandendaugh, 2000):

- Fácil de aprender y rápido: Como este lenguaje es muy sencillo de aprender se puede empezar a trabajar con él desde el principio. Es ideal para agregar ciertas funciones rápidas a una página web. No se necesita una fase de compilación y para su uso no es necesario cargar ninguna máquina virtual para ejecutar el código.
- Usabilidad: JavaScript es el lenguaje de programación que más se utiliza en la web. Hay publicadas millones de páginas web que incorporan elementos que lo usan. La mayoría de los navegadores web pueden trabajar con él.
- Reducción de la carga del servidor: Este lenguaje se puede hacer cargo de una gran parte de las funciones del cliente de las cuales se encargaba el servidor, siendo un ejemplo de esto las validaciones.

jQuery v 1.8

jQuery es una biblioteca para JavaScript que ayuda al desarrollador a trabajar con los documentos HTML, manipular los objetos del DOM (Document Object Model) y a interactuar con los eventos de una página web.

Entre las principales características de esta biblioteca se encuentran (Álvarez, 2012):

- Implementa una serie de clases que permiten que el programador no tenga que preocuparse por el navegador que está usando el usuario.
- Ofrece una infraestructura que brinda mayor facilidad para la creación de aplicaciones complejas proporcionándole al programador ayuda en la creación de interfaces de usuario, los efectos dinámicos y las peticiones AJAX.
- Posee una licencia gratuita para su utilización en cualquier plataforma personal o comercial.
- Es pequeño, ocupando unos 56 KB y no retrasa la carga de la página web debido a que el servidor le enviará los archivos del framework al cliente la primera vez y luego los tomará de la caché.

Lenguaje de programación PHP v 5.3

En (Doyle, 2011) se define a PHP (Hypertext Preprocessor) como un lenguaje de programación utilizado para la construcción de sitios web dinámicos e interactivos. Entre las principales características que presenta este lenguaje se encuentran:

- El código PHP puede ser embebido dentro del HTML de las páginas web, permitiendo la creación de contenido dinámico.
- Permite la lectura y procesamiento de formularios enviados por el usuario.
- Manejo de los datos almacenados en una base de datos.
- Es un lenguaje de programación multiplataforma.
- Se integra con los servidores web más comunes como Apache e Internet Information Server (IIS).
- No es necesario una licencia comercial para su uso.

1.10 Marco de trabajo

Un marco de trabajo (framework) simplifica el desarrollo de una aplicación mediante la automatización de algunos de los patrones utilizados para resolver las tareas comunes. Además, un framework proporciona estructura al código fuente, ayudando al desarrollador a crear código más legible y más fácil de mantener.

También facilita la programación de aplicaciones encapsulando las operaciones complejas en instrucciones sencillas (Potencier and Zaninotto, 2008).

Symfony v 2.1

Symfony es un framework diseñado para optimizar el desarrollo de aplicaciones web. Separa la lógica del negocio, la lógica del servidor y la presentación de la aplicación web. Además proporciona varias herramientas y clases encaminadas a reducir el tiempo de una aplicación web compleja además de automatizar las tareas más comunes, permitiéndole al desarrollador centrarse por completo en los aspectos específicos de cada aplicación. Entre las principales características de este marco de trabajo se encuentran:

- Es independiente del sistema gestor de base de datos.
- Sigue la mayoría de las buenas prácticas y patrones de diseño para la web.
- Está preparado para aplicaciones empresariales y es adaptable a las arquitecturas y políticas propias de cada empresa.
- Es sencillo de usar en la mayoría de los casos, pero es lo suficientemente flexible como para adaptarse a los casos más complejos. (Potencier and Zaninotto, 2008)

Bootstrap v 2.1

Bootstrap es un framework para CSS desarrollado por Mark Otto y Jacob Thornton. Este proyecto surge como una necesidad de estandarizar la interfaz realizada por los distintos ingenieros de Twitter. Desde su lanzamiento este marco de trabajo ha evolucionado de ser un proyecto que controla todo el CSS relacionado a la vista de una aplicación web a incluir una serie de bibliotecas JavaScript e iconos unidos a una serie de botones y formularios. Bootstrap es completamente adaptable a las necesidades específicas de los proyectos permitiendo que los desarrolladores elijan cuales características de CSS o JavaScript desean incluir en su aplicación (Spurlock, 2013).

1.11 Sistema gestor de bases de datos

Un Sistema gestor de bases de datos (SGBD) consiste en una colección de datos interrelacionados y un conjunto de programas para acceder a dichos datos. El objetivo principal de un SGBD es proporcionar una forma de almacenar y recuperar la información de una base de datos de manera que sea tanto práctica como eficiente. Los sistemas de bases de datos se diseñan para gestionar grandes cantidades de

información. La gestión de los datos implica tanto la definición de estructuras para almacenar la información como la provisión de mecanismos para la manipulación de la información (Sillberchatz *et al.*, 2002).

PostgreSQL v 9.2

PostgreSQL es un sistema de gestión de bases de datos objeto-relacional, distribuido bajo licencia BSD y con su código fuente disponible libremente. Es el sistema de gestión de bases de datos de código abierto más utilizado actualmente (PostgreSQL, 2014). Entre las principales características que posee este SGBD se encuentran:

- Multiplataforma.
- Permite la creación de funciones/procedimientos almacenados en numerosos lenguajes de programación.
- Numerosos tipos de datos y posibilidad de definir nuevos tipos.
- Soporta el almacenamiento de objetos binarios grandes.
- Permite el acceso encriptado vía SSL.
- Creación de disparadores (triggers) comunes, por columna o condiciones.

1.12 Herramienta de mapeo objeto relacional

Las herramientas de mapeo objeto relacional (ORM, por sus siglas en inglés) constituyen una técnica de programación utilizada para convertir datos entre el lenguaje de programación orientado a objetos utilizado y el sistema de base de datos utilizado en la aplicación. Actualmente las base de datos relacionales solo pueden almacenar datos primitivos y no los objetos que se vayan creando en la aplicación, para esto se crean los ORM, evitándole al desarrollador convertir esos datos primitivos en objetos (Carrero, 2013)

Doctrine v 2.0.

Doctrine 2.0 es un ORM para PHP 5.3 que provee una persistencia transparente para los objetos PHP. Una de sus principales características es la opción de escribir las consultas a la base de datos en un lenguaje orientado al SQL llamado Doctrine Query Language (DQL) permitiéndole al desarrollador la creación de consultas en un simple y flexible entorno debido a que el lenguaje lo abstrae del mapeo entre las columnas de la base de datos y los objetos (Doctrine-Projects, 2013).

1.13 Servidor web

Un servidor web es un programa que atiende y responde a las diversas peticiones de los navegadores, proporcionándoles los recursos que solicitan mediante el protocolo HTTP o el protocolo HTTPS (la versión segura, cifrada y autenticada de HTTP). Un servidor web básico tiene un esquema de funcionamiento muy sencillo, ejecutando de forma infinita el bucle siguiente (Mateu, 2004):

1- Espera peticiones en el puerto TCP.
2- Recibe una petición.
3- Busca el recurso en la cadena de petición.
4- Envía el recurso por la conexión por donde ha recibido la petición.
5- Vuelve al punto 2.

Apache v 2.0

Apache es un servidor web HTTP de código abierto y multiplataforma que es altamente configurable debido a su diseño modular, permitiendo de esta forma que las personas con conocimientos en Pearl o C puedan crear e incluir módulos existentes al servidor. Entre las principales características de este servidor web se encuentran (Kabir, 2003):

- Soporte al último protocolo HTTP 1.1: Es totalmente compatible con el nuevo estándar HTTP 1.1 y al mismo tiempo sigue siéndolo con el HTTP 1.0.
- Sencillo, con la configuración en un solo archivo: el servidor Apache no presenta una interfaz visual para su configuración, sino se trata de un solo archivo llamado httpd.conf.
- Soporte de host virtuales: Es uno de los primeros servidores en soportar tanto servidores basados en IP como en host virtuales, permitiendo la creación de más de un servidor en una máquina.
- Soporte de script PHP: Ofrece un amplio soporte de PHP utilizando el módulo mod_php.

1.14 Entorno de desarrollo integrado

Un Entorno de desarrollo integrado (IDE por sus siglas en inglés) es un entorno de programación que ha sido empaquetado como un programa de aplicación para mejorar el desarrollo de aplicaciones. Ofrecen un marco de trabajo para la mayoría de los lenguajes de programación. Los principales componentes que posee un IDE son (Maldonado, 2007):

- Editor de texto

- Compilador
- Intérprete
- Posibilidad de ofrecer un sistema de control de versiones
- Factibilidad para ayudar en la construcción de interfaces gráficas de usuario

Netbeans IDE v 7.2

Netbeans es un IDE de desarrollo creado en Java que permite el desarrollo de aplicaciones de escritorio en Java, aplicaciones Web, HTML5, JavaScript y CSS. Brinda una serie de herramientas para los desarrolladores de C, C++ y PHP. Netbeans es una herramienta libre, de código abierto, multiplataforma y posee una gran comunidad de desarrolladores y usuarios en el mundo. Entre las principales características que posee este IDE se encuentran (NetBeans-Project, 2013):

- El mejor soporte para las últimas tecnologías Java.
- La edición rápida e inteligente del código fuente.
- Administración fácil y eficiente de los proyectos.
- Desarrollo rápido de interfaces de usuario.
- Posee una gran comunidad que provee plugins para el IDE.

1.15 Patrones de diseño

Un patrón de diseño es una buena práctica documentada de una solución que ha sido aplicada exitosamente en múltiples ambientes para resolver problemas que ocurren en un conjunto de situaciones. Puede ser visto como una encapsulación de solución reusable para darle solución a un problema común en el diseño y como una expresión documentada de resolver un problema que es observado o descubierto durante el estudio o construcción de un software (Kuchana, 2004).

El principal objetivo de un patrón de diseño es ayudar a mejorar la calidad de un software en términos de que sea reusable, de fácil mantenimiento y extensible, además de reducir el tiempo de desarrollo. Son usualmente independientes del lenguaje de programación a utilizar. Los patrones de diseño son utilizados generalmente durante el proceso de creación de un framework o sea estos internamente encierran muchos patrones de diseño (Kuchana, 2004).

Patrones GRASP

Los patrones GRASP son una ayuda en el aprendizaje que permiten al desarrollador entender lo esencial del diseño de objetos y a aplicar el razonamiento de una forma metódica, racional y explicable. Este enfoque se entiende en el uso de los principios del diseño basado en patrones para la asignación de responsabilidades a las clases y objetos de una aplicación (Noorullah, 2011). Los principales patrones GRASP se muestran a continuación:

Creador

El patrón creador guía la asignación de responsabilidades relacionadas con la creación de objetos. El principal propósito de este patrón es encontrar un creador que se debe conectar con el objeto producido en cualquier evento. Brinda soporte a un bajo acoplamiento, suponiendo menos dependencias respecto al mantenimiento y mejores oportunidades de reutilización (Craig, 1999).

Experto

Es un patrón que se usa más que cualquier otro al asignar responsabilidades, es un principio básico que suele utilizarse en la programación orientada a objetos. Este patrón favorece al encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide. También soporta un bajo acoplamiento, favoreciendo la creación de sistemas más robustos y de fácil mantenimiento (Craig, 1999).

Controlador

La mayor parte de los sistemas reciben eventos de entrada externa, los cuales generalmente incluyen una interfaz gráfica para el usuario (GUI) operado por una persona. Debido a esto si se recurre a un diseño orientado a objetos, hay que elegir los controladores que manejen esos eventos de entrada. 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 (Craig, 1999).

Bajo acoplamiento

El acoplamiento es una medida de la fuerza con que una clase está conectada a otras clases, con que las conoce y con que recurre a ellas. Una clase con bajo acoplamiento no depende de muchas otras. El bajo acoplamiento soporta el diseño de clases más independientes, que reducen el impacto de los cambios y

también más reutilizables para de esta forma acrecentar la oportunidad de una mayor productividad (Craig, 1999).

Alta cohesión

La cohesión es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme, poseyendo estas clases responsabilidades moderadas en un área funcional y colabora con otras para llevar a cabo las tareas (Craig, 1999).

Patrones GoF

En (Gamma *et al.*, 1997) se propone un total de 23 patrones que pueden ser utilizados para el correcto diseño de un software, estos patrones son reconocidos como los patrones GoF (Gang of Four). A continuación se presentan las formas de clasificar estos patrones según su propósito:

- Creacionales: Conciernen al procesos de creación de objetos.
- Estructurales: Tratan con la composición de las clases o los objetos.
- Comportamiento: Caracterizan la vía en que las clases u objetos interactúan y distribuyen las responsabilidades.

Patrón Decorador

El patrón Decorador proporciona una manera más flexible de que una subclase herede funcionalidad, permitiendo de esta forma que se le pueda agregar responsabilidades a una clase dinámicamente. Con su uso las responsabilidades pueden ser añadidas o eliminadas en tiempo de ejecución (Gamma *et al.*, 1997).

Inyección de dependencia

La inyección de dependencia es un patrón que permite el desarrollo de código con bajo acoplamiento, permitiendo de esta forma que los servicios de un sistema puedan ser cambiados por otros, el código pueda ser reutilizado, extendido e implementado en paralelo, facilita el mantenimiento del sistema y permite que las clases puedan ser probadas unitariamente (Seeman, 2012).

1.16 Arquitectura de software

La arquitectura de software de un programa o sistema computacional es la estructura o estructuras del sistema, esta comprende los componentes del software, las propiedades de visibilidad externa de esos componentes y la relación existente entre ellos (Bass *et al.*, 2013).

La arquitectura describe cuales son los componentes de alto nivel que posee la aplicación, así como las responsabilidades que estos componentes poseen con respecto a otros. También describe como estos componentes están organizados desde un nivel conceptual o una descomposición detallada. Esta define cuáles son las interfaces que los componentes presentan a otros y cuáles son las interfaces y/o componentes que usa (Martensson, 2006).

La arquitectura no es el software operacional, sino una representación que permite (Pressman, 2010):

- Analizar la efectividad del diseño
- Considerar alternativas arquitectónicas en un escenario donde hacer cambios en el diseño es relativamente fácil.

Patrones arquitectónicos

Un patrón arquitectónico expresa un esquema fundamental para la organización estructural de los sistemas de software. Proporciona un conjunto de subsistemas predefinidos, especifica sus responsabilidades e incluye reglas y directrices para organizar la relación entre ellos. Estos patrones son plantillas para arquitecturas de software concretas. Ellos especifican las propiedades estructurales de todo el sistema de una aplicación y tienen un impacto en la arquitectura del subsistema (Buschmann *et al.*, 1996).

Arquitectura en capas

La arquitectura basada en capas es una de las técnicas más comunes utilizadas por los diseñadores de software para separar un sistema de software complejo. En esta arquitectura las capas más altas pueden utilizar servicios de la capa más baja pero esta no puede acceder a los servicios de la capa superior.

El uso de esta arquitectura brinda una serie de beneficios, algunos de los cuales son mencionados a continuación (Fowler, 2002):

- Una capa puede ser como un todo coherente sin tener mucho conocimiento de las otras.

- Las capas pueden ser sustituidas con implementaciones alternativas del mismo servicio.
- Se minimiza la dependencia entre las capas.
- Una vez construida una capa, puede ser utilizada para muchos servicios de un nivel más alto.

Patrón Modelo-Vista-Controlador (MVC)

El MVC es uno de los patrones más utilizados en la actualidad. Este define tres roles:

- Modelo: Es un objeto que representa cierta información del dominio. Constituye un objeto no visual y contiene todos los datos y comportamiento del mismo.
- Vista: La vista representa la muestra del modelo mediante la interfaz visual.
- Controlador: Se encarga de cambiar cualquier información existente en el modelo. Toma las entradas del usuario, manipula el modelo y actualiza la vista apropiadamente.

Uno de los aspectos más importantes que proporciona la división en estos roles es la separación de dependencias debido a que como la vista depende del modelo, no ocurriendo de forma contraria, permite que los desarrolladores que estén trabajando en el modelo desconozcan que presentación será utilizada. Esto también brinda la oportunidad de que se puedan realizar cambios en la vista sin afectar el modelo y simplifica el trabajo de los que estén trabajando en ambos roles (Fowler, 2002).

Estilo Arquitectónico Llamada y Retorno

Esta familia de estilos enfatiza la modificabilidad y la escalabilidad. Son los estilos más generalizados en sistemas en gran escala. Miembros de la familia son las arquitecturas de programa principal y subrutina, los sistemas basados en llamadas a procedimientos remotos, los sistemas orientados a objeto y los sistemas jerárquicos en capas (Reynoso and Kicillof, 2004).

Conclusiones parciales del capítulo

Luego del análisis realizado se llegaron a las siguientes conclusiones:

- Se analizaron los procesos Súplica y Desistimiento, lo cual permitió una mayor comprensión del contexto donde se desarrolla la investigación.
- De los sistemas jurídicos estudiados solo Minerva-NOJ implementa los procesos de Súplica y Desistimiento. Sin embargo esta herramienta es privativa, y fue desarrollada con tecnologías

propietarias que no cumplen con las políticas de desarrollo del proyecto SITPC, atentando contra el proceso de migración a software libre que se está llevando a cabo en el país.

- Se describieron las tecnologías que van a ser utilizadas en la investigación, sentándose las bases para el desarrollo de los componentes.

CAPÍTULO 2: DESCRIPCIÓN DE LA SOLUCIÓN PROPUESTA

Introducción

En el presente capítulo se detallan los aspectos referentes al sistema a desarrollar, partiendo de los requisitos funcionales y no funcionales. También se describen los principales aspectos referentes al diseño del sistema, la arquitectura, patrones arquitectónicos, patrones de diseño y las métricas a utilizar para su validación.

2.1 Requisitos funcionales de los procesos Súplica y Desistimiento

Antes de comenzar el diseño e implementación de los componentes Súplica y Desistimiento es necesario conocer los requisitos funcionales de cada uno de los procesos.

Del proceso de Súplica se obtuvieron 8 funcionalidades que se resumen en los 8 casos de usos que se describen a continuación:

Tabla 2 Casos de uso del proceso Súplica. Fuente: (Vargas and Fustiel, 2013).

Caso de uso	Descripción	Complejidad
Registrar recurso de Súplica.	El sistema permitirá registrar un recurso de Súplica.	Baja
Disponer sobre recurso de Súplica.	El sistema permitirá disponer sobre un recurso de Súplica.	Media
Registrar escrito contestando la Súplica.	El sistema permitirá registrar el escrito contestando la Súplica.	Baja
Resolver recurso de Súplica.	El sistema permitirá resolver el recurso de Súplica.	Alta
Registrar escrito de protesta.	El sistema permitirá registrar el escrito de protesta.	Baja
Generar providencia teniendo por presentada la protesta.	El sistema permitirá generar la providencia una vez que esté presentada la protesta.	Media

Generar providencia teniendo por presentada la contestación a la Súplica.	El sistema permitirá generar la providencia una vez que esté presentado el escrito contestando a la Súplica.	Media
Disponer sobre vencimiento para subsanar el escrito de Súplica.	El sistema permitirá disponer sobre el escrito de Súplica cuando se haya vencido el término para subsanarlo	Media

Del proceso de Desistimiento se obtuvieron 9 funcionalidades, las cuales se resumen en los 9 casos de usos que se listan a continuación:

Tabla 3 Casos de uso del proceso Desistimiento. Fuente: (Vargas and Fustiel, 2013).

Caso de uso	**Descripción**	**Complejidad**
Registrar escrito de contestación subsanado.	El sistema permitirá registrar el escrito de contestación subsanado.	Baja
Registrar escrito de contestación.	El sistema permitirá registrar el escrito de contestación.	Baja
Crear oficio de personería.	El sistema permitirá crear el oficio de personería.	Baja
Registrar escrito de Desistimiento.	El sistema permitirá registrar el escrito de Desistimiento.	Baja
Registrar escrito de Desistimiento subsanado.	El sistema permitirá registrar el escrito de Desistimiento subsanado.	Baja
Disponer sobre escrito de contestación.	El sistema permitirá disponer sobre el escrito de contestación.	Alta
Disponer sobre vencimiento del término para subsanar.	El sistema permitirá disponer sobre el vencimiento del término para subsanar.	Media
Disponer sobre el escrito de Desistimiento.	El sistema permitirá disponer sobre el escrito de Desistimiento.	Alta
Disponer sobre el Desistimiento.	El sistema permitirá disponer sobre el	Media

	Desistimiento.	

2.2 Requisitos no funcionales del sistema

Los requisitos no funcionales fueron definidos por el equipo de arquitectura del proyecto SITPC para todo el sistema (Roque and Brito, 2013). A continuación se describen los principales.

Usabilidad:

- El sistema debe poseer facilidad de aprendizaje.
- Se deben visualizar todos los mensajes en idioma español. La tipografía debe ser uniforme, de un tamaño adecuado. Aumentar y/o disminuir el tamaño de los caracteres en los textos. Debe tener un contraste adecuado que resalte los textos.
- Debe ofrecer una interfaz amigable, fácil de operar. Igualmente tiene que mantener la línea de diseño establecida la cual mantiene la uniformidad y representatividad de la solución.

Confiabilidad:

- El sistema contará con una solución que será capaz de registrar trazas de error, de rendimiento y de acceso, con el objetivo de poder auditar el recorrido histórico de los recursos y los usuarios del sistema.
- Debe estar disponible durante el horario laboral, efectuándose en períodos de tiempos definidos el proceso de actualización de la información del servidor local de base de datos con el servidor de la instancia superior.
- Se debe configurar un sistema local de recuperación ante fallos del servidor que garantice una copia de la aplicación así como del estado diario de la base de datos.
- El sistema tendrá un tiempo de inactividad de 10 minutos y una vez que quede inactivo el usuario deberá autenticarse nuevamente.

Seguridad:

- Se establecerán mecanismos de autenticación personalizada para todos los usuarios que harán uso de la aplicación mediante dirección o rango de direcciones IP autorizadas a hacer uso del sistema.

- Se definirá una jerarquía de usuarios en correspondencia con las acciones que puedan realizar, estos podrán ser activados y desactivados por el administrador a través del sistema.
- Las políticas de acceso serán definidas por el administrador del sistema, los permisos serán limitados basados en los roles definidos y el usuario no podrá gestionarlos. Dichos permisos se harán corresponder con los niveles de acceso a la información que se define como clasificada.

2.3 Arquitectura de software

En la solución propuesta se aplican dos patrones arquitectónicos fundamentales: Modelo-Vista-Controlador y basado en capas. Ambos se encuentran dentro de la familia del estilo arquitectónico Llamada y Retorno. Esta es la arquitectura que utiliza el sistema con el que se tienen que integrar los componentes a desarrollar. La arquitectura propuesta es sustentada principalmente por el marco de trabajo Symfony 2 que brinda una estructura de paquetes que define correctamente cuales son las clases que forman parte del modelo, la vista y el controlador.

A continuación en la Figura 2 se muestra una imagen que describe la arquitectura del sistema.

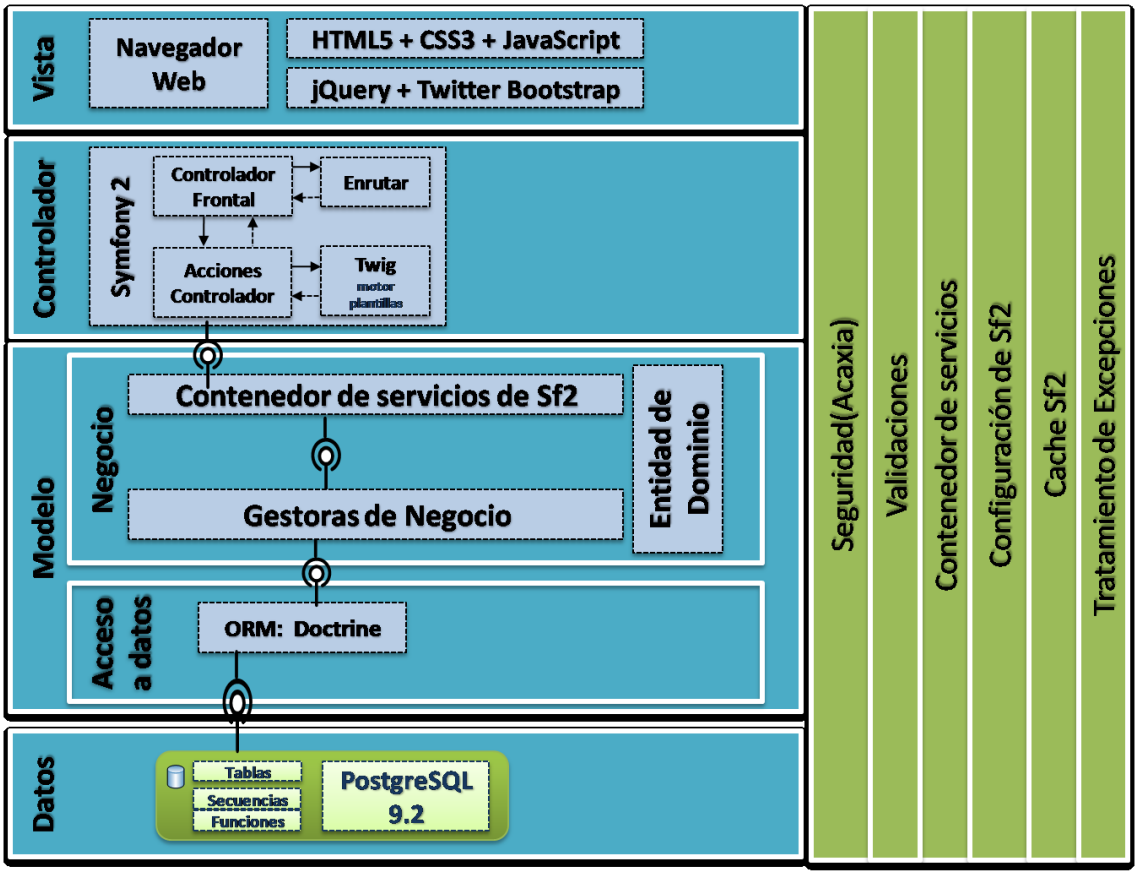

Figura 2 Arquitectura del sistema. Fuente: (Domínguez, 2014)

Vista

La vista se encarga de la interacción del usuario con el sistema. En esta capa se encuentran todas las vistas e interfaces del sistema las cuales deben ser entendibles y amigables, usando para esto imágenes, estilos CSS y funciones JavaScript. Estas interfaces son generadas mediante el motor de plantillas Twig, definiéndose para esto una plantilla base que se encuentra en la dirección "SIT/app/Resources/views" y otro conjunto de plantillas que se encuentran en "SIT/src/SIT/SRV/ComunesBundle/Resources/views", ambas se encargan de transformar la información del modelo para mostrárselo al usuario.

Controlador

Esta capa se encarga de responder a las peticiones realizadas por el usuario para mostrar y cambiar datos en el modelo. Symfony 2 propone en su estructura de paquetes una dirección donde se deben

ubicar todas las clases controladoras. En el sistema todas las controladoras se encuentran en la siguiente dirección: “SIT/src/SIT/SRV/ComunesBundle/Controller”. En esta capa también se encuentra el motor de plantillas Twig, encargándose de generar las vistas en código HTML para que pueda ser interpretada por el navegador.

Modelo

El modelo representa un objeto no visual, almacenando la información y comportamiento del mismo. En el sistema esta capa se divide en dos, la de negocio y la de acceso a datos.

En la capa de negocio se encuentra el contenedor de servicios de Symfony mediando entre las clases controladoras y las clases gestoras. También se encuentra en esta capa las clases gestoras que están ubicadas en la siguiente dirección “SIT/src/SIT/SRV/ComunesBundle/Negocio/Gestor”. Además de ubicarse en el negocio las clases entidades del dominio que representan las tablas de la base de datos ya mapeadas por Doctrine, estas entidades se pueden encontrar en “SIT\vendor\Base\ComunBundle\Entity”.

La capa de acceso a datos está comprendida por el ORM Doctrine el cual se encarga de acceder a los datos almacenados, en ella se encuentran las clases Repository donde se definen las consultas para acceder a los datos del modelo, dichas clases se encuentran en “SIT/vendor/Base/ComunBundle/Repository”.

Capa de datos

La capa de datos es donde se almacena la información del negocio, esta se compone por las tablas, procedimientos, esquemas y funciones que se encuentran en el servidor de base de datos.

2.4 Patrones de Diseño

A continuación se exponen los patrones de diseño utilizados en la solución, además de una descripción donde se evidencia su uso.

Experto

En la solución propuesta este patrón se evidencia en las clases entidades, por ejemplo, la clase *PersonaNatural* posee la información del número de identificación, el primer y segundo nombre y el

primer y segundo apellido, siendo la experta en calcular la edad, fecha de nacimiento y el nombre completo de una persona.

Creador

En la solución propuesta este patrón fue utilizado en las clases gestoras para la creación de las entidades que van a ser almacenadas en base de datos y en las clases controladoras para la creación de los formularios.

Controlador

En el sistema este patrón es utilizado en las clases controladoras. Las clases controladoras se encargan de recibir los datos introducidos por el usuario en el sistema y enviarlos a las distintas clases en dependencia de lo que se desee realizar. Esto permite lograr una separación entre la capa de presentación y la capa de negocio, aumentando la reutilización de código y brindando un mayor control sobre los cambios.

Bajo acoplamiento

En el sistema las entidades son las clases más reutilizadas y en la arquitectura se puede apreciar que la vista se comunica con la controladora, ésta a su vez con las clases gestoras que se comunican con las entidades. De esta forma se puede apreciar que las entidades no están relacionadas con las vistas ni los controladores, siendo necesario cuando se vaya a realizar algún cambio en las entidades modificar solamente las clases gestoras.

Alta cohesión

En la solución propuesta se utiliza el patrón experto para asignarle las responsabilidades que les corresponde a cada clase estableciéndose condiciones para que cada clase colabore con las restantes y darle solución a las tareas que implican a todas y que no sean capaces de resolver por sí solas.

Decorador

En la solución propuesta este patrón es utilizado en el trabajo con plantillas, las cuales son manejadas utilizando el motor de plantillas Twig e implementando una herencia entre las plantillas de la

aplicación. De esta forma se crea una plantilla base que contiene todos los elementos comunes del sistema y definir los bloques que cada una de las plantillas descendientes pueden modificar.

La plantilla base del sistema se encuentra en “SIT/app/Resources/views” la cual define un HTML básico que puede ser utilizado para una página de dos columnas (menú lateral izquierdo y contenido), además de poseer otros bloques que son comunes para todos los subsistemas y módulos: el banner, el menú superior y el pie de página. Las plantillas pertenecientes a cada subsistema descienden de esta plantilla base y se encargan de redefinir el menú lateral izquierdo y las plantillas creadas en cada caso de uso heredan de la plantilla del subsistema al que pertenecen y redefinen el bloque de contenido.

Inyección de dependencia

Symfony2 posee un generador de *bundles* que crea el fichero *service.yml*. En este fichero se detallan todos los servicios de ese *bundle* y sus dependencias. El fichero se compone por dos secciones, el *parameters* donde se declaran los parámetros de configuración de cada servicio y el *service* donde se declara cada servicio.

Este marco de trabajo también proporciona la clase *Controller* que se encuentra en “Symfony\Bundle\FrameworkBundle\Controller” la cual posee el atributo *container* que es una instancia del contenedor de dependencias. De esta forma desde cualquier clase descendiente de esta controladora se puede instanciar cualquier servicio existente en la aplicación a través del método get() del *container*, en la Figura 3 se observa un ejemplo.

```
public function getGestor() {
    if (!$this->container->has('comunes.disponerSobreEscritoDesistimientoGtr')) {
        throw new \LogicException('Este servicio no esta registrado en la aplicación');
    }
    return $this->container->get('comunes.disponerSobreEscritoDesistimientoGtr');
}
```

Figura 3 Método *getGestor()* de la clase *DisponerSobreEscritoDesistimientoController.*

Esto se evidencia en la aplicación cuando se va a acceder a las clases gestoras. Las clases controladoras le piden al *container* un servicio determinado y este accede al archivo de configuración del *bundle*. Como se observa en la Figura 4, la declaración de los servicios asociados a las clases

gestoras poseen una opción llamada *pattern*, la cual le dice al servicio que esa clase hereda de la clase *BaseGtr* y que también obtenga los servicios de configuración de esa clase.

```
comunes.disponerSobreEscritoDesistimientoGtr:
        class: %comunes.disponerSobreEscritoDesitimiento.gtr.class%
        parent: arquitectura.basegtr
```

Figura 4 Configuración del servicio *comunes.disponerSobreEscritoDesistimientoGtr*

Posteriormente el contenedor de servicios sabe que para instanciar esa clase necesita inyectarle el *container* y se lo pasa a los argumentos del constructor. En la Figura 5 se observa la configuración del servicio arquitectura.basegtr.

```
arquitectura.basegtr:
    class: %arquitectura.base.gtr.class%
    arguments: [@service_container]
    abstract: true
```

Figura 5 Configuración del servicio *arquitectura.basegtr*

2.5 Modelo de Diseño

Los lenguajes de modelado, como UML, son usados en la construcción de los modelos de sistemas de software. El modelo de diseño va a producir una representación visual del software a desarrollar, permitiendo guiar la fase de implementación.

2.5.1 Diagramas de Clases del Diseño

Los diagramas de clase del diseño describen gráficamente las especificaciones de las clases de software en una aplicación. Están compuestos por las clases, asociaciones, atributos, interfaces, métodos, entre otros. Un diagrama de este tipo contiene las definiciones de las entidades del software en vez de conceptos del mundo real (Craig, 1999).

En la Figura 6 se muestra el diagrama de clases asociado al CU Registrar escrito de Súplica. El resto de los diagramas pueden verse en los anexos del 6 al 21.

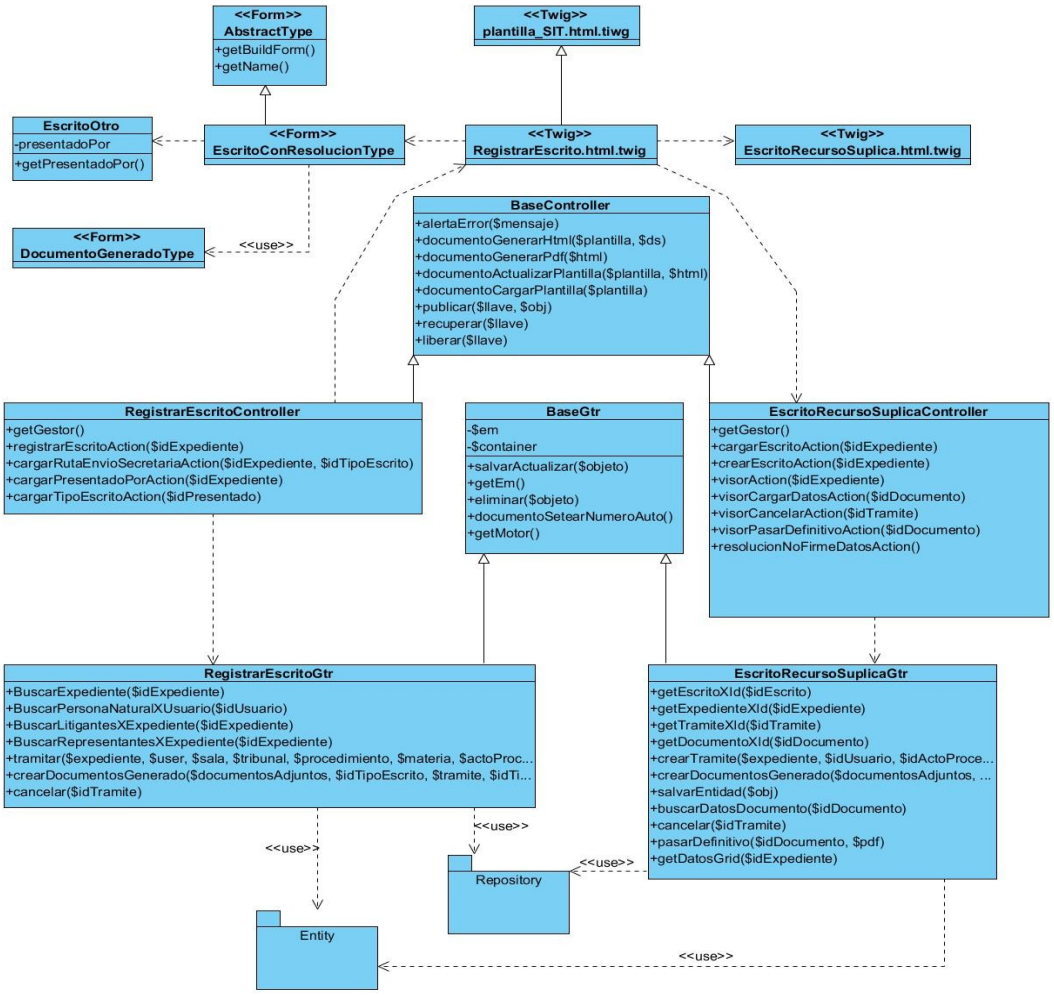

Figura 6 Diagrama de clases del caso de uso Registrar escrito de Súplica.

2.5.2 Diagramas de Secuencia

Un diagrama de secuencia es una forma de diagrama de interacción que muestra los objetos como líneas de vida a lo largo de la página y con sus interacciones en el tiempo representadas mediante mensajes dibujados como flechas desde la línea de vida origen hasta la línea de vida destino. Los diagramas de secuencia son buenos para mostrar qué objetos se comunican con qué otros objetos y qué mensajes disparan esas comunicaciones (System, 2014).

A continuación se muestra el diagrama de secuencia asociado al CU Disponer sobre escrito de Desistimiento, el cual es uno de los más significativos. Los restantes diagramas se encuentran en el expediente de proyecto.

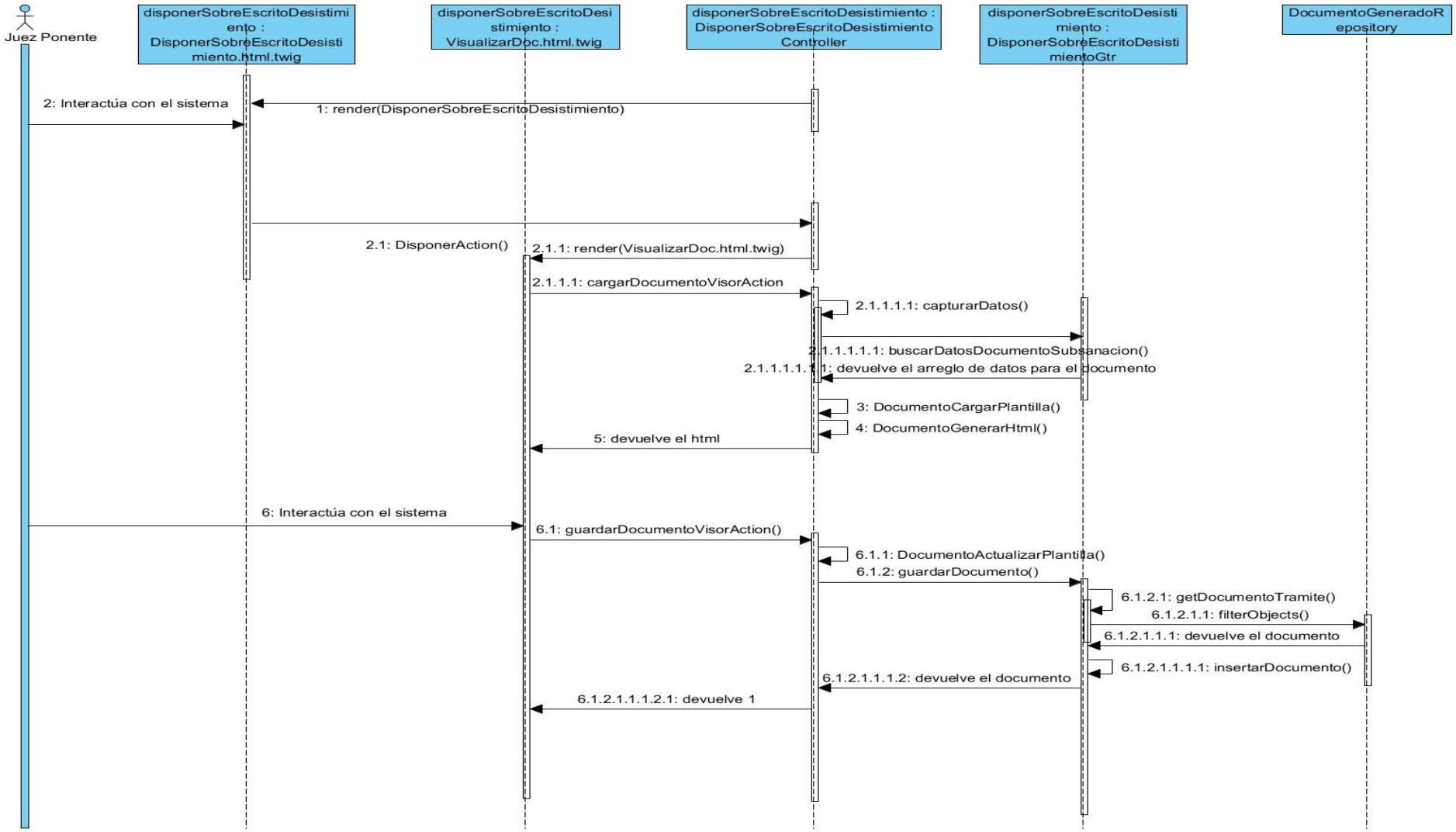

Figura 7 Diagrama de secuencia del caso de uso Disponer sobre escrito de Desistimiento.

2.5.3 Diagrama de paquetes

Cualquier sistema grande se debe dividir en unidades más pequeñas, de modo que las personas puedan trabajar con una cantidad de información limitada. La gestión del modelo consiste en paquetes y relaciones de dependencia entre paquetes. Los paquetes contienen elementos del modelo al más alto nivel, tales como clases y sus relaciones (Rumbaugh *et al.*, 2000).

A continuación se muestra el diagrama de paquetes de los componentes a desarrollar.

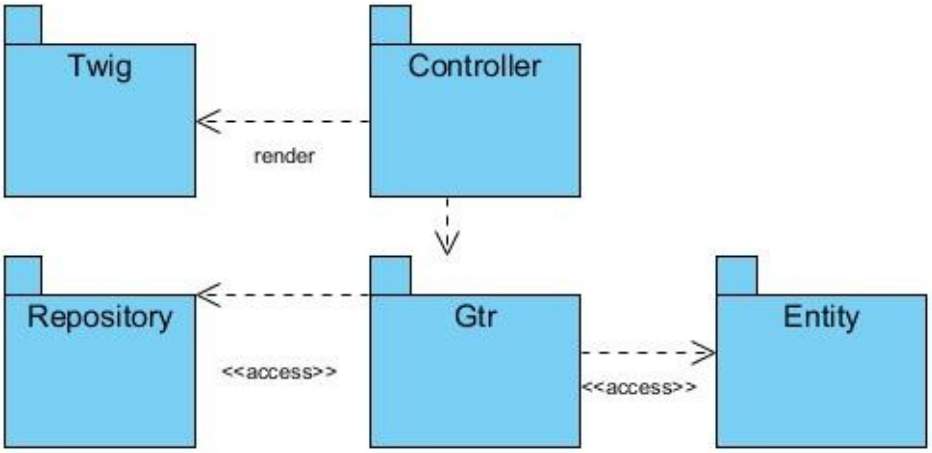

Figura 8 Diagrama de paquetes.

En el paquete Twig se encuentran todas las plantillas que serán renderizadas por el navegador web para la interacción del usuario con el sistema (Ver Anexo 1). El paquete Gtr está compuesto por las clases gestoras del negocio, que heredan de *BaseGtr* (Ver Anexo 2). El paquete Repository recoge las clases donde se almacenan las consultas de acceso a datos. Al igual que en el paquete Gtr, todas heredan de *EntityRepository* (Ver Anexo 3). Igualmente, en el paquete Controller se puede observar que todas las clases heredan de BaseController (Ver Anexo 4). Por otro lado, las clases entidades del dominio se pueden observar en el Anexo 5.

2.5.4 Modelo de datos

Un modelo de datos es una definición lógica, independiente y abstracta de los objetos, operadores y demás que en conjunto constituyen la máquina abstracta con la que interactúan los usuarios. Los

objetos permiten modelar la estructura de los datos y los operadores su comportamiento (Date, 2001). En la Figura 9 se muestra el modelo de datos.

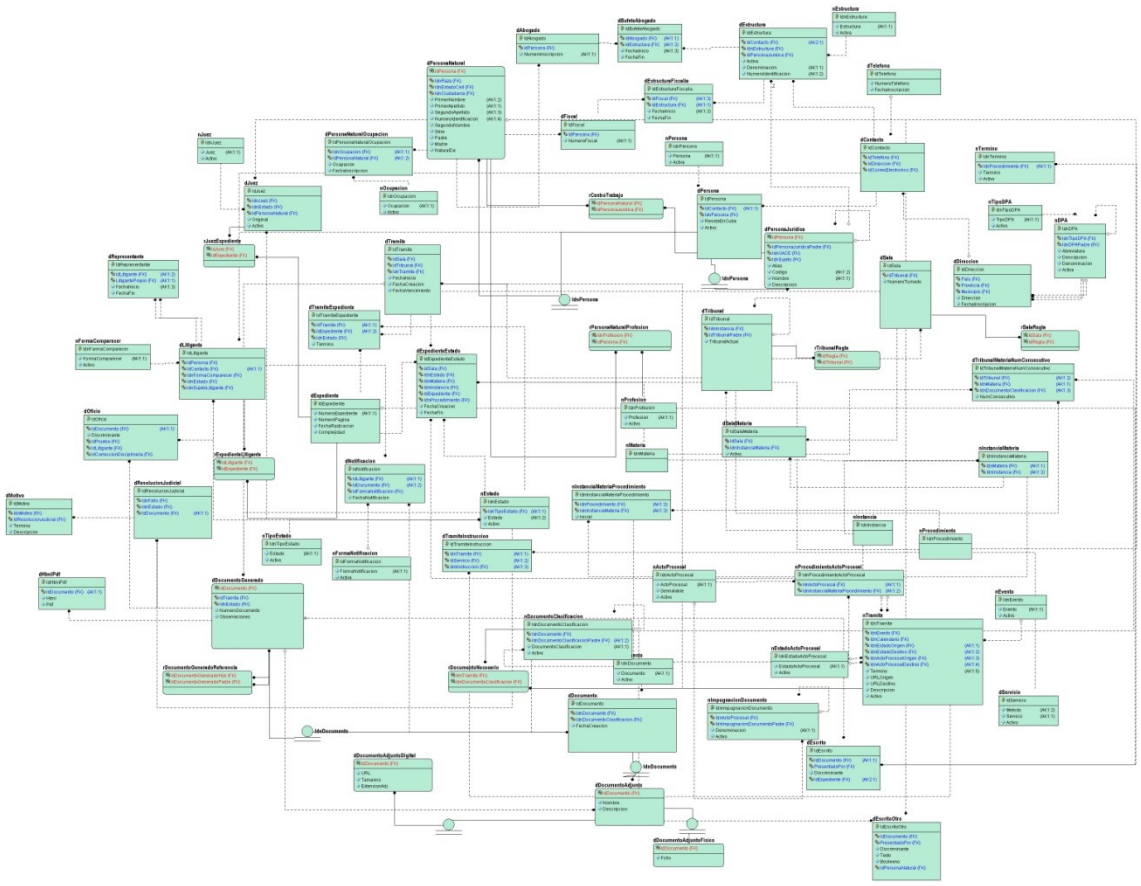

Figura 9 Modelo de datos del sistema. Fuente: (Gómez, Yosvany, 2013).

2.6 Validación del Diseño

Para validar el diseño realizado se aplicaron un conjunto de métricas. Los resultados obtenidos por cada una de las métricas utilizadas se muestran a continuación.

2.6.1 Métrica Tamaño Operacional de Clases (TOC)

Las métricas basadas en el tamaño operacional de las clases se enfocan en contar la cantidad de atributos y operaciones para una clase individual y los valores promedios para el sistema orientado a

objetos en su totalidad. El tamaño de clase puede ser determinada usando las siguientes medidas (Pressman, 2010):

- El número total de operaciones (las operaciones privadas de la clase así como las heredadas) que están encapsuladas en ella.
- El número de atributos (ya sean los atributos heredados como los propios) que están encapsulados dentro de la clase.

Para medir la responsabilidad, complejidad de implementación y la reutilización se definieron los umbrales que se muestran en la Tabla 4.

Tabla 4 Indicadores de calidad a evaluar y sus umbrales.

Indicadores de Calidad	**Categoría**	**Criterio**
Responsabilidad	Baja	< =Prom.
	Media	Entre Prom. y 2* Prom.
	Alta	> 2* Prom.
Complejidad de implementación	Baja	< =Prom.
	Media	Entre Prom. y 2* Prom.
	Alta	> 2* Prom.
Reutilización	Baja	> 2*Prom.
	Media	Entre Prom. y 2* Prom.
	Alta	<= Prom.

Para evaluar el tamaño de las clases se tuvo en cuenta que todas las clases controladoras y gestoras heredan de *BaseController* y *BaseGtr* respectivamente y debido a que las clases mencionadas anteriormente fueron definidas por el equipo de arquitectura del proyecto, las clases hijas poseían un tamaño de clase elevado. En la Tabla 5 se muestran los criterios tenidos en cuenta para definir el tamaño de una clase.

Tabla 5 Umbrales para definir el tamaño de clase.

Categoría	**Criterio**
Bajo	Menor que 13
Medio	Entre 13 y 17
Alto	Más de 17

Luego de aplicada la métrica a las 36 clases diseñadas se obtuvo que de ellas 9 poseen un tamaño bajo, 21 un valor de medio y 6 alto. En la Figura 10 se reflejan estos resultados.

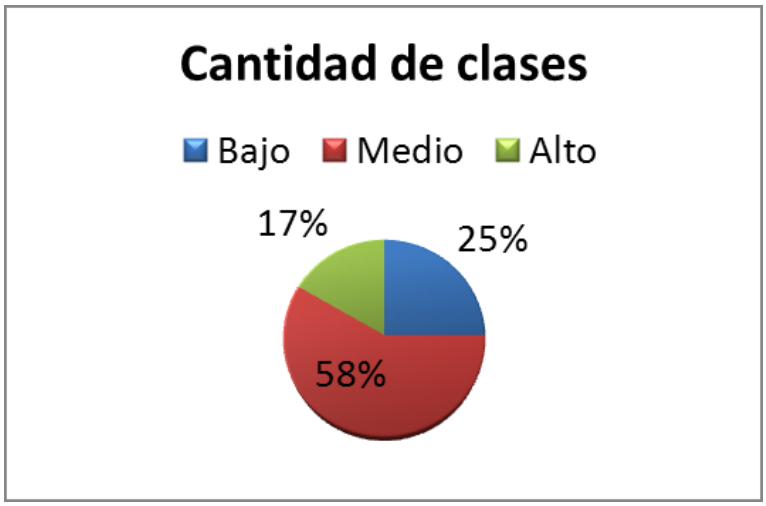

Figura 10 Resultados de la métrica Tamaño de clases.

Con respecto a la reutilización, como se puede observar en la Figura 11, se obtuvo que de las 36 clases 18 poseen una alta reutilización y 18 un valor medio para el indicador.

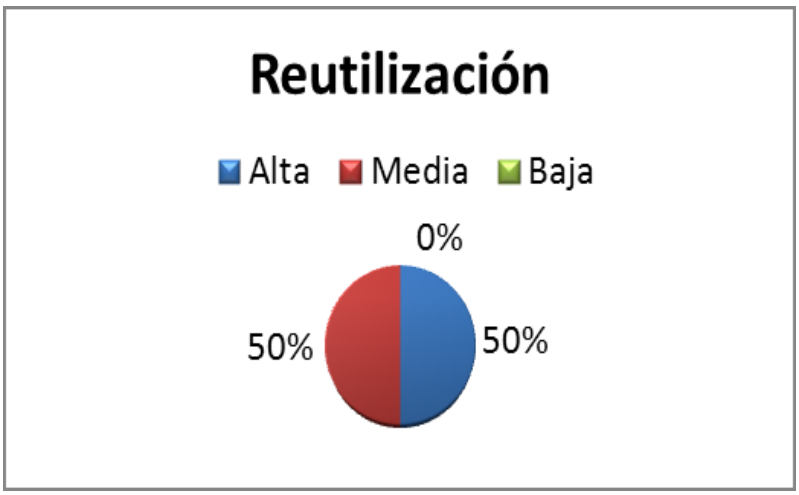

Figura 11 Resultados de la métrica Tamaño de clases para la reutilización.

Para la responsabilidad asociada a las clases se obtuvo que 18 de ellas poseen una baja responsabilidad y las restantes responsabilidad media. Los porcientos obtenidos se reflejan en la Figura 12.

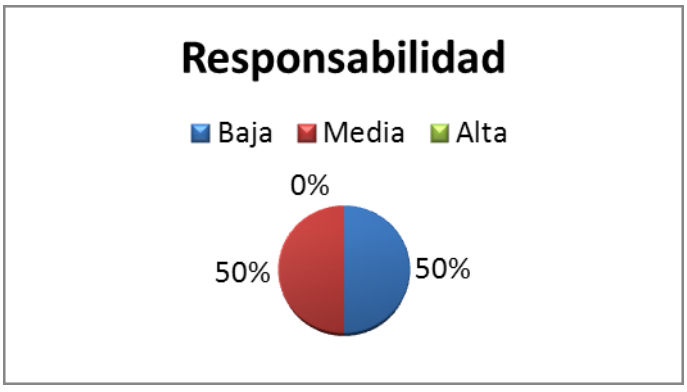

Figura 12 Resultados de la métrica Tamaño de clases para la responsabilidad.

Los valores obtenidos asociados a la complejidad de implementación fueron que 18 clases poseen una baja complejidad y el resto una complejidad media. En la Figura 13 se observan los valores en por ciento.

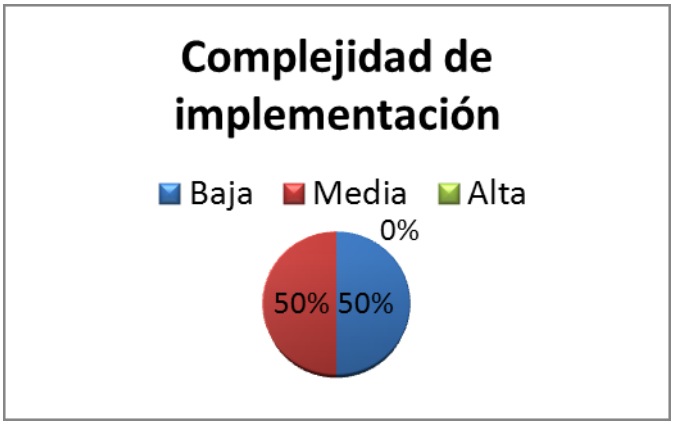

Figura 13 Resultados de la métrica Tamaño de clases para la complejidad de implementación.

2.6.2 Métrica Acoplamiento entre Objetos (AEO)

El AEO de una clase es el número de clases a las cuales una clase está ligada. Se da dependencia entre dos clases cuando una clase usa métodos o variables de la otra clase. Las clases relacionadas por herencia no se tienen en cuenta. Mientras más acoplamiento exista en una clase, más difícil será reutilizarla. Además, las clases con un excesivo acoplamiento dificultan la comprensibilidad y hacen más difícil el mantenimiento por lo que será necesario un mayor esfuerzo y riguroso testeo (Rodríguez and Harrison, 2000).

Para definir el umbral se tuvo en cuenta que todas las clases gestoras se conectan con el paquete Repository y el Entity, razón por la cual va a existir para algunas clases una relación con otras dos, los umbrales a utilizar se muestran en la Tabla 6.

Tabla 6 Umbrales asociados al acoplamiento.

Criterio	Categoría
2 dependencias(o menos)	Bueno
3 dependencias	Regular
4 dependencias	Malo

Más de 4 dependencias	Muy Malo

Se definieron los indicadores de calidad: acoplamiento, complejidad de mantenimiento, reutilización y cantidad de pruebas. Los umbrales a utilizar para medir estos atributos se observan en la Tabla 7.

Tabla 7 Umbrales asociados a los atributos de calidad.

Indicadores de Calidad	**Categoría**	**Criterio**
Acoplamiento	Ninguno	0
	Bajo	<3
	Medio	3
	Alto	>3
Complejidad de mantenimiento	Baja	<= Prom.
	Media	Entre Prom. y 2*Prom.
	Alta	> 2*Prom.
Reutilización	Baja	>2* Prom.
	Media	Entre Prom. y 2*Prom.
	Alta	<= Prom.
Cantidad de Pruebas	Baja	<= Prom.
	Media	Entre Prom. y 2*Prom.
	Alta	> 2*Prom.

La aplicación de las métricas arrojó como resultado un total de 26 clases con dos o menos dependencias, 9 con tres y 1 clase con 4 dependencias. En la Figura 14 se muestran estos resultados en porciento.

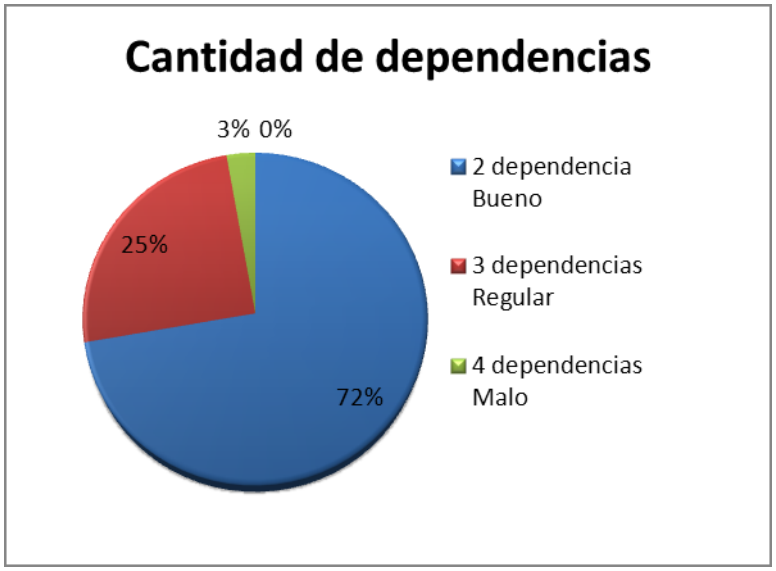

Figura 14 Resultados de la métrica AEO, cantidad de dependencias entre las clases.

Para el acoplamiento se obtuvieron 4 clases con ningún acoplamiento, 22 con bajo acoplamiento y 9 con un valor de medio. Este resultado se observa en la Figura 15.

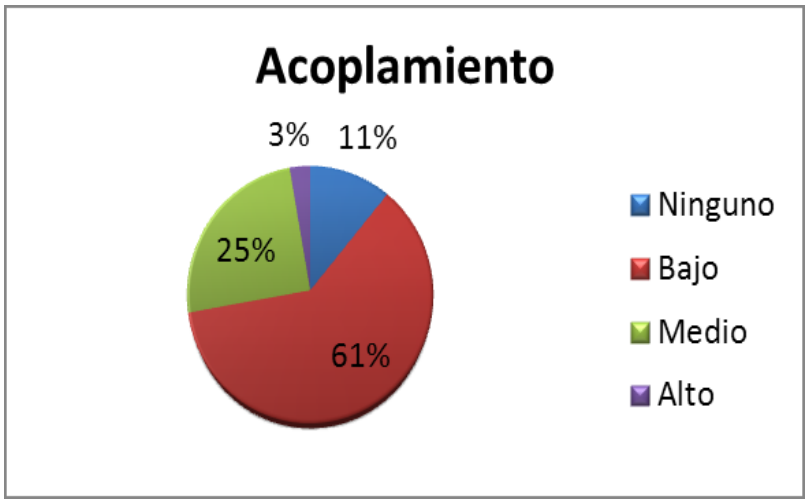

Figura 15 Resultados de la métrica AEO, acoplamiento entre los objetos.

Para la complejidad de mantenimiento se tuvo como resultado 9 clases con baja complejidad, 26 con media y 1 con una alta complejidad. Esta información en porciento se puede observar en la Figura 16.

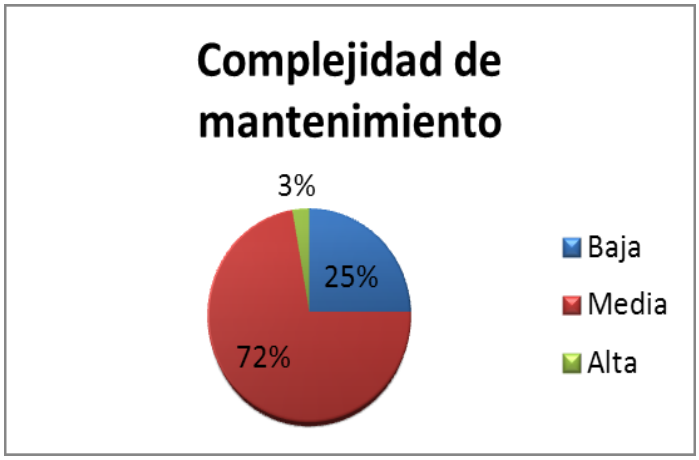

Figura 16 Resultados de la métrica AEO, complejidad de mantenimiento.

El análisis del indicador cantidad de pruebas arrojó como resultado un total de 9 clases con un valor bajo, 26 con medio y una clase con una cantidad alta.

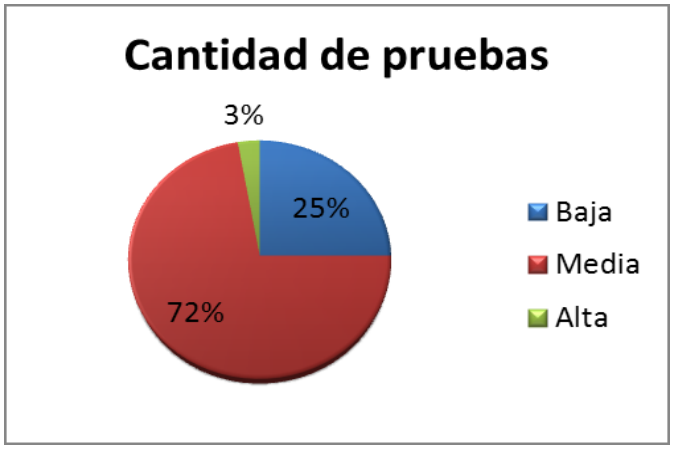

Figura 17 Resultados de la métrica AEO, cantidad de pruebas.

Para la reutilización se obtuvo como resultado 1 clase con una baja reutilización, 26 con media y 9 con alta.

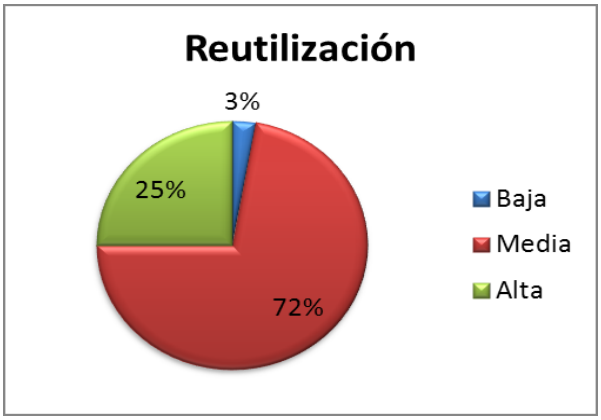

Figura 18 Resultados de la métrica AEO, reutilización.

2.6.3 Métrica árbol de profundidad de herencia (APH)

El árbol de profundidad de herencia es una métrica que brinda una medida de cuantas clases padres afectan potencialmente las clases hijas. Esta se mide como el tamaño máximo desde un nodo hasta la raíz del árbol. El impacto de la profundidad de la herencia se puede se puede observar desde diferentes puntos de vista (Chidamber and Kemerer, 1994):

- Mientras más profunda se encuentre la clase en la jerarquía, aumenta la cantidad de métodos que hereda, haciéndola más compleja de predecir su comportamiento.
- Mientras más profundo sea el árbol, más complejo será su diseño.
- Mientras más profunda sea una clase en la jerarquía tiene un mayor potencial de rehusar las operaciones heredadas.

Debido a que todas las clases gestoras y controladoras deben heredar de *BaseGtr* y *BaseController* se definió como criterio para evaluar esta métrica el que se muestra en la Tabla 8.

Tabla 8 Criterio para evaluar el APH.

Categoría	**Criterio**
Mucha profundidad	APH>1
Poca profundidad	APH<=1

La aplicación de la métrica arrojó como resultado un total de 36 clases con poca profundidad y ninguna con mucha.

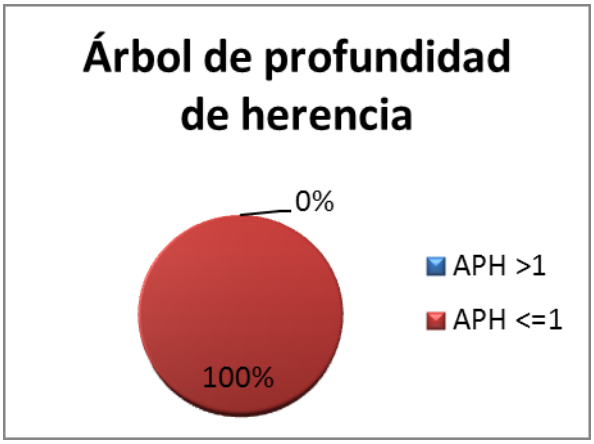

Figura 19 Resultados de la métrica Árbol de profundidad de herencia.

2.6.4 Métrica carencia de cohesión en los métodos (CCM)

La carencia de cohesión de los métodos cuenta la cantidad de parejas de métodos cuya similitud es cero menos la cantidad de métodos cuya semejanza no es cero. Mientras más grande sea la cantidad de métodos similares, mas cohesión presentará esa clase (Chidamber and Kemerer, 1994).

El valor CCM proporciona una medida de la naturaleza dispar de los métodos en la clase. Un número más pequeño de pares disjuntos implica una mayor similitud entre los métodos. CCM está íntimamente ligada a las variables de instancia y métodos de una clase, por lo tanto es una medida de los atributos de una clase de objeto (Chidamber and Kemerer, 1994).

Para la métrica se definieron las categorías que se muestran a continuación:

Tabla 9 Categoría para medir la CCM.

Categoría	Criterio
Alta	CCM = 0
Baja	CCM >0

Como resultado se obtuvo que 35 clases tienen una baja carencia y 1 posee alta carencia.

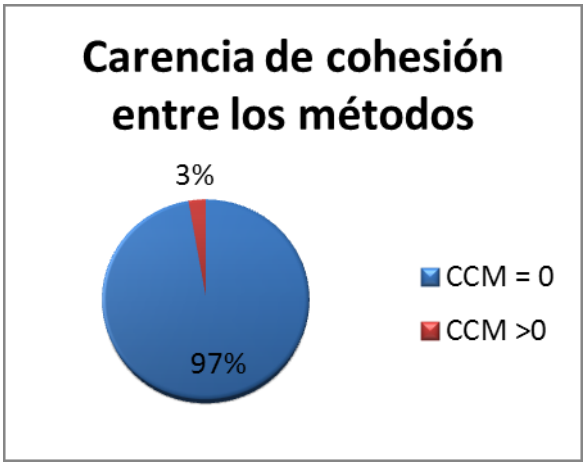

Figura 20 Resultados de la métrica Carencia de cohesión entre los métodos.

La aplicación de las métricas arrojó como resultado que las clases del diseño propuesto poseen baja complejidad de mantenimiento e implementación, baja cantidad de pruebas, baja carencia de cohesión entre los métodos y baja complejidad del diseño. Por otro lado, la responsabilidad es media y la reutilización es alta, quedando demostrada la calidad del diseño.

Conclusiones parciales del capítulo

Luego de realizadas las actividades correspondientes al diseño del sistema se llegaron a las siguientes conclusiones:

- Los artefactos descritos son de gran importancia para la construcción del sistema, facilitando una definición en detalle para permitir su interpretación e implementación.
- El uso de patrones de diseño posibilitó modelar una solución teniendo como objetivo la reutilización de código y la solución a problemas en contextos similares durante el desarrollo de software.
- El Modelo del diseño permitió materializar los requisitos del cliente sirviendo como guía para las actividades de implementación, al producir una representación del software a desarrollar.
- El diagrama de componentes mostró la estructura lógica del sistema, permitiendo encapsular las funcionalidades y lograr un proceso de implementación más organizado y productivo.
- Las métricas utilizadas arrojaron resultados satisfactorios.

CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBAS AL SISTEMA

Introducción

En el presente capítulo se detalla la información referente a los artefactos generados durante la implementación, así como el uso de los estándares de codificación seleccionados y los resultados obtenidos después de la realización de pruebas al software.

3.1 Estándares de codificación

Con el objetivo de facilitar el análisis del código y el mantenimiento del sistema, en la implementación de los componentes se utilizará el estándar de codificación del lenguaje PHP unido a las pautas que se describen a continuación. Esta información se encuentra en el documento *CEGEL_SITPC_Estándares de codificación v1.0* del expediente del proyecto SITPC (Domínguez, 2013).

Cabecera del archivo:

Es importante que todos los archivos .php inicien con una cabecera específica que indique el autor de los últimos cambios. Cada equipo de desarrollo decide según las necesidades de documentación de código, agregar o modificar esta información (ver Figura 21).

Figura 21 Ejemplo del estándar Cabecera de archivo.

Comentario en las funciones:

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

Ubicación y denominación de archivos:

Se ubicarán los archivos según las convenciones establecidas por Symfony2 o según las especificaciones del equipo de arquitectura.

Para la denominación de los archivos se seguirán las convenciones establecidas por Symfony2 o el equipo de arquitectura. Ejemplo:

- Para las clases de gestión del negocio se usará el sufijo Gtr, ejemplo: AcusadoGtr.
- Para los tablemodel definidos para los grid se usará el sufijo Tm, ejemplo: AcusadoTm.
- Para las entidades de presentación se usará el sufijo Ep, ejemplo: AcusadoEp.

En las plantillas html.twig definidas para la vista el nombre del archivo debe seguir el estándar de la denominación de las clases. Ejemplo:

- Acusados.html.twig
- En caso de estar formado por más de una palabra: AcusadosRebeldes.html.twig

Clases:

Las clases serán colocadas en un archivo .php aparte, donde sólo se colocará el código de la clase. El nombre del archivo será el de la clase. Las clases siguen las reglas de las funciones, por tanto, debe colocarse un comentario antes de la declaración de la clase explicando su utilidad. Los nombres de las clases deben de iniciar con letra mayúscula. Para los nombres compuestos por más de una palabra, la primera letra de cada nueva palabra debe capitalizarse. No se permiten las letras capitalizadas sucesivas; por ejemplo, una clase "SymfonyPDF" no se permite, mientras " SymfonyPdf" es aceptable.

Nombre de variables:

Los nombres deben ser descriptivos y concisos. No usar ni grandes frases ni pequeñas abreviaciones para las variables. Siempre es mejor saber qué hace una variable con sólo conocer su nombre. Esto aplica para los nombres de variables, funciones, argumentos de funciones y clases. Los nombres de las variables y de las funciones pueden iniciar con letra minúscula, pero si estas tienen más de una palabra, cada nueva palabra debe iniciar con letra mayúscula. Las constantes deben de escribirse siempre en mayúsculas y tanto estas como las variables globales deben de tener como prefijo el nombre de la clase a la que pertenecen.

Definiciones de la función:

Los nombres de la función pueden contener sólo caracteres alfanuméricos y comenzar con letra minúscula. Cuando el nombre de una función está compuesto por más de una palabra, la primera letra de cada nueva palabra debe capitalizarse. En la Figura 22 se puede ver un ejemplo.

```
public function indexAction($idTramite) {
}
```

Figura 22 Ejemplo del estándar Definiciones de la función.

Siempre incluir las llaves:

En todo momento a la hora de codificar un bloque de instrucciones, éste debe ir encerrado entre llaves, aun cuando conste de una sola línea. Un ejemplo correcto se observa en la Figura 23.

```
if ($tmp->getTipoFormaComparecer()->getId() == EFormaComparecer::Demandado) {
    $representaAcusado = true;
}
```

Figura 23 Ejemplo del estándar Siempre incluir las llaves.

Llaves:

Las llaves de apertura irán al final de la sentencia que delimitan y las de cierre estarán alineadas con el inicio de la sentencia en una nueva línea (Ver Figura 24).

```
foreach ($jueces as $tmp) {
    if ($tmp->getTipoJuez()->getId() != ETipoJuez::JuezPonente) {
        $juez[] = $tmp->getPersonaNatural()->getNombreCompleto();
    }
}
```

Figura 24 Ejemplo del estándar Llaves.

No utilizar variables sin inicializar:

Si no se tiene control sobre el valor de una variable, se debe verificar que esté inicializada. Esto lo permite PHP de la forma representada en la Figura 25.

```
if (isset($doc) && $doc != null) {
```

Figura 25 Ejemplo del estándar No utilizar variables sin inicializar.

Pero sólo se debe usar esta opción cuando no se tenga el control o no se esté seguro del valor que pueda tener la variable (Como en variables que llegan por parámetro o por GET).

3.2 Diagrama de despliegue

Un Diagrama de despliegue modela la arquitectura en tiempo de ejecución de un sistema. Esto muestra la configuración de los elementos de hardware (nodos) y cómo los elementos y artefactos del software se trazan en esos nodos (System, 2014). A continuación se presenta el diagrama de despliegue del sistema.

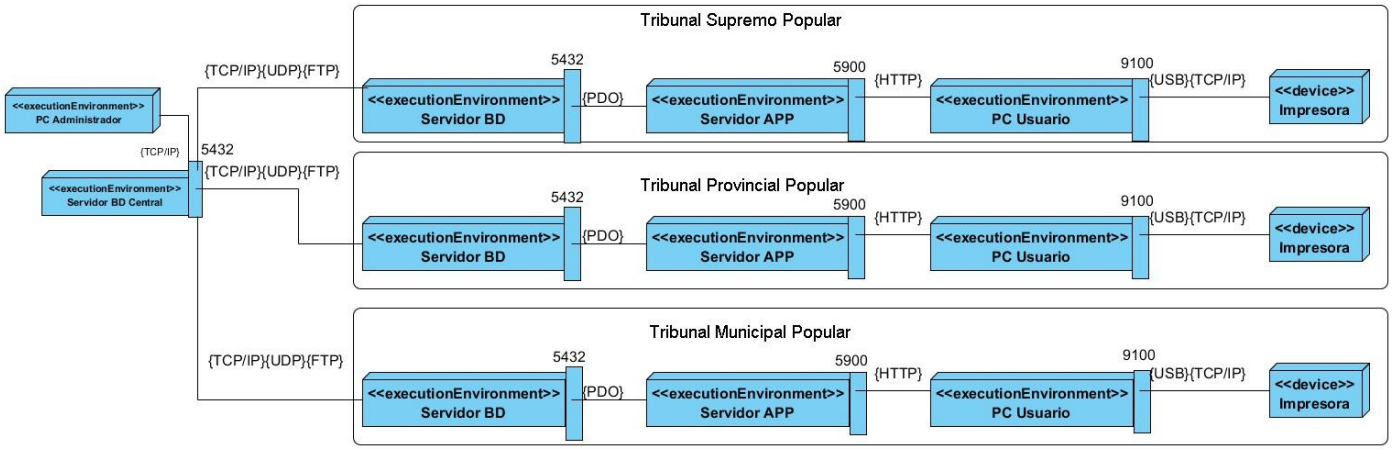

Figura 26 Diagrama de despliegue del sistema

El diagrama de despliegue anteriormente presentado, se compone por una PC-Cliente que se encarga de la gestión del servidor central de datos, esta conexión se realiza mediante un protocolo TCP/IP, dicho servidor está conectado mediante los protocolos TCP/IP, UDP y FTP, por el cual se realiza el acceso de la información, a cada servidor de BD que se encuentra en cada una de las instancias de los tribunales, estos se conectan con un servidor de aplicaciones mediante la extensión PHP Data Object (PDO), y estos a su vez están conectados a las PC Usuario existentes en los tribunales mediante el protocolo HTTP, estas PC se encuentran conectadas mediante una conexión TCP/IP o USB a una impresora ubicada en el tribunal donde se emiten los documentos necesarios.

3.3 Seguridad

En el SITPC se realiza la gestión de los usuarios mediante el sistema Acaxia debido a que este permite gestionar el control de acceso de varias aplicaciones de forma simultánea en entornos multidominios. La arquitectura del sistema está dividida en nueve módulos fundamentales que le permite garantizar el control de acceso a sí misma y a otros sistemas que formen parte de su dominio de aplicación (Gómez, Oiner, 2012). Entre los principales módulos se encuentran:

- Gestión de organizaciones y dominios: tiene la responsabilidad de proveer los mecanismos necesarios para que se establezcan desde el inicio las restricciones y relaciones jerárquicas entre los distintos tipos de organizaciones y los atributos que se van a gestionar por cada una de ellas.
- Gestión de servidores: este módulo brinda la posibilidad de configurar los servidores de base de datos y de autenticación que van a utilizar cada uno de los sistemas para la gestión de los datos y la identificación y autenticación de los usuarios respectivamente.
- Gestión de sistemas: se encarga de gestionar las estructuras de los sistemas (sistemas, subsistemas, funcionalidades) a los cuales se necesita proveer seguridad.
- Gestión de roles y usuarios: tiene la responsabilidad de establecer la mayor parte de las políticas de acceso, apoyándose en las configuraciones realizadas en los demás módulos.

3.4 Pruebas de software

Para la verificación de los componentes desarrollados se realizaron pruebas utilizando los métodos de caja blanca y caja negra.

3.4.1 Pruebas de caja blanca

La prueba de caja blanca es un método que usa la estructura de control descrita como parte del diseño al nivel de componentes para derivar los casos de pruebas. Se basa en un examen cercano al detalle procedimental, donde se prueban las rutas lógicas del software y la colaboración entre componentes (Pressman, 2010).

La técnica de la ruta básica permite la obtención de una medida de la complejidad lógica de un diseño procedimental y que use esta medida como guía para definir un conjunto básico de rutas de ejecución. Los casos de prueba derivados para ejercitar el conjunto básico deben garantizar que se ejecute cada instrucción del programa por lo menos una vez durante la prueba.

A continuación se muestra el grafo de la ruta básica perteneciente al método *insertarDocumento($html,$datos)* de la clase *DisponerSobreEscritoDesistimientoGtr*.

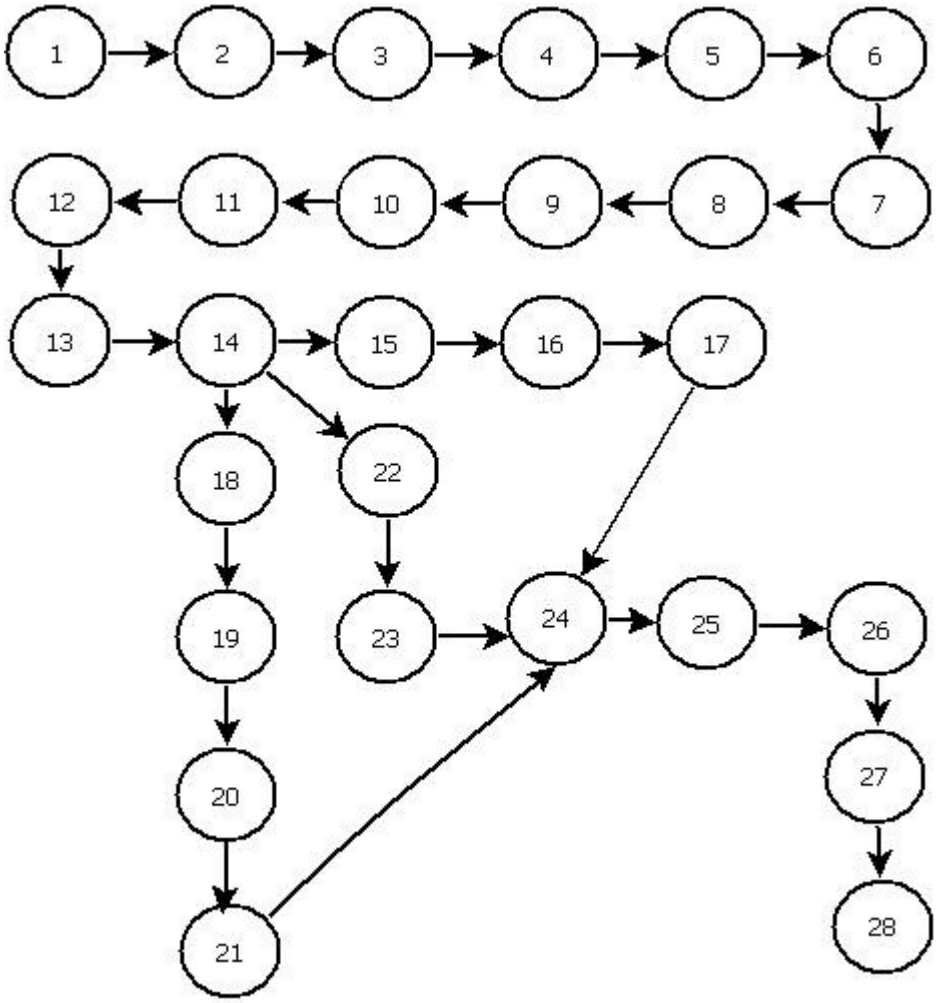

Figura 27 Ruta básica del método *insertarDocumento.*

Luego de analizada la ruta básica de la Figura 27 se procedió a calcular la complejidad ciclomática del método, esto se hace siguiendo la ecuación:

V(G)=A-N+2 donde A es la cantidad de aristas del grafo y N la cantidad de nodos.

En el caso del método que se está analizando la complejidad quedaría de la siguiente manera:

V(G)=29-28+2= 3

El resultado obtenido luego de calculada la complejidad ciclomática indica que existen tres caminos posibles que se describen a continuación:

a) 1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-24-25-26-27-28

b) 1-2-3-4-5-6-7-8-9-10-11-12-13-14-18-19-20-21-24-25-26-27-28
c) 1-2-3-4-5-6-7-8-9-10-11-12-13-14-22-23-24-25-26-27-28

Luego de determinados los caminos se realizaron los casos de prueba para cada uno. En la Tabla 10 se muestra el caso de prueba para la tercera ruta definida anteriormente.

Tabla 10 Diseño de un caso de prueba del método *insertarDocumento*.

Entrada	**Resultados esperados**	**Condiciones**
Para la correcta ejecución de la función es necesario el html con el que se desea crear el documento y un arreglo de datos donde viene como parámetro el tipo de documento clasificación.	Se genera un documento de tipo Auto de rechazo de desistimiento con un motivo de rechazo.	$datos['tipoDoc'] == ETipoResolucionAuto:Rechazo_Desistimiento

3.4.2 Pruebas de caja negra

Las pruebas de caja negra se concentran en los requisitos funcionales del software, permitiendo que se obtengan un conjunto de condiciones de entrada que ejercitarán por completo todos los requisitos funcionales del sistema. Al aplicar técnicas de caja negra se obtienen un conjunto de casos de prueba (Pressman, 2010).

La partición equivalente es una técnica de prueba de caja negra que divide el dominio de entrada de un programa en clases de datos a partir de las cuales pueden derivarse casos de prueba. El método se esfuerza por definir un caso de prueba que descubra ciertas clases de errores, reduciendo así el número total de casos de prueba que deben desarrollarse. Una clase de equivalencia representa un conjunto de estados válidos y no válidos para las condiciones de entrada (Pressman, 2010).

Para la realización de las pruebas de caja negra a la solución fueron analizadas todas las funcionalidades del sistema, realizándose 15 diseños de casos de prueba. Se efectuaron un total de 3 iteraciones para poder alcanzar resultados satisfactorios desde el punto de vista funcional, atendiendo al correcto comportamiento del sistema ante diferentes situaciones (entradas válidas y no válidas). En la Figura 28 se realiza una descripción de las no conformidades detectadas en cada iteración de las pruebas.

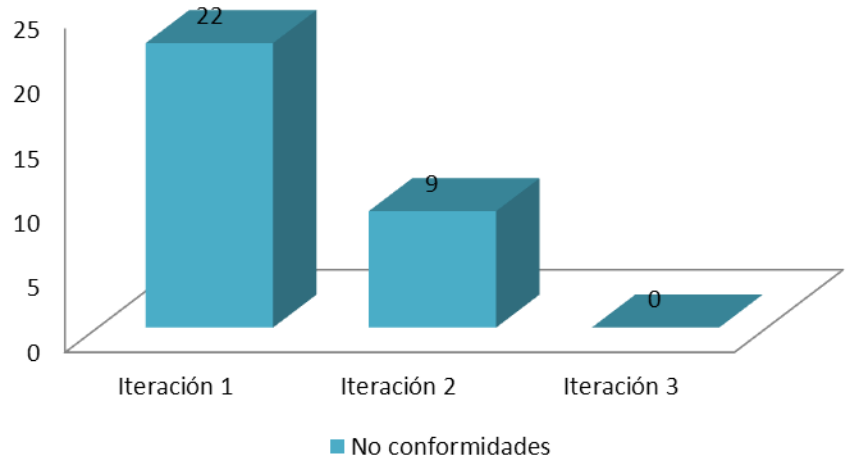

Figura 28 Resultados de las iteraciones realizadas en las pruebas de caja negra.

3.5 Validación de las variables de la investigación

Para desarrollar la investigación fueron identificados problemas en el control y disponibilidad de la información en los procesos de Súplica y Desistimiento de los TPC. Estas dificultades fueron dadas por:

- La demora en las búsquedas y obtención de la información.
- El vencimiento de los términos sin que el proceso sea resuelto.

En la Tabla 11 se describen como fueron erradicados estos problemas con la implementación de la solución propuesta.

Tabla 11 Validación de las variables de la investigación.

Variables	**Proceso actual de los TPC**	**Proceso con la utilización de los paquetes implementados.**
Disponibilidad de la información	Los expedientes están en formato duro, lo que puede provocar rotura y pérdida de la información.	Los expedientes están almacenados de forma digital en la herramienta, preservándose de romperse. La información guardada persiste en la base de datos, a la cual se le realizan copias de seguridad, evitándose la pérdida de información.

	Los expedientes se encuentran almacenados en estantes, lo que dificulta su búsqueda para la consulta de su información.	Los expedientes se encuentran disponibles en el sistema, tanto durante el proceso como una vez culminado el mismo.
	El acceso a los expedientes se realiza a través de la secretaria.	El acceso a los expedientes puede ser realizado por los usuarios según el permiso que tengan en el sistema, en cualquier momento.
	No se puede realizar el acceso simultáneo por dos personas o más a los expedientes, debido a que existe un único expediente en cada proceso.	En el sistema igualmente existe un único expediente, pero debido a que está digital puede ser consultado simultáneamente por el personal autorizado.
Control de la información	Los expedientes tienen un número que los identifica. Debido a como se generan y almacenan los expedientes puede darse el caso de que este número distintivo se repita.	Los números identificativos de los expedientes son generados automáticamente por el sistema, validándose que no existan expedientes distintos con el mismo número.
	Los trámites de Súplica y Desistimiento tienen un plazo para realizarse. Si estos se vencen la secretaria debe avisar al juez de este vencimiento. Actualmente, por el alto contenido de trabajo, se vencen los términos sin que el proceso sea resuelto.	Los trámites de Súplica y Desistimiento son controlados por el sistema. Mediante la utilización de semáforos se le indica al usuario el estado en que se encuentra cada uno de los trámites del expediente.

Conclusiones parciales del capítulo

Después de desarrollada la aplicación, las pruebas a los componentes implementados y realizada la validación de las variables de la investigación se arribó a las siguientes conclusiones:

- Los estándares de programación utilizados permitieron realizar una implementación legible y entendible, para su futuro mantenimiento.
- Se realizó el diagrama de despliegue el cual permitió comprender como funcionará el sistema una vez desplegado.
- Los resultados obtenidos en las pruebas de caja blanca demostraron que no existe código innecesario dentro de los paquetes implementados. Los resultados arrojados en las pruebas de caja negra validaron que las funcionalidades desarrolladas son correctas.
- Se validaron las variables que forman parte del problema de la investigación, demostrándose que con el sistema se contribuye a a elevar la disponibilidad y control de la información en los procesos de Súplica y Desistimiento del TPC.

CONCLUSIONES

Finalizado el desarrollo del presente trabajo de diploma se concluye:

- La definición del marco teórico permitió una mayor comprensión del contexto donde se desarrolla la investigación, definiéndose las tecnologías adecuadas para desarrollar el sistema propuesto.
- El estudio de los sistemas jurídicos utilizados en Cuba y el mundo indicó que ninguno es adecuado para la investigación, quedando demostrada la necesidad de implementar los componentes Súplica y Desistimiento.
- El diseño realizado permitió materializar los requisitos, sirviendo de guía para las actividades de implementación. Su validación se realizó mediante métricas, las cuales arrojaron resultados satisfactorios.
- Los resultados obtenidos en las pruebas de caja blanca demostraron que no existe código innecesario dentro de los componentes implementados. Los resultados arrojados en las pruebas de caja negra validaron que las funcionalidades desarrolladas son correctas.
- Se validaron las variables que forman parte del problema de la investigación, demostrándose que con el sistema se contribuye a a elevar la disponibilidad y control de la información en los procesos de Súplica y Desistimiento del TPC, cumpliéndose el objetivo general de la investigación.

RECOMENDACIONES

- Se recomienda que se tenga en cuenta la integración de los componentes Súplica y Desistimiento en los módulos que serán desarrollados en las próximas fases del proyecto SITPC.

BIBLIOGRAFÍA

Alegsa. *Clasificación de los lenguajes de programación*, alegsaonline.com, 2006. [Disponible en:

Álvarez, M. *Manual de jQuery*, 2012. [2013]. Disponible en: http://www.desarrolloweb.com/manuales/manual-jquery.html

Amaro Salup, J. R. *Ley de procedimiento civil, administrativo, laboral y económico (LPCALE)*. La Habana, Cuba, Consejo de Gobierno del Tribunal Supremo Popular, 1996.

Aronson, L. *HTML manual of style : a clear, concise reference for hypertext markup language (including HTML5)*. 4ta edicion. EEUU, Weasley, Addison, 2011. 315 p. 978-0-321-71208-0

Atrio, C. *Sistema seguro de intercambio de documentos Lexnet*. Montevideo, 2011. 2013.

Barrios, A. *Reglas del negocio del proyecto SITPC*. La Habana, Cuba, Proyecto de Informatización de los Tribunales Populares Cubanos, Centro de Gobierno Electrónico, Universidad de las Ciencias Informáticas, 2013.

Bass, L.; P. Clements*, et al. Software Architecture in Practice*. Tercera. EEUU, Addison Wesley, 2013. 588 p. 978-0-321-81573-6

Ben-Ari, M. *Understanding Programming Languages*. John Wiley & Sons, 1996. 376 p. 978-0471958468

Brandendaugh, J. *Aplicaciones JavaScript* Madrid, España, O'Reilly & Associates, Inc, 2000. 540 p. 84-415-1070-9

Brindys. *Características Técnicas Gedex*, 2013a. [2013]. Disponible en: http://www.brindys.com/ca/caracteristiques-tecniques.html

Brindys. *GEDEX-Manual de Usuario*, 2013b. [2013]. Disponible en: http://www.brindys.com/docs/cas10001050.html

Buschmann , F.; R. Meunier*, et al. Pattern- Oriented Software Architecture: A System of Patterns*. New York, EEUU, JOHN WILEY & SONS, 1996. 467 p. 0-471-95889 -7

C. Mitchell, J. *Concepts in Programming Languages*. Universidad de Cambridge, Reino Unido, 2004. 529 p. 0-521-78098-5

Carrero, A. *Conceptos básicos de ORM (Object Relational Mapping)*, 2013. [2013]. Disponible en: http://www.programacion.com/articulo/conceptos_basicos_de_orm_object_relational_mapping_349

Craig, L. *UML y Patrones: Introducción al análisis y diseño orientado a objetos*. Mexico, Prentice Hall, 1999. 536 p. 970-17-0261-1

Chidamber, S. and F. Kemerer. *A Metrics Suite for Object Oriented Design*. *IEEE Transactions on Software Engineering*, 1994. 20: 18.

Date, C. J. *Introducción a los sistemas de bases de datos*. 7ma. México, Pearson Education, 2001. p. 968-444-419-2

De Sousa´S, S. *The Advantages and Disadvantages / Best Practices of RUP Software Development*, 2009. 2013.

Doctrine-Projects. *Doctrine 2 ORM 2.0.0 documentation*, 2013. [2013]. Disponible en: http://docs.doctrine-project.org/en/2.0.x/reference/introduction.html

Domínguez, Y. *Arquitectura del Sistema de Informatización de los Tribunales Populares Cubanos*. La Habana, Cuba, Centro de Gobierno Electrónico, Universidad de las Ciencias Informáticas, 2014.

Domínguez, Y. *CEGEL_SITPC_Estándares de codificación v1.0*. La Habana, Cuba, Centro de Gobierno Electrónico, Universidad de las Ciencias Informáticas, 2013.

Doyle, M. *Beginning PHP 5.3*. Indiana, EEUU, Wrox, 2011. 840 p. 978-0-470-41396-8

Estévez, D. and P. Romero. *IMPLEMENTACIÓN DEL PROCEDIMIENTO DILIGENCIAS PREVIAS DEL SUBSISTEMA ECONÓMICO DEL PROYECTO DE INFORMATIZACIÓN DE LOS TRIBUNALES POPULARES CUBANOS*. La Habana, Cuba, Universidad de las Ciencias Infórmaticas, 2012. 69. p.

Fowler, M. *Patterns of Enterprise Application Architecture*. Primera. EEUU, Addison Weasley, 2002. p. 978-0321127426

Gabbrielli, M. and S. Martini. *Programming Languages: Principles and Paradigms*. 10. Bologna, Italia, 2006. 440 p. 978-1-84882-913-8

Gamma, E.; R. Helm*, et al. DESIGN PATTERNS : ELEMENTS OF REUSABLE OBJECT-ORIENTED SOFTWARE*. Addison Weasley, 1997. 395 p. 978-0-201-63361-0

Garzás, J. *Informe CHAOS: Visión y críticas sobre el éxito de los proyectos software*, 2010. 2013.

Gespro. *CEGEL - Centro de Gobierno Electrónico: Misión*, 2013. [2013]. Disponible en: http://gespro.cegel.prod.uci.cu/

Gómez, O. *CAEM: Modelo de control de acceso para sistemas de información en entornos multidominios.*: Departamento de Tecnología, Centro para la Informatización de la Gestión de Entidades. La Habana, Cuba, Universidad de las Ciencias Informáticas, 2012. p.

Gómez, Y. *Modelo de datos del Proyecto Sistema de Informatización de los Tribunales Populares Cubanos.* La Habana, Cuba, Centro de Gobierno Electrónico, Universidad de las Ciencias Informáticas, 2013.

González, D. and J. González. *Sistema para la Tramitación de Procesos Penales.*, 2008. p.

Inei. *Herramientas CASE*, Instituto Nacional de Estadística e Informática, 1999. 52.

Isdefe. *Anexo Técnico para el Servicio de Asistencia Técnica para la realización de las actividades de Desarrollo Informático necesarias para el mantenimiento correctivo y evolutivo del aplicativo Minerva-NOJ. Expediente 2013-00218*. Madrid, España, Ingeniería de Sistemas para la Defensa de España., 2013.

Jacobson, I.; G. Booch*, et al. El Proceso Unificado de Desarrollo de Software*. OTERO, A. Madrid, 2000. 464.

Judicial, D. *Si se cayó el sistema, se trabaja después de hora*, 2009. [2013]. Disponible en: http://www.diariojudicial.com/contenidos/2009/03/18/noticia_0004.html

Jurisoft. *Infolex Abogados*, Jurisoft, 2012. [2013]. Disponible en: http://www.infolex.es/ie/infolexabogados_modulos.aspx?menu=3

Justica, P. A. *¿Qué es Lexnet?*, 2013. [2013]. Disponible en: https://www.administraciondejusticia.gob.es/paj/publico/ciudadano/informacion_institucional/modernizacion/modernizacion_tecnologica/infolexnet/que_es/

Kabir, M. *La Biblia del Servidor Apache 2*. Anaya Multimedia, 2003. 845 p. 8441514682

Kuchana, P. *Software Architecture Design Patterns in Java*. Florida, EEUU, 2004. 476 p. 0-8493-2142-5

Maldonado, D. *¿Qué son los IDE de programación?*, www.elcodigok.com.ar, 2007. [2013]. Disponible en: www.elcodigok.com.ar

Martensson, F. *SOFTWARE ARCHITECTURE QUALITY EVALUATION APPROACHES IN AN INDUSTRIAL CONTEXT*. Karlskrona, Suecia, Blekinge Institute of Technology, 2006. 138.

Mateu, C. *Desarrollo de aplicaciones web*. Primera. Barcelona, España, Eureca Media, 2004. 377 p.

Netbeans-Project. *NetBeans IDE - The Smarter and Faster Way to Code* 2013. [2013]. Disponible en: https://netbeans.org/features/index.html

Noorullah, R. M. *Grasp and GOF Patterns in Solving Design Problems*. *International Journal of Engineering and Technology*, IJET Publications UK, 2011. 1: 196-205.

Paradigm, V. *Visual Paradigm for UML - UML tool for software application development*, 2013. [2013]. Disponible en: http://www.visual-paradigm.com/product/vpuml/

Ponjuán, G. *Gestión de información en las organizaciones. Principios, conceptos y aplicaciones*. Santiago de Chile, CECAPI, 1998. p.

Postgresql. *About PostgreSQL*, Sitio oficial de PostgreSQL, 2014. [2014]. Disponible en: http://postgresql.org/about

Potencier, F. *Twig Documentation*, 2011. [2014]. Disponible en: http://twig.sensiolabs.org/doc/intro.html

Potencier, F. and F. Zaninotto. *Symfony, la guía definitiva*. New York. EEUU, 2008. 435 p. 1590597869

Pressman, R. S. *Software Engineering: A Practitioner´s Approach*. Séptima edición. New York, EEUU, McGraw-Hill, 2010. 870 p. 978-0-07-337597 -7

Reynoso, C. and N. Kicillof. *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft*

Argentina, Universidad de Buenos Aires, 2004. 76.

Rodríguez, D. and R. Harrison. *Medición para la Gestión en la Ingeniería del Software*. España, RA-MA, 2000. p. 84-7897-403-2

Roque, Y. and M. Brito. *Especificación de requisitos no funcionales del sistema. Proyecto de Informatización de los Tribunales Populares Cubanos.* La Habana, Cuba, Centro de Gobierno Electrónico, Universidad de las Ciencias Informáticas, 2013. 1-12.

Rumbaugh, J.; I. Jacobson*, et al. El Lenguaje Unificado de Modelado. Manual de referencia.* Madrid, España, Addison, Weasley, 2000. 552.

Sebasta, R. W. *Concepts of programming languages*. 10. Colorado, EEUU, 2012. 795 p. 978-0-13-139531-2

Seeman, M. *Dependency Injection in .NET*. EEUU, Manning Publications Co., 2012. 552 p. 9781935182504

Sillberchatz, A.; H. Korth*, et al. Fundamentos de bases de datos*. 4ta edición. Madrid, España, McGRAW-HILL, 2002. 787 p. 0-07-228363-7

Spurlock, J. *Bootstrap*. Primera edición. EEUU, O'REILLY, 2013. 128 p. 978-1-44934-34391-0

System, S. *Diagrama de Secuencia UML 2*, 2014. [2014]. Disponible en: http://www.sparxsystems.com.ar/resources/tutorial/uml2_sequencediagram.html

Unitech. *IURIX. La solución integral para informatizar la Gestión Judicial*, 2011. [2013]. Disponible en: http://www.unitech.com.ar/productos/iurix/

Vargas, E. and Y. Fustiel. *Especificación de casos de uso. Proyecto de Informatización de los Tribunales Populares Cubanos.* La Habana, Cuba, Centro de Gobierno Electrónico, Universidad de las Ciencias Informáticas, 2013.

Watt, D. A. *Programming language design concepts*. 2004. p. 0-470-85320-4