Universidad de las Ciencias Informáticas

Facultad 3

Tema: Diseño e implementación del subproceso Pruebas Documental y Libro 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

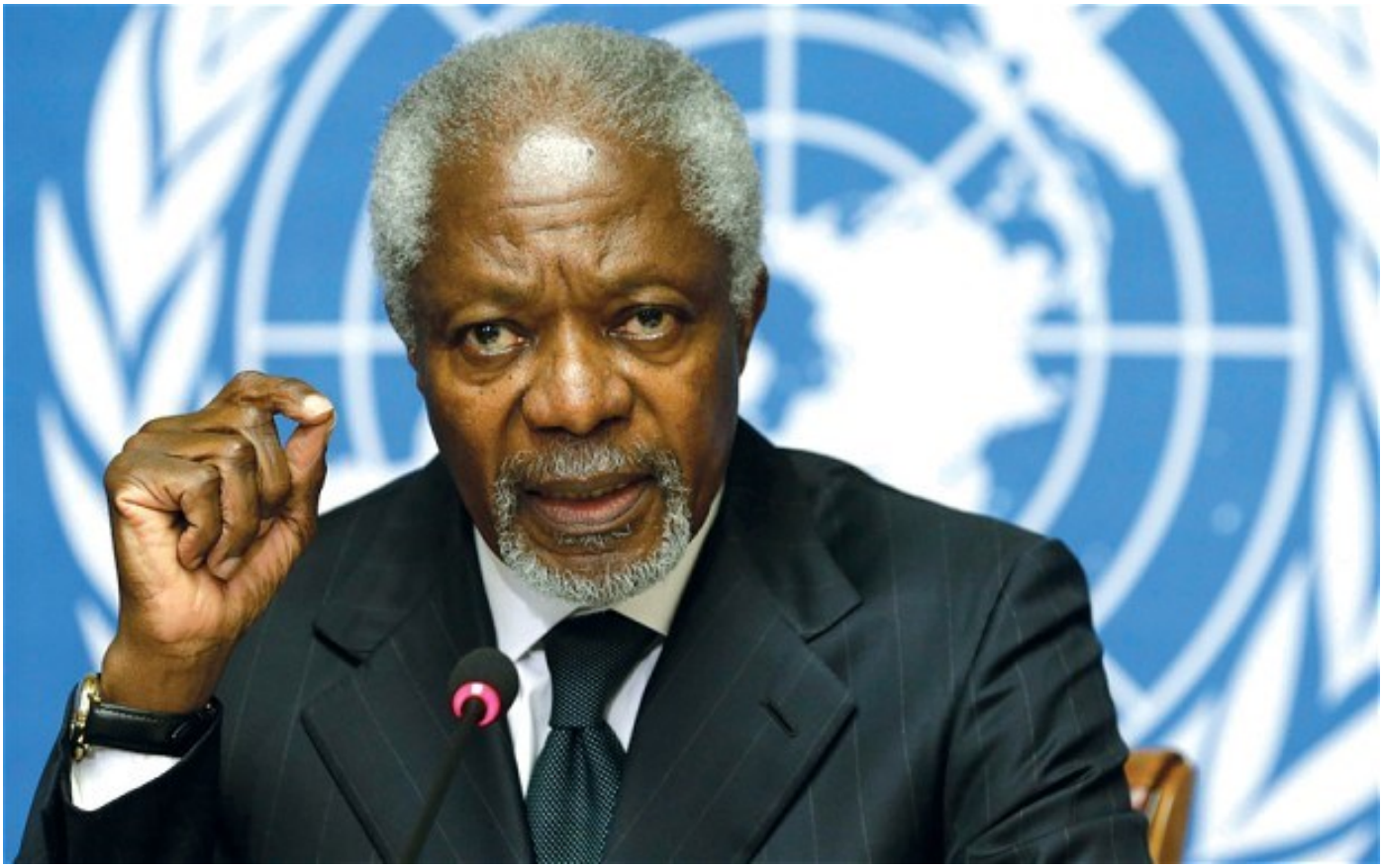
Autor(es): Anidey Machado Escudero.

Javier Borrego Asteasuainzarra.

Tutor: Ing. Yinett Hernández Hernández.

Co-Tutor: Ing. Pedro Frank Cadenas del Llano.

La Habana junio de 2014



“Las tecnologías de la información y la comunicación no son ninguna panacea ni fórmula mágica, pero pueden mejorar la vida de todos los habitantes del planeta. Se dispone de herramientas para llegar a los Objetivos de Desarrollo del Milenio, de instrumentos que harán avanzar la causa de la libertad y la democracia y de los medios necesarios para propagar los conocimientos y facilitar la comprensión mutua”

Kofi Annan.

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 _____.

_____ _____

Anidey Machado Escudero (Autor)

Javier Borrego Asteasuainzarra (Autor)

_____ _____

Ing. Yinett Hernández Hernández (Tutor)

Ing. Pedro Frank Cadenas del Llano (Co-Tutor)

Autores

Anidey Machado Escudero

Correo electrónico: aescudero@estudiantes.uci.cu

Javier Borrego Asteasuainzarra

Correo electrónico: jborregoa@estudiantes.uci.cu

Tutor: Yinett Hernández Hernández

Correo electrónico: yhhernandez@uci.cu

Co-Tutor: Pedro Frank Cadenas del Llano

Correo electrónico: pfcadenas@uci.cu

A mi mamá que ha sido la persona que me ha guiado en todo momento por el buen camino con su amor y cariño, y será siempre para mí un ejemplo de madre a seguir.

A mi abuela que ha sido otra madre para mí, y siempre me ha brindado su apoyo y sus consejos.

A mis tíos y tías, que han sido padres y madres para mí, siempre los he tenido al alcance de mi mano para lo que necesite, y creo que si hoy estoy aquí es en gran medida gracias a ellos.

A todos mis primos, los cuales considero mis hermanitos y hermanitas, los quiero mucho.

A toda mi familia de manera general que siempre me han ayudado en todo.

A la persona que me ha acompañado en estos últimos 4 años de mi vida, y me ha apoyado incondicionalmente en momentos buenos y malos, gracias mi amor por estar siempre al alcance mi mano.

A mi compañero de tesis, Javier gracias por brindarme tu apoyo y amistad en todo este tiempo de trabajo juntos.

A nuestra tutora Yinett y co-tutor Pedro Frank por toda su ayuda y el tiempo que nos dedicaron.

A los compañeros del proyecto por brindarnos su ayuda cuando lo necesitamos.

A todos mis compañeros de estudio y compañeras de apartamento en estos largos 5 años, donde compartimos momentos especiales, Yamilka, Aymée, Odeysi, Idelmis, Yorguy, Andrés, Lenier.

A todas mis amistades que no se encuentran en la universidad y me apoyaron cuando los necesité.

A todos los profesores que de alguna forma u otra han contribuido a mi formación como profesional.

Anidey Machado Escudero

A mis padres, que este resultado es el reflejo de la buena educación que me han dado, por estar conmigo en los momentos difíciles, por todo su amor y apoyo.

A mi abuela linda y majadera por todo su inmenso amor, cariño y peleítas.

A mis hermanos, en especial a mi hermano Pedri, por toda su dedicación y por ser más que un hermano.

A la futura madre de mis hijos Karla, por estar conmigo en los momentos tanto buenos como malos, por toda la confianza y apoyo que ha depositado en mí.

A gloria y a mis padrinos por todo su apoyo.

A los hermanos de vida: Orlandito, por sus consejos incluidos en cada chiste que me hacía, a Osmail, a Yoelvis, a Pimentel, a Félix, a Kenia, a Sarita, a Yasel, en fin, a todos mis compañeros y compañeras de aula y de apartamento, que los considero como mi familia, y a todos los amigos que han compartidos todos estos años conmigo.

A mi compañera de tesis Anidey por sus espaguetis, por sus cafecitos y por formar parte de este logro, muchísimas gracias.

A mis compañeros de proyecto, al profesor Maykel, Reinier y demás por todas las molestias que pude ocasionar.

A mi tutora Yinett por todo el tiempo que ha dedicado, por apoyarnos en cada momento aun cuando el tiempo y el trabajo se interponían.

A Pedro Frank, mi co-tutor por ser una persona indispensable, por haber estado en los momentos más difíciles por los que atravesamos.

A mis suegros y mi hermanito chiquito Robertico.

A gallego, por estar pendiente a mi formación como estudiante y por sus consejos.

A mis vecinos, tatatía, a Juan Manuel, Kenia y demás por su apoyo.

A todos mis santos.

A todos los que han contribuido a mi formación, Muchas Gracias.

Javier Borrego Asteasuainzarra

A mi mamá y a mi abuela en especial, y a mi familia de manera general por toda su ayuda.

A todas mis amistades y a las personas que han contribuido a mi formación como persona y profesional.

A Angel por brindarme su apoyo y amor incondicional.

Anidey Machado Escudero

A mis padres y a mi hermano Pedri, por todo el amor y confianza que me han dado, por su apoyo incondicional, por todo el sacrificio y dedicación durante estos largos años como estudiante.

A mi abuela por apoyarme y estar ahí siempre que he necesitado su apoyo.

A mis padrinos por contribuir a mi formación.

A mi novia Karla por todo el amor que me ha dado.

Javier Borrego Asteasuainzarra

Los Tribunales Populares Cubanos (TPC) atienden diferentes materias, estas materias abarcan varias funcionalidades que se realizan de forma similar, un ejemplo son los subprocesos Prueba Documental y de Libro. La gestión de las Pruebas Documental y de Libro se realizan de forma manual, lo que implica que los tiempos de realización de los trámites, así como los de búsqueda de información son muy extensos, además pueden aparecer errores humanos como la duplicación de información y el vencimiento de términos, el gasto de material de oficina es elevado y la pérdida de información por el almacenamiento en papel siempre está vigente. El presente trabajo de diploma propone una solución para informatizar los subprocesos Prueba Documental y de Libro y mejorar la gestión de la información en los TPC.

Para la construcción de la aplicación se realizó un estudio de la metodología que guiará el proceso de desarrollo de software, además de las diferentes tecnologías y herramientas utilizadas en la implementación de la solución. Se describe la arquitectura que da soporte al sistema, así como los diferentes artefactos generados en la fase de diseño. El diseño y la implementación propuesta fueron evaluados cuantitativamente a través de las métricas de diseño y las pruebas de caja blanca y caja negra. Con el desarrollo de los subprocesos Prueba Documental y de Libro los TPC obtendrán una herramienta que les permita mejorar la gestión de la información.

INTRODUCCIÓN 1
CAPÍTULO 1: Fundamentación teórica 4
1.1. Proceso de pruebas judiciales en Cuba 4
1.2. Las pruebas judiciales en los sistemas informáticos 5
1.3. Metodología de desarrollo de software 6
1.4. Lenguajes de programación 7
1.4.1. Hyper Text Markup Language (HTML) 8
1.4.2. Hypertext Preprocessor (PHP) 8
1.4.3. JavaScript 9
1.4.4. Twig 9
1.4.5. Cascade Style Sheets (CSS) 10
1.5. Arquitectura de software 10
1.5.1. Patrones arquitectónicos 10
1.5.1.1. Modelo Vista Controlador 10
1.5.1.2. Arquitectura en capas 11
1.5.2. Marcos de trabajo 12
1.5.2.1. JQuery 12
1.5.2.2. Symfony 2 12
1.5.2.3. Bootstrap 13
1.5.2.4. Doctrine 14
1.6. Herramientas de desarrollo 14
1.6.1. Entornos de desarrollo Integrado (por sus siglas en inglés IDE) 14
1.6.1.1. Netbeans 14
1.6.2. Sistema Gestor de Bases de Datos 15
1.6.2.1. PostgreSQL 15
1.6.3. Apache 16
1.6.4. Visual Paradigm 16
1.6.5. Subversion 17
1.7. Conclusiones parciales 17

CAPÍTULO 2: Solución propuesta 19
2.1. Arquitectura del sistema 19
2.2. Estándares de codificación 22
2.3. Patrones de diseño 28
2.3.1. Patrones GRASP utilizados en la solución del proyecto SITPC 28
2.3.1.1. Patrón Creador 28
2.3.1.2. Patrón Experto 29
2.3.1.3. Patrón Controlador 30
2.3.1.4. Bajo acoplamiento 31
2.3.1.5. Alta cohesión 31
2.3.2. Patrones GOF utilizados en la solución del proyecto SITPC 32
2.3.2.1. Decorador 32
2.3.2.2. Factory Method 32
2.3.3. Otros patrones 32
2.3.3.1. Inyección de dependencias 32
2.4. Modelo de diseño 33
2.4.1. Diagramas de secuencia 33
2.4.2. Diagramas de clases del diseño 34
2.5. Modelo de implementación 36
2.5.1. Diagrama de componentes 36
2.6. Diagrama de despliegue 37
2.7. Interfaces de la aplicación 38
2.8. Conclusiones parciales 40
CAPÍTULO 3: Análisis de resultados 42
3.1. Validación del diseño. Métricas de diseño. 42
3.1.1. Familia de métricas CK (propuestas por Chidamber y Kemerer) 42
3.1.1.1. Árbol de profundidad de herencia (APH) 42
3.1.1.2. Carencia de cohesión en los métodos (CCM) 43
3.1.2. Familia de métricas LK (propuestas por Lorenz y Kidd) 47

3.1.2.1. Tamaño de clase (TC) 47
3.1.2.2. Relaciones entre clases (RC) 50
3.2. Validación de la implementación 52
3.2.1. Pruebas de caja blanca 52
3.2.2. Pruebas de caja negra 54
3.3. Conclusiones parciales 59
CONCLUSIONES GENERALES 60
RECOMENDACIONES 61
BIBLIOGRAFÍA 62

INTRODUCCIÓN

En la actualidad, debido al desarrollo de las Tecnologías de la Información y las Comunicaciones (TICs) la sociedad es cada vez más dinámica y está sujeta a cambios frecuentes que involucran a todas las esferas de la vida. El sector jurídico a pesar de ser uno de los más reservados, no está ajeno a estas transformaciones, su informatización constituye un paso de avance en el mejoramiento de la calidad y la tramitación de los procesos. Dentro de las proyecciones estratégicas de este sector se encuentra que el sistema de tribunales continúe desarrollando una infraestructura tecnológica para asegurar su gestión, administración y explotación en los principios de la seguridad informática.

El sistema de tribunales en Cuba está compuesto por tres instancias, el Tribunal Supremo Popular, los Tribunales Provinciales Populares y los Tribunales Municipales Populares. Su objetivo principal es impartir justicia para contribuir a la seguridad jurídica y a la preservación y desarrollo de la sociedad socialista. En las diferentes instancias se resuelven asuntos de las materias: Civil, Administrativo, Laboral, Económico y Penal, siguiendo las leyes procesales vigentes. Todas las disposiciones se registran en autos, providencias o sentencias, resoluciones que deben ser notificadas a las partes, lo que genera un sinnúmero de documentación así como desgaste humano. Los reportes estadísticos se realizan manualmente y están sujetos a períodos fijos, dificultando su obtención dinámica.

Los TPC necesitan para su desarrollo y éxito organizacional la gestión de la información, que como enuncia (Capote Marrero, y otros, 2006) es el proceso que se encarga de suministrar los recursos necesarios para la toma de decisiones, así como para mejorar los procesos, productos y servicios de la organización. Tiene como objetivo principal optimizar recursos a través de un adecuado análisis de las necesidades de información para que la organización y/o sus usuarios puedan alcanzar sus metas.

Con el propósito de ganar en tiempo y recursos se concibió un acuerdo de colaboración entre el Tribunal Supremo Popular y la Universidad de las Ciencias Informáticas (UCI), específicamente el Centro de Gobierno Electrónico (CEGEL), para construir una solución informática para la tramitación electrónica de todos los procesos que se realizan en la institución, y proporcionar mayor agilidad, uniformidad y transparencia a la justicia. Surge así, el Proyecto de Informatización para la Gestión de los Tribunales Populares Cubanos (SITPC), que para facilitar su implementación se divide en los siguientes subsistemas: Administrativo, Laboral, Civil, Económico y Penal.

Luego de analizar detalladamente cada uno de los procesos que se llevan a cabo en los diferentes subsistemas, el equipo de trabajo identificó un conjunto de trámites que se realizaban de manera similar. Se decide implementar un nuevo subsistema que abarque estos trámites, y pudiera ser utilizado posteriormente por los demás, surge de esta forma Común. Uno de los procesos que se desarrollan en dicho subsistema es Pruebas, el cual está compuesto por los distintos tipos de pruebas llevadas a cabo en los tribunales, entre ellas las Pruebas Documental y de Libro, que permiten corroborar la veracidad en los hechos judiciales mediante documentos y libros.

En las Pruebas Documental y de Libro se captan desde la fase inicial de los procesos que se tramitan grandes cantidades de datos, los que deben ser repetidos en cada documento, pues se presenta un número elevado de documentación que servirá de prueba en los procesos judiciales, esto puede conllevar a que se cometan errores como la duplicación de información. Son muchas las personas implicadas en estos trámites: abogados, fiscales, jueces, incluyendo instituciones como el banco y las prisiones, y todas las notificaciones se realizan de forma personal. Los expedientes se forman en formato duro, los cuales con el paso del tiempo se deterioran y se corre el riesgo de perder la información que contienen. Asimismo existe un excesivo consumo de recursos de material de oficina. Producto de esta tramitación engorrosa de captar la información, almacenarla y notificar a las personas implicadas, puede ocurrir lentitud en la ejecución de los trámites que se realizan, y hasta manifestarse la violación en el cumplimiento de términos.

Ante la situación descrita se define el siguiente **problema de investigación**: ¿Cómo mejorar los subprocesos Prueba Documental y de Libro para contribuir a la gestión de la información en los Tribunales Populares Cubanos?

Se define como **objeto de estudio**: La gestión jurídica.

Para dar respuesta a la situación planteada se tiene como **objetivo general**: Informatizar los subprocesos Prueba Documental y de Libro para contribuir a la gestión de la información en los Tribunales Populares Cubanos.

Definiendo como **campo de acción**: Los subprocesos Prueba Documental y de Libro en los Tribunales Populares Cubanos.

Como **idea a defender** se plantea: Con la informatización de los subprocesos Prueba Documental y de Libro se contribuirá a la gestión de la información en los Tribunales Populares Cubanos.

Para dar cumplimiento al objetivo general se han derivado los siguientes **objetivos específicos:**

- Confeccionar el marco teórico de la investigación.
- Obtener el modelo de diseño.
- Implementar las funcionalidades.
- Validar los resultados obtenidos.

Tareas a desarrollar para dar cumplimiento a los objetivos específicos:

- Caracterizar el sistema de los TPC y el proceso de Pruebas, específicamente los subprocesos Prueba Documental y de Libro.
- Identificar diferentes sistemas informáticos existentes a nivel nacional e internacional que implementen el proceso de Pruebas.
- Caracterizar las metodologías de desarrollo de software, los lenguajes de desarrollo y las herramientas utilizadas.
- Valorar las plataformas de desarrollo libre y los paradigmas de programación.
- Seleccionar los patrones de diseño más factibles para la propuesta de solución.
- Obtener el modelo de diseño.
- Seleccionar y aplicar las métricas y pruebas de software para la validación del diseño.
- Implementar los CU.
- Obtener el diagrama de despliegue.
- Diseñar los casos de prueba.
- Probar los componentes y solucionar No conformidades.

CAPÍTULO 1: Fundamentación teórica

En el presente capítulo se puntualizan los elementos teóricos necesarios para el desarrollo de los procesos de la investigación. Se realiza el estudio de la metodología de software, los lenguajes de programación, los elementos de la arquitectura y las herramientas de desarrollo a utilizar con el fin de justificar su empleo, así como una investigación de las aplicaciones que existen en el mundo y su correspondencia con el tratamiento de pruebas judiciales en Cuba.

1.1. Proceso de pruebas judiciales en Cuba

Las pruebas judiciales según (Echandía, 1981) son el conjunto de reglas que regulan la admisión, producción, asunción y valoración de los diversos medios que pueden emplearse para llevar al juez la convicción sobre los hechos que interesan al proceso.

El proceso Pruebas se utiliza en los subsistemas Civil, Administrativo, Laboral y Económico de los TPC. Implementa tres actividades fundamentales: la presentación del escrito de proposición de pruebas; la disposición sobre el escrito, en la cual el juez podrá Admitir, mandar a Subsanar o Rechazar; y la práctica de pruebas. Posteriormente a estas tres actividades el proceso está listo para la sentencia. Está compuesto por diferentes tipos de pruebas, entre ellas Prueba Documental y de Libro. A continuación se describe el flujo de actividades de cada una de ellas.

El subproceso Prueba Documental comienza con la presentación del escrito y posteriormente el juez procede a disponer sobre él. Esta prueba no se practica, por lo que, admitido el documento se ganan 2 días para que las partes si lo desean puedan impugnar, en cuanto a ello la Ley de Procedimiento Civil Administrativo Laboral y Económico (LPCALE) en su ARTÍCULO 285 respalda que, la impugnación podrá hacerse por falta de legitimidad, de autenticidad o de exactitud de los documentos públicos y privados. También en su ARTÍCULO 286 defiende que admitida la impugnación por falta de autenticidad o exactitud de los documentos públicos, el Tribunal dispondrá su cotejo con los originales, y lo realizará, con citación de las partes, por sí mismo o por medio de Secretario. Si carecieren de originales o estos hubieran desaparecido, se les tendrá por eficaces salvo prueba en contrario. (Salup, 1996)

En el subproceso Prueba de Libro una vez presentado el escrito de proposición de pruebas se realiza un señalamiento, donde se cita con fecha y hora a la persona responsable del libro. Según el ARTÍCULO 291 enunciado en (Salup, 1996) para la práctica de la prueba de libro se constituirá el Secretario, asistido de las partes, en la oficina o lugar en que se hallen y se extenderá un acta en la que transcribirá literalmente

el contenido del asiento objeto de la prueba y hará constar los demás particulares que tengan relación con el mismo.

1.2. Las pruebas judiciales en los sistemas informáticos

Después de una detallada búsqueda de sistemas internacionales que implementaran dichos subprocesos, se obtuvo como resultado varios sistemas informáticos que incluyen la presentación de escritos y documentos judiciales, es decir, la presentación de la Prueba Documental.

Lexnet

El sistema Lexnet, desarrollado en España para la presentación de escritos y envío de notificaciones judiciales en el ámbito de la Administración de Justicia, es un medio de transmisión seguro de información, que mediante el uso de firma electrónica reconocida, en los términos establecidos en la Ley 59/2003, satisface las características de autenticación, integridad y no repudio. (Justicia, 2007)

Para la presentación de escritos y documentos en la aplicación informática, se requiere por parte de los usuarios del sistema según (Justicia, 2007) la previa cumplimentación de todos los campos de datos obligatorios. Además el usuario puede incorporar, conjuntamente con el documento electrónico anexo, en el que se contenga el propio acto procesal objeto de transmisión, otros anexos, y se le permite visualizar los documentos electrónicos incorporados, a efectos de comprobación, antes de proceder a su envío.

Electronic Filing System (EFS)

EFS es un componente del sistema informático LawNet, fue implementado como parte de la estrategia de manejo de casos de la Corte Suprema de Singapur, y consiste en una red nacional de computación diseñada para facilitar una conducción expedita y efectiva de la litigación civil. El sistema provee varios servicios entre los cuales está la posibilidad de que los litigantes ingresen escritos judiciales a través de un sistema web durante las 24 horas del día. Además ofrece el Centro de Servicios Electrónicos de Documentos, el cual permite a los abogados enviar copia de los documentos presentados a la Corte de forma automática a sus contrapartes mediante correo electrónico. (Becerra, 2008)

H@bilus

H@bilus es un sistema informático desarrollado en Portugal, permite que todas las notificaciones se hagan de forma electrónica y que los operadores cuenten con una zona dedicada a las notificaciones

recibidas en sus procesos con alertas automáticas. El sistema brinda la facilidad de realizar las pruebas documentales con los documentos electrónicos contenidos en su base de datos o aquellos presentados por los abogados. (Becerra, 2008)

Estos sistemas a pesar de permitirle al usuario la presentación de la Prueba Documental, no cumplen con las expectativas como solución informática que pretende el proyecto Tribunales, pues requieren ciertas condiciones tecnológicas las cuales el país no está capacitado para ofrecer, como es el uso de la firma digital de los documentos. Además no se rigen por las mismas leyes que los Órganos de Justicia de Cuba.

Se desarrollará una aplicación para dar solución al objetivo planteado, a continuación se define la metodología de software que guiará el proceso de desarrollo de dicha aplicación.

1.3. Metodología de desarrollo de software

El equipo de desarrollo de software necesita una forma coordinada de trabajar, un proceso que integre las múltiples facetas de desarrollo, un método común que:

- Proporcione una guía para ordenar las actividades de un equipo.
- Dirija las tareas de cada desarrollador por separado y del equipo como un todo.
- Especifique los artefactos que deben desarrollarse.
- Ofrezca criterios para el control, la medición y actividades del producto. (Jacobson, y otros, 2000)

Se hace necesario seleccionar una metodología que se ajuste con las características del proyecto, pues ella es quien se encarga de brindar estas ventajas. El SITPC presenta un equipo de desarrollo numeroso, donde la mayoría no tienen alta experiencia en el dominio de aplicación, además es necesario desarrollar un producto de gran complejidad, duración y alta criticidad. Por lo que es precisa una metodología que brinde una planificación total del trabajo a realizar.

Las metodologías ágiles están basadas en heurísticas provenientes de prácticas de producción de software. Son aplicadas principalmente en proyectos preparados a los cambios, siendo un proceso poco controlado y con escasos principios. El cliente es parte del equipo de desarrollo, el cual cuenta con un número pequeño de integrantes, los cuales generalmente trabajan en el mismo sitio. Es una metodología que genera pocos artefactos y roles, y hace menos énfasis en la arquitectura de software.

Las metodologías tradicionales están basadas en normas provenientes de estándares seguidos por el entorno de desarrollo. Presenta resistencia al cambio, siendo un proceso muy controlado y con muchas

políticas y normas. El cliente interactúa con el equipo de desarrollo mediante reuniones, el cual está compuesto por grupos grandes de personas y posiblemente distribuidos por varios puestos de trabajo. Es una metodología que genera gran cantidad de artefactos, documentación y roles. La arquitectura de software es esencial en su desarrollo, ya que es una de sus tres características fundamentales.

De acuerdo con las características del equipo de desarrollo la mejor opción sería el uso de las metodologías tradicionales, la cuales se centran especialmente en el control del proceso, mediante una rigurosa definición de roles, actividades, artefactos, herramientas, notaciones para el modelado y documentación detallada.

En el SITPC se definió como metodología Rational Unified Process (RUP), metodología tradicional que brinda una necesaria documentación para todos los miembros del equipo de desarrollo, además de que se adapta fácilmente a proyectos grandes y de larga duración como es el caso. Es válido destacar que el SITPC no utiliza en su totalidad todos los artefactos que genera RUP, pues se rige por las normas que se le exigen en el Expediente de Proyecto según el Programa de Mejoras establecido por el Centro Nacional de Calidad de Software (CALISOFT).

El proceso propuesto por RUP tiene tres características esenciales: dirigido por CU, centrado en la arquitectura e iterativo e incremental. Según expresa (Jacobson, y otros, 2000) un CU es una técnica de captura de requisitos y representa los requisitos funcionales del sistema, en RUP además, es la guía para el diseño, la implementación y pruebas. Otro aspecto fundamental es el establecimiento temprano de una arquitectura robusta que no se vea fuertemente impactada ante cambios posteriores durante la construcción y el mantenimiento, y que permita una estrecha interacción con los CU para que todos puedan ser desarrollados. Por otra parte, el trabajo se divide en partes más pequeñas o mini proyectos, permitiendo que el equilibrio entre los CU y la arquitectura se vaya logrando en cada mini proyecto durante todo el proceso de desarrollo.

Estas tres características son significativamente importantes para el desarrollo de la aplicación, pues las tres actúan vinculadas para lograr la eficiencia del producto. Una vez definida la metodología de software es necesaria la selección de los diferentes lenguajes de programación que se utilizarán en el desarrollo.

1.4. Lenguajes de programación

Los lenguajes de programación permiten describir el conjunto de acciones que se necesita que el ordenador ejecute, por lo que son imprescindibles para la creación de aplicaciones.

1.4.1. Hyper Text Markup Language (HTML)

HTML es un lenguaje para escritura de hipertexto, es decir, documentos de texto estructurados, incluye enlaces que conducen a otros documentos o a otras fuentes de información, y permite la inclusión de información por formularios, entre otros.
HTML en su quinta versión, pretende remplazar al actual Lenguaje de Marcado de Hipertexto Extensible (XHTML), corrigiendo problemas actuales, así como rediseñar el código actualizándolo a nuevas necesidades que demanda la web. Se caracteriza por ofrecer:

- Una mejor estructura: Elimina el uso excesivo de las etiquetas div, con el objetivo de que la web sea más coherente y comprensible.
- Mejoras en los formularios: El elemento input adquiere gran relevancia al ampliarse los elementos que se permitirán en el "type".
- Otros Elementos: Nuevos elementos que permitirán incrustar un contenido multimedia de sonido o de vídeo, respectivamente. (Cantón, 2006)

Para la generación de las vistas del SITPC junto con las plantillas Twig también se utiliza HTML5, debido a la incorporación de las novedades antes mencionadas que proporcionan beneficios en el desarrollo de la aplicación y que van a ser aplicadas en el sistema informático que se desea, además que es totalmente compatible con todos los navegadores, siendo esta característica fundamental debido al alcance que se aspira que tenga el sistema.

1.4.2. Hypertext Preprocessor (PHP)

PHP es un lenguaje de código abierto, especialmente adecuado para desarrollo web y que puede ser incrustado en HTML. Se distingue por ejecutar el código en el servidor, generando HTML y enviándolo al cliente. El cliente recibirá los resultados de ejecutar el script, sin ninguna posibilidad de determinar qué código ha producido el resultado recibido. Permite procesar la información de formularios, generar páginas con contenidos dinámicos, o enviar y recibir cookies. Además se caracteriza por su soporte para una gran cantidad de bases de datos y puede comunicarse con otros servicios usando protocolos tales como Protocolo Ligero de Acceso a Directorios (por sus siglas en inglés LDAP). (Achour, y otros, 2013)
Por tanto el lenguaje PHP en su versión 5.3 constituye uno de los lenguajes idóneos para la creación del sistema informático que se aspira, debido a que está completamente orientado al desarrollo de

aplicaciones web dinámicas con acceso a información almacenada en una base de datos. Permite el uso de técnicas de programación orientada a objetos, ganando de esta manera en organización y limpieza en el código. Es multiplataforma y hace que la programación en PHP sea segura y confiable, pues el código fuente escrito en este lenguaje es invisible al navegador y al cliente.

1.4.3. JavaScript

JavaScript es un lenguaje de programación interpretado, por lo que no es necesario compilar los programas para ejecutarlos. En otras palabras, los programas escritos con JavaScript se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios. Se utiliza principalmente para crear páginas web dinámicas. (Eguíluz, 2007)

Se utiliza en su forma del lado del cliente, implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas, aunque existe una forma de JavaScript del lado del servidor.

Por lo que se concluye que este lenguaje de programación interpretado es fundamental en el desarrollo del sistema, debido a que ofrece mejoras significativas en las interfaces de usuario, pues es eficiente para la realización de validaciones y en el desarrollo de web dinámicas. Además de ser un lenguaje orientado a eventos que cuando un usuario realiza alguna acción en un determinado elemento HTML, se pueden producir diferentes eventos y a través de él se pueden desarrollar scripts que ejecuten funciones en respuesta a estos eventos. Favorece al rendimiento de la aplicación a la que se aspira puesto que al ser un lenguaje interpretado no necesita ser compilado.

1.4.4. Twig

Twig es un flexible, rápido y seguro motor de plantillas para PHP. Según (Pacheco, 2011) se caracteriza por ser:

- Rápido: Compila las plantillas hasta código PHP regular optimizado. El costo general en comparación con código PHP regular se ha reducido al mínimo.
- Seguro: Tiene un modo de recinto de seguridad para evaluar el código de plantilla que no es confiable. Esto te permite utilizar Twig como un lenguaje de plantillas para aplicaciones donde los usuarios pueden modificar el diseño de la plantilla.
- Flexible: Es alimentado por flexibles analizadores léxico y sintáctico. Esto permite al desarrollador definir sus propias etiquetas y filtros personalizados.

Para la generación de las vistas del SITPC, se decidió el uso Twig en su versión 1.0, puesto que al emplear el Symfony 2, se recomienda su utilización para la creación de las plantillas de la aplicación y por las grandes potencialidades que ofrece, tal como la herencia entre plantillas, que permite crear una estructura base que contenga todos los elementos comunes del sitio y define los bloques que las plantillas descendientes pueden sustituir, garantizando así organización en el código y facilidad de mantenimiento, que al ser un proyecto grande este proceso puede resultar engorroso.

1.4.5. Cascade Style Sheets (CSS)

CSS es un lenguaje de hojas de estilos creado para controlar la presentación de los documentos electrónicos definidos con HTML y XHTML. Es la mejor forma de separar los contenidos y su presentación, y es imprescindible para la creación de páginas web complejas. El uso de CSS mejora la accesibilidad del documento, reduce la complejidad de su mantenimiento y permite visualizar el mismo documento en infinidad de dispositivos diferentes.

CSS3 incorpora nuevos mecanismos para mantener un mayor control sobre el estilo con el que se muestran los elementos de las páginas. Añade un grupo de propiedades a los bordes, fondos, color, texto, interfaz, selectores, modelo de caja básico, entre otros.

Por tanto este lenguaje de hojas de estilo es esencial en el desarrollo del SITPC, debido a las potencialidades que brinda en el control centralizado de la presentación, además permite lograr estilos y efectos visuales sin necesidad de tecnologías adicionales, ofrece nuevos estilos que proporciona un entorno amigable y contribuyendo así de forma significativa en el diseño de la aplicación.

1.5. Arquitectura de software

La Arquitectura del Software tiene como objetivo determinar el diseño de la estructura de la aplicación, permite entender la formación del sistema, así como organizar su desarrollo y el cumplimiento de sus funcionalidades.

1.5.1. Patrones arquitectónicos

Los patrones arquitectónicos son imprescindibles para la construcción y diseño de un sistema de software, pues ofrecen la organización estructural de la aplicación. A continuación se listan los patrones utilizados en la solución.

1.5.1.1. Modelo Vista Controlador

El patrón arquitectónico Modelo Vista Controlador (MVC) divide una aplicación interactiva en tres componentes: el modelo, las vistas y los controladores. Estos dos últimos componentes, en conjunto, forman la interfaz del usuario, asegurando la consistencia entre esta y el modelo.

El modelo encapsula los datos centrales y tiene la funcionalidad de la aplicación. Es un componente totalmente independiente de las representaciones específicas de salidas o del comportamiento de la entrada.

Los controladores reciben la entrada, normalmente como eventos que codifican los movimientos del mouse o entrada del teclado. Los eventos son traducidos para servir a las demandas del modelo o las vistas. El usuario interactúa con el sistema solamente a través de las vistas de la aplicación.

Diferentes vistas presentan la información del modelo al usuario de distintas maneras. Pueden existir múltiples vistas de un mismo controlador, pero cada vista tiene una relación uno a uno con un controlador. Cada vista define un procedimiento de actualización que se activa por el mecanismo de propagación de cambios. Cuando es llamado el procedimiento de actualización, una vista recupera los valores de datos actuales del modelo para ser mostrados, y los pone en la pantalla. (Sandra Almeira, y otros, 2007)
Dicho patrón permite dividir la lógica de negocio del diseño, haciendo el proyecto más escalable, además de permitir la reutilización de los componentes, simplicidad en el mantenimiento de los sistemas y facilidad para desarrollar prototipos rápidos.

1.5.1.2. Arquitectura en capas

En el estilo en capas como organización jerárquica, cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior. Las capas suelen ser entidades complejas, compuestas de varios paquetes o subsistemas. (Shaw, 1994)
El uso de arquitecturas en capas, explícitas o implícitas, es frecuente en la actualidad y entre las principales ventajas que brinda se encuentran:

- Soporta un diseño basado en niveles de abstracción crecientes, lo cual a su vez permite a los implementadores la partición de un problema complejo en una secuencia de pasos incrementales.
- Admite naturalmente optimizaciones y refinamientos.
- Proporciona amplia reutilización. Al igual que los tipos de datos abstractos, se pueden utilizar diferentes implementaciones o versiones de una misma capa en la medida que soporten las

mismas interfaces de cara a las capas adyacentes. Esto conduce a la posibilidad de definir interfaces de capa estándar, a partir de las cuales se pueden construir extensiones o prestaciones específicas. (Ávila, 2010)

1.5.2. Marcos de trabajo

Un marco de trabajo es un conjunto de herramientas, librerías, convenciones y buenas prácticas que pretenden encapsular las tareas repetitivas en módulos genéricos fácilmente reutilizables. (Pérez, 2009)

1.5.2.1. JQuery

JQuery es una biblioteca JavaScript rápida, pequeña y rica en funciones. Hace manipulación y recorrido de documentos HTML, control de eventos, efectos y animaciones personalizadas, selección de elementos del Modelo de Objetos del Dominio (por sus siglas en inglés DOM) y manipulación de la hoja de estilos CSS, con un Application Programming Interface (API) que funciona a través de muchos navegadores. Brinda varias utilidades como obtener información del navegador, operar con vectores y funciones para rutinas comunes. (Foundation, 2013)

Por tanto el uso de jQuery en su versión 1.8 como librería Javascript, es una aceptada selección pues ofrece una infraestructura que facilita la creación de aplicaciones complejas del lado del cliente. Ayuda en la creación de interfaces de usuario, efectos dinámicos y en el uso de Ajax[1], permite la abstracción casi total del uso de este recurso y al pretenderse que el proyecto SITPC posea un gran alcance con un gran número de usuarios, contribuye a la abstracción entre las diferencias entre navegadores. Por otra parte garantiza la creación de todo tipo de funcionalidades en el lado cliente de manera sencilla, y favorece en el rendimiento por el manejo rápido de propiedades y CSS.

1.5.2.2. Symfony 2

Symfony es un completo marco de trabajo diseñado para optimizar, gracias a sus características, el desarrollo de las aplicaciones web; separa la lógica del negocio, del servidor y la presentación de la aplicación web. Es una herramienta que proporciona varias clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja.

[1]Las funciones y métodos que están dentro nos permiten cargar datos del servidor sin tener que actualizar la página en el navegad or.

Symfony 2 es una nueva versión de Symfony que se caracteriza por ser:

- Útil: Permite al desarrollador abstraerse de los retos de la programación, tales como la seguridad, rendimiento, validaciones, entre otros, y de esta manera reducir el tiempo de desarrollo.
- Flexible: Permite al desarrollador seleccionar los archivos de configuración que desee (Lenguaje de Marcado Ligero (por sus siglas en inglés YAML), Lenguaje de Marcas Extensible (por sus siglas en inglés XML), PHP), al igual que las plantillas (Twig y PHP), y almacenamiento (Lenguaje de Consulta Estructurado (por sus siglas en inglés SQL) y NoSQL[2]).
- Rendimiento: Exige el uso de PHP 5.3, todo se compila a PHP, archivos de configuración, anotaciones del código, plantillas Twig, todo en general. Utiliza un *reverse Proxy* que permite que no se hagan consultas anteriormente hechas a la base de datos en un determinado período de tiempo.
- Documentación: Existe bibliografía fiable y concreta que facilita el uso y aprendizaje del marco de trabajo. (Eguíluz, 2011)

Por las características antes expuestas se puede concluir que Symfony en su versión 2.1 se ajusta a la solución informática propuesta, fundamentalmente por su arquitectura basada en MVC y por la inclusión de componentes, además facilita la creación del sistema que se pretende debido a que favorece en gran medida en el rendimiento, característica que fue abordada anteriormente y reduce el tiempo de programación, por el uso del paradigma de programación orientada a objetos.

1.5.2.3. Bootstrap

Bootstrap es una librería CSS o herramienta para el desarrollo de aplicaciones y sitios web. Fomenta las buenas prácticas de diseño y desarrollo web, conforme a estándares W3C[3]. Permite diseñar webs adaptables y fluidas, visualizables correctamente en múltiples dispositivos. Permite aprovechar los beneficios de HTML5, CSS3 y Javascript, también incluye elementos de diseño, tipografías, tablas, formularios, navegación, alertas, etc.

El uso de Bootstrap en su versión 2.1 como librería CSS en el proyecto Tribunales permite la integración del marco de trabajo de forma sencilla, además de ofrecer una elegante tipografía, formas, botones,

[2] Sistemas de gestión de bases de datos que no utilizan SQL como principal lenguaje de consultas.
[3] World Wide Web Consortium desarrolla especificaciones técnicas y directrices a través de un proceso que ha sido diseñado para maximizar el consenso sobre el contenido de un informe técnico, de forma que se pueda asegurar la alta calidad técnica y editorial, así como obtener un mayor apoyo desde el W3C y desde la comunidad en general.

cuadros y navegación, permite abstraerse de la definición de estilos, contribuyendo de esta manera a la reducción del tiempo de desarrollo.

1.5.2.4. Doctrine

Doctrine es un asignador objeto-relacional (por sus siglas en inglés ORM) para PHP 5.3.0. Se sitúa en la parte superior de una poderosa capa de abstracción de base de datos. La principal tarea de los asignadores objeto-relacionales es la traducción transparente entre objetos PHP y las filas relacionales de la base de datos.

Doctrine posee la habilidad de escribir consultas a la base de datos a partir de la programación orientada a objetos, llamada Doctrine Query Language (DQL). (Team, 2011)

Por las características propias de Doctrine tales como el bajo nivel de configuración que se necesita para comenzar un proyecto, la generación de clases a partir de una base de datos creada y en cuanto a las facilidades que le brinda al programador para especificar relaciones y agregar funcionalidades comunes para las clases generadas, es apropiado para el desarrollo de la solución informática que se pretende desarrollar. Se utiliza la versión 2.0.

Una vez definida la arquitectura que será utilizada son necesarias las herramientas de desarrollo que permitirán la construcción del software.

1.6. Herramientas de desarrollo

Las herramientas de desarrollo incluyen aquellos programas o aplicaciones que van a permitirle al programador la construcción del sistema, a continuación se describen algunas de las que serán utilizadas.

1.6.1. Entornos de desarrollo Integrado (por sus siglas en inglés IDE)

Un IDE es un entorno de programación que le proporciona al desarrollador un conjunto de herramientas y que contiene un editor de código, un compilador, un depurador, un constructor de interfaz gráfica, que van a facilitar las tareas de desarrollo y mantenimiento de las aplicaciones. Entre los entornos de desarrollo integrados que permiten trabajar con PHP como lenguaje de programación se encuentran: Geany, CodeRun, Eclipse y NetBeans. Por parte del equipo de arquitectura del proyecto se decidió utilizar para el desarrollo del SITPC este último en su versión 7.4.

1.6.1.1. Netbeans

NetBeans IDE está escrito en Java, pero puede servir para cualquier otro lenguaje de programación. Existe además un número importante de módulos para extenderlo. Es un producto de código abierto y gratuito para uso tanto comercial como no comercial. El código fuente está disponible para su reutilización de acuerdo con la Common Development and Distribution License (CDDL) v1.0 and the GNU General Public License (GPL) v2. (Team, 2013)

Es un IDE que provee un conjunto de herramientas que permiten crear aplicaciones profesionales. Consta de una comunidad en constante crecimiento, lo que le ha favorecido, al igual que muchos otros sistemas libres, el progreso paulatino de sus prestaciones y la eliminación de errores de programación que pudiesen existir. Por otra parte la plataforma NetBeans permite que las aplicaciones se desarrollen a partir de un conjunto de módulos o componentes de software, es fácil de instalar y de uso instantáneo y es multiplataforma.

1.6.2. Sistema Gestor de Bases de Datos

Los Sistemas Gestores de Bases de Datos (SGBD) son un conjunto de programas cuya función principal es permitir el almacenamiento, la modificación y extracción de datos en una Base de Datos (BD), proporcionando un acceso controlado a la misma. Entre sus servicios se encuentran:

- Definición y creación de la BD.
- Manipulación de los datos realizando consultas, inserciones y actualizaciones.
- Acceso controlado a los datos mediante mecanismos de seguridad de acceso a los usuarios.
- Mantener la integridad de los datos.
- Controlar la concurrencia a la BD.
- Mecanismos de copias de respaldo y recuperación para restablecer la información en caso de fallos en el Sistema. (Ramírez, 2008)

Por parte del equipo del proyecto se decidió utilizar para el desarrollo del SITPC PostgreSQL en su versión 9.2.

1.6.2.1. PostgreSQL

PostgreSQL es un SGBD objeto-relacional con su código fuente disponible libremente. Es el SGBD de código abierto más utilizado en el ámbito del desarrollo de software.

Según (Team, 2009-2013) dentro de las características que identifican al PostgreSQL se encuentran:

- Es una base de datos 100% ACID (Atomicity, Consistency, Isolation and Durability: Atomicidad, Consistencia, Aislamiento y Durabilidad).
- Es altamente escalable, tanto en la cantidad de datos como en el número de usuarios concurrentes que puede atender, realizando esto de forma más eficiente que muchos otros gestores.
- Implementa el uso de subconsultas y transacciones, haciendo su funcionamiento mucho más eficaz, ofreciendo soluciones en disimiles campos.
- Múltiples métodos de autentificación.
- Acceso encriptado a través de la vía Capa de Conexión Segura (por sus siglas en inglés SSL).
- Disponible para Linux y UNIX en todas sus variantes y Windows 32/64bit.

Con la implementación del proyecto SITPC se pretende manipular un alto volumen de información y una alta concurrencia de usuarios, por tanto es necesario disponer de un SGBD robusto y potente, por lo que el uso de PostgreSQL resulta ser muy necesario en el desarrollo del mismo. Además que es un sistema multiplataforma y de código abierto.

1.6.3. Apache

El Servidor Apache es un servidor web Hypertext Transfer Protocol (HTTP) de código abierto para plataformas Unix (Berkeley Software Distribution o BSD, GNU/Linux, etc.), Microsoft Windows, Macintosh y otras, que implementa el protocolo HTTP/1.1. Este servidor web es muy potente, gratuito y flexible debido a que es altamente configurable y de diseño modular, lo que posibilita la selección de los módulos del proyecto a ser incluidos y ejecutados en el servidor, evitando la sobrecarga del mismo con módulos innecesarios en el momento de ejecutar alguna acción. Apache es el servidor web de millones de servidores en el mundo debido a su robustez y estabilidad, y es el que brinda servicio al equipo de desarrollo de SITPC en su versión 2.2.

1.6.4. Visual Paradigm

Esta herramienta ha sido concebida para soportar el ciclo de vida completo del proceso de desarrollo del software (análisis y diseño orientados a objetos, construcción, pruebas y despliegue) a través de la representación de todo tipo de diagramas. Constituye una herramienta de probada utilidad para el analista. Fue diseñado para una amplia gama de usuarios interesados en la construcción de sistemas de software de forma fiable a través de la utilización de un enfoque orientado a objetos.

Esta herramienta presenta algunas características como:

- Disponibilidad en múltiples plataformas.
- Diseño centrado en CU y enfocado al negocio que generan un software de mayor calidad.
- Capacidades de ingeniería directa e inversa.
- Disponibilidad de múltiples versiones, con diferentes especificaciones.
- Diagramas de flujo de datos.
- Diagramas de Procesos de Negocio - Proceso, Decisión, Actor de negocio, Documento. (Paradigm, 2011)

Visual Paradigm 8.0 fue la herramienta seleccionada para el modelado de los procesos de negocios, ya que brinda la posibilidad de realizar todos los diagramas de la fase de elaboración propuesta por la metodología de desarrollo, además es multiplataforma.

1.6.5. Subversion

Subversion es una herramienta de código abierto para el control de versiones. Está basada en un repositorio[4], cuyo funcionamiento se asemeja enormemente al de un sistema de ficheros.

En el proyecto este repositorio es accedido solo por las personas que poseen permisos, además solo acceden a la información que se corresponde con el rol que desempeña.

1.7. Conclusiones parciales

Según lo expuesto en el capítulo se arriba a las siguientes conclusiones:

- Los sistemas informáticos analizados incluyen el desarrollo de la Prueba Documental pero no se corresponden con las condiciones del proyecto.
- La metodología de software que guiará el proceso de desarrollo es RUP, dada la complejidad del negocio, el equipo de desarrollo es amplio y el tiempo de duración prolongado.
- Se realizó una aplicación web para facilitar el despliegue, la actualización del software y la comunicación entre las diferentes instancias de los tribunales.

[4] Depósito donde se almacena la información digital.

- Se utilizó el marco de trabajo jQuery para las validaciones del lado del cliente, Symfony 2 para el desarrollo de la aplicación, Bootstrap para el diseño de las vistas y Doctrine para el mapeo de la base de datos.
- Para el desarrollo de la aplicación se utilizaron los lenguajes de programación PHP del lado del servidor, JavaScript del lado del cliente permitiendo realizar las validaciones, Twig y HTML 5 para lograr una mayor organización del código y CSS 3 manteniendo el control centralizado de la presentación.
- Se definió como IDE de desarrollo Netbeans, SGBD PostgreSQL y servidor web Apache.

CAPÍTULO 2: Solución propuesta

En el presente capítulo se puntualizan los elementos relacionados con el diseño e implementación de las Prueba Documental y de Libro. Se expone cómo fueron utilizados los diferentes patrones de diseño en el proceso de desarrollo. Además se muestran los diagramas que forman parte del modelo de diseño, los diagramas de secuencia y los diagramas de clases del diseño. Asimismo se presenta el diagrama de despliegue y el diagrama de componentes.

2.1. Arquitectura del sistema

Apoyados en los principios y arquitectura que propone el marco de trabajo Symfony 2, para el desarrollo del SITPC se define una arquitectura en capas basada en el patrón MVC, debido a las facilidades que ofrece tanto en términos de escalabilidad, pues divide la lógica del negocio de la lógica del diseño como en la facilidad de mantenimiento, además de mantener un bajo acoplamiento. Este patrón se muestra de forma implícita en el uso del marco de trabajo Symfony 2, definiendo este último dónde se ubican las clases del modelo, las de la vista y las del controlador.

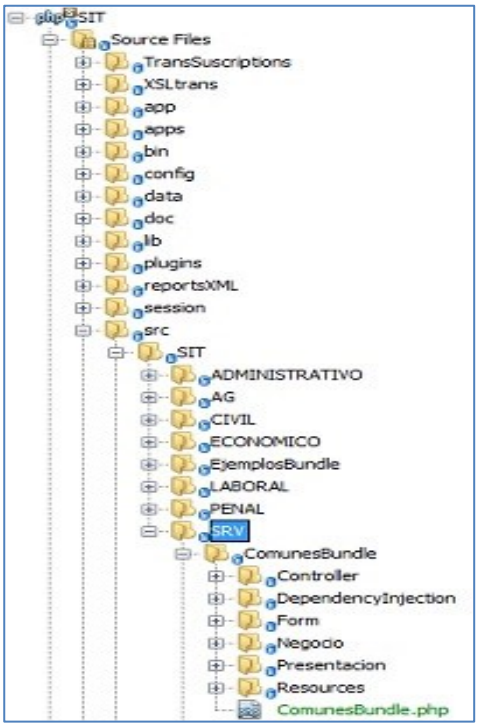


Figura 1: Organización propuesta por Symfony 2

A continuación se muestra la arquitectura propuesta por SITPC.

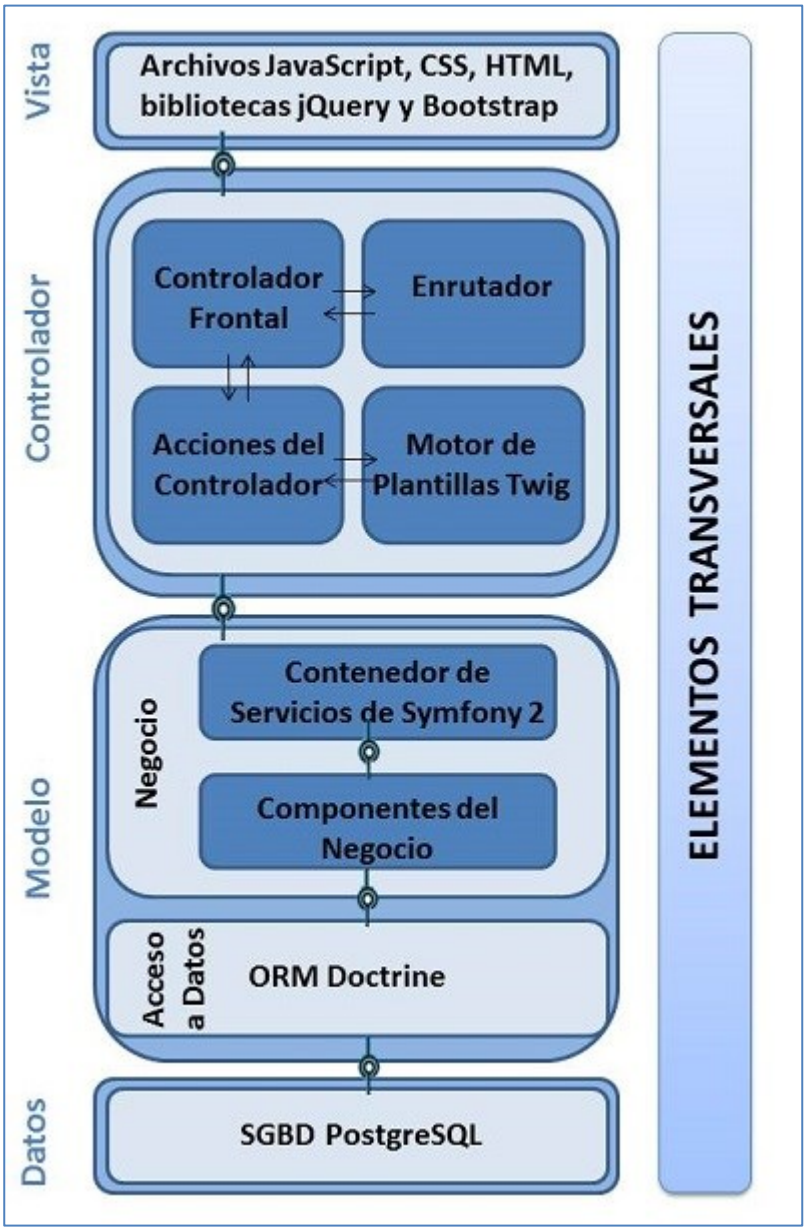


Figura 2: Arquitectura del SITPC

Vista

La vista es la capa con la que interactúan directamente los usuarios finales, dentro de ella se ubica todo lo referente a la visualización de la información en las páginas de la aplicación.

En Symfony 2 se evidencia esta capa a través de la biblioteca jQuery, en conjunto con los archivos CSS, JavaScript y HTML, siendo los encargados de crear las páginas de la aplicación. Dichos archivos se encuentran ubicados en el directorio /Resources/views dentro de cada bundle de la aplicación y específicamente en “SIT/src/SIT/SRV/ComunesBundle/Resources/views”.

Controlador

El controlador es la capa intermedia entre la vista y el modelo. Es la encargada de responder a las acciones del usuario e invocar cambios en el modelo o generar la vista apropiada, dependiendo de las peticiones del usuario.

En Symfony 2 como marco de trabajo para el desarrollo de la solución, queda evidenciada esta capa a través del Controlador Frontal, mediante los archivos app.php para entornos de producción y app_dev.php para entornos de desarrollo, los cuales se conciben para la ejecución de la aplicación en el servidor de producción y para el uso de los programadores respectivamente, pues les provee de información útil en todas las páginas (sobre todo, las páginas de error). Los archivos de enrutamiento también forman parte de esta capa intermedia los cuales permite determinar qué controlador está asociado con la página que se solicita. También forma parte de esta capa el motor de plantillas Twig, responsable de transformar cada archivo Twig en HTML. Por último las clases Controller, que controlarán el flujo de información que se recibe y se envía hacia la vista, y estarán ubicadas dentro del paquete “SIT/src/SIT/SRV/ComunesBundle/Controller” junto a las demás clases que se localizan en el directorio /Controller/ de cada bundle en el SITPC.

Modelo

En el SITPC esta capa está dividida en 2, la capa de negocio y la capa de acceso a datos.

Capa de Negocio

En la capa de negocio se localiza los componentes en los que se encuentran las entidades y entidades de presentación. Las entidades son la representación de las tablas de la base de datos previamente mapeadas por el ORM Doctrine, y las entidades de presentación una combinación entre dos o más entidades. En el proyecto las primeras se encuentran en “SIT/vendor/Base/ComunBundle/Entity”, mientras que las otras están ubicadas en “SIT/src/SIT/SRV/ComunesBundle/Presentacion”, y también se pueden encontrar dentro del directorio /Presentacion/ de cada bundle del SITPC. Forman parte de esta capa

también las clases gestoras, las cuales reciben instrucciones e información a través de las clases controladoras, y se localizan en "SIT/src/SIT/SRV/ComunesBundle/Negocio/Gestor", además en los directorios /Gestor/ de cada bundle del SITPC, siendo las responsable del manejo de la lógica del negocio.

Capa de acceso a datos

La capa de acceso a datos es la encargada de establecer la conexión con la base de datos. Dentro de esta capa se encuentra ubicado el ORM Doctrine, que posibilita la separación en la aplicación del SGBD mediante su lenguaje propio de consultas DQL, y las clases repositorios. En estas últimas clases se encuentran las consultas más complejas que se realizan a la base de datos, y se localizan dentro del directorio vendor, donde se encuentran las dependencias de terceros en Symfony 2, específicamente en "SIT/vendor/Base/ComunBundle/Repository".

Elementos transversales

Los elementos transversales son un conjunto de operaciones que se ejecutan de forma transversal, ya que se pueden ejecutar en cualquiera de las capas de la arquitectura. Entre ellos se encuentra el contenedor de servicios, la caché y las configuraciones de Symfony 2, así como el tratamiento de excepciones, las validaciones y la seguridad del sistema.

2.2. Estándares de codificación

Un estándar de codificación comprende todas las reglas para la generación de código. Un código fuente completo debe reflejar un estilo armonioso, como si un único programador hubiera escrito todo el código de una sola vez.

Entre los estándares de codificación definidos por el equipo de arquitectura de SITPC se encuentran:

- **Identación.**

El contenido siempre se identará con tabs, nunca utilizando espacios en blanco.

- **Cabecera del archivo.**

Es importante que todos los archivos .php inicien con una cabecera específica que indique información de la versión, autor de los últimos cambios, etc. Es decisión de cada equipo si se quieren o no agregar más datos.

/**

* @Clase controladora Prueba Libro. " PruebasLibrosController.php"

* @modificado: 24 de Febrero del 2014

* @autor: Javier Borrego Asteasuanizarra

*/

- **Comentarios en las funciones.**

Antes de declarar una función se debe añadir un comentario, explicando en qué consiste la misma. 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 ser suficiente para entender el código.

- **Ubicación y denominación de archivos.**

Se ubicarán los archivos según la estructura propuesta por Symfony 2 o según las especificaciones del equipo de arquitectura.

Para la denominación de los archivos se seguirán las convenciones establecidas por Symfony 2 o el equipo de arquitectura. Ejemplo:

Para las clases de gestión del negocio se usará el sufijo Gtr: PruebasGtr

Para los table model definidos para los grid se usará el sufijo TM: PruebasTM

Para las entidades de presentación se usará el sufijo Ep: PruebasEp

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

En caso de estar formado por más de una palabra: PruebasLibros.html.twig

Para la denominación de los parámetros se utilizará la siguiente nomenclatura (todo en minúscula utilizando como separador el carácter punto (.))

Ejemplo:
comunes.pruebalibrogtr.class:
comunes.tm.pruebalibro.class:

Para los servicios se utilizará la siguiente nomenclatura (todo en minúscula utilizando como separador el carácter punto (.))

Ejemplo:
comunes.pruebalibrogtr:
comunes.tm.pruebalibro:

Para las rutas se utilizará la siguiente nomenclatura (todo en minúscula utilizando como separador el caracter guión bajo (_))

Ejemplo:
comunes_pruebas_libro:

- **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 mismo del de la clase. Las clases siguen las mismas reglas de las funciones, por tanto, debe colocarse un comentario antes de la declaración de la clase explicando su utilidad. Los nombres de las clases deben iniciar con letra mayúscula. Si un nombre de clase se comprende de más de una palabra, la primera letra de cada nueva palabra debe comenzar con letra mayúscula. No se permiten las letras mayúsculas sucesivas; por ejemplo, una clase "SymfonyPDF" no se permite, mientras " SymfonyPdf" es aceptable.

Siempre utilizar las etiquetas <?php ?> para abrir un bloque de código. No utilizar el método de etiquetas cortas.

Estilo y reglas de escritura de código PHP.

- **Nombres de variables**.

Los nombres deben ser descriptivos y concisos. No usar grandes frases ni pequeñas abreviaciones para las variables. Siempre es mejor saber qué hace una variable con solo conocer su nombre. Esto se 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 escribirse siempre en mayúsculas y tanto estas como las variables globales deben tener como prefijo el nombre de la clase a la que pertenecen.

- **Las definiciones de la función**

Los nombres de las funciones pueden contener solo caracteres alfanuméricos y siempre deben empezar en letras minúsculas. Cuando el nombre de una función conste de más de una palabra, la primera letra de cada nueva palabra debe comenzar con mayúscula. Ejemplo:

```
gridPruebasLibroAction() {
        instrucciones
}
```

- **Llamadas a funciones**

Deben llamarse las funciones sin los espacios entre el nombre de la función, el paréntesis de la apertura, y el primer parámetro; los espacios entre las comas y cada parámetro, y ningún espacial entre el último parámetro, el paréntesis del cierre, y el punto y coma. Como por ejemplo:

```
<?php
 $var = foo($bar, $baz, $guux);
 ?>
```

- **Siempre incluir las llaves.**

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

Incorrecto:

```
if(condición) function();
```

Correcto :

```
if (condición) {
        function();
}
```

- **Colocar correctamente las llaves**

Las llaves de apertura irán al final de la sentencia que delimitan, y las de cierre alineadas con el inicio de la sentencia en una nueva línea.

```
if (condición) {
        for (iteración) {
                //código
        }
}
while (condición) {
        function();
}
```

- **Poner espacios entre signos.**

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

Esto está mal:	Esto está bien:
$a=0;	$a = 0;
for($i=5;$i<=$j;$i++)	for ($i = 5; $i <= $j; $i++)

Nota: De manera automática en el caso del IDE "NetBeans", al presionar las teclas Alt+Shift+f se autoformatea el contenido del fichero activo, haciendo cumplir estas 2 últimas pautas.

- **Precedencia de operadores**

Lo mejor es siempre usar paréntesis para estar seguro de la precedencia de los operadores.

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

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

- **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 siguiente manera:

Mal hecho:

```
if ($cliente == 5) ...
```

Bien hecho:

```
$a = 0;
if (isset($cliente) && $cliente == 5) ...
```

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 POST o por GET, etc.).

- **Instrucción "switch".**

Cuando se utilice, se deberá seguir el siguiente estilo:

```
switch ($modo) {
    case 'modo1':
        // Instrucción 1
        break;
    case 'modo2':
        // Instrucción 2
        break;
    default:
        // Código a ejecutar si todo falla
        break;
}
```

(SITPC, 2012)

2.3. Patrones de diseño

El patrón es una pareja de problema/solución con un nombre y que es aplicable a otros contextos, con una sugerencia sobre la manera de usarlo en situaciones nuevas. (Larman, 1999)

Entre los diversos patrones para el desarrollo de software se encuentran los patrones de diseño. Para que una solución sea considerada un patrón de diseño debe poseer ciertas características, una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores. Además debe ser reusable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias. Los patrones no se proponen descubrir ni expresar nuevos principios de la ingeniería del software. Todo lo contrario, intentan codificar el conocimiento, las expresiones y los principios ya existentes: cuanto más trillados y generalizados, tanto mejor. (Larman, 1999)

Dentro de las clasificaciones de los patrones están los patrones Generales de Software para Asignar Responsabilidades (GRASP), ellos describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones, además de los patrones Gang Of Four (GOF), que solucionan problemas de creación de instancias.

2.3.1. Patrones GRASP utilizados en la solución del proyecto SITPC

2.3.1.1. Patrón Creador

El patrón creador se encarga de asignarle a la clase B la responsabilidad de crear una instancia de la clase A en uno de los siguientes casos:

- B agrega los objetos A.
- B contiene los objetos A.
- B registra las instancias de los objetos A.
- B utiliza específicamente los objetos A.
- B tiene los datos de inicialización que serán transmitidos a A cuando este objeto sea creado (así que B es un experto respecto a la creación de A). (Larman, 1999)

El patrón creador guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El propósito fundamental de este patrón es encontrar

un creador que debemos conectar con el objeto producido en cualquier evento. Al escogerlo como creador, se da soporte al bajo acoplamiento, constituyendo su principal beneficio, lo cual supone menos dependencias respecto al mantenimiento y mejores oportunidades de reutilización. (Larman, 1999)

Este patrón se evidencia en el desarrollo del SITPC en las clases Controller, que son las responsables de la construcción de los formularios que se visualizarán en las vistas de la aplicación. Además se evidencian en las clases gestoras en el instante en el que se crean nuevas instancias de las clases Entidades.

```
public function indexAction() {
    $modelolibro = $this->get('comunes.tm.pruebalibro');
    $modeloEncargado = $this->get('comunes.tm.encargadolibro');
    $modeloEncargadoEntidad = $this->get('comunes.tm.encargadolibroEntidad');
    $rutas = new RutasGrid();
    $rutas->setRutaDatosAjax('comunes_pruebas_librosAjax');
    $form = $this->createForm(new \SIT\SRV\ComunesBundle\Form\Pruebas\PruebaLibroType());
    $direccion = new \Base\ComunBundle\Entity\AG\Direccion();

    $formDir = $this->createForm(new \SIT\SRV\ComunesBundle\Form\AG\DireccionPType(), $direccion);
```

Figura 3: Uso del patrón Creador en la clase PruebasLibrosController

2.3.1.2. Patrón Experto

El patrón experto se encarga de asignar una responsabilidad al experto en información, la clase que cuenta con la información necesaria para cumplir la responsabilidad. Experto es un patrón que se usa más que cualquier otro, es un principio básico que suele utilizarse en el diseño orientado a objetos.

Algunas ventajas del patrón experto son:

- Se conserva el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide. Esto soporta un bajo acoplamiento, lo que favorece al hecho de tener sistemas más robustos y de fácil mantenimiento.
- El comportamiento se distribuye entre las clases que cuentan con la información requerida, alentando con ello definiciones de clases “sencillas” y más cohesivas que son más fáciles de comprender y de mantener. (Larman, 1999)

Este patrón se evidencia en el desarrollo del SITPC en la creación de las vistas que serán mostradas al usuario, siendo las clases Controller las expertas en información, y por tanto, las encargadas de suministrarle toda información necesaria para la generación de la vista que será mostrada al usuario.

```
public function indexAction() {
    $modelolibro = $this->get('comunes.tm.pruebalibro');
    $modeloEncargado = $this->get('comunes.tm.encargadolibro');
    $modeloEncargadoEntidad = $this->get('comunes.tm.encargadolibroEntidad');

    $rutas = new RutasGrid();
    $rutas->setRutaDatosAjax('comunes_pruebas_librosAjax');
    $form = $this->createForm(new \SIT\SRV\ComunesBundle\Form\Pruebas\PruebaLibroType());
    $direccion = new \Base\ComunBundle\Entity\AG\Direccion();
    $formDir = $this->createForm(new \SIT\SRV\ComunesBundle\Form\AG\DireccionPType(), $direccion);
    return $this->render('ComunesBundle:Paginas\Pruebas\PruebaLibro:PruebasLibros.html.twig', array(
                'modelo' => $modelolibro,
                'modeloE' => $modeloEncargado,
                'modeloEE' => $modeloEncargadoEntidad,
                'ruta' => $rutas,
                'formulario' => $form->createView(),
                'formDir' => $formDir->createView()
    ));
}
```

Figura 4: Uso del patrón Experto en la clase PruebasLibrosController

2.3.1.3. Patrón Controlador

El patrón controlador se encarga de asignar la responsabilidad del manejo de un mensaje de los eventos de un sistema a una clase que represente una de las siguientes opciones:

- El “sistema” global (controlador de fachada).
- La empresa u organización global (controlador de fachada).
- Algo en el mundo real que es activo (por ejemplo, el papel de una persona) y que pueda participar en la tarea (controlador de tareas).
- Un manejador artificial de todos los eventos del sistema de un CU, generalmente denominados “Manejador<NombreCasodeUso>” (controlador de CU). (Larman, 1999)

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.

Este patrón se evidencia a través de las clases controladoras que se localizan en el directorio Controller perteneciente a cada Bundle del SITPC y en el uso del controlador frontal ubicado en el directorio web de cada aplicación concebida en Symfony 2. El principio fundamental de las clases controladoras se encuentra basado en que la lógica del negocio debe estar separada de la capa de presentación, con el propósito de aumentar la reutilización de código y tener un mayor control sobre los cambios.

```
class PruebasLibrosController extends BaseController {

    private function getGestor() {...}
    public function indexAction() {...}
    public function cargarModalAction() {...}
    public function gridPruebasLibroAction() {...}
    public function datosPersonaAction() {...}
    public function modDireccionAction() {...}
    public function salvarPruebaLibroAction() {...}
    public function eliminarPruebaLibroAction() {...}
    public function modPruebaLibroAction() {...}
    public function modificadoPruebaLibroAction() {...}
}
```

Figura 5: Uso del patrón Controlador en la clase PruebasLibrosController

2.3.1.4. Bajo acoplamiento

El acoplamiento según (Larman, 1999) es una medida de la fuerza con que una clase está conectada a otras clases, con que las conoce y con que recurre a ellas. Una clase con bajo (o débil) acoplamiento no depende de muchas otras; “muchas otras” depende del contexto. El bajo acoplamiento soporta el diseño de clases más independientes, que reducen el impacto de los cambios, y también más reutilizables, que acrecientan la oportunidad de una mayor productividad. No puede considerarse en forma independiente de otros patrones como experto o alta cohesión, sino que más bien ha de incluirse como uno de los principios del diseño que influyen en la decisión de asignar responsabilidades.

2.3.1.5. Alta cohesión

En la perspectiva del diseño orientado a objetos, la cohesión (o, más exactamente, la cohesión funcional) 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. (Larman, 1999)

Algunas ventajas del patrón alta cohesión son:

- Mejoran la claridad y la facilidad con que se entiende el diseño.
- Se simplifican el mantenimiento y las mejoras en funcionalidad.
- A menudo se genera un bajo acoplamiento.

- La ventaja de una gran funcionalidad soporta una mayor capacidad de reutilización, porque una clase muy cohesiva puede destinarse a un propósito muy específico.

2.3.2. Patrones GOF utilizados en la solución del proyecto SITPC

2.3.2.1. Decorador

Extiende la funcionalidad de un objeto dinámicamente de tal modo que es transparente a sus clientes, utilizando una instancia de una subclase de la clase original que delega las operaciones al objeto original. Provee una alternativa muy flexible para agregar funcionalidad a una clase. (Gamma, y otros, 1995)

En la implementación de SITPC el patrón decorador se evidencia en la potencialidad que posee Twig relacionado con la herencia entre plantillas, es decir, en el uso de una plantilla global que contendrá los elementos comunes del sitio para las vistas que decorarán las demás páginas de la aplicación.

```
{% extends 'ComunBundle::plantilla_SIT.html.twig' %}

{%block contenido%}

  {% include 'ComunesBundle:Paginas/Pruebas:PruebaSimple.html.twig' %}
```

Figura 6: Uso del patrón Decorador en la vista PruebasIndex.html.twig

2.3.2.2. Factory Method

Factoy Method permite organizar una clase de tal forma que se puedan instanciar otras clases sin depender de ninguna de las clases que puedan ser instanciadas. La clase reutilizable es capaz de permanecer independiente de las otras clases instanciadas delegando la elección de que clase instanciar a otros objetos y referirse a los objetos recién creados a través de una interfaz común. Define una interfaz para crear un objeto, dejando a las subclases decidir el tipo específico. (Gamma, y otros, 1995)

Este patrón se evidencia a la hora de crear los servicios de las clases Repository, pues estas clases van a contener las consultas a la base de datos y cuando alguna clase gestora requiera el uso de alguna consulta, este patrón permite llamar al servicio del Repository donde se encuentra la consulta deseada sin necesidad de crear una instancia de la clase que contiene a dicha consulta.

2.3.3. Otros patrones

2.3.3.1. Inyección de dependencias

La inyección de dependencias consiste en pasar (inyectar) a las clases todos los objetos que necesitan (dependencias) ya creados y configurados. (Eguíluz, 2007) Dicho patrón garantiza que se suministren objetos a una clase en lugar de ser la propia clase quien cree el objeto, según las relaciones que fueron establecidas entre las clases.

Este patrón es la clave para la compresión del funcionamiento del marco de trabajo, permitiendo que este sea rápido y flexibles, se evidencia en los servicios, que no son más que un objeto PHP que se crea cuando se es necesario utilizar la función a la que hace referencia dicho servicio y en los contenedores de servicios los cuales son objetos PHP que gestionan la creación de instancias de servicios, es decir, objetos. Sin el contenedor de servicios sería necesario crear cada servicio manualmente. En el SITPC se evidencia en la relación que se establece entre las clases gestoras y las clases controladoras, pues cada clase controladora tiene asociado una clase gestora que va a ser la encargada de manejar la lógica del negocio. Estos gestores son publicados como servicios en "SIT/src/SIT/SRV/ ComunesBundle/Resources/config/services.yml" y cuando una clase controladora necesite hacer uso de su gestor; en lugar de crear una instancia del mismo, simplemente llama a un método previamente declarado denominado getGestor() el cual accede al contenedor de servicios, que es el encargado de hacer la instancia del gestor solicitado en tiempo de ejecución y de destruirla posteriormente cuando ya no la necesite.

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

public function indexAction() {
    $modelolibro = $this->get('comunes.tm.pruebalibro');
```

Figura 7: Uso del patrón Inyección de dependencias en la clase PruebasLibrosController

2.4. Modelo de diseño

2.4.1. Diagramas de secuencia

La finalidad del diseño orientado a objetos es definir los objetos software y sus colaboraciones. Una notación habitual para ilustrar estas colaboraciones es el diagrama de interacción. Muestra el flujo de mensajes entre los objetos software y, por tanto, la invocación de métodos. (Larman, 1999)

Dentro de los diagramas de interacción se encuentran los diagramas de secuencia, ellos son un dibujo que muestra, para un escenario específico de un CU, los eventos que generan los actores externos, el orden y los eventos entre los sistemas. Todos los sistemas se tratan como cajas negras, los diagramas destacan los eventos que cruzan los límites del sistema desde los actores a los sistemas.

A continuación se muestra el diagrama de secuencia correspondiente al escenario Registrar escrito de impugnación documental del CU Registrar escrito de impugnación de documentos.

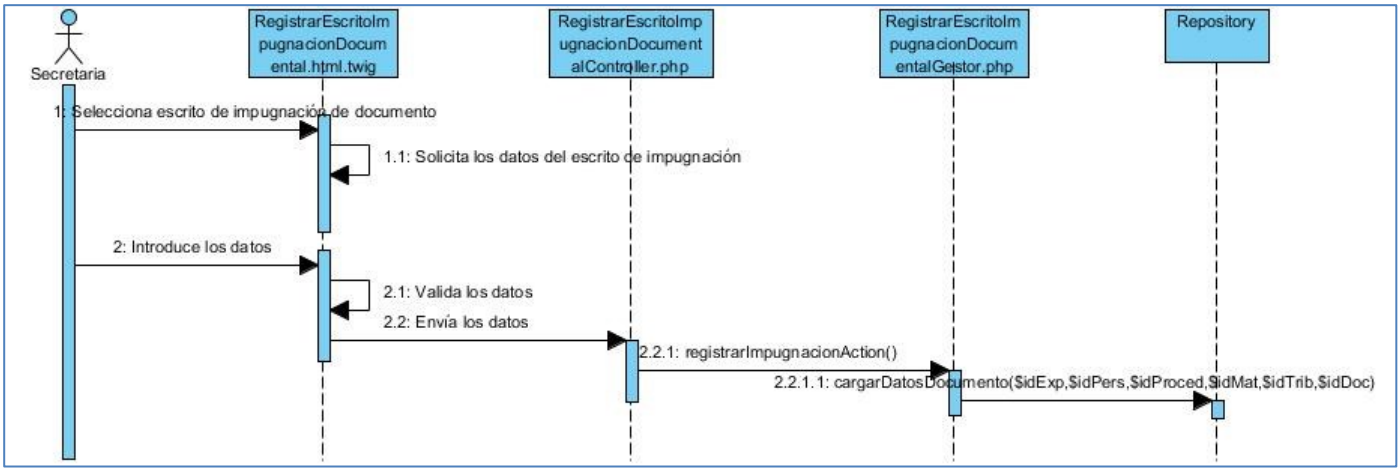


Figura 8: Diagrama de secuencia del CU Registrar escrito de impugnación de documentos

2.4.2. Diagramas de clases del diseño

El diagrama de clases del diseño describe gráficamente las especificaciones de las clases de software y de las interfaces (las de Java, por ejemplo) en una aplicación. Normalmente contiene la siguiente información:

- Clases, asociaciones y atributos.
- interfaces, con sus operaciones y constantes.
- Métodos.
- Información sobre los tipos de atributos.
- Navegabilidad y dependencias. (Larman, 1999)

A continuación se muestra el diagrama de clases del diseño correspondiente al CU Registrar escrito de impugnación de documentos.

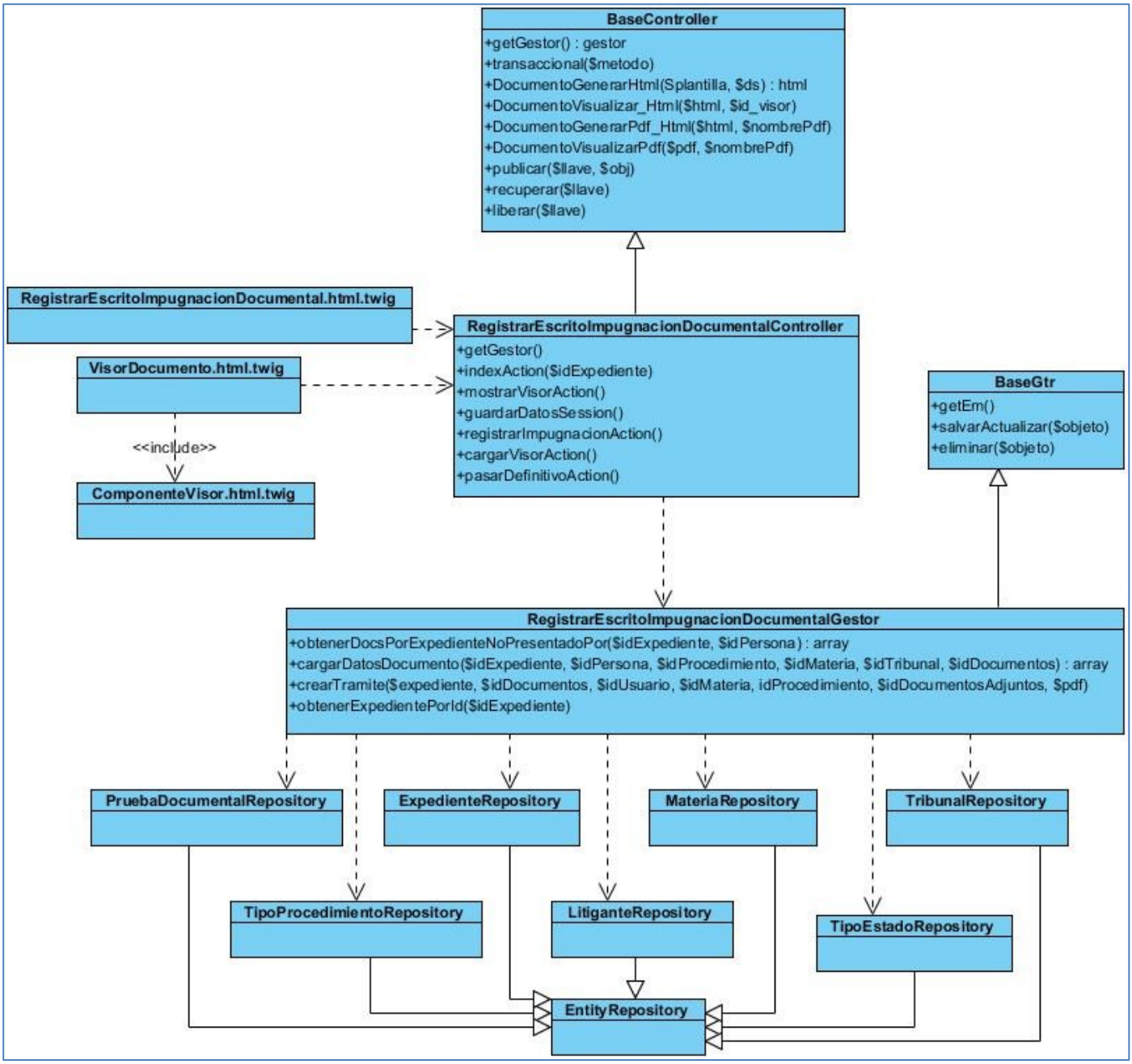


Figura 9: Diagrama de clases del diseño del CU Registrar escrito de impugnación de documentos

2.5. Modelo de implementación

2.5.1. Diagrama de componentes

Un componente es una parte física de un sistema, por lo que se puede decir que es la materialización de una o más clases. A continuación se muestra el diagrama de componentes correspondiente a los subprocesos Prueba Documental y de Libro, en el cual se representa los diferentes componentes que intervienen, desde los más generales ubicados en el paquete SITPC, tales como: seguridad, servicios, mensajería para la gestión de los mensajes que serán mostrados, componente para la captura y tratamiento de excepciones, controlador frontal y archivos de configuración de Symfony 2, y el enrutador del proyecto, hasta componentes específicos dentro del paquete Común, tales como: servicios, enrutador y el paquete Prueba Documental y de Libro, compuesto por los paquetes vista, modelo, controlador y datos.

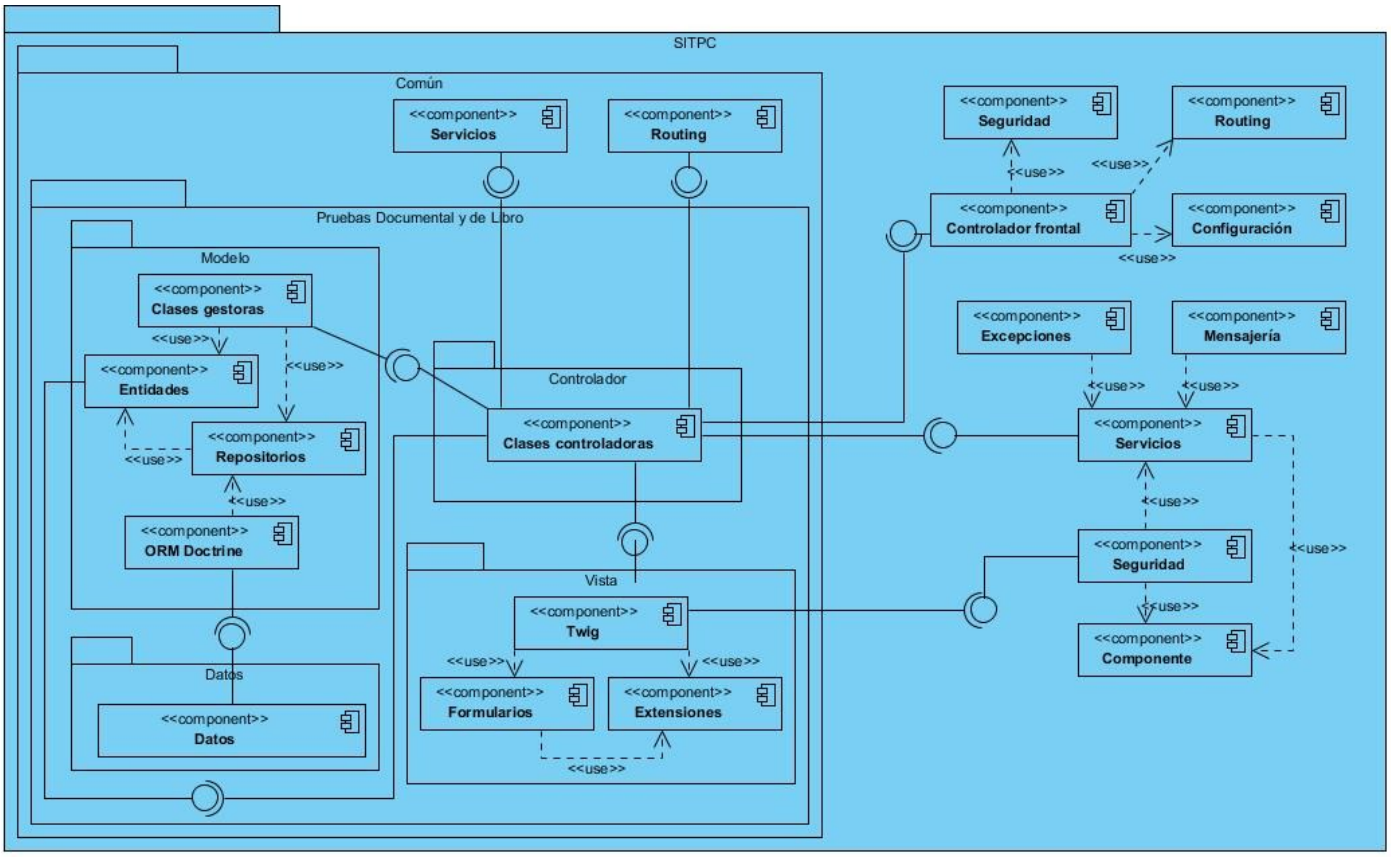


Figura 10: Diagrama de componentes

2.6. Diagrama de despliegue

El diagrama de despliegue tiene como objetivo modelar el hardware utilizado en las implementaciones de sistemas y las relaciones entre sus componentes. Muestra las relaciones físicas de los distintos nodos que componen un sistema y el reparto de los componentes sobre dichos nodos. Un nodo es un recurso de ejecución tal como un computador, un dispositivo o memoria. Los nodos son conectados por asociaciones de comunicación tales como enlaces de red, conexiones TCP/IP.

A continuación se muestra el diagrama de despliegue para el SITPC, el mismo está compuesto por una PC[5] cliente conectada a través del protocolo HTTP a un servidor web, que se comunica con el servidor de bases de datos mediante la extensión PDO[6], además de una impresora que se conecta mediante USB o el protocolo TCP/IP a la PC cliente, esto ocurre en cada una de las instancias. El servidor de bases de datos establece una conexión con el Centro de datos al cual envía toda la información que se procesa y a su vez obtiene aquella que necesite.

[5] Personal Computer
[6] PHP Data Objects

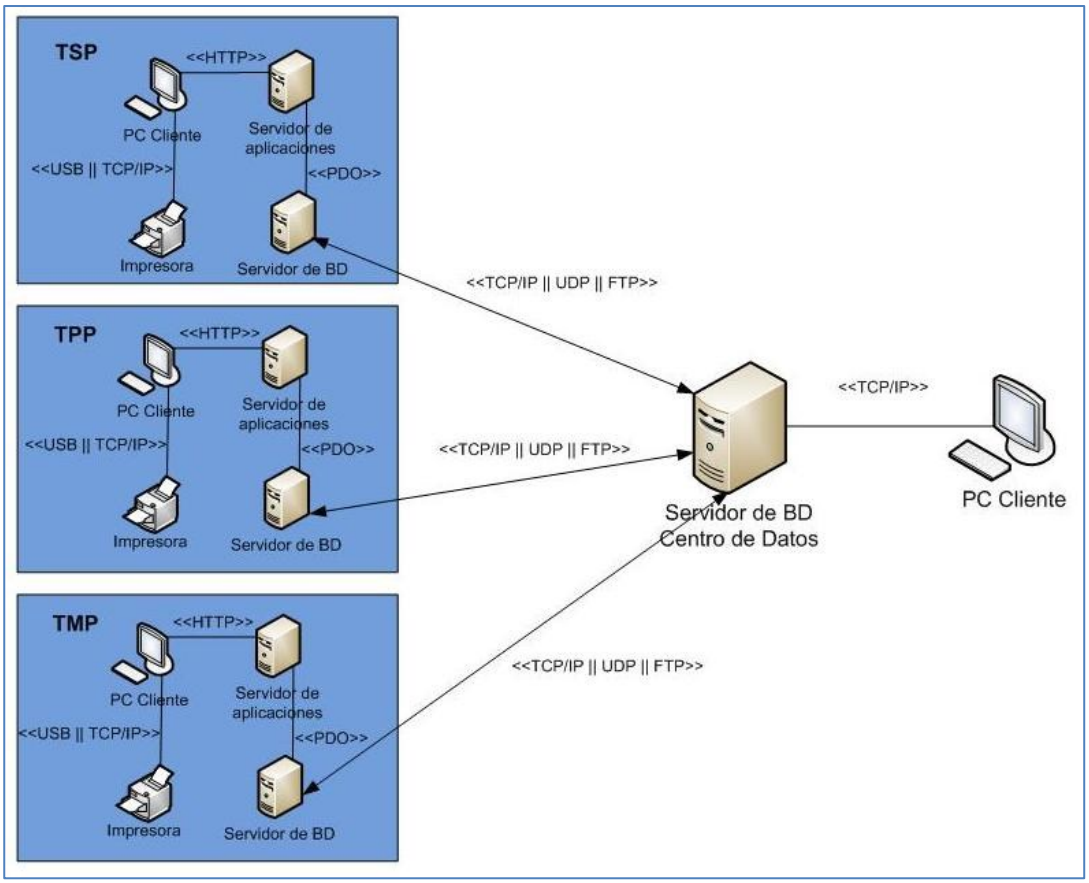


Figura 11: Diagrama de despliegue

2.7. Interfaces de la aplicación

A continuación se muestran algunas de las interfaces de la aplicación.

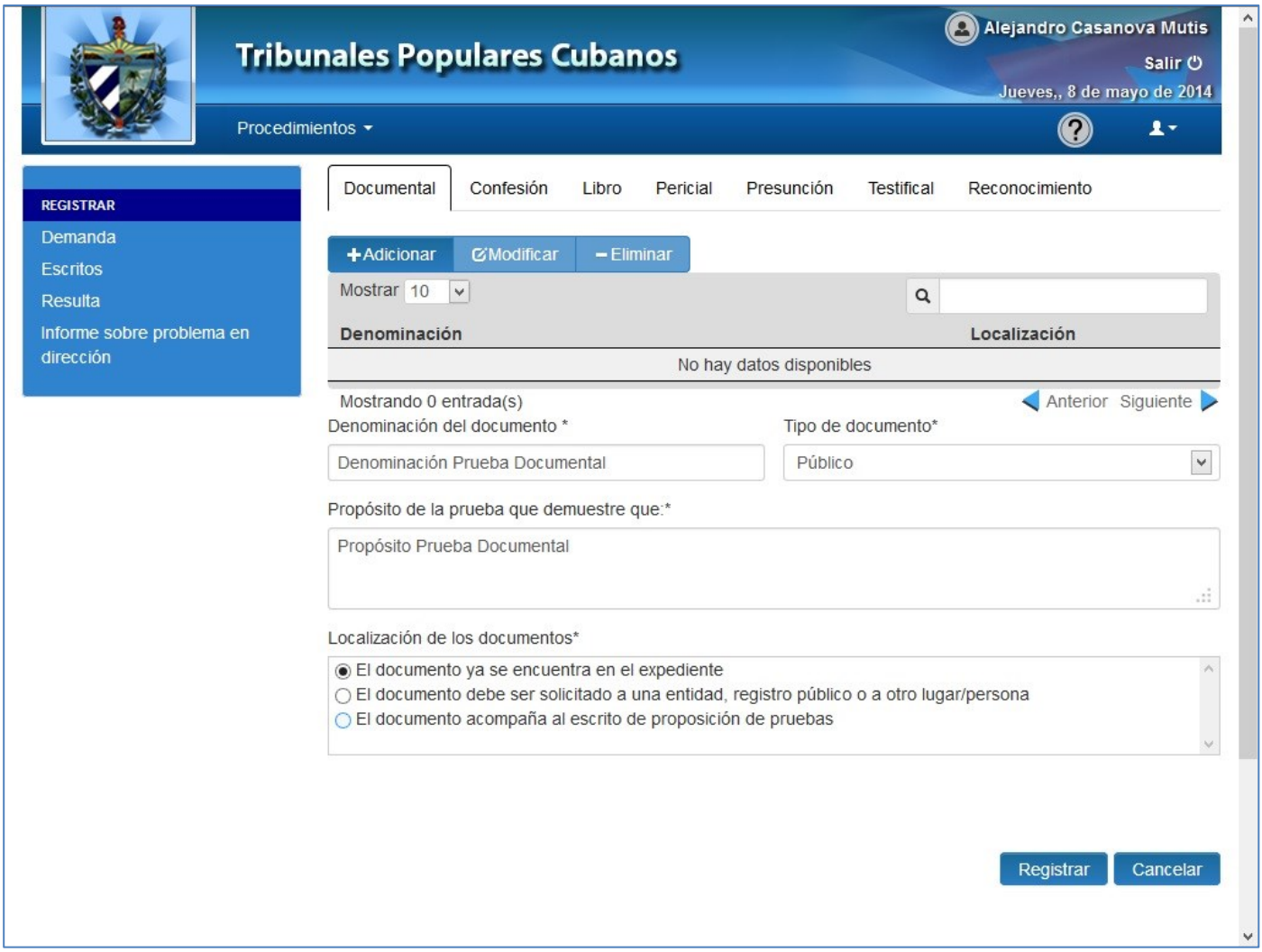


Figura 12: CU Gestionar Prueba documental

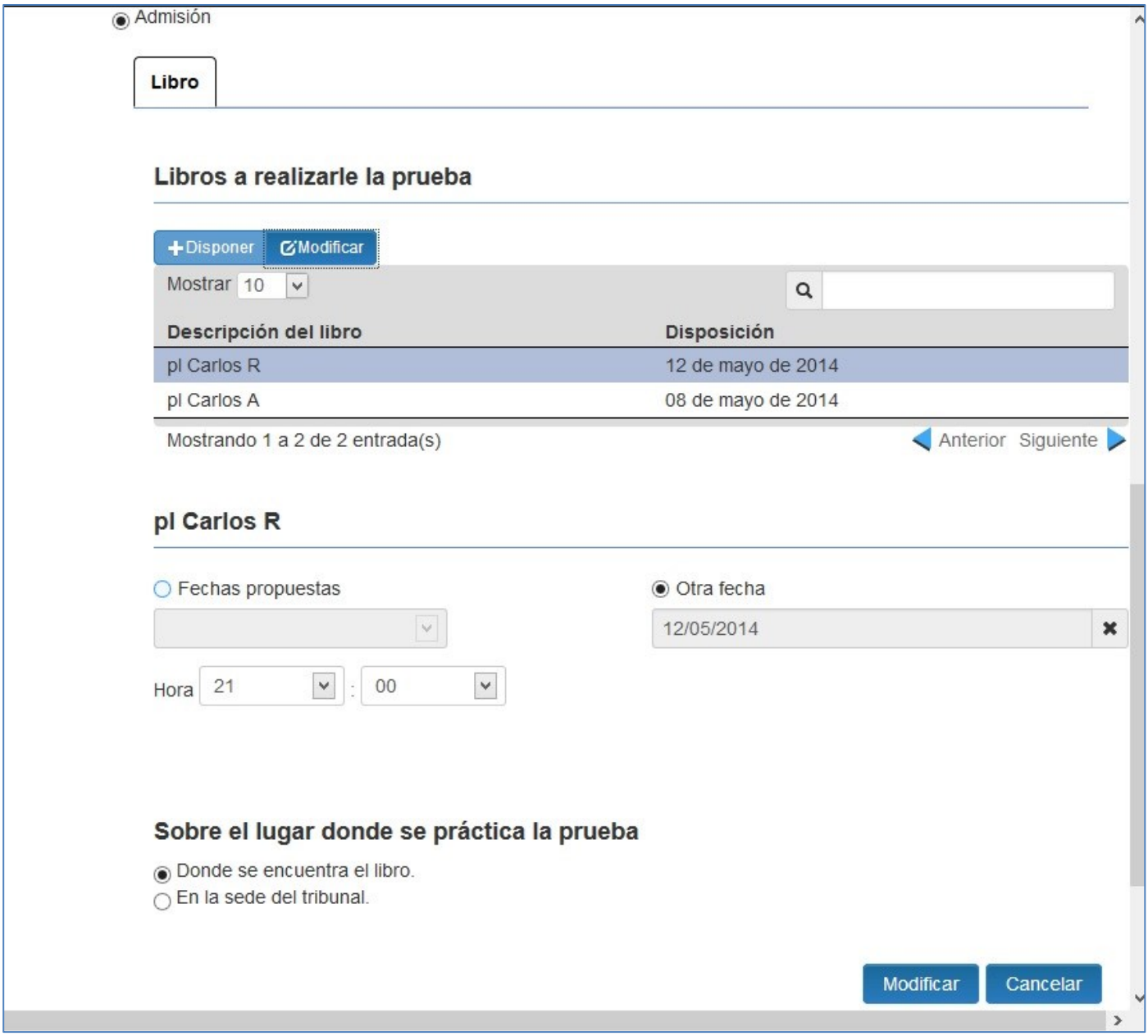


Figura 13: Señalar Prueba de libro

2.8. Conclusiones parciales

Según lo expuesto en el capítulo se arriba a las siguientes conclusiones:

- El sistema presenta una arquitectura en capas basada en el patrón MVC.

- Los estándares de codificación utilizados contribuyeron a lograr una mejor organización, uniformidad y entendimiento del código
- Entre los patrones utilizados se encuentran los patrones GRASP y los GOF, así como el patrón inyección de dependencias.
- El modelo de diseño permitió un mayor entendimiento de las funcionalidades del sistema, facilitando la elaboración de las actividades de implementación.
- El modelo de despliegue contribuyó a concebir la configuración de los elementos del hardware disponible y las conexiones entre estos elementos del sistema.

CAPÍTULO 3: Análisis de resultados

En el presente capítulo se realiza la validación de la solución propuesta, con el objetivo de evaluar los resultados obtenidos en el diseño e implementación del sistema. Se analizarán las diferentes pruebas de software que fueron aplicadas para medir la calidad de la aplicación, así como algunas métricas del diseño.

3.1. Validación del diseño. Métricas de diseño.

Para validar el diseño de la solución propuesta se utilizan métricas de diseño. Con el objetivo de medir el nivel de relaciones entre las clases y evaluar la complejidad de cada clase por separado, algunas de las métricas seleccionadas fueron: Tamaño de clases (TC), Relaciones entre clases (RC), Árbol de Profundidad de Herencia (APH) y Carencia de Cohesión en los métodos (CCM).

3.1.1. Familia de métricas CK (propuestas por Chidamber y Kemerer)

Uno de los conjuntos de métricas de software orientado a objetos a los que se hace referencia más ampliamente, es el propuesto por Chidamber y Kemmerer. Estas 6 métricas propuestas se refieren al diseño de las clases, a las cuales suele aludirse con el nombre de conjunto de métricas CK, y ellas son:

- Métodos ponderados por clase (MPC).
- Árbol de profundidad de herencia (APH).
- Número de descendiente (NDD).
- Respuesta para una clase (RPC).
- Acoplamiento entre clases objeto (ACO).
- Carencia de cohesión en los métodos (CCM). (Pressman, 2005)

Para la validación del diseño se utilizarán Árbol de profundidad de herencia (APH) y Carencia de cohesión en los métodos (CCM).

3.1.1.1. Árbol de profundidad de herencia (APH)

Esta métrica representa la longitud máxima desde el nodo hasta la raíz del árbol. A medida que crece la APH, es posible que las clases del nivel inferior hereden mucho más métodos. Una jerarquía de clases

profunda (su APH es mayor) también se presta a una mayor complejidad de diseño. Por otro lado, valores grandes de APH indican que se podrían reutilizar una gran cantidad de métodos.

A continuación se muestra el ejemplo de la aplicación de esta métrica al CU Crear acta de cotejo con originales.

Clase	Nivel de profundidad de herencia
BaseController	**0**
CrearActaCotejoOriginalesController	**1**
BaseGtr	**0**
CrearActaGtr	**1**
LitiganteRepository	**1**
ExpedienteRepository	**1**
TramiteRepository	**1**
CrearActaCotejo.html.twig	**0**

Tabla 1: Aplicación de la métrica APH

Luego de aplicar esta métrica se observa que un total de 3 clases presentan un APH de cero, y 5 clases presentan APH igual a uno. Se puede llegar a la conclusión que existe un bajo nivel de reutilización de los métodos en cuanto a la herencia entre clases, además el diseño tiene una complejidad media, lo que posibilita un mejor entendimiento del proyecto. El resto de las clases del diseño se comportan de manera similar al ejemplo.

3.1.1.2. Carencia de cohesión en los métodos (CCM)

La CCM es el número de métodos que acceden a uno o más de los mismos atributos. Si ningún método accede a los mismo atributos, entonces CCM = 0. Si la CCM es alta los métodos pueden acoplarse entre

sí mediante atributos, esto aumenta la complejidad del diseño de clase. Lo deseable es mantener alta la cohesión, es decir, conservar baja la CCM. (Pressman, 2005)

A continuación se muestra una tabla con la aplicación de dicha métrica a la clase “Expediente” la cual es una de las más utilizadas en el subproceso Pruebas Documental y de Libro. En la primera tabla se le otorga un identificador a cada atributo, y después en la segunda tabla, se representa la relación de los métodos con el atributo que utilizan.

Atributos	Identificadores
id	**A**
juez	**B**
escrito	**C**
usuariosProhibidos	**D**
numeroExpediente	**E**
complejidad	**F**
numeroPagina	**G**
fechaRadicacion	**H**
tramiteExpediente	**I**
expedienteEstado	**J**
usuarioExpedienteRegla	**K**
litigante	**L**

materia	M
tipoProcedimiento	N
tramite	O

Tabla 2: Atributos de la clase Expediente

Métodos	Atributos
getMateria	M
setTipoMateria	M
getProcedimiento	N
setTipoProcedimiento	N
getId	A
setNumeroExpediente	E
getNumeroExpediente	E
setNumeroPagina	G
getNumeroPagina	G
setFechaRadicacion	H
getFechaRadicacion	H
addJuez	B
removeJuez	B

getJuez	B
addExpedienteEstado	J
removeExpedienteEstado	J
getExpedienteEstado	J
addTramiteExpediente	I
removeTramiteExpediente	I
getTramiteExpediente	I
addTramite	O
removeTramite	O
getTramite	O
addUsuarioExpedienteRegla	K
removeUsuarioExpedienteRegla	K
getUsuarioExpedienteRegla	K
setComplejidad	F
getComplejidad	F
addLitigante	L
removeLitigante	L

getLitigante	**L**
addUsuariosProhibido	**D**
removeUsuariosProhibido	**D**
getUsuariosProhibidos	**D**
addEscrito	**C**
removeEscrito	**C**
getEscrito	**C**

Tabla 3: Métodos de la clase Expediente

Se puede concluir que la clase Expediente muestra un nivel de CCM igual 3, ya que los atributos no son accedidos por más de 3 métodos. Por tanto disminuye la complejidad del diseño de clases y favorece una alta cohesión, comportamiento que es similar en las demás clases.

3.1.2. Familia de métricas LK (propuestas por Lorenz y Kidd)

Lorenz y Kidd dividen las métricas basadas en clases en cuatro amplias categorías, cada una con un diseño al nivel de componentes: tamaño, herencia, valores internos y valores externos.

Para la validación del diseño se toma en consideración las métricas orientadas al tamaño de clase, las cuales se concentran en el conteo de atributos y operaciones en una clase individual, así como en valores promedio para el sistema orientado a objetos como un todo.

3.1.2.1. Tamaño de clase (TC)

Según (Pressman, 2005) el tamaño general de una clase se determina con las siguientes medidas:

- El número total de operaciones que están encapsuladas en una clase.
- El número de atributos que están encapsulados por la clase.

Los valores grandes de TC indican que tal vez una clase tenga demasiada responsabilidad, esto reduce la posibilidad de reutilización de la clase y complica la implementación y la prueba.

A continuación se muestra un ejemplo de las medidas de los parámetros de calidad obtenidas luego de la aplicación de la métrica TC.

Clase	Cantidad de Procedimientos	Responsabilidad	Complejidad	Reutilización
CrearActaCotejoOriginalesController	2	Baja	Baja	Alta
CrearActaGtr	2	Baja	Baja	Alta
ActaExhibicionDocumentosController	9	Baja	Baja	Alta
ActaExhibicionDocumentoGtr	11	Media	Media	Media
CrearActaPruebaLibroController	10	Baja	Baja	Alta
ActaPruebaLibroGtr	12	Media	Media	Media
OficioPruebaDocumentalController	9	Baja	Baja	Alta
DisponerSobrePruebaDocumentalController	21	Media	Media	Media
DisponerSobrePruebaDocumentalGtr	21	Media	Media	Media
PruebasLibroController	10	Baja	Baja	Alta
PruebasLibroGtr	12	Media	Media	Media
RegistrarEscritoPDocumentalController	8	Baja	Baja	Alta
RegistrarEPDocumentalGtr	10	Baja	Baja	Alta
RegistrarEscritoImpugnacionDocumentalController	9	Baja	Baja	Alta
RegistrarEscritoImpugnacionDocumentalGtr	12	Media	Media	Media

Tabla 4: Aplicación de la métrica TC

Para el ejemplo se aplicó la métrica a una muestra de 15 clases, para un total de 158 procedimientos y un promedio de 10.53 procedimientos aproximadamente.

Para evaluar los resultados de la aplicación de la métrica TC se aplicarán los siguientes umbrales definidos por Lorenz y Kidd.

TC	Umbral
Pequeño	<=20
Medio	>20 y <=30
Grande	>30

Tabla 5: Umbrales para la métrica TC

Se obtuvo como resultado que 13 clases presentan un tamaño pequeño y 2 clases con tamaño medio. Además analizando cada uno de los parámetros de calidad se demuestran bajo niveles de responsabilidad y complejidad de implementación (60%), y altos niveles de reutilización (60%). Estos datos evidencian un resultado positivo, ya que el objetivo estaba enmarcado en lograr bajos niveles de responsabilidad y complejidad de las clases, y valores elevados de reutilización, facilitando la implementación y entendimiento del proyecto. A continuación se muestran en gráficos los resultados antes abordados.

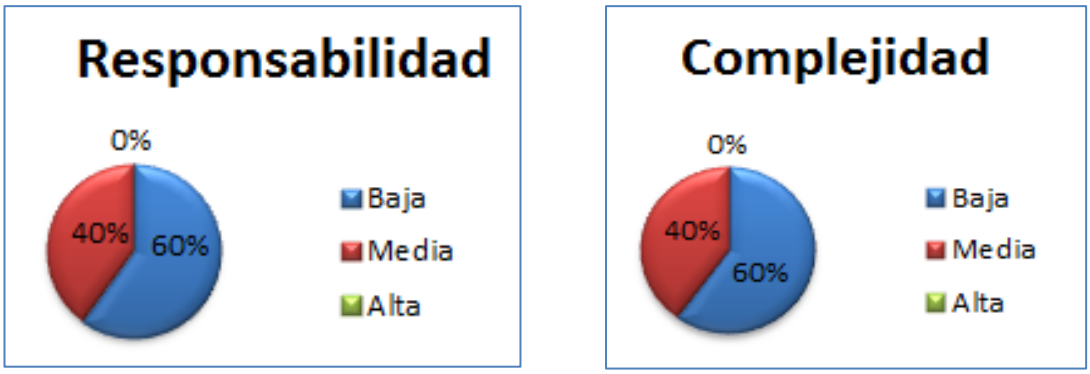


Figura 14: Resultados de la métrica TC en los atributos de calidad Responsabilidad y Complejidad

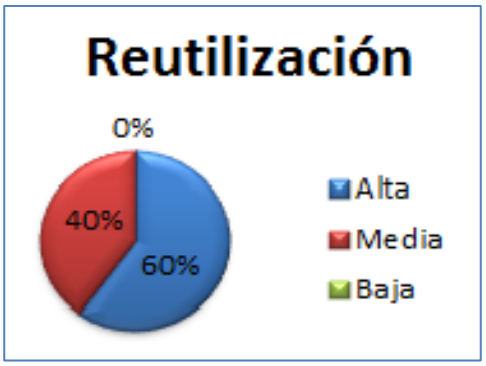


Figura 15: Resultados de la métrica TC en el atributo de calidad Reutilización

3.1.2.2. Relaciones entre clases (RC)

El resultado de la métrica RC viene dado por el número de relaciones de uso de una clase con otras. Al aplicar dicha métrica se evalúan atributos como el Acoplamiento, la Complejidad de Mantenimiento, la Reutilización y la Cantidad de Pruebas, a mayor número de relaciones de uso entre clases mayor será el Acoplamiento, la Complejidad de Mantenimiento y la Cantidad de Pruebas, mientras que su Reutilización disminuye.

A continuación se muestra un ejemplo de las medidas de los parámetros de calidad obtenidas luego de la aplicación de la métrica RC.

Clase	Relaciones de uso	Acoplamiento	Complejidad Mantenimiento	Reutilización	Cant. de Pruebas
CrearActaCotejoOriginalesController	1	Bajo	Baja	Alta	Baja
CrearActaGtr	1	Bajo	Baja	Alta	Baja
ActaExhibicionDocumentosController	1	Bajo	Baja	Alta	Baja
ActaExhibicionDocumentoGtr	10	Alto	Alta	Baja	Alta
CrearActaPruebaLibroController	2	Medio	Baja	Alta	Baja
ActaPruebaLibroGtr	2	Medio	Baja	Alta	Baja

OficioPruebaDocumentalController	1	Bajo	Baja	Alta	Baja
DisponerSobrePruebaDocumentalController	2	Medio	Baja	Alta	Baja
DisponerSobrePruebaDocumentalGtr	2	Medio	Baja	Alta	Baja
PruebasLibroController	2	Medio	Baja	Alta	Baja
PruebasLibroGtr	9	Alto	Alta	Baja	Alta
RegistrarEscritoPDocumentalController	2	Medio	Baja	Alta	Baja
RegistrarEPDocumentalGtr	11	Alto	Alta	Baja	Alta
RegistrarEscritoImpugnacionDocumentalController	1	Bajo	Baja	Alta	Baja
RegistrarEscritoImpugnacionDocumentalGtr	7	Alto	Media	Media	Media

Tabla 6: Aplicación de la métrica RC

En los siguientes gráficos se puede observar la distribución de los resultados obtenidos.

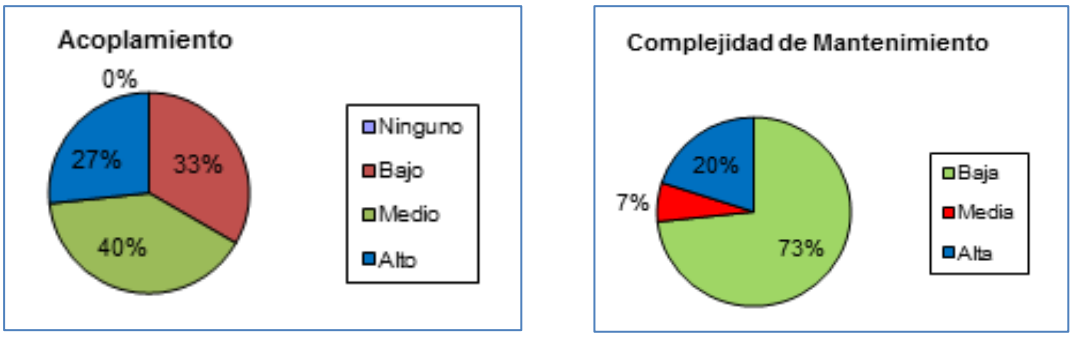


Figura 16: Resultados de la métrica TC en los atributos de calidad Acoplamiento y Complejidad de Mantenimiento

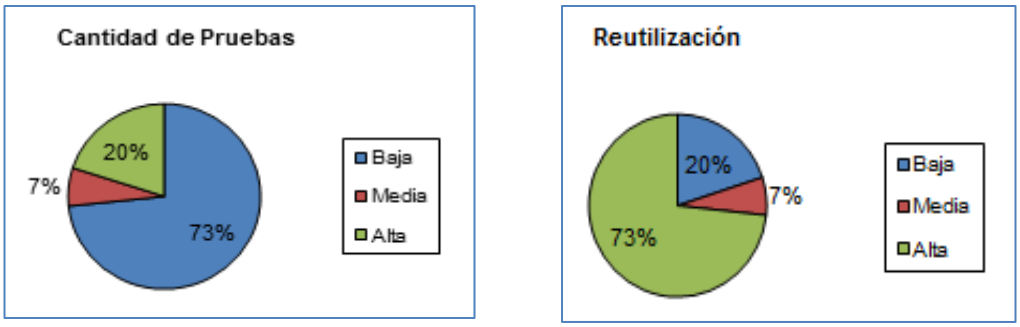


Figura 17: Resultados de la métrica TC en los atributos de calidad Cantidad de pruebas y Reutilización

Se puede llegar a la conclusión que el diseño presenta un acoplamiento relativamente bajo, la complejidad de mantenimiento se comporta de forma satisfactoria, pues aproximadamente el 73% de las clases de la muestra son de fácil soporte, se necesita una baja cantidad de pruebas para validar el diseño, ya que solo un 20% requiere de un alto esfuerzo a la hora de verificar la calidad del producto, y la mayoría de las clases favorecen la reutilización.

3.2. Validación de la implementación

Para validar la implementación se realizan diferentes pruebas de software, las cuales permiten determinar la calidad del producto. En este proceso se ejecutan pruebas dirigidas a componentes del software o al sistema en su totalidad, con el objetivo de medir el grado en que el software cumple con los requerimientos.

Hay dos maneras de probar cualquier producto construido: uno si se conoce la función específica para la que se diseñó el producto, se aplican pruebas que demuestren que cada función es plenamente operacional, mientras que se buscan los errores de cada función; dos si se conoce el funcionamiento interno, se aplican pruebas para asegurarse que “todas las piezas encajan”, es decir que las operaciones internas se realizan de acuerdo con las especificaciones, y se han probado todos los componentes internos de manera adecuada. Al primer enfoque de prueba se le denomina prueba de caja negra, y al segundo prueba de caja negra. (Pressman, 2005)

3.2.1. Pruebas de caja blanca

La prueba de caja blanca del software se basa en un examen cercano al detalle procedimental. Se prueban las rutas lógicas del software y la colaboración entre componentes, al proporcionar casos de pruebas que ejerciten conjuntos específicos de condiciones, bucles o ambos. Al emplear los métodos de

prueba de caja blanca el ingeniero del software podrá derivar casos de prueba que: 1) garanticen que todas las rutas independientes dentro del módulo se han ejercido al menos una vez. 2) ejerciten los lados verdadero y falso de todas las decisiones lógicas. 3) ejecuten todos los bucles en sus límites y dentro de sus límites operacionales y 4) ejerciten estructuras de datos internos para asegurar su validez. (Pressman, 2005)

Para ejecutar este tipo de pruebas se utilizó la técnica del Camino básico, propuesta por Tom McCabe, la cual permite obtener una medida de la complejidad lógica del procedimiento y utilizarla para definir los casos de prueba, de tal forma que se ejecute cada instrucción del programa al menos una vez.

A continuación se pone como ejemplo la aplicación de esta técnica en el método **modDireccionAction** de la clase **PruebaLibroController**.

La gráfica de flujo obtenida es la siguiente:

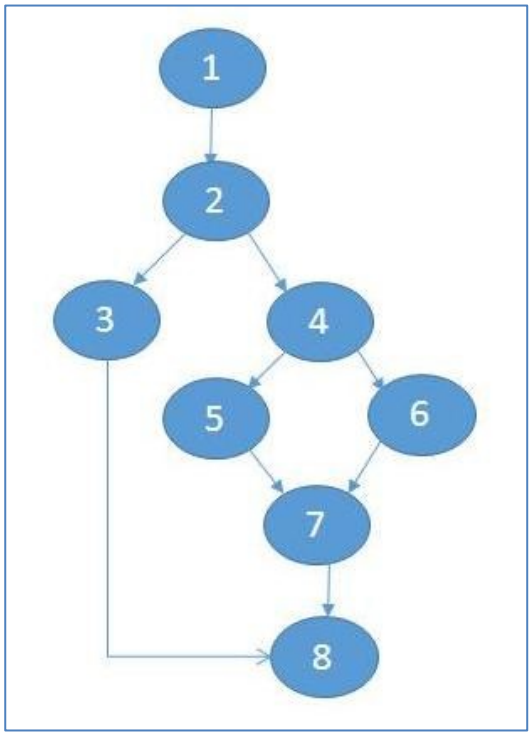


Figura 18: Grafo para el método modDireccionAction()

Luego se calcula el valor de la complejidad ciclomática, la cual según (Pressman, 2005) es una métrica de software que proporciona una medida cuantitativa de la complejidad lógica de un programa. El valor resultante define el número de rutas independientes[7] en el conjunto básico de la aplicación.

La complejidad ciclomática V(G) de un grafo G, se define como:

V(G) = E – N + 2, donde E es el número de aristas, y N el número de nodos de la gráfica de flujo.

La gráfica de flujo construida tiene 9 aristas y 8 nodos, por lo que:

V(G) = 9 – 8 + 2 = 3

Por tanto existen 3 rutas independientes en el método analizado, entonces serán necesarios igual número de casos de pruebas para verificar que se prueben todos los posibles datos de entradas en el método.

Los caminos de las rutas independientes son:

Camino 1: 1 – 2 – 3 – 8

Camino 2: 1 – 2 – 4 – 5 – 7 – 8

Camino 3: 1 – 2 – 4 – 6 – 7 – 8

En la tabla se muestra un ejemplo de un caso de prueba para el camino 1:

Entrada	Se haya registrado anteriormente alguna prueba de libro.
Resultados esperados	Carga los datos de la dirección para que pueda ser modificada.
Condiciones	$idlitigante == null && $idTestigo == null

Tabla 7: Caso de prueba de caja blanca

Una vez ejecutados los casos de prueba, se llegó a la conclusión que todos los caminos básicos identificados fueron probados satisfactoriamente, demostrando que no existe código innecesario.

3.2.2. Pruebas de caja negra

[7] En un grafo es el recorrido que incluye una arista que no se haya recorrido antes.

Las pruebas de caja negra, también denominadas, pruebas de comportamiento, se concentran en los requisitos funcionales del software. Permiten derivar conjuntos de condiciones de entrada que ejercitarán por completo todos los requisitos funcionales del programa. (Pressman, 2005)

Para la realización de estas pruebas se diseñó un caso de prueba por cada CU, basado en la técnica de partición de equivalencia, la cual divide el dominio de entrada de un programa en clases de datos a partir de las cuales pueden derivarse los casos de pruebas. El objetivo es ejecutar los casos de pruebas y encontrar la mayor cantidad de no conformidades, mientras existe alguna no conformidad el sistema no puede ser desplegado, por lo que es obligatorio realizar todas las iteraciones que sean necesarias para alcanzar una aplicación libre de errores.

A continuación se muestra un ejemplo de caso de prueba para el CU Gestionar prueba documental.

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	Denominación del documento (Variable1)	Campo de texto	Obligatorio	Se escribe el nombre que va a tener el documento. Puede tener cualquier tipo de caracteres.
2	Tipo de documento (Variable2)	Campo de selección múltiple	Obligatorio	Lista las opciones: Público y Privado.
3	Propósito de la prueba (Variable3)	Campo de texto	Obligatorio	Permite cadena de caracteres de números y letras. Comienza con mayúscula.
4	El documento ya se encuentra en el expediente (Variable4)	Campo de selección única	Obligatorio	Podrá seleccionarse siempre que las opciones: *El documento acompaña escrito de proposición de prueba* y *El documento debe ser solicitado a una entidad, registro público o a otro lugar/persona* no estén seleccionados.
5	El documento acompaña escrito de proposición de prueba (Variable5)	Campo de selección única	Obligatorio	Podrá seleccionarse siempre que las opciones: *El documento ya se encuentra en el expediente* y *El documento debe ser solicitado a una entidad, registro público o a otro lugar/persona* no estén seleccionados.
6	El documento debe ser solicitado a una entidad, registro público o a otro lugar/persona(Variable6)	Campo de selección única	Obligatorio	Podrá seleccionarse siempre que las opciones: *El documento acompaña escrito de proposición de prueba* y *El documento ya se encuentra en el expediente* no estén seleccionados.

Tabla 8: Descripción de las variables

Adicionar prueba documental

Escenario	Descripción	(Variable1)	(Variable2)	(Variable3)	(Variable4)	(Variable5)	(Variable6)	Respuesta del sistema	Flujo central
EC 1.1 Adicionar prueba documental.	El sistema permitirá adicionar una o varias pruebas documentales.	*V*	*V*	*V*	*V*	*V*	*V*	Se muestra en la tabla la prueba creada con la denominación del documento y la localización del mismo.	Selecciona la pestaña de Documental, Selecciona la opción Adicionar, llena los datos correspondientes y luego la opción Registrar.
		Prueba documental.	público	Sirve para probar	VERDADERO	FALSO	FALSO		
EC 1.2 Los datos son incorrectos.	Introduce datos incorrectos.	*I*	*V*	*V*	*V*	*V*	*V*	Indica que los datos son incorrectos y no guarda los cambios.	
		No registra dato	público	Sirve para probar	VERDADERO	FALSO	FALSO		
		V	*V*	*I*	*V*	*V*	*V*		
		Prueba documental 2	privado	3333333333	FALSO	VERDADERO	FALSO		
EC 1.3 La operación es cancelada.	La operación es cancelada.	*N/A*						Cierra la interfaz y no guarda los cambios.	Selecciona la pestaña de Documental, clic en el botón Adicionar y selecciona la opción Cancelar.

Tabla 9: Escenario de caso de prueba para adicionar una prueba documental

Modificar prueba documental

Escenario	Descripción	Variable(1)	Variable(2)	Variable(3)	Variable(4)	Variable(5)	Variable(6)	Respuesta del sistema	Flujo central
EC 1.1 Modificar prueba documental.	El sistema después de haber registrado una prueba documental, permitirá modificarle los datos que esta posee.	*V*	*V*	*V*	*V*	*V*	*V*	Guarda los datos modificados.	Selecciona la pestaña de Documental, selecciona la prueba que se quiera modificar, selecciona la opción Modificar, llena los datos correspondientes y luego la opción Registrar.
		Prueba documental 1	público	Ver variable	FALSO	FALSO	VERDADERO		
EC 1.2 Los datos son incorrectos.	Introduce datos incorrectos.	*V*	*V*	*V*	*V*	*V*	*I*	Indica que los datos son incorrectos y no guarda los cambios.	
		deja en blanco este campo	público	Escribe caracteres extraños	FALSO	FALSO	No selecciona ningún campo		
EC 1.3 La operación es cancelada.	La operación es cancelada.	*N/A*						Cierra la interfaz y no guarda los cambios.	Selecciona la pestaña de Documental, clic en el botón Adicionar y selecciona la opción Cancelar.

Tabla 10: Escenario de caso de prueba para modificar una prueba documental

Eliminar prueba documental

Escenario	Descripción	Prueba documental	Respuesta del sistema	Flujo central
EC 1.1 Eliminar	El sistema permitirá eliminar una prueba	*V*	Elimina la prueba seleccionada de	Selecciona la pestaña de Documental, selecciona

prueba documental.	documental seleccionada.	Selecciona la prueba que desea eliminar.	la tabla donde se muestra.	la prueba documental que desea eliminar, selecciona la opción Eliminar.
EC 1.2 Eliminar incorrectamente.	No permite eliminar.	/	Muestra un mensaje comunicando que debe ser seleccionada la prueba que se desea eliminar.	Selecciona la pestaña de Documental, no selecciona la prueba documental que desea eliminar, selecciona la opción Eliminar.
		No selecciona la prueba que desea eliminar		

Tabla 11: Escenario de caso de prueba para eliminar una prueba documental

Una vez aplicados todos los casos de pruebas se obtuvo en la primera iteración un total de 38 no conformidades, ya para la segunda iteración se redujo a 18 no conformidades, y en la tercera iteración, la aplicación estaba libre de errores.

En la siguiente gráfica se muestran los resultados antes mencionados.

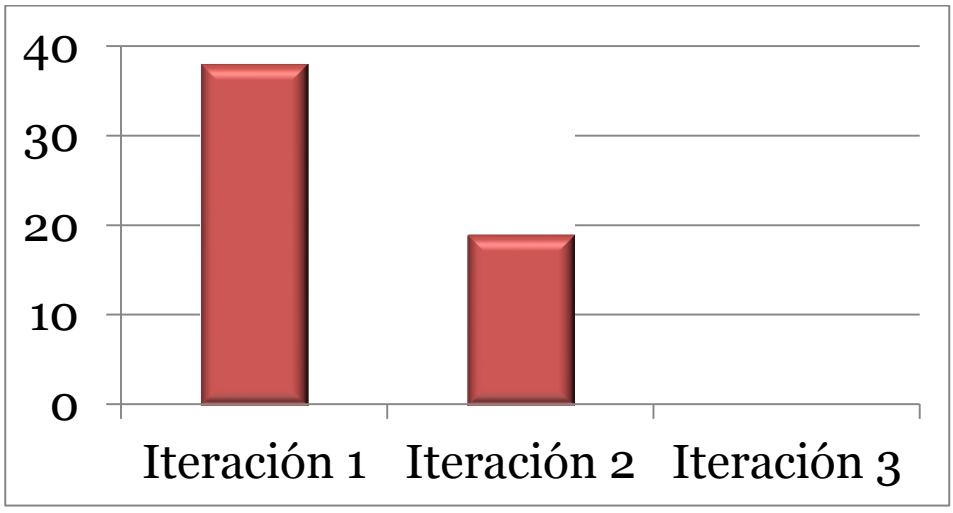


Figura 19: Representación de las no conformidades del sistema

3.3. Conclusiones parciales

Según lo expuesto en el capítulo se arriba a las siguientes conclusiones:

- El resultado obtenido con la aplicación de las métricas APH, CCM, TC y RC demuestran que el diseño de la aplicación generalmente presenta baja complejidad de las clases, bajo acoplamiento, alta cohesión y valores elevados de reutilización.
- Para la validar la implementación se aplicaron las pruebas de caja blanca y caja negra, utilizando las técnicas de camino básico y partición de equivalencia respectivamente.

CONCLUSIONES GENERALES

Con la culminación del presente trabajo se desarrollaron los subprocesos Prueba Documental y de Libro, contribuyendo a la gestión de la información en los Tribunales Populares Cubanos. Por lo que se puede llegar a la conclusión que:

- A través del estudio del estado del arte se evidenció que de las herramientas informáticas analizadas no existe ninguna solución que pudiera ser tomada como referencia para el desarrollo de la aplicación.
- Se confeccionó el diseño de la solución, mediante el cual se obtuvo el flujo de actividades de cada CU, y permitió tener una visión de cómo deberían quedar implementado los requisitos del sistema.
- Con la correcta selección de lenguajes, tecnologías y herramientas se implementó la aplicación, obteniéndose un software que satisface las necesidades del cliente.
- Se aplicaron métricas de diseño y pruebas de software que permitieron medir cuantitativamente la calidad de la solución.

RECOMENDACIONES

Se recomienda la integración del proceso Pruebas específicamente las Pruebas Documental y de Libro a los demás subsistemas que componen el SITPC y aún no se han implementado, pues actualmente solo se encuentra integrado con el subsistema Económico.

BIBLIOGRAFÍA

Achour, Mehdi, Betz, Friedhelm y Dovgal, Antony. 2013. *Manual de PHP.* 2013.

Álvarez, Miguel Ángel. 2011. desarrolloweb.com. [En línea] 3 de 11 de 2011. www.desarrolloweb.com/manuales/css3.html.

Annan, Kofi. 2003. Discurso inaural de la primera fase de la WSIS. 2003.

Ávila, Rodolfo Pérez. 2010. *Arquitectura de Software Vista de Integración.* La Habana : s.n., 2010.

Becerra, María Dolores Peche. 2008. Foro e-Gobierno OEA. [En línea] 2008. http://www.suboletin.com/contentsoea/docs/Boletin_65/TemadelMes65.htm.

Cantón, Alejandro Castillo. 2006. Axtro. [En línea] 2006. www.theproc.es.

Company, The Reuse. 2006. Adictos al Trabajo. [En línea] 22 de junio de 2006. [Citado el: 29 de noviembre de 2013.] http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=reuse.

Echandía, Hernando Devis. 1981. *Compendio de la prueba judicial.* Bogotá : s.n., 1981.

Eguíluz, Javier. 2011. *Desarrollo Web Ágil con Symfony 2.* 2011.

Foundation, JQuery. 2013. JQuery write less, do more. [En línea] 2013. http://api.jquery.com.

Gamma, Erich, y otros. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software.* 1995.

García, Victor. 2011. Programación en Java. [En línea] 13 de diciembre de 2011. [Citado el: 26 de noviembre de 2013.] http://programarjava.wordpress.com/2011/12/13/reutilizacion-de-software-ventajas-y-desventajas/.

Ing. Pedro Ariel Negro, Dra. Roxana Giandini. 2008. Umbrales para Métricas Orientadas a Objetos. Mar del Plata, Argentina : s.n., 2008.

Jacobson, Ivar, Booch, Grady y Rumbaugh, James. 2000. *El Proceso Unificado de Desarrollo de Software.* s.l. : Addison Wesley, 2000.

JavaScript, Introducción a. 2007. *Javier Eguiluz Pérez.* 2007.

Juristo, Natalia , Moreno, Ana y Vegas, Sira. 2005. *Técnicas de evaluación de Software.* 2005.

Justicia, Órgano de Ministerio de. 2007. Noticias Jurídicas. [En línea] 14 de febrero de 2007. [Citado el: 15 de marzo de 2014.] http://noticias.juridicas.com/base_datos/Admin/rd84-2007.html#cpa2.

Kruchten. 2000. *The Rational Unified Process: An introduction.* s.l. : Addison Wesley, 2000.

Larman, Craig. 1999. *UML y Patrones.* México : s.n., 1999.

Luño, Antonio Enrique Pérez. 1996. *Manual de Informático y Derecho.* Barcelona : Ariel, 1996.

Pacheco, Nacho. 2011. *Manual de Twig.* 2011.

Paradigm, International Visual. 2011. Boots Productivity with Innovative and Intuitive Technologies. [En línea] 2011. http://www.visual-paradigm.com/support/documents/vpumluserguide/12/13/5963_visualparadi.html.

Peñalvo, Francisco José García. 2004. *Tratamiento en Java.* Salamanca : s.n., 2004.

Pérez, Javier Eguíluz. 2009. *CSS Avanzado.* 2009.

Pressman, Roger S. 2005. *Ingeniería de software: Enfoque Práctico 6ta edición.* 2005.

Ramírez, Raquel Zambrano. 2008. *Sistemas gestores de bases de datos.* Granada : s.n., 2008.

Salup, Dr. José R. Amaro. 1996. Ley de Procedimiento Civil, Administrativo, Laboral y Económico. Cuidad de La Habana : s.n., 1996.

Sandra Almeira, Adriana y Pérez Cavenano, Varina. 2007. *Arquitectura de software: Estilos y Patrones.* San Juan Bosco, Argentina : s.n., 2007.

Shaw, David Garlan y Mary. 1994. *An introduction to software architecture.* 1994.

SITPC, Equipo de arquitectura. 2012. *Plan de Administración de la Conexión.* La Habana : s.n., 2012.

Software, Pruebas de. 2005. Pruebas de Software. [En línea] España, 2005. [Citado el: 26 de noviembre de 2013.] pruebasdesoftware.com.

Team, Doctrine Project. 2011. *Doctrine 2 ORM Documentation.* 2011.

Team, NetBeans. 2013. NetBeans. [En línea] 2013. netbeans.org/index_es.html.

Team, PostgreSQL. 2009-2013. PostgreSQL - es. [En línea] 2009-2013. www.postgresql.org.es/sobre_postgresql.