Universidad de las Ciencias Informáticas

FACULTAD 3

Diseño e implementación del subproceso Pruebas de Confesión del proyecto de Informatización para la Gestión de los Tribunales Populares Cubanos

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

Autores:

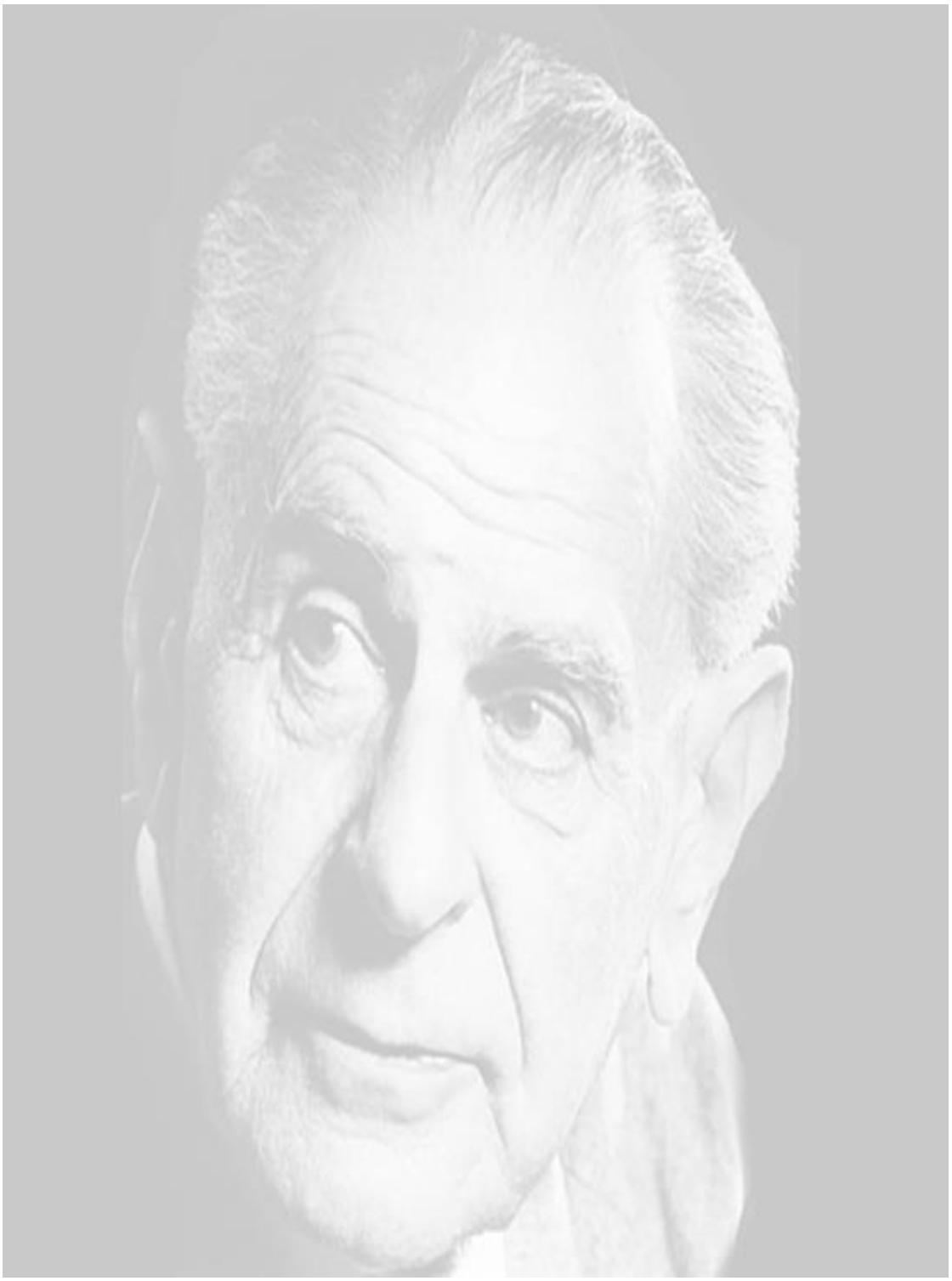
Yamilka Rios La Hoz

Willian Gómez Díaz

Tutora: Ing. Diana Valdés González

Co-tutor: Ing. Manuel Urquiza Rodríguez

La Habana, junio de 2014



"Lo que caracteriza al hombre de ciencia no es la posesión del conocimiento o de verdades irrefutables, sino la búsqueda desinteresada e incesante de la verdad."

Karl Popper

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

_____	_____
Yamilka Rios La Hoz	Willian Gómez Díaz
Autor	Autor
_____	_____
Ing. Diana Valdés González	Ing. Manuel Urquiza Rodríguez
Tutor	Co-tutor

Autores:

Yamilka Rios La Hoz

Correo Electrónico: ylahoz@estudiantes.uci.cu

Willian Gómez Díaz

Correo Electrónico: wgomez@estudiantes.uci.cu

Tutor: Ing. Diana Valdés González

Correo Electrónico: dvaldesg@uci.cu

Co-tutor: Ing. Manuel Urquiza Rodríguez

Correo Electrónico: murquiza@uci.cu

Agradezco:

A mis padres: mami y papi, son lo más grande que tengo en la vida, gracias por su apoyo y su dedicación, por estar siempre al pendiente de mí, por dejar a un lado sus necesidades cuando de satisfacer las mías se trata. Doy gracias a la vida por tenerlos y espero se sientan orgullosos de mí, hoy y siempre.

A mi hermano: Yano, has sido mi ejemplo e inspiración durante todos estos años y seguir tus pasos me ha traído hasta aquí, así que este logro también es tuyo.

A mis abuelas, especialmente a Cacá, gracias por consentirme siempre, por ser mi viejita linda y apoyarme en todo, por aguantar mis malcriadeces.

A mis tíos y padrinos, Berta y Orlando, creo que nunca podré llegar retribuirles todo lo que han hecho por mí, gran parte de este triunfo es de ustedes que han sido un apoyo fundamental en mi vida.

A mis sobrinos Jonathan y Jeffry, por llenarme de alegría y sonrisas, por regalarme su cariño.

A tía Tery, tío Julito, Juli, Lidibet, la china y Jeni, por ser tan especiales conmigo, por hacerme sentir como en mi casa cuando estoy con ustedes.

A mi familia en general, por su apoyo y preocupación en todo momento.

A Anidey, por soportarme estos 5 años y hacerme saber en todo momento que cuento con tu amistad incondicional.

A Albert, gracias taquiti por tantos momentos lindos que hemos pasado juntos, por estar siempre dispuesto a ayudarme para lo que sea, por haberte sentado ese primer día de clases a mi lado (aunque tú digas que fui yo).

A los otros integrantes de mi F4 (ahora F5): Fidel, Yordanka y Dania, por compartir conmigo sus alegrías y sus problemas, por escucharme y aconsejarme.

A Misael, por brindarme tu amor y cariño, por tu apoyo en todos los sentidos, porque sin ti todo hubiese sido más difícil, te quiero.

A Daniel y Liliana por todo lo que vivimos juntos en el IPVCE y por llegar hasta acá conmigo.

A todos mis compañeros del 3501 y del antiguo 3101, especialmente a Andrés, Jesús, Charli y el Sama.

A mi compañero de tesis por tener siempre palabras de aliento para consolarme, incluso robarme una sonrisa, en los momentos de mayor estrés.

A mis tutores por todo el apoyo brindado en la realización de este trabajo.

A los miembros del tribunal por contribuir a elevar la calidad de la tesis, especialmente a la oponente: la profe Maribel, por brindarnos su ayuda y consejos siempre que lo necesitamos.

A los profesores que contribuyeron a mi formación profesional durante todos estos años.

Yamilka

Agradezco:

A Belkis Díaz, the best que tengo en la vida.

A Angel Luis.

A mi abuela Felicia, a mi tia Zule, a mis primos Zurelis y Alex.

A mis hermanos Adrian y Anita.

A mi team de asalto y combate, Irene, Lisandra, José Miguel, René, Randy.

A todos los que hoy no están aquí pero estuvieron en ese primer año en el 4103, a Leonardo, a Maine, a Solanyi, a Yuliet, a Gonzalo.

A mis compañeros del IPVCE Carlos Marx.

A mis profesores, Yalice, Yoan, Rosalina, Mónica, Hilda, Eutimio.

Al tribunal por sus atenciones hacia nosotros en estos días y en especial la Ing. Maribel por habernos dedicado tanto de su tiempo.

A mis tutores por haberme ayudado tanto, y darme consejos para terminar este trabajo.

A mi compañera de tesis y no por ser la última, es menos importante; que ha sabido tener paciencia y cargar conmigo.

Willian

Dedicatoria:

A mami y papi por ser mis tesoros más preciados, mis ángeles guardianes.

A mi hermano por ser faro y guía.

A Cacá por consentirme y mimarme siempre.

A tía Berta y tío Orlando por quererme como una hija y ayudarme tanto.

Yamilka

Dedicatoria:

A mi mamá, quien ha sabido siempre estar ahí para mí, y me ha permitido llegar hasta aquí por mis propias decisiones, acertadas o no, siempre mías. Por haber siempre confiado en ellas y haberme aconsejado en todo momento.

Willian

RESUMEN

El Sistema de Informatización para la Gestión de los Tribunales Populares Cubanos contribuye a elevar la calidad de la actividad jurisdiccional en todo el país. Uno de los subprocesos que se llevan a cabo en los tribunales son las Pruebas de Confesión. Este subproceso es común para varios módulos del proyecto y su informatización de manera independiente por cada uno de ellos generaría una gran pérdida de tiempo y recursos innecesariamente, además podría provocar la duplicidad de funcionalidades y la no uniformidad de la solución. Es por ello que el presente trabajo de diploma tiene como objetivo general realizar el diseño e implementación del subproceso Pruebas de Confesión de manera que satisfaga los requisitos acordados y contribuya al desarrollo de los módulos Administrativo, Disciplina y Derecho laboral y Ordinario civil del proyecto de Informatización para la Gestión de los Tribunales Populares Cubanos.

Para dar cumplimiento al objetivo planteado se realizó un estudio y descripción de la metodología de desarrollo a utilizar, así como de las principales herramientas y lenguajes de programación. Además se describió la arquitectura base para el desarrollo de la solución y se validó el resultado obtenido, mostrándose el análisis correspondiente. Con esta propuesta de solución se espera contribuir al desarrollo de los módulos Administrativo, Disciplina y Derecho laboral y Ordinario civil, al permitir la reutilización de las Pruebas de Confesión por cada uno de ellos, disminuyendo así el tiempo de desarrollo del proyecto, mejorando la mantenibilidad del código y ahorrando recursos humanos y materiales.

Palabras claves: Pruebas de Confesión, Tribunales Populares Cubanos.

TABLA DE CONTENIDOS

INTRODUCCIÓN

En la actualidad se hace cada vez más necesario el uso de las Tecnologías de la Información y las Comunicaciones (TICs) en las distintas esferas en las que se desenvuelve el hombre. El desarrollo de la informática ha posibilitado la informatización oportuna de grandes procesos, contribuyendo de esta manera a la disminución de los costos y el aumento de la productividad. Diversos son los sectores de la sociedad que se han beneficiado con el avance tecnológico, el sector empresarial se ha integrado, de manera efectiva, a todo este fenómeno que se llama tecnología, logrando así informatizar gran parte de sus procesos garantizando una mayor calidad de los mismos.

Existen diversas empresas cubanas en las que se llevan a cabo procesos de gran complejidad, que aún se realizan de forma manual, lo que afecta en gran medida el desempeño de las mismas. No obstante, desde hace algunos años se han dado pasos en el proceso de informatización de la sociedad con el uso de las Tecnologías de la Información y las Comunicaciones, para dar solución a las necesidades de información y conocimiento de las diferentes esferas de la sociedad. Siguiendo esta premisa surge, en el año 2002 la Universidad de la Ciencias Informáticas (UCI), primera universidad nacida al calor de la Batalla de Ideas.

La UCI, en la actualidad, posee gran prestigio dentro de la industria cubana del software, pues a lo largo de todos estos años la universidad ha participado en numerosos programas de informatización con ministerios y entidades nacionales, destacándose en el desarrollo de sistemas para la salud, educación, automatización, prensa, bioinformática, realidad virtual, entre otros. Esto ha posibilitado que dichas entidades aumenten la eficiencia y agilidad de sus procesos contribuyendo a que el país dé un paso de avance en cuanto a su desarrollo tecnológico. El sector jurídico no se ha quedado al margen de estas transformaciones, identificando varias entidades que pueden ser informatizadas, como es el caso del Ministerio de Justicia, los bufetes colectivos, las notarías, los registros civiles, la Fiscalía General de la República, la cual está dando hoy sus primeros pasos en la utilización de un sistema de gestión fiscal producido en la UCI, y el Sistema de Tribunales Populares Cubanos que constituye el objetivo del presente trabajo.

Una de las proyecciones estratégicas del Sistema de Tribunales Populares Cubanos es continuar desarrollando su infraestructura tecnológica y asegurar su gestión, administración y explotación en los principios de la Seguridad Informática. Para el logro de este propósito se concibió el acuerdo de colaboración con el Centro de Gobierno Electrónico (CEGEL) de la Universidad de las Ciencias Informáticas para trabajar en proyectos mutuamente beneficiosos en el ámbito de la Formación, Producción e Investigación Científica. Surge así el Sistema de Informatización para la Gestión de los Tribunales Populares Cubanos (SITPC), el cual contempla varios subsistemas y módulos entre

los que se encuentra el módulo Común que informatiza procesos del negocio comunes para todos los subsistemas.

Uno de los procesos que se desarrollan en el módulo Común es el proceso Pruebas, el cual es priorizado por su importancia en el derecho. En distintos contextos se puede entender la palabra prueba de manera diferente; algunas de las definiciones que da La Real Academia de la Lengua Española sobre la palabra prueba son:

1. Razón, argumento, instrumento u otro medio con que se pretende mostrar y hacer patente la verdad o falsedad de algo.
2. *Del Derecho.* La que, por diferencia de la regla de su libre estimación por el juez, la ley exige específicamente para poder acreditar un hecho determinado.

Las pruebas (en el contexto judicial) son clasificadas en distintos tipos: Pericial, Documental, de Libro, Testifical, de Presunción, de Reconocimiento y de Confesión.

En el subproceso prueba de confesión se hace referencia solo a los involucrados en el caso. Donde todo litigante (persona natural o jurídica), está obligado a comparecer para prestar confesión cuando así lo solicite su contrario o de oficio lo disponga el Tribunal; según lo estipulado en el artículo 262, SECCIÓN TERCERA de la Ley de Procedimiento Civil, Administrativo, Laboral y Económico (1). El día –previamente establecido por el Tribunal– en el que se practicarán las pruebas, se abrirá el sobre que contenga las posiciones (preguntas), delante de las partes que concurriesen. En el caso de dos litigantes que hayan de declarar sobre las mismas posiciones, el Tribunal deberá tomar las medidas pertinentes para que no puedan comunicarse, ni enterarse previamente de las mismas. Siempre que para la práctica de prueba se libre despacho a otro Tribunal, se acompañará el interrogatorio en sobre cerrado, una vez declaradas pertinentes las posiciones que el litigante haya de contestar. De existir alguna circunstancia especial (el litigante no puede abandonar el domicilio, se necesita un intérprete de lenguajes de señas o idioma, etc.), el interrogatorio se llevará a cabo en el domicilio, mediante un intérprete, o en el caso que fuere un sordomudo y supiese leer, se harán las preguntas por escrito. En los procesos en que sea parte el Estado, sus órganos y organismos o las empresas estatales, no se les pedirá confesión. En su lugar, la parte contraria propondrá por escrito las preguntas que quiera hacer, las cuales serán contestadas por vía de informe por los funcionarios a quienes conciernan los hechos; según el artículo 279. Se podrán presentar solicitud de Pruebas de Confesión fuera del período estipulado en cada una de las materias, cuando se demuestren nuevas circunstancias o aspectos a tratar que no se hubieran tratado y fuera relevante para el caso. (2)

El desarrollo de una aplicación informática que informatice este subproceso sería de gran utilidad para el módulo Común y de manera general para el SITPC. Al ser este subproceso común a la mayoría de los subsistemas se garantiza su reutilización, permitiendo la disminución del tiempo de desarrollo de estos. Actualmente existen tres módulos en el proyecto que no se encuentran implementados, Administrativo, Disciplina y Derecho laboral y Ordinario civil, cuya implementación demoraría 2, 4 y 3 meses respectivamente, utilizando para ello 35 personas. Con la informatización del subproceso Pruebas de Confesión de Común, se espera disminuir el tiempo y la cantidad de personas necesarios para la implementación de estos.

A partir de los requerimientos del proceso se comenzó la informatización del subproceso Pruebas de Confesión. Hasta el momento se cuenta con la especificación de los requisitos funcionales y no funcionales con los que debe cumplir el sistema, firmados por el cliente. Además se especificaron cada uno de los casos de uso correspondientes, determinando su complejidad y priorizándolos según los criterios establecidos en el programa de mejora de CALISOFT[1]. Posteriormente se dio paso a la creación de los prototipos de interfaz de usuario a partir de talleres realizados por el equipo de desarrollo, y luego a la validación de los mismos por parte del cliente. Se considera pertinente continuar con el diseño e implementación de estos requisitos funcionales para satisfacer las necesidades de los TPC informatizando las Pruebas de Confesión, contribuyendo de esta forma al desarrollo de los módulos Administrativo, Disciplina y Derecho laboral y Ordinario civil, y de manera general al Sistema de Informatización para la Gestión de los Tribunales Populares Cubanos.

Ante la problemática anterior surge el siguiente **problema a resolver**:

¿Cómo satisfacer los requisitos acordados para el subproceso Pruebas de Confesión de forma tal que contribuya al desarrollo de los módulos Administrativo, Disciplina y Derecho laboral y Ordinario civil del proyecto de Informatización para la Gestión de los Tribunales Populares Cubanos?

Teniendo en cuenta lo anteriormente expuesto se define como **objeto de estudio:**

Proceso Prueba de los Tribunales Populares Cubanos.

Para dar respuesta a la interrogante planteada, el presente trabajo tiene como **objetivo general:**

Realizar el diseño e implementación del subproceso Pruebas de Confesión de manera que satisfaga los requisitos acordados y contribuya al desarrollo de los módulos Administrativo, Disciplina y Derecho laboral y Ordinario civil del proyecto de Informatización para la Gestión de los Tribunales Populares Cubanos.

[1] Centro Nacional de Calidad de Software

Se determina entonces como **campo de acción:**

Diseño e implementación del subproceso Pruebas de Confesión del módulo Común perteneciente al proyecto de Informatización para la Gestión de los Tribunales Populares Cubanos.

Como **idea a defender** se define que**:**

Si se realiza el diseño y la implementación del subproceso Pruebas de Confesión, se contribuirá a la satisfacción de los requisitos acordados y se facilitará el desarrollo de los módulos Administrativo, Disciplina y Derecho laboral y Ordinario civil del proyecto de Informatización para la Gestión de los Tribunales Populares Cubanos.

A partir del objetivo general se derivan los siguientes **objetivos específicos:**

- Elaborar el marco teórico de la investigación.
- Obtener el Modelo de diseño.
- Realizar la implementación.
- Validar los resultados obtenidos.

Con vista a dar cumplimiento a los objetivos planteados, se deciden las siguientes **tareas de investigación:**

1. Caracterización del Sistema de Tribunales Populares Cubanos.
2. Análisis del proceso Pruebas de los Tribunales Populares Cubanos.
3. Análisis del subproceso Pruebas de Confesión de los Tribunales Populares Cubanos.
4. Caracterización de la metodología de desarrollo de software RUP[2] en el marco del proyecto de Informatización para la Gestión de los Tribunales Populares Cubanos.
5. Caracterización del Lenguaje Unificado de Modelado (UML) según los artefactos que se obtienen de la aplicación de la metodología del proyecto.
6. Descripción de la herramienta CASE[3] Visual Paradigm para la representación gráfica de los artefactos que se obtienen en el proyecto.
7. Caracterización de los marcos de trabajo Symfony2 y Bootstrap para la implementación de la solución propuesta.

[2] Rational Unified Process (proceso unificado de desarrollo)
[3] Computer Aided Software Engineering (ingeniería de software asistida por computadora)

8. Caracterización de PHP como lenguaje de programación del proyecto y JQuery para el trabajo con HTML[4].
9. Descripción de PostgreSQL y Doctrine como sistema gestor de BD y diseñador del modelo de datos y consultas respectivamente.
10. Descripción de la arquitectura de software base para la implementación del sistema.
11. Resumen de métodos y técnicas de pruebas de software que se usarán para la validación del sistema.
12. Realización de los diagramas de clases del diseño del subproceso Pruebas de Confesión.
13. Realización de los diagramas de secuencia del diseño del subproceso Pruebas de Confesión.
14. Implementación de los casos de uso del subproceso Pruebas de Confesión.
15. Realización del diagrama de despliegue.
16. Diseño de casos de prueba para la validación del subproceso Pruebas de Confesión.
17. Realización de pruebas funcionales a los requisitos acordados para el subproceso Pruebas de Confesión.
18. Solución de no conformidades al finalizar cada iteración de pruebas.

El trabajo está estructurado de la siguiente manera:

El capítulo 1 hace alusión a la fundamentación teórica del trabajo, estableciendo cada elemento teórico que servirá de base para darle solución a la problemática planteada. Se fundamentan las tecnologías, lenguajes de programación, herramientas y metodologías utilizadas durante el proceso de desarrollo del software.

El capítulo 2 presenta un análisis detallado de la solución que se propone, exponiendo para ello la arquitectura base sobre la cual se desarrolló el sistema. Así como diagrama de clases y secuencia del diseño además del diagrama de despliegue.

El capítulo 3 detalla la validación realizada a la aplicación obtenida, así como los casos de pruebas realizados para comprobar que se implementaron todas las funcionalidades especificadas y que estas funcionan correctamente.

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

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

En el presente capítulo se exponen los conceptos fundamentales que servirán de base para un mayor entendimiento del subproceso Pruebas de Confesión del Sistema de Tribunales Populares Cubanos. Además se fundamenta la utilización de la metodología, paradigmas de programación, herramientas, patrones de diseño y lenguajes de programación que fueron definidos por el equipo de arquitectura del proyecto y que contribuyen al desarrollo de la aplicación.

1.2 Sistema de Tribunales Populares Cubanos (TPC)

En Cuba, el sistema judicial está estructurado en tres instancias: los Tribunales Municipales Populares (TMP), los Tribunales Provinciales Populares (TPP) y el Tribunal Supremo Popular (TSP). Los TMP ejercen su jurisdicción en todo el territorio municipal correspondiente, sus sedes radican generalmente en la cabecera municipal, aunque pueden situarse en cualquier otro lugar del municipio por decisión del Consejo de Gobierno del Tribunal Supremo Popular. Los TPP ejercen su jurisdicción en todo el territorio provincial correspondiente y sus sedes radican donde el Consejo de Gobierno del Tribunal Supremo Popular decida. El TSP es el órgano encargado de ejercer la máxima autoridad judicial en la República de Cuba, tiene su sede en La Habana y ejerce jurisdicción en todo el territorio nacional.

Los tribunales populares cubanos constituyen un sistema de órganos estatales, estructurado con independencia funcional de cualquier otro, y subordinado jerárquicamente a la Asamblea Nacional del Poder Popular y al Consejo de Estado. Se rigen por los principios consagrados en la Constitución, que norman la organización y el funcionamiento de los órganos estatales. (2)

1.3 Informatización de los TPC

La informatización de los TPC es un proceso que se lleva a cabo en los tribunales a todas las instancias e incluye varios proyectos de informatización entre otros aspectos para la gestión estadística, almacén de datos y la gestión jurídica. Este último a través del Sistema de Informatización para la Gestión de los Tribunales Populares Cubanos.

Actualmente los TPC llevan a cabo la mayoría de sus procesos manualmente, lo que ocasiona una gran pérdida de tiempo y recursos. Debido a esto el Tribunal Supremo Popular llegó a un acuerdo con la Universidad de las Ciencias Informáticas para realizar un software que informatizara sus procesos, imprimiéndole así fiabilidad y celeridad a los mismos. A partir de la propuesta realizada por parte del TSP, la UCI escogió al Centro de Gobierno Electrónico (CEGEL) para desarrollar

dicha solución. Surge así, el “Sistema de Informatización para la Gestión de los Tribunales Populares Cubanos”. El proyecto está constituido por 5 subsistemas, dentro de los cuales se encuentra el módulo Común que, como su nombre lo indica, implementa los procesos de negocio con sus respectivos actos procesales, que son comunes para los diferentes subsistemas. Dentro de los subprocesos que contempla Común se encuentra el proceso Pruebas.

1.4 Proceso Pruebas

En cada uno de los subsistemas del proyecto de Informatización para la Gestión de los Tribunales Populares Cubanos las pruebas se encuentran presentes, pues son un proceso común que, a pesar de tener sus especificidades en cada subsistema, por lo general siguen el mismo flujo. Las pruebas pueden ser clasificadas en distintos tipos según su naturaleza y los involucrados; es por esto que se hace necesario definir subprocesos que ayuden a organizar y entender este gran proceso, entre los cuales se pueden contar Prueba Documental, de Libro, Pericial, de Presunción, de Reconocimiento, Testifical y de Confesión.

1.4.1 Subproceso Pruebas de Confesión Judicial

El subproceso comienza con la solicitud de prueba vía escrito por parte del abogado en representación de la parte. Posteriormente corresponde al juez disponer sobre este escrito ya sea admitiendo la prueba, rechazándola o mandando a subsanar. Una vez sea admitido el escrito, el juez señala día y hora en que se practicará la prueba. La Figura 1 representa el diagrama de procesos correspondiente al subproceso Pruebas de Confesión.

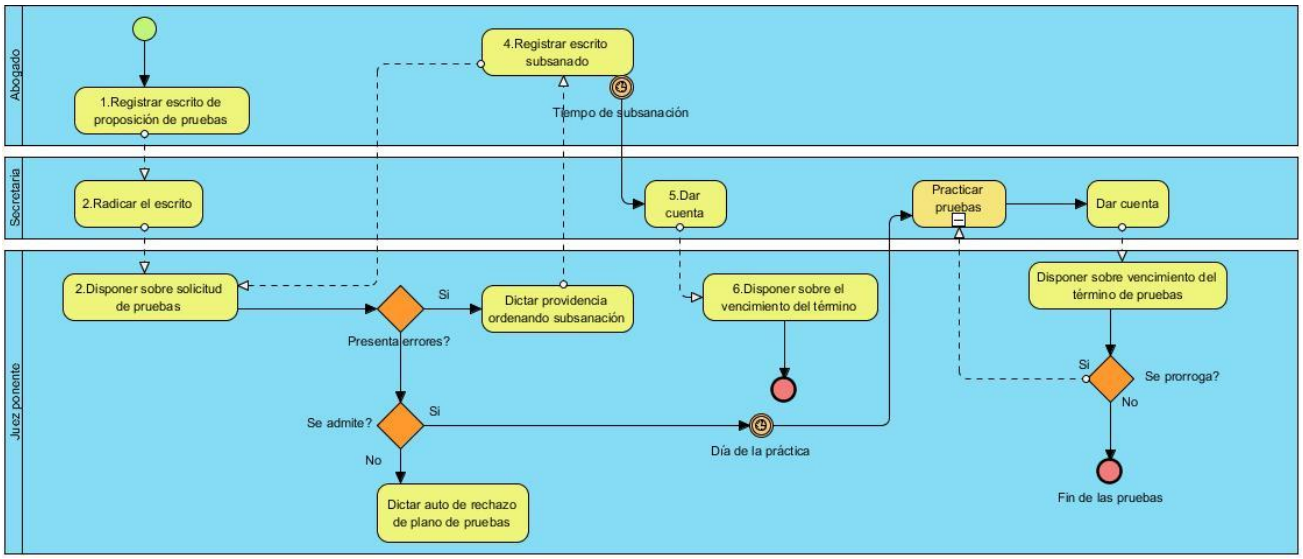


Figura 1: Diagrama de procesos correspondiente al subproceso Pruebas de Confesión

Para lograr la correcta informatización del subproceso anteriormente descrito, se hace necesaria la selección de una metodología y tecnologías de desarrollo adecuadas, según las características del proyecto. Estas contribuyeron con las primeras etapas: modelado del negocio y levantamiento de requisitos y guiarán además todo el proceso de desarrollo para construir un software de calidad.

1.5 Tecnologías de desarrollo

1.5.1 Metodología de desarrollo de software

Las metodologías de desarrollo de software "son todas las actividades necesarias para transformar los requisitos de un usuario en un sistema de software". (3) Proporcionan una guía para desarrollar el producto informático deseado, indicando paso a paso las actividades que se deben realizar, qué personas deben realizarlas y el rol que desempeñan cada una de ellas.

1.5.1.1 Proceso Unificado de Desarrollo (RUP)

RUP es un proceso unificado de desarrollo de software, que proporciona un enfoque disciplinado para la asignación de tareas y responsabilidades dentro de un equipo de desarrollo. Su objetivo es asegurar la producción de software de alta calidad, que satisfaga las necesidades de sus usuarios finales, con el tiempo y presupuesto requeridos. RUP mejora la productividad del equipo proporcionando a cada miembro fácil acceso a una base de conocimientos con las directrices, plantillas y herramientas para todas las actividades críticas de desarrollo. Al acceder todos los miembros del equipo a la misma base de conocimientos, no importa si se trabaja con los requisitos, con el diseño, pruebas, administración de la configuración o cualquier otra área, esto garantiza que todos compartan un lenguaje, proceso y vista común sobre cómo desarrollar el software. (4)

La vida de un software se divide en ciclos, cada ciclo representa una nueva generación del producto. RUP divide un ciclo de desarrollo en cuatro fases consecutivas:

- Inicio: Tiene como finalidad principal el alcanzar un acuerdo entre todos los interesados respecto a los objetivos del ciclo vital para el proyecto.
- Elaboración: El principal propósito de esta fase es el establecimiento de una línea base para la arquitectura del sistema y proporcionar una base para el diseño y la implementación de la fase de construcción.
- Construcción: Se busca completar el desarrollo del sistema.
- Transición: Se garantiza que el software esté disponible para los usuarios.

Existen tres aspectos fundamentales que definen este proceso: es dirigido por casos de uso, centrado en la arquitectura e iterativo e incremental. RUP cuenta además en su ciclo de vida con 9

flujos de trabajo principales dentro de los que se encuentran 6 flujos de ingeniería y 3 flujos de soporte, relacionados todos con las 4 fases antes mencionadas. La Figura 2 muestra dicha relación:

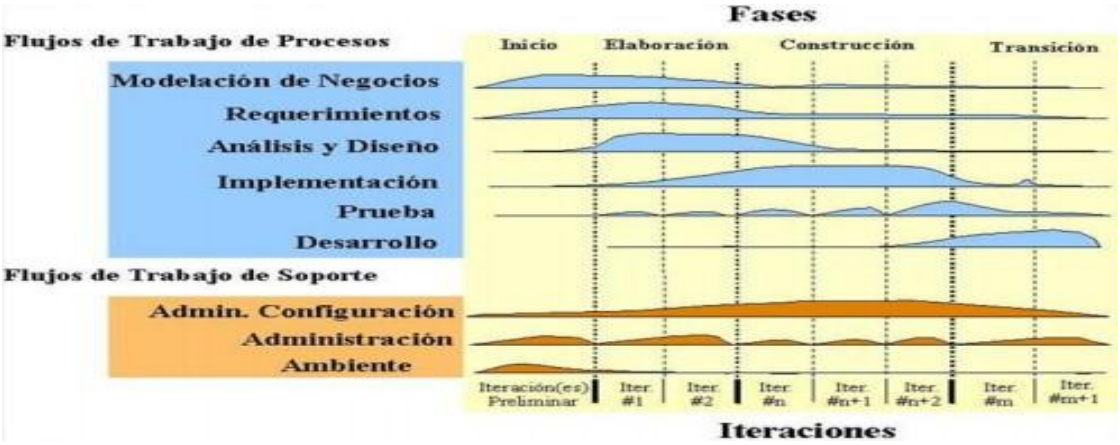


Figura 2: Relación entre las fases y flujos de trabajo de RUP (4)

A partir de lo anteriormente expuesto se decide escoger RUP como metodología de desarrollo, ya que la misma es robusta y está diseñada principalmente para proyectos complejos y de larga duración, como es el caso. Se caracteriza por generar un gran cúmulo de documentación, característica esta muy importante si se tiene en cuenta la complejidad del producto a desarrollar, la gran cantidad de personal con poca experiencia en el desarrollo de software de esta envergadura, además de que el equipo de trabajo puede variar fácilmente. Por tanto es importante tener bien documentado cada paso del proceso, guiando de esta manera también al cliente que no puede estar a tiempo completo durante el desarrollo del sistema.

RUP utiliza UML como lenguaje de notación, siendo una guía para saber cómo utilizar eficazmente este lenguaje en la construcción de los artefactos requeridos.

1.5.2 Lenguaje Unificado de Modelado (UML) 2.0

UML (Unified Modeling Language) por sus siglas en inglés es un lenguaje que permite visualizar, especificar, construir y documentar los artefactos que se crean durante el proceso de desarrollo. No es un método ni una metodología, tampoco es una guía para realizar el análisis y diseño orientado a objetos, o sea no es un proceso, es el lenguaje que permite la modelación de sistemas con tecnología orientada a objetos (aprobado como estándar por OMG[5] en noviembre de 1997). Para

[5] Object Management Group

un mejor resultado en la aplicación de UML son utilizadas con frecuencia las llamadas herramientas CASE. (5)

1.5.3 Herramientas CASE

Las herramientas CASE proporcionan un conjunto de herramientas bien integradas que, enmarcadas dentro de una determinada metodología, permiten automatizar las fases del ciclo de vida de un sistema software. (6)

Esta automatización total busca una mejora en la productividad y en la calidad del producto final, reduciendo el tiempo, coste de desarrollo y mantenimiento de las aplicaciones informáticas. Actualmente existen gran variedad de herramientas CASE, dentro de las que se puede citar Visual Paradigm.

1.5.3.1 Visual Paradigm 8.0

Visual Paradigm for UML es una herramienta CASE que soporta el modelado mediante UML y proporciona asistencia a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del ciclo de vida de desarrollo de un software.

Las ventajas que proporciona Visual Paradigm for UML son:

- **Dibujo.** Facilita el modelado de UML, ya que proporciona herramientas específicas para ello. Esto también permite la estandarización de la documentación, ya que la misma se ajusta al estándar soportado por la herramienta.
- **Corrección sintáctica.** Controla que el modelado con UML sea correcto.
- **Coherencia entre diagramas.** Al disponer de un repositorio común, es posible visualizar el mismo elemento en varios diagramas, evitando duplicidades.
- **Trabajo multiusuario.** Permite el trabajo en grupo, proporcionando herramientas de compartición de trabajo. (7)

Los artefactos desarrollados mediante el modelado con Visual Paradigm constituyen la base para comenzar la implementación de los casos de uso definidos. Para ello se hace necesaria la utilización de marcos de trabajo que propicien y faciliten llevar a buen término la informatización del subproceso Pruebas de Confesión.

1.5.4 Marcos de trabajo

Un marco de trabajo simplifica el desarrollo de una aplicación mediante la automatización de algunos de los patrones utilizados para resolver las tareas comunes. Además, un marco de trabajo

proporciona estructura al código fuente, forzando al desarrollador a crear código más legible y más fácil de mantener. Por último, un marco de trabajo facilita la programación de aplicaciones, ya que encapsula operaciones complejas en instrucciones sencillas. (8)

1.5.4.1 Symfony2

Symfony2 es un marco de trabajo diseñado para optimizar el desarrollo de las aplicaciones web. Para empezar, separa la lógica de negocio, la lógica de servidor y la presentación de la aplicación web. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja. Además, automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación. (8)

Symfony2 está desarrollado completamente con PHP 5. Ha sido probado en numerosos proyectos reales y se utiliza en sitios web de comercio electrónico de primer nivel. Es compatible con la mayoría de gestores de Bases de Datos, como MySQL, PostgreSQL, Oracle y SQL Server de Microsoft. Se puede ejecutar tanto en plataformas *nix (Unix, Linux, etc.) como en plataformas Windows. (8)

Este marco de trabajo cuenta con las siguientes características:

- Sencillo de usar en la mayoría de casos, pero lo suficientemente flexible como para adaptarse a los casos más complejos.
- Sigue la mayoría de mejores prácticas y patrones de diseño para la web.
- Preparado para aplicaciones empresariales y adaptables a las políticas y arquitecturas propias de cada empresa, además de ser lo suficientemente estable como para desarrollar aplicaciones a largo plazo.
- Fácil de extender, lo que permite su integración con librerías desarrolladas por terceros. (8)

Las características antes mencionadas se ajustan perfectamente a las necesidades del SITPC y por tanto a la solución propuesta. Es un marco de trabajo sencillo de utilizar que permite que el equipo de trabajo se familiarice fácilmente con él, al mismo tiempo que puede adaptarse a entornos complejos como es el caso del SITPC que es un proyecto de gran envergadura debido a la complejidad del negocio que se informatiza. Además permite la utilización de servicios desarrollados por terceros, característica muy importante y que posibilita el uso de Acaxia para gestionar la seguridad del sistema, el Generador Dinámico de Reportes (GDR), además del ORM

Doctrine. Symfony2 permite el uso de diferentes patrones, entre ellos el patrón arquitectónico MVC[6] y el patrón Inyección de dependencia que son dos de los utilizados en la solución propuesta. Se decide, a partir de lo anteriormente expuesto, utilizar Symfony2 como marco de trabajo para el desarrollo de la aplicación.

1.5.4.2 Bootstrap 2.1

Bootstrap es un marco de trabajo CSS o herramienta para el desarrollo de aplicaciones y sitios web. Permite desarrollar webs adaptables y fluidas. Además posee una serie de plantillas CSS y ficheros JavaScript que permiten integrar el marco de trabajo de forma sencilla y potente en proyectos web.

Posee las siguientes características:

- Proporciona la facilidad de construir sitios rápidamente y prototipar sitios webs y aplicaciones con un bajo coste.
- Ofrece un diseño sólido usando estándares como CSS3/HTML5.
- Se integra perfectamente con las principales librerías Javascript, por ejemplo JQuery.
- Funciona con todos los navegadores, incluido Internet Explorer usando HTML Shim para que reconozca las etiquetas HTML5. (9)

1.5.4.3 JQuery 1.8

JQuery es una biblioteca JavaScript rápido, pequeño y rico en funciones. Hace las cosas como documento HTML de recorrido y la manipulación, manejo de eventos, animación y Ajax mucho más simple con un API[7] fácil de usar que funciona a través de una multitud de navegadores. (10)
Algunas de sus características fundamentales son:

- Permite localizar partes específicas de la estructura de un documento HTML sin necesidad de tantas líneas de código, ya que posee un robusto y eficiente mecanismo selector para recuperar exactamente la pieza del documento que vaya a ser inspeccionado o manipulado.
- JQuery puede modificar el contenido de un documento en sí mismo con sólo pulsar unas teclas. Todo el texto puede ser cambiado, las imágenes se pueden insertar o permutar, las listas pueden ser reordenadas, o toda la estructura del código HTML puede ser reescrita y ampliada.

[6] Modelo Vista Controlador
[7] *API* (del inglés *Application Programming Interface*) o Interfaz de programación de aplicaciones (IPA).

- La biblioteca jQuery ofrece una forma elegante para interceptar una amplia variedad de eventos, tal como que un usuario haga clic en un enlace, y sin la necesidad de recargar el propio código HTML con controladores de eventos. (11)

Para el desarrollo de la solución propuesta, haciendo uso de los marcos de trabajo anteriormente mencionados, se hace necesario el dominio de ciertos lenguajes de programación que permitan, a través de palabras, símbolos y reglas sintácticas, la correcta implementación de la aplicación.

1.5.5 Lenguajes de programación

Los lenguajes de programación son un conjunto de reglas, herramientas y condiciones que permiten la creación de programas o aplicaciones dentro de una computadora. Poseen reglas acerca de cómo se deben escribir las sentencias. (12)
Pueden usarse para crear programas que controlen el comportamiento físico y lógico de una máquina, para expresar algoritmos con precisión o como modo de comunicación humana. Permite especificar de manera precisa sobre qué datos debe operar una computadora, cómo deben ser almacenados o transmitidos y qué acciones debe tomar bajo una variada gama de circunstancias. (13)

1.5.5.1 Hypertext Preprocessor (PHP) 5.3

PHP, acrónimo de "PHP: Hypertext Preprocessor", es un lenguaje de propósito general y de código abierto que está especialmente pensado para el desarrollo web y que puede ser embebido en páginas HTML. Su sintaxis recurre a C, Java y Perl, y es fácil de aprender. La meta principal de este lenguaje es permitir a los desarrolladores web escribir dinámica y rápidamente páginas web, aunque se puede hacer mucho más con PHP. (14)

Sus características lo hace un lenguaje ideal tanto para el desarrollo web de aplicaciones rápidas y sencillas como para la construcción de sistemas complejos. Entre sus características más relevantes se encuentran:

- Es un lenguaje multiplataforma.
- Permite establecer conexión con distintos gestores de Bases de Datos que se utilizan en la actualidad, destaca su conectividad con MySQL y PostgreSQL.
- Está orientado al desarrollo de aplicaciones web dinámicas que tienen acceso a información almacenada en una Base de Datos.
- En PHP es el servidor quien se encarga de ejecutar el código fuente escrito enviando el resultado HTML al navegador, siendo así invisible este código tanto al navegador como al cliente.

- No requiere definición de tipos de variables.

1.5.5.2 JavaScript

Es un lenguaje de programación que se utiliza principalmente para crear páginas web dinámicas. Técnicamente, 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. (15)

1.5.5.3 HTML

HTML son las siglas de “HyperText Markup Language” (Lenguaje de Marcado de Hipertexto) y es el lenguaje con el que se "escriben" la mayoría de las páginas web. Es un estándar reconocido en todo el mundo y cuyas normas define un organismo sin ánimo de lucro llamado World Wide Web Consortium. (16)

HTML ofrece a los autores los medios para:

- Publicar documentos en línea con encabezados, textos, tablas, listas, fotos.
- Recuperar información en línea a través de enlaces de hipertexto, en el tecleo de un botón.
- Diseñar formularios para realizar transacciones con servicios remotos, para su uso en la búsqueda de información, hacer reservas y pedir productos. (17)

1.5.6 Sistema Gestor de Base de Datos

Los Sistemas de Gestión de Bases de Datos (SGBD) son sistemas de software centralizados o distribuidos que ofrecen facilidades para la definición de Bases de Datos, para la selección de las estructuras de datos necesarias para el almacenamiento y búsqueda de los datos, lo mismo interactivamente que mediante un lenguaje de programación. (18)

1.5.6.1 PostgreSQL 9.2

PostgreSQL es un Sistema Gestor de Bases de Datos de SQL avanzado, disponible en una amplia gama de plataformas y se está convirtiendo rápidamente en uno de los servidores de Bases de Datos más populares del mundo, con una envidiable reputación de rendimiento, estabilidad, y una enorme variedad de características avanzadas. PostgreSQL es uno de los proyectos más antiguos de código abierto, de uso completamente gratuito, y se desarrolló por una comunidad muy diversa en todo el mundo.

PostgreSQL tiene las siguientes características principales:

- Excelente cumplimiento de normas SQL hasta SQL 2008.
- Diseño altamente concurrente en donde los lectores y los escritores no bloquean entre sí.
- Altamente configurable y extensible para muchos tipos de aplicaciones. (19)

1.5.7 Mapeo objeto-relacional

Mapeo objeto-relacional (ORM[8] por sus siglas en inglés), es un método para el modelado de Bases de Datos y las consultas a nivel conceptual. Su notación gráfica suele ser mucho más expresiva que otras notaciones ya que está orientada a los hechos para el modelado de la información. A diferencia del modelado entidad-relación y los diagramas de clases del Lenguaje Unificado de Modelado, el modelado orientado a los hechos es libre de atributos, tratando a todos los hechos elementales como las relaciones. (20)

1.5.7.1 Doctrine 2.0

Doctrine 2 es un mapeador objeto-relacional (ORM) para PHP 5.3 que proporciona persistencia transparente de objetos PHP. El beneficio de Doctrine para el programador se basa en la capacidad de centrarse en la lógica de negocio orientado a objetos y que la preocupación por la persistencia sea un problema secundario. Esto no significa que la persistencia es minimizada por Doctrine 2, sin embargo, existen beneficios considerables para la Programación Orientada a Objetos, si la persistencia y entidades se mantienen separados. (21)

Doctrine cuenta con las siguientes características:

- Cuenta desde el 2006 con un código base muy estable y de alta calidad.
- Posee un mapeador de objetos y consultas extremadamente flexibles y de gran alcance.
- Está integrado con diferentes frameworks como Symfony, Zend Framework, CodeIgniter, FLOW3, Lithium y otros. (22)

1.5.8 Servidor web

Se hace necesaria la utilización de un servidor web que atienda y responda a las diversas peticiones de los distintos navegadores que harán uso de la aplicación, proporcionando los recursos que se solicitan mediante el protocolo HTTP[9] o el protocolo HTTPS[10] (la versión segura, cifrada y autenticada de HTTP).

[8] Object Relational Mapper
[9] Hypertext Transfer Protocol (en español, protocolo de transferencia de hipertexto)
[10] Hypertext Transfer Protocol Secure (en español, protocolo seguro de transferencia de hipertexto)

Un servidor web básico tiene un esquema de funcionamiento muy sencillo, ejecutando de forma infinita el siguiente bucle:

1. Espera peticiones en el puerto TCP[11] asignado (el estándar para HTTP es el 80).
2. Recibe una petición.
3. Busca el recurso en la cadena de petición.
4. Envía el recurso por la misma conexión por donde ha recibido la petición.
5. Vuelve al punto 2.

A partir del esquema anterior se han diseñado y construido todos los programas servidores de HTTP que existen, variando sólo el tipo de peticiones que pueden atender. (23)

Debido a esta necesidad, el equipo de trabajo, luego de un estudio preliminar, decidió utilizar como servidor web HTTP Apache en su versión 2.2.

1.5.8.1 Servidor HTTP Apache 2.2

Apache es un servidor web de código libre, robusto, cuya implementación se realiza de forma colaborativa, con prestaciones y funcionalidades equivalentes a las de los servidores comerciales. El proyecto está dirigido y controlado por un grupo de voluntarios de todo el mundo que, usando Internet y la web para comunicarse, planifican y desarrollan el servidor y la documentación relacionada. (23)

Se decide utilizar HTTP Apache 2.2 como servidor web por sus ventajosas características al ser una herramienta modular, que permite incorporarle nuevas funcionalidades. Además de contar con abundante documentación y ser capaz de soportar varios lenguajes de programación, dentro de los que se encuentran los seleccionados para la solución propuesta, así como el marco de trabajo Symfony2. Cabe destacar también su flexibilidad, rapidez, además de ser una herramienta multiplataforma, características estas que promueven la eficiencia y rendimiento del sistema, el cual debe ser capaz de responder a las peticiones con un nivel aceptable de desempeño.

1.5.9 Control de versiones

Debido a la importancia del subproceso que se desea implementar, se hace necesario el uso de un sistema para el control de versiones.

El control de versiones es el arte del manejo de los cambios en la información. Ha sido durante mucho tiempo una herramienta crítica para los programadores, quienes normalmente empleaban su tiempo haciendo pequeños cambios en el software y después deshaciendo esos cambios al día

[11] Transmisson Control Protocol

siguiente. Pero la utilidad del software de control de versiones se extiende más allá de los límites del mundo del desarrollo de software. (24)

Un sistema de control de versiones permite seguir y controlar los cambios realizados en los ficheros del proyecto, fundamentalmente en el código fuente, la documentación y en las páginas web. Permitiendo además la comunicación entre los desarrolladores, administración de fallos y atribución y autorización en los cambios de los desarrolladores. El sistema para el control de versiones utilizado en el desarrollo de la solución propuesta es el Subversion 1.5.

1.5.9.1 Subversion 1.5

Subversion es un sistema de control de versiones libre y de código fuente abierto que permite manejar ficheros y directorios a través del tiempo. Hay un árbol de ficheros en un repositorio central. El repositorio es prácticamente lo mismo que un servidor de ficheros ordinario, salvo que recuerda todos los cambios que se hayan hecho a sus ficheros y directorios. Esto permite que pueda recuperar versiones antiguas de los mismos y examinar la historia de cuándo y cómo cambiaron sus datos, y quién hizo el cambio. (25)

La utilización de Subversion como sistema de control de versiones para el desarrollo de la aplicación permitirá el acceso al repositorio por parte de los desarrolladores desde distintos ordenadores, pudiendo modificar los mismos datos de forma paralela, garantizando y fomentando de esta manera la colaboración. Además, al mantener todo el trabajo bajo el control de versiones, se evitan las inconsistencias de los datos y permite al equipo de trabajo recuperar versiones más antiguas de un archivo, ver un registro de cambios y realizar otras tareas útiles según lo necesitado.

Analizados los elementos anteriores, para lograr la correcta implementación de la solución propuesta urge entonces definir la arquitectura de software que permita al equipo compartir una misma línea de trabajo, estableciendo la estructura, funcionamiento e interacción entre las diferentes partes del sistema.

1.6 Arquitectura de software

La arquitectura del software es el diseño de más alto nivel de la estructura de un sistema, programa o aplicación y tiene la responsabilidad de definir los módulos principales, definir las responsabilidades que tendrá cada uno de estos módulos y definir la interacción que existirá entre dichos módulos. Aporta una visión abstracta de alto nivel, posponiendo el detalle de cada uno de los módulos definidos a pasos posteriores del diseño. Según la IEEE Std 1471-2000: “La Arquitectura del Software es la organización fundamental de un sistema formada por sus

componentes, las relaciones entre ellos y el contexto en el que se implantarán, y los principios que orientan su diseño y evolución". (26)

1.6.1 Estilos arquitectónicos

Un estilo arquitectónico es una transformación impuesta al diseño de todo un sistema. El objetivo es establecer una estructura para todos los componentes del sistema. En caso de que una arquitectura existente se vaya a someter a reingeniería, la imposición de un estilo arquitectónico desembocará en cambios fundamentales en la estructura del software, incluida una reasignación de la funcionalidad de los componentes. (27)

Según su clasificación, los estilos arquitectónicos existentes son los siguientes:

Estilos de flujo de datos

- Tuberías y Filtros

Estilos centrados en datos

- Arquitecturas de Pizarra o repositorio

Estilos de llamada y retorno

- Modelo Vista Controlador (MVC)
- Arquitectura en Capas
- Arquitectura Orientada a Objetos
- Arquitectura basada en Componentes

Estilo de código móvil

- Arquitectura de Máquinas Virtuales

Estilos Punto a Punto

- Arquitectura basada en Eventos
- Arquitecturas Orientas a Servicios (SOA)

1.6.1.1 Patrón Modelo Vista Controlador (MVC)

La arquitectura MVC separa la lógica de negocio (el modelo) y la presentación (la vista) por lo que se consigue un mantenimiento más sencillo de las aplicaciones. Si por ejemplo una misma aplicación debe ejecutarse tanto en un navegador estándar como en un navegador de un dispositivo móvil, solamente es necesario crear una vista nueva para cada dispositivo; manteniendo el controlador y el modelo original. El controlador se encarga de aislar al modelo y a la vista de los

detalles del protocolo utilizado para las peticiones (HTTP, consola de comandos, email, etc.). El modelo se encarga de la abstracción de la lógica relacionada con los datos, haciendo que la vista y las acciones sean independientes de, por ejemplo, el tipo de gestor de Bases de Datos utilizado por la aplicación. (28)

1.6.1.2 Arquitectura en Capas

La idea de dividir el sistema en capas está basada en una estrategia de asignar responsabilidades por niveles. De esta forma cada uno de los niveles tiene solamente la lógica que le compete, se relacionan entre sí pero el cambio que pueda ocurrir en alguno de ellos no compromete a los demás. (29)

Entre los motivos por los cuales se recurre a la arquitectura multicapas se cuentan los siguientes:

- Aislamiento de la lógica de aplicaciones en componentes independientes susceptibles de reutilizarse después en otros sistemas.
- Distribución de las capas en varios nodos físicos de cómputo y en varios procesos. Esto puede mejorar el desempeño, la coordinación y el compartir la información en un sistema cliente-servidor.
- Asignación de los diseñadores para que construyan determinadas capas; por ejemplo, un equipo que trabaje exclusivamente en la capa de presentación. Así se brinda soporte a los conocimientos especializados en las habilidades de desarrollo y también a la capacidad de realizar actividades simultáneas en equipo. (29)

El equipo de trabajo seleccionó una arquitectura en capas, basada en el patrón arquitectónico Modelo Vista Controlador, siendo este último implementado por el marco de trabajo Symfony2, seleccionado para el desarrollo de la aplicación. Teniendo en cuenta que se definió para un proyecto de gran envergadura, la arquitectura en capas resulta de vital utilidad, puesto que separa el desarrollo de la aplicación en varias capas, siendo esto fundamental en el desarrollo de arquitecturas consistentes, reutilizables y más fáciles de mantener, lo que se traduce en una disminución de tiempo y recursos.

1.6.2 Patrones de diseño

En la constante lucha para mejorar la calidad del software, se han desarrollado varias estrategias, muchas de las cuales son producto de mejoras realizadas sobre estrategias anteriores. Una de estas, y probablemente la más versátil son los patrones de diseño, que consisten en la reutilización de soluciones, no solamente de código fuente, sino, soluciones integrales a problemas repetitivos en el desarrollo de software. (30)

Los patrones GRASP[12] describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. El nombre se eligió para indicar la importancia de captar (grasping) estos principios, si se quiere diseñar eficazmente el software orientado a objetos. Dentro de los patrones que responden a este grupo se pueden mencionar:

- Experto
- Creador
- Alta Cohesión
- Bajo Acoplamiento
- Controlador
- Polimorfismo
- Fabricación Pura
- Indirección
- No Hables con Extraños (29)

Existen 23 patrones GOF[13] que se clasifican según su propósito en Creacionales, Estructurales y de Comportamiento y según su ámbito en Objeto y de Clase. Dentro de sus múltiples ventajas se pueden citar las siguientes:

- Se potencia el encapsulamiento, puesto que se aísla a los clientes de las implementaciones.
- La separación entre interfaz e implementación permite configurar (y cambiar) esta relación dinámicamente, en tiempo de ejecución (y no de compilación). Esta independencia favorece una mejor estructuración en niveles del sistema.
- Mayor flexibilidad, puesto que se puede añadir o modificar la capacidad de atender una petición, simplemente haciendo cambios en la cadena de responsabilidades dinámicamente (en tiempo de ejecución). (31)

En la solución propuesta se utilizan Experto, Creador, Alta Cohesión, Bajo Acoplamiento y Controlador como parte de los patrones GRASP, Fábrica de métodos y Decorador dentro del grupo de los GOF, además del patrón orientado a objetos Inyección de Dependencia.

1.7 Pruebas de software

[12] General Responsibility Asignment Software Patterns (patrones generales de software para asignar responsabilidades)
[13] Gang of Four (grupo de los cuatro)

Las pruebas del software son un elemento crítico para la garantía de calidad del software y representa una revisión final de las especificaciones, del diseño y de la codificación. Una vez generado el código fuente, el software debe ser probado para descubrir (y corregir) el máximo de errores posibles antes de su entrega al cliente. El objetivo es diseñar una serie de casos de prueba que tengan una alta probabilidad de encontrar errores. Como ventaja secundaria la prueba demuestra hasta qué punto las funciones del software parecen funcionar de acuerdo con las especificaciones y parecen alcanzarse los requisitos de rendimiento. Además los datos que se van recogiendo a medida que se lleva a cabo la prueba proporcionan una buena indicación de la fiabilidad del software y de alguna manera indican la calidad del software como un todo. (27)

Pruebas de caja negra

Las pruebas de caja negra también denominadas, pruebas de comportamiento, se concentran en los requisitos funcionales del software. Permiten al ingeniero del software derivar conjuntos de condiciones de entrada que ejercitarán por completo todos los requisitos funcionales de un programa. (27)

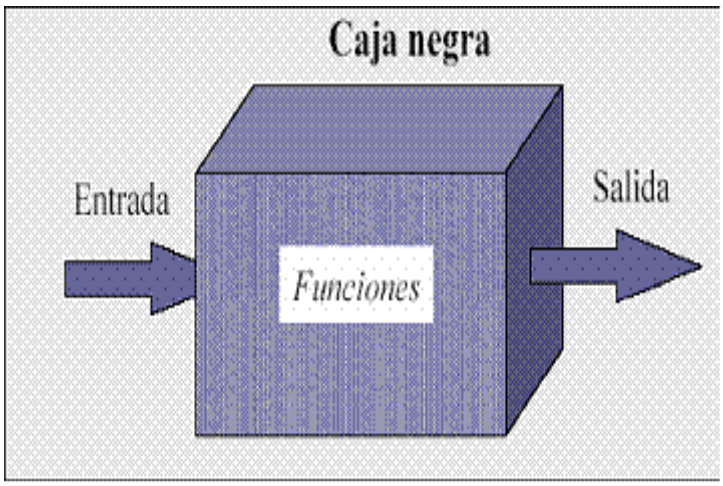


Figura 3: Representación gráfica de las Pruebas de caja negra

Para validar la implementación de la solución propuesta se aplicarán las Pruebas de caja negra, que permitirán probar el comportamiento del software a partir de un conjunto de datos de entrada, todo esto a partir de la técnica de particiones equivalentes. Estas pruebas permitirán ejercitar completamente todos los requisitos funcionales implementados, centrándose en “qué es lo que hace” el sistema pero sin darle importancia a “cómo lo hace”.

1.8 Conclusiones parciales

- El estudio de los conceptos fundamentales asociados al negocio permitió comprender y demostrar la necesidad de continuar con el desarrollo del subproceso Pruebas de Confesión a partir de los requisitos funcionales especificados.
- El análisis del proceso de desarrollo del SITPC y las tecnologías a usar en el mismo permitió sentar las bases desde el punto de vista técnico para las fases de diseño e implementación del subproceso Pruebas de Confesión.
-

CAPÍTULO 2: PROPUESTA DE SOLUCIÓN

2.1 Introducción

En el presente capítulo se describe la propuesta de solución técnica de la investigación. Se realiza un análisis detallado sobre la arquitectura base del sistema a desarrollar. Contempla, además, el Modelo de Diseño donde se analizan los patrones de diseño utilizados, así como los estándares de codificación tenidos en cuenta para el proceso de desarrollo y se modelan los diagramas de clases y los de secuencia, a partir de los requisitos especificados, permitiendo obtener una visión más clara de la implementación del sistema.

2.2 Principales funcionalidades

El diseño y la implementación del sistema requieren tener como base una correcta especificación de los requisitos y casos de uso obtenidos durante el análisis. El subproceso Pruebas de Confesión está compuesto por 16 casos de uso, los cuales se muestran brevemente descritos a continuación:

- **Gestionar prueba de Confesión Judicial:** Comienza cuando el abogado necesita gestionar una prueba de confesión en un escrito de proposición de pruebas. Puede adicionarla, modificarla o eliminarla del escrito.
- **Gestionar pliego de preguntas para confesión:** Permite adicionar, modificar o eliminar un pliego de preguntas en una prueba de confesión.
- **Disponer sobre solicitud de confesión:** El caso de uso se inicia cuando el Juez necesita disponer de una prueba de confesión, admitiéndola o rechazándola.
- **Crear acta de confesión:** Permite a la secretaria registrar los datos para crear el acta de prueba de confesión y la declaración del confesante.
- **Crear acta de confesión en el domicilio:** Permite a la secretaria registrar los datos para crear el acta de prueba de confesión en el domicilio.
- **Declarar pertinencia de preguntas vía informe:** El caso de uso se inicia cuando el Juez necesita declarar la pertinencia de las preguntas a realizarle al confesante.
- **Registrar informe de confesión:** Permite a la secretaria registrar un informe de una prueba de confesión.
- **Registrar solicitud de aclaración de confesión:** Permite al abogado registrar una solicitud de aclaración de una prueba de confesión.

- **Disponer solicitud de aclaración de confesión:** El caso de uso se inicia cuando el Juez necesita disponer sobre una solicitud de aclaración de una prueba de confesión.
- **Registrar confesión judicial por el 262:** El caso de uso se inicia cuando el abogado necesita registrar una prueba de confesión, amparado en el artículo 262 de la LPCALE[14].
- **Disponer sobre la solicitud de confesión por el 262:** El caso de uso se inicia cuando el Juez necesita disponer de una prueba de confesión, admitiéndola, rechazándola o mandándola a subsanar.
- **Señalar confesión por el 262:** El caso de uso se inicia cuando el juez necesita disponer señalamiento para la práctica de una prueba de confesión.
- **Disponer sobre confesión judicial vía informe:** El caso de uso se inicia cuando el Juez necesita disponer sobre la recepción de la prueba de confesión entregada vía informe para mandarla a aclarar o simplemente tenerla por recibida.
- **Crear acta de aclaración de confesión:** Permite a la secretaria registrar los datos para crear el acta de aclaración de una prueba de confesión que haya sido realizada en el domicilio, a una persona jurídica o en otro tribunal que no es en el que se está llevando a cabo el proceso judicial.
- **Disponer sobre preguntas antes de la confesión:** El caso de uso se inicia cuando el Juez necesita disponer sobre las preguntas antes de la confesión.
- **Crear resolución teniendo el informe de aclaración de confesión:** El caso de uso se inicia cuando el Juez necesita disponer sobre las preguntas antes de la confesión.

2.3 Arquitectura base

El desarrollo del SITPC responde a una arquitectura en capas, implementando el patrón arquitectónico Modelo-Vista-Controlador (MVC), el cual viene integrado a Symfony2 que fue el marco de trabajo seleccionado para la implementación de la aplicación. La arquitectura está compuesta por varias capas: la capa vista, la capa controladora, la capa del modelo (compuesta por la capa de negocio y la capa de acceso a datos) y la capa de datos. En la Figura 4 se observa cómo se encuentran estructuradas y relacionadas cada una de estas capas. Se evidencian además varios complementos que son transversales, garantizando así seguridad, tratamiento de excepciones y validaciones.

[14] Ley de Procedimiento Civil, Administrativo, Laboral y Económico

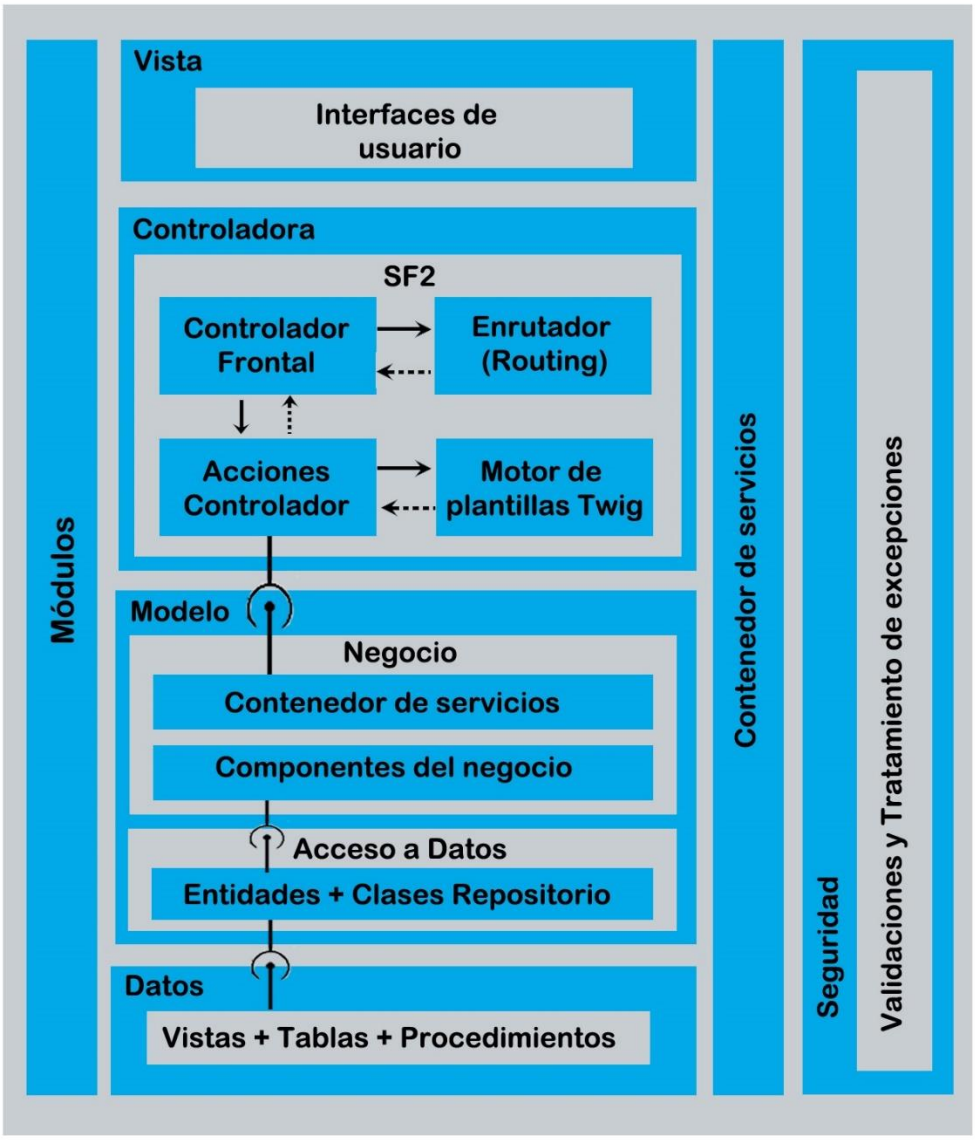


Figura 4: Arquitectura correspondiente al SITPC

Capa vista

Esta capa permite la interacción del usuario con el sistema, capturando o mostrando la información que este introduce o solicita respectivamente mediante las interfaces de usuario, las cuales deben ser amigables, o sea, entendibles y fáciles de usar. Además contiene todo lo referente a la visualización de la información, el diseño, los colores, los estilos y la estructura visual de las páginas. La implementación de esta capa se encuentra ubicada en el paquete *Resources* de cada uno de los módulos, específicamente en el paquete *views,* donde se encuentran todas las páginas

de la aplicación, las cuales importan las imágenes, archivos CSS y JavaScript necesarios, como lo muestra la Figura 5

En el desarrollo de SITPC aprovechando la herencia de plantillas que permite twig, se crea una herencia a tres niveles desde la arquitectura base general donde se crea la plantilla *base.html.twig*, que define las áreas de trabajo en sentido general, la *plantilla_SIT.html.twig*, que hereda de la primera e incluye los menú y funcionalidades en dependencia de cada materia y otras especificaciones y las del tercer nivel que son las de cada uno de los módulos que solo implementan el contenido del área de trabajo.

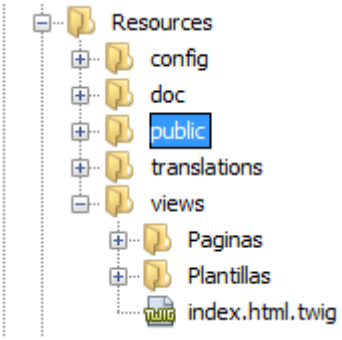


Figura 5: Contenido de las carpetas Resources y views

Capa controladora

En esta capa se maneja la lógica de control, así como la construcción de páginas y formularios. Está compuesta por el Controlador Frontal de Symfony2, el cual maneja todas las peticiones realizadas por el usuario al ser el único punto de entrada a la aplicación en un entorno determinado. La petición del usuario es enviada por el controlador frontal al enrutador o routing, este se encarga de enviar la ruta correspondiente, indicando así cuál es el controlador responsable de realizar la acción. Este va a acceder a la capa de negocio y va a mostrar la información correspondiente a través de las vistas, haciendo uso del motor de plantillas Twig. Cada uno de estos controladores se encuentra ubicado en la carpeta *Controller* como muestra la Figura 6:

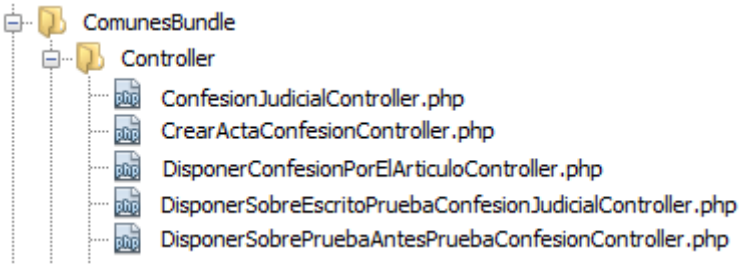


Figura 6: Contenido de la carpeta Controller

Capa de negocio

Esta capa está compuesta por los componentes del negocio, donde se encuentran las clases gestoras, encargadas del manejo de la lógica de negocio y ubicadas en la carpeta *Gestor* como muestra la Figura 7. El acceso a estas clases gestoras por parte del controlador no se realiza directamente, para ello se hace uso del contenedor de servicios de Symfony2.

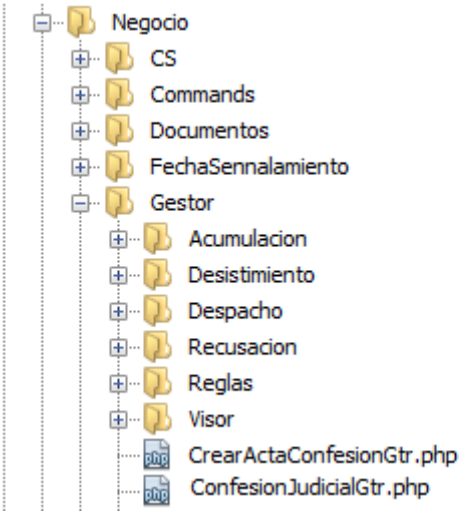


Figura 7: Contenido de la carpeta Gestor

Capa de acceso a datos

Esta capa gestiona las peticiones de la capa de negocio al consultar la Base de Datos y devolver los datos requeridos al gestor correspondiente. Está compuesta por las entidades y los repositorios. Las clases repositorios contienen un conjunto de consultas para acceder y realizar acciones sobre los datos de tablas específicas de la Base de Datos. Es importante destacar que aunque las entidades del negocio se encuentran contenidas en esta capa, también pueden ser accedidas desde las capas superiores.

Capa de datos

Esta capa contiene los esquemas, tablas, vistas y procedimientos, que garantizan el almacenamiento y persistencia de la información utilizada por la aplicación.

Seguridad

El sistema Acaxia es el que se encarga de la seguridad en la aplicación, garantizando Autenticación, Autorización, Auditoría, Administración de perfil y Administración de conexiones. La Autenticación garantiza la fortaleza de la contraseña estableciendo parámetros a cumplir por la misma, como longitud (más de 8 caracteres), vencimiento de la contraseña cada 90 días, tenencia

de mayúscula, números y caracteres especiales. La Autorización, permite gestionar los permisos para cada uno de los usuarios que accedan a la aplicación, teniendo en cuenta el principio de mínimo privilegio. Para realizar la Auditoría se hace uso del control de trazas, permitiendo saber que usuario se autenticó en el sistema, el día y el tiempo que estuvo conectado, así como las acciones realizadas por el mismo. La Administración de perfil permite al usuario la personalización de su perfil en el sistema, posibilitándole modificar datos personales, así como el cambio de su contraseña cada cierto tiempo.

Validaciones

Las validaciones se realizan tanto del lado del cliente como del lado del servidor, para asegurar la entrada de valores correctos a la aplicación, garantizando un correcto funcionamiento de la misma.

Tratamiento de excepciones

El tratamiento de excepciones lo brinda Symfony2 de forma interna, se encuentra de forma transversal en toda la aplicación pues en cualquiera de las capas se puede lanzar una excepción, que la clase del núcleo la captura y remite a un controlador que la procesa y determina que plantilla de error mostrar. En la arquitectura propia del SITPC lo que se realizó fue adaptar la plantilla de error por defecto a mostrar, la cual se encuentra en el directorio: *SIT\app\Resources\TwigBundle\views\Exception\error.html.twig*.

Contenedor de servicios

El contenedor de servicios es un objeto especial de PHP para Symfony2, el mismo permite que las respuestas de la aplicación sean más rápidas, potencia la reutilización de código, además de que estandariza y centraliza la forma de construir objetos en toda la aplicación. En la solución propuesta los servicios se encuentran declarados y publicados en el archivo *services.yml* de cada uno de los módulos, los cuales pueden ser instanciados desde cualquier clase controladora haciendo uso del atributo *container*.

2.4 Modelo de diseño

El modelo de diseño es un modelo de objetos que describe la realización física de los casos de uso, centrándose en cómo los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tienen impacto en el sistema a considerar. Además, el modelo de diseño sirve de abstracción de la implementación del sistema y es, de ese modo, utilizada como una entrada fundamental de las actividades de implementación. (3)

2.4.1 Patrones de diseño

Experto: Es un patrón que se usa al asignar responsabilidades; es un principio básico que suele utilizarse en el diseño orientado a objetos. Con él no se pretende designar una idea oscura ni extraña; expresa simplemente la "intuición" de que los objetos hacen cosas relacionadas con la información que poseen. Al aplicarlo 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. (29)

En la solución propuesta para la investigación se evidencia el uso de este patrón, específicamente en las clases controladoras, como es el caso de *ConfesionJudicialController.php* al crear una vista. Se utiliza además en las clases gestoras, por ejemplo en *ConfesionJudicialGtr.php*, puesto que la misma contiene toda la información correspondiente a las entidades y es la encargada de gestionar toda la información necesaria acerca de ellas.

Creador: 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 se debe conectar con el objeto producido en cualquier evento. Al escogerlo como creador, se da soporte al bajo acoplamiento. (29)

La utilización de este patrón se evidencia en la clase controladora *ConfesionJudicialController.php* a la hora de crear los formularios y en la clase gestora *ConfesionJudicialGtr.php* para la creación de las entidades.

Bajo Acoplamiento: Es un principio que se debe recordar durante las decisiones de diseño: es la meta principal que es preciso tener presente siempre. Es un patrón evaluativo que el diseñador aplica al juzgar sus decisiones de diseño. 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. (29)

En la solución propuesta se pone de manifiesto el bajo acoplamiento, ya que las entidades son las clases más reutilizadas, y al analizar la arquitectura del sistema se observa que las vistas se comunican con las clases controladoras, estas a su vez se comunican con las clases gestoras y son estas últimas las que establecen relación con las entidades, por tanto las entidades no tienen relación directa ni con las vistas ni con las clases controladoras.

Alta Cohesión: Como el patrón Bajo Acoplamiento, también Alta Cohesión es un principio que se debe tener presente en todas las decisiones de diseño: es la meta principal que ha de buscarse en

todo momento. Es un patrón evaluativo que el desarrollador aplica al valorar sus decisiones de diseño. Una clase con mucha cohesión es útil porque es bastante fácil darle mantenimiento, entenderla y reutilizarla. Su alto grado de funcionalidad, combinada con una reducida cantidad de operaciones, simplifica el mantenimiento y los mejoramientos. La ventaja que significa una gran funcionalidad también soporta un aumento de la capacidad de reutilización. (29)

Cada una de las clases de la solución propuesta tiene bien definidas cuáles son sus responsabilidades, permitiéndoles trabajar en conjunto para dar solución a las funcionalidades requeridas bajo el principio de la Alta cohesión

Controlador: La mayor parte de los sistemas reciben eventos de entrada externa, los cuales generalmente incluyen una interfaz gráfica para el usuario (IGU) operado por una persona. Otros medios de entrada son los mensajes externos -entre ellos un conmutador de telecomunicaciones para procesar llamadas- o las señales procedentes de sensores como sucede en los sistemas de control de procesos. En todos los casos, si se recurre a un diseño orientado a objetos, hay que elegir los controladores que manejen esos eventos de entrada. Este patrón ofrece una guía para tomar decisiones apropiadas que generalmente se aceptan. (29)

En el sistema este patrón se evidencia en las clases controladoras y en el controlador frontal de Symfony2. Todas las peticiones realizadas por el usuario son manejadas por el controlador frontal que se encuentra ubicado en el directorio "*SIT/web/app.php*" siendo este el único punto de entrada a la aplicación en un entorno determinado. Se encarga de recibir los datos que el usuario introduce al sistema y enviarlos a las distintas clases de acuerdo a la tarea que se desea realizar. Su funcionamiento está basado en que la lógica de negocios debe estar separada de la capa de presentación y así aumentar la reutilización de código y tener un mayor control sobre los cambios. En el sistema todas las clases controladoras se encuentran dentro de una carpeta llamada *Controller* ubicada en la siguiente dirección "*SIT/src/SIT/SRV/ComunesBundle/Controller*".

Fábrica de Método: Symfony2 brinda una forma confiable para manejar la creación de objetos, así como una manera sencilla de especificar los parámetros necesarios para la creación de estos, en ocasiones se hace necesario más, es por esto que se hace uso de una fábrica que se encargue de tal fabricación (creación) y decir al contenedor de servicios que implementa Symfony2 que haga uso de los métodos de dicha fábrica en vez de instanciar al propio objeto. Estos servicios son publicados en archivos a través de los cuales el contenedor de servicios se encarga de brindar una instancia de cada uno de los objetos asociados a estos.

El EntityManager[15] implementado por Doctrine hace uso de la fábrica de método para proporcionar una instancia de las clases repositorios asociadas a cada una de las entidades, que son las encargadas del proceso de búsqueda y filtrado relacionadas con la Base de Datos. Las imágenes siguientes ilustran cómo se evidencia lo explicado anteriormente en el código de la solución propuesta.

```
28  class ConfesionJudicialGtr extends BaseGtr {
29
30      public function getTiposProcedimiento() {
31          $repo = $this->getEm()->getRepository('ComunBundle:AG\TipoProcedimiento');
32          return $repo->findAll();
33      }
```

Figura 8: Aplicación de la Fábrica de método en la clase gestora ConfesionJudicialGtr

```
755  comun.servicios.Tribunal:
756          class: Base\ComunBundle\Repository\TipoProcedimientoRepository
757          factory_service: doctrine.orm.default_entity_manager
758          factory_method: getRepository
759          arguments: [Base\ComunBundle\Entity\AG\TipoProcedimiento]
```

Figura 9: Declaración de servicios

Decorador: Este patrón responde a la necesidad de añadir responsabilidades a una o más clases de forma dinámica. Aporta una mayor flexibilidad que la herencia estática, permitiendo, entre otras cosas, añadir una funcionalidad dos o más veces. Además evita concentrar en lo alto de la jerarquía clases “guiadas por las responsabilidades”, es decir, que pretenden (en vano) satisfacer todas las posibilidades. De esta forma las nuevas funcionalidades se componen de piezas simples que se crean y se combinan con facilidad, independientemente de los objetos cuyo comportamiento extienden. (31)

En Symfony2 el motor de plantillas twig implementa la herencia de plantillas, a través de la cual es posible utilizar el patrón decorador; permitiendo la declaración de manera general de la organización del sistema y definiendo los bloques que podrán ser redefinidos por sus descendientes. De esta forma puede ser definida la estructura del sistema en la plantilla *SIT/app/Resources/views/base.html.twig*, dejando a las plantillas descendientes de esta la redefinición de los bloques que competen en cada nivel. Las plantillas de cada uno de los diferentes módulos heredan de la plantilla base y se encargan de redefinir el menú lateral izquierdo definido como bloque en la plantilla heredada. A su vez cada plantilla utilizada en los casos de uso

[15] Administrador de entidades

hereda de las plantillas respectivas de sus módulos y son las encargadas de la redefinición del bloque contenido, también declarado desde la base.

En la Figura 10 es posible ver los bloques que se redefinen en cada uno de los niveles de herencia. Señalado en color verde se observa la definición de la plantilla base, de color rojo el bloque redefinido por las plantillas de cada módulo y en color azul las plantillas de cada caso de uso.

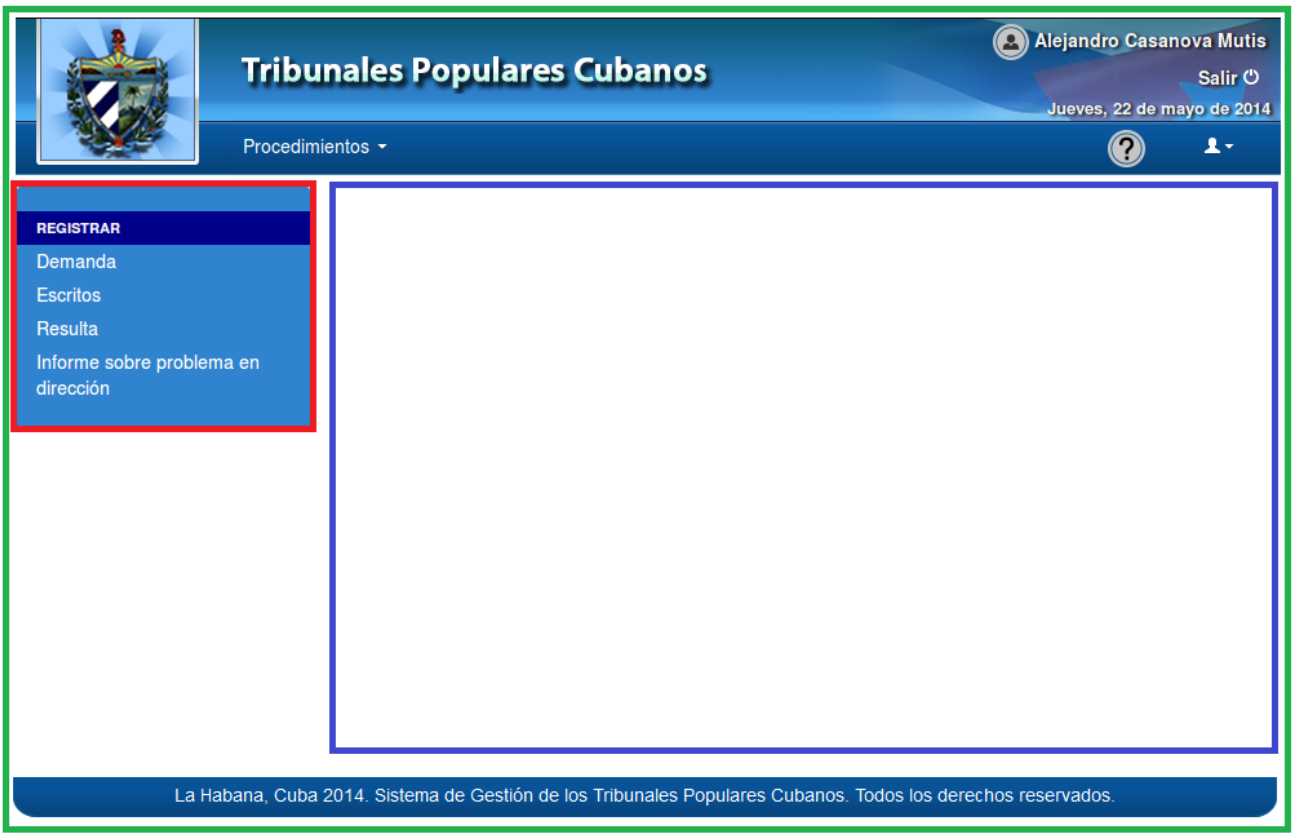


Figura 10: Uso del patrón decorador en la solución propuesta

Inyección de dependencia: La inyección de dependencias se basa en que, en vez de que una clase vaya a buscar los objetos que necesita llamando a métodos que se los proporcionan, se hace al revés, siendo estos últimos los que se asignan automáticamente al objeto. (32)

Symfony2 en su estructura de bundles proporciona un archivo llamado *service.yml*, en el cual se declaran y describen los servicios que pertenecen al bundle, así como sus dependencias. Este fichero está estructurado en dos secciones: *parameters* y *services*. En la primera de estas secciones se declaran los parámetros de configuración y en la segunda, los propios servicios.

```
parameters:
    comunes.confesionjudicialgtr.class: SIT\SRV\ComunesBundle\Negocio\Gestor\ConfesionJudicialGtr

services:
    comunes.confesionjudicialgtr:
        class: %comunes.confesionjudicialgtr.class%
        parent: arquitectura.basegtr
```

Figura 11: Declaración de servicios en el archivo services

La clase *Controller.php* del marco de trabajo, posee un atributo público llamado *container*, que no es más que una instancia del contenedor de servicios de Symfony2. De esta forma desde cualquiera de las clases que hereden de esta se puede acceder a una instancia de dicho contenedor de dependencias, lo que permite instanciar cualquier servicio previamente declarado haciendo uso del método *get()*.

```
class ConfesionJudicialController extends BaseController {

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

Figura 12: Uso de la Inyección de dependencia en el controlador

2.4.2 Estándares de codificación

En el desarrollo del sistema se utilizó el estándar de codificación del lenguaje de programación PHP en conjunto con el definido en las pautas que se presentan a continuación.

Identación

El contenido siempre se identó 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 de cada equipo decidir si se quieren o no agregar más datos.

/**

*

* @Clase controladora acusado. "*AcusadoController.php*"

* @modificado: 1 de Septiembre del 2012

* @autor: Freeman

*

*/

- **Comentarios en las funciones**

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

- **Ubicación y denominación de archivos**

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

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

Para las clases de gestión del negocio se usó el sufijo Gtr: AcusadoGtr

Para los table model definidos para los grid se usó el sufijo Tm: AcusadoTm

Para las entidades de presentación se usó el sufijo Ep: AcusadoEp

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: *Acusados.html.twig*

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

- **Clases**

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

Siempre se utilizaron las etiquetas <?php ?> para abrir un bloque de código. Nunca usando el método de etiquetas cortas.

Estilo y reglas de escritura de código PHP

- **Nombres de variables**

Los nombres deben ser descriptivos y concisos. No se usaron ni grandes frases ni pequeñas abreviaciones para las variables. Siempre es mejor saber qué hace una variable con sólo conocer su nombre. Esto aplicó 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 se inició con letra mayúscula. Las constantes se escribieron siempre en mayúsculas y tanto estas como las variables globales tienen como prefijo el nombre de la clase a la que pertenecen.

- **Las definiciones de la función**

Los nombres de la función contienen sólo caracteres alfanuméricos. Los nombres de la función siempre se empezaron en letras minúsculas. Cuando un nombre de la función consistía de más de una palabra, la primera letra de cada nueva palabra se capitalizó. Ejemplo:

```
listarUsuariosAction() {
        instrucciones
}
```

- **Llamadas a funciones**

Se llamaron 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. Ejemplo:

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

- **Siempre incluir las llaves**

En todo momento a la hora de codificar un bloque de instrucciones, éste se encerró entre llaves, aún cuando conste de una sola línea. Ejemplo:

```
if ($cosa) {
        function();
}
```

- **Llaves. ¿Dónde colocarlas?**

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

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

- **Poner espacios entre signos**

En el caso de tener un signo y operador binario, se colocó espacios a ambos lados. De ser un signo unario, se colocó espacios a uno de sus lados. En términos más simples, se programó como si se escribiera bien en español. Este elemento es algo muy sencillo que ayuda a la legibilidad del código. Ejemplo:

```
$a = 0;
for ($i = 5; $i <= $j; $i++)
```

- **Precedencia de operadores**

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

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

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

- **No utilizar variables sin inicializar**

Si no se tenía control sobre el valor de una variable, se verificó que esta estuviera inicializada. Esto lo permite PHP de la siguiente manera. Ejemplo:

```
if (isset($cliente) && $cliente == 5)
```

Sólo se usó esta opción cuando no se tenía el control o no se estaba seguro del valor que podía tener la variable (Como en variables que llegan por parámetro o por GET, etc.).

- **Instrucción "switch"**

Cuando se hizo uso de ella, se intentó seguir el siguiente estilo:

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

2.4.3 Diagramas de clases

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. A diferencia del modelo conceptual, un diagrama de este tipo contiene las definiciones de las entidades del software en vez de conceptos del mundo real. El UML no define concretamente un elemento denominado

"diagrama clases del diseño", sino que se sirve de un término más genérico: "diagrama de clases". (29)

El Anexo 5 muestra el diagrama de clases correspondiente al caso de uso Gestionar prueba de Confesión Judicial. A continuación se presentan fragmentos del mismo donde se evidencian cada una de las clases que intervienen en el caso de uso y las relaciones entre ellas.

La Figura 13 representa las clases de la vista que intervienen en el caso de uso antes mencionado y las relaciones que se establecen entre ellas. La clase *ConfesionJudicial.html.twig* incluye las clases *GridCrud.html.twig* y *GridCrudOffline.html.twig* y hace uso de *PruebaConfesionJudicialType.php* y *DireccionPType.php* las cuales heredan de *AbstractType.php*. Al mismo tiempo *ConfesionJudicial.html.twig* establece una relación de uso con la clase *ConfesionJudicialController.php*.

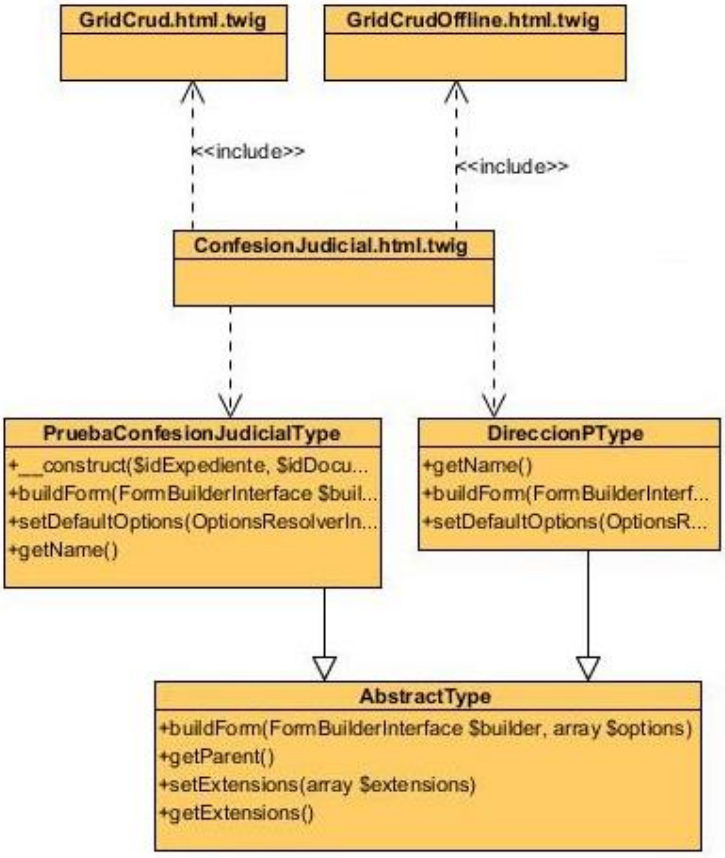


Figura 13: Clases de la vista pertenecientes al caso de uso Gestionar prueba de Confesión Judicial

La Figura 14 muestra las clases controladoras que intervienen en el caso de uso Gestionar prueba de Confesión Judicial. La clase *ConfesionJudicialController.php* hereda de *BaseController.php* y establece relaciones de uso con *ConfesionJudicial.html.twig* y *ConfesionJudicialGtr.php*.

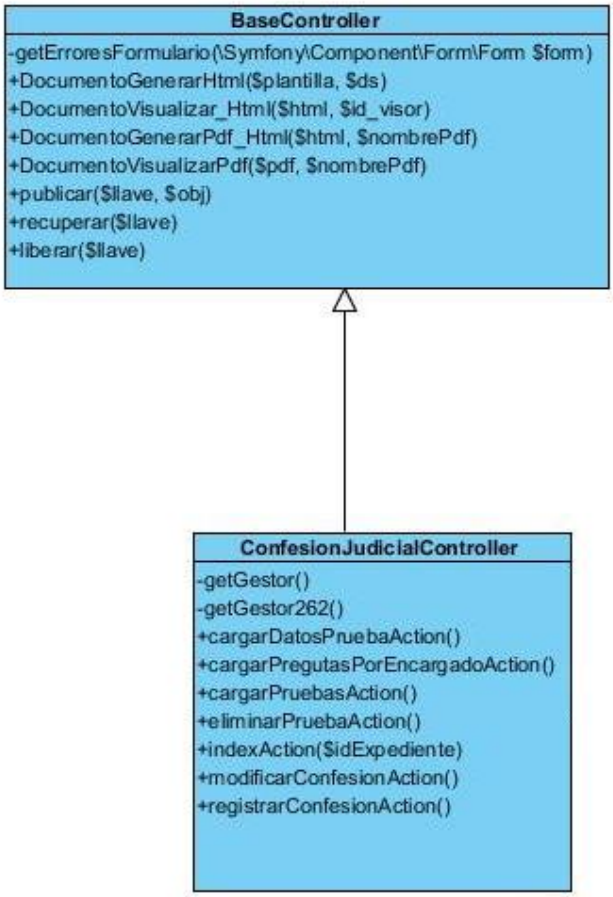


Figura 14: Clases controladoras pertenecientes al caso de uso Gestionar prueba de Confesión Judicial

La Figura 15 representa las clases gestoras y entidades que intervienen en el caso de uso Gestionar prueba de Confesión Judicial. La clase *ConfesionJudicialGtr.php* hereda de *BaseGtr.php* y establece relaciones de uso con las clases entidades correspondientes.

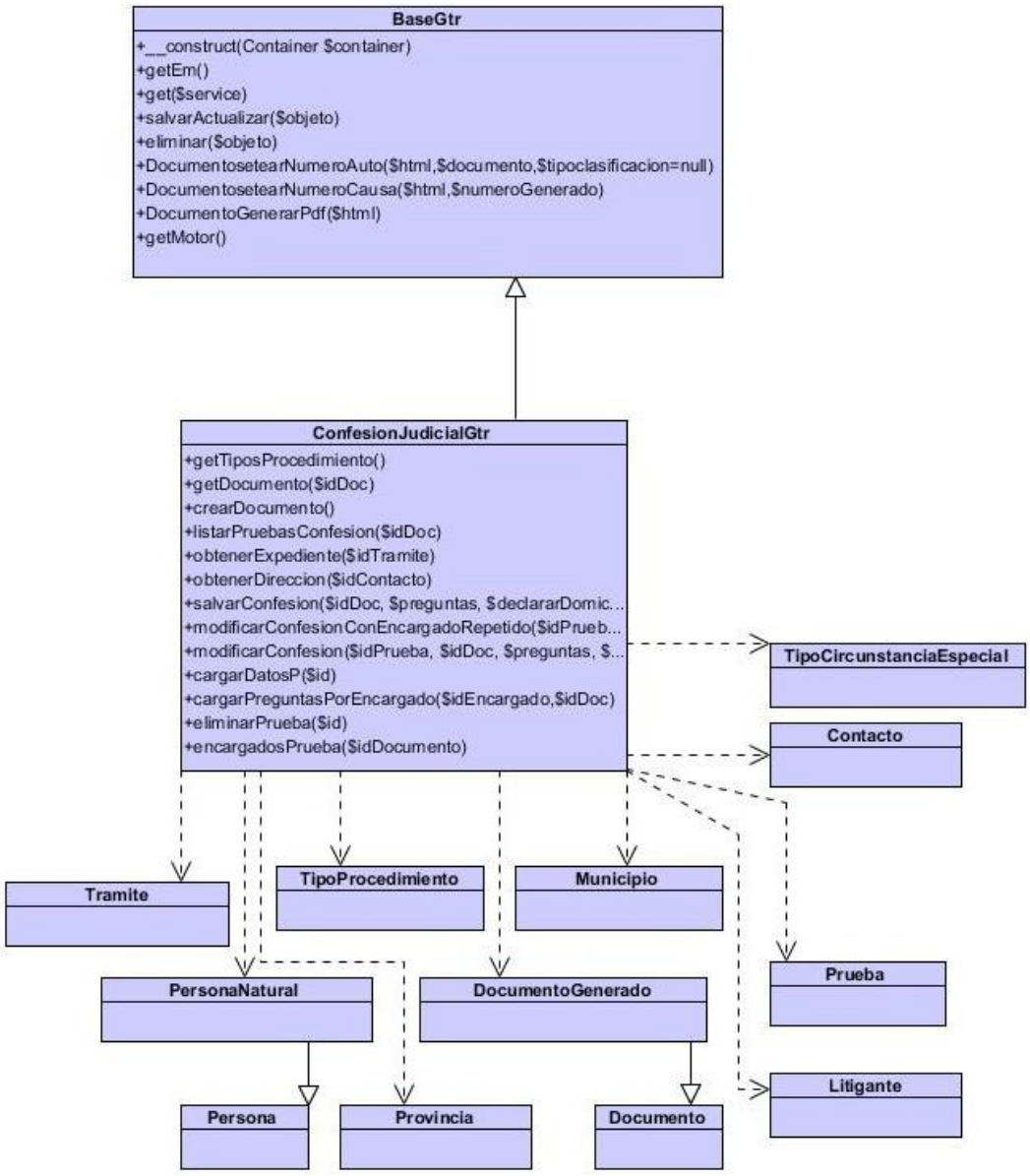


Figura 15: Clases del negocio pertenecientes al caso de uso Gestionar prueba de Confesión Judicial

2.4.4 Diagramas de secuencia

Un diagrama de secuencia muestra un conjunto de mensajes, dispuestos en una secuencia temporal. Cada rol en la secuencia se muestra como una línea de vida, es decir, una línea vertical que representa el rol durante cierto plazo de tiempo, con la interacción completa. Los mensajes se muestran como flechas entre las líneas de vida. Un diagrama de secuencia puede mostrar un escenario, es decir, una historia individual de una transacción. (29)

El Anexo 6 muestra el diagrama de secuencia correspondiente al caso de uso Gestionar prueba de Confesión Judicial, específicamente el escenario Adicionar.

2.4.5 Diagrama de despliegue

El diagrama de despliegue se utiliza para modelar la distribución física del hardware para desplegar el sistema y las relaciones entre sus componentes. Está compuesto por nodos unidos por conexiones de comunicación, donde un nodo es un recurso de ejecución, desde un computador hasta un dispositivo o memoria y se conectan por asociaciones de comunicación tales como enlaces de red, conexiones TCP/IP[16].

El diagrama de despliegue correspondiente al SITPC, el cual está compuesto en cada una de las instancias de los TPC (Tribunal Municipal Popular (TMP), Tribunal Provincial Popular (TPP) y Tribunal Supremo Popular (TSP)) por Computadoras Personales (PC) clientes conectadas a través del protocolo HTTPS a un servidor web, este a su vez se encuentra conectado mediante la extensión PDO[17] a un servidor de Bases de Datos. Además se cuenta con una impresora conectada por USB o el protocolo TCP/IP a la PC cliente. Cada uno de los servidores de Bases de Datos establecerá conexión con un Centro de datos a través de los protocolos TCP/IP, UDP[18] o FTP[19], enviando a este toda la información que se procese en cada una de las instancias y obteniendo toda la información que necesite. La Figura 16 muestra el diagrama de despliegue realizado para el SITPC donde se representa la distribución física de los nodos, que se repite para las diferentes instancias de los tribunales.

Descripción de los nodos físicos:

Los nodos son los elementos de hardware que soportarán el software del sistema desarrollado, a continuación se describen los nodos que aparecen en la Figura 16:

- **Servidor de Base de Datos (ubicado en el Centro de Datos):** En este servidor se encontrará la Base de Datos que contendrá la información referente a cada una de las instancias de los tribunales.
- **Servidor de Base de Datos (ubicado en los tribunales Supremo, Provinciales y Municipales):** En este servidor se encontrará la Base de Datos que contendrá la información referente a la instancia que corresponda. Se sincronizará con el servidor del Centro de Datos en horario no laboral para enviar la información diaria.

[16] Transmisson Control Protocol/ Internet Protocol
[17] PHP Data Object
[18] User Datagram Protocol
[19] File Transfer Protocol

- **Servidor de aplicaciones:** Contendrá todo lo referente a la aplicación, incluyendo los archivos que sean necesarios para que los usuarios puedan tener acceso a esta. Tendrá almacenada la configuración general de sistema, bibliotecas externas y los archivos de instalación de la aplicación.
- **PC cliente:** Tendrá instalado el sistema operativo Linux y el navegador web Mozilla Firefox, a través del cual los usuarios podrán acceder a la aplicación y hacer uso de esta.
- **Impresora:** Dispositivo de hardware necesario para imprimir los diferentes documentos que se generan como resultado de las distintas tramitaciones en los tribunales y que son necesarios para el funcionamiento normal de estos.

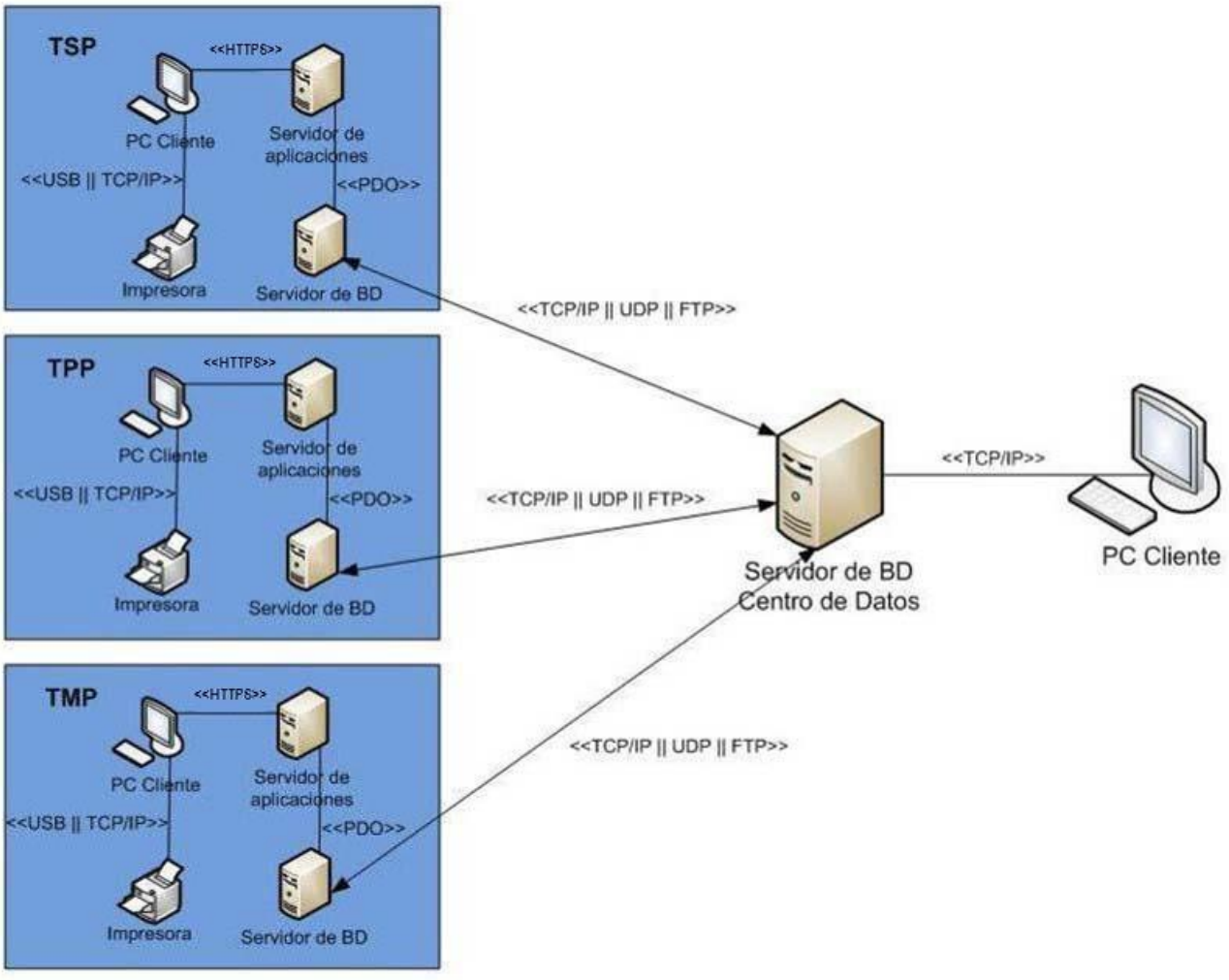


Figura 16: Diagrama de despliegue para el SITPC

2.4.6 Interfaces de la aplicación

A continuación se presentan algunas de las interfaces de la aplicación correspondientes a los casos de uso de alta prioridad:

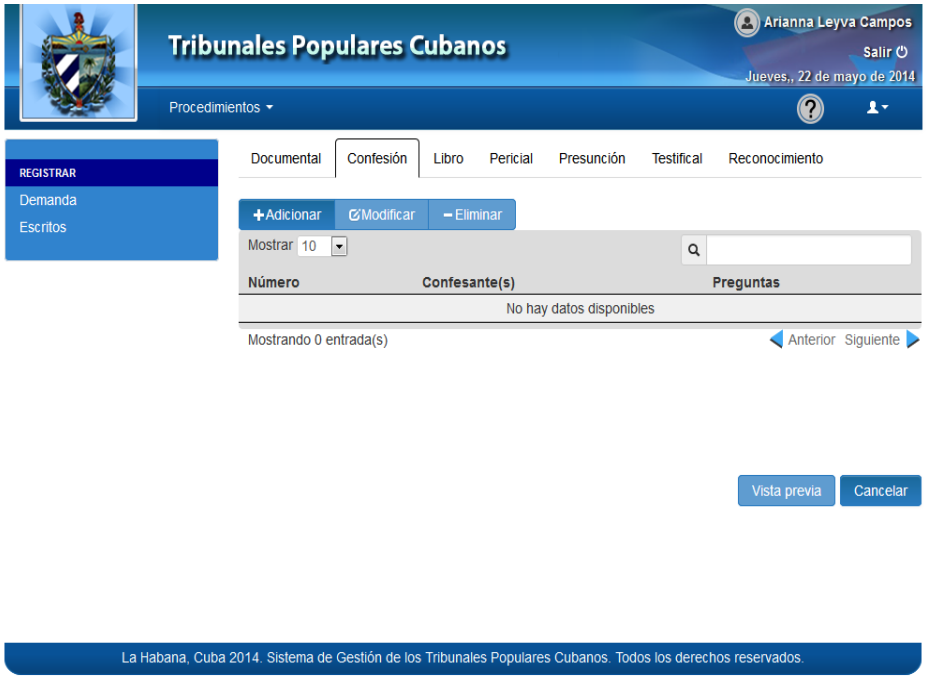


Figura 17: Interfaz correspondiente al caso de uso Gestionar prueba de Confesión Judicial

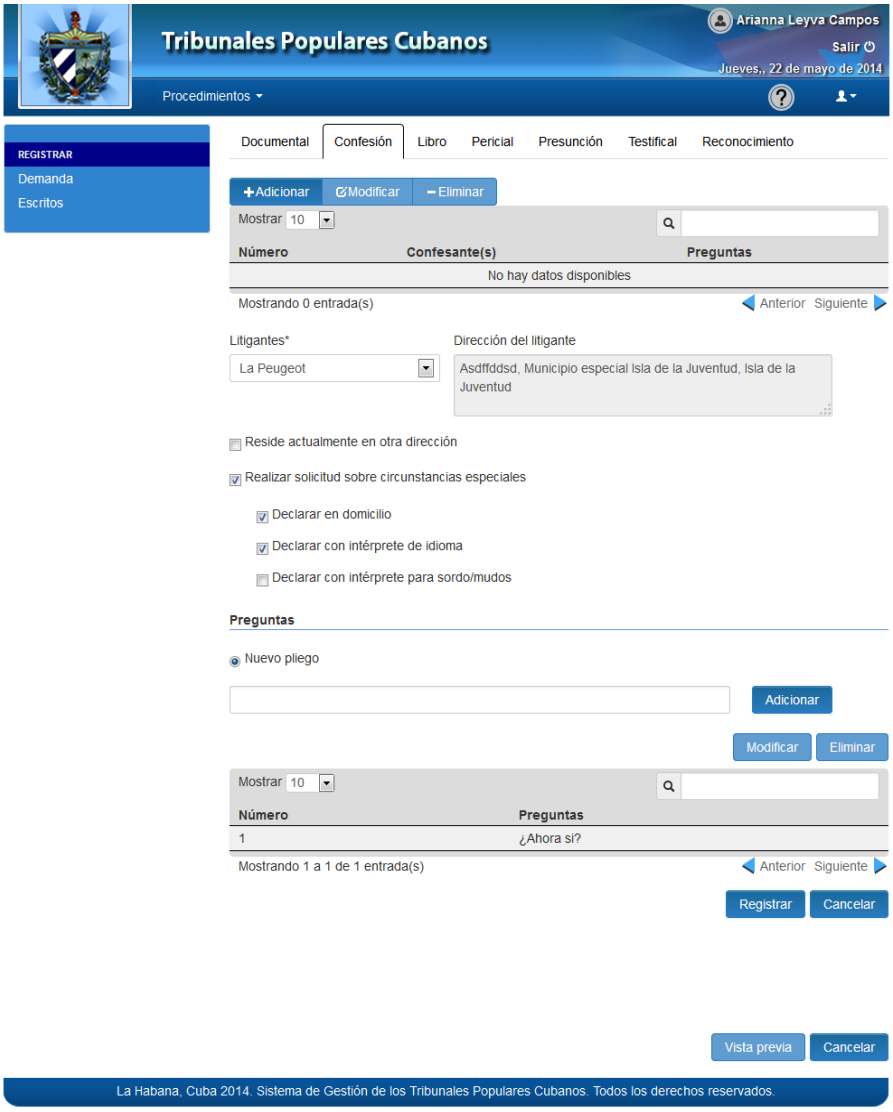


Figura 18: Interfaz correspondiente al caso de uso Gestionar prueba de Confesión Judicial

2.5 Conclusiones parciales

- La descripción de la arquitectura del sistema y el desarrollo de los diagramas de clases, secuencia y despliegue, posibilitó establecer una guía para lograr la correcta implementación del subproceso Pruebas de Confesión, teniendo siempre en cuenta los patrones de diseño definidos, así como los estándares de implementación.
- Se obtuvo la propuesta de solución, permitiendo sentar las bases para realizar posteriormente la validación del sistema en cuanto a diseño e implementación.

CAPÍTULO 3: VALIDACIÓN

3.1 Introducción

En el presente capítulo se realiza la validación del diseño y la implementación de la solución propuesta, a partir de la aplicación de las métricas seleccionadas para evaluar el diseño y la ejecución de los casos de prueba para determinar el correcto funcionamiento de la aplicación. Se muestra el análisis realizado a los resultados obtenidos en la aplicación de las métricas y la ejecución de las pruebas de funcionalidad.

3.2 Validación del diseño

Para validar el diseño de la solución propuesta, se estudiaron las diferentes métricas de diseño y sus características. Se seleccionaron cuatro métricas que permiten medir el nivel de relaciones entre las clases y el nivel de complejidad de cada clase por separado. Estas métricas son: Relaciones entre Clases (RC), Tamaño de Clase (TC), Carencia de Cohesión en los Métodos (CCM) y Árbol de Profundidad de Herencia (APH).

3.2.1 Métricas de diseño

Las métricas del producto para el software de computadora, proporcionan una manera sistemática de evaluar la calidad a partir de un conjunto de reglas definidas con claridad. También proporcionan al ingeniero de software información inmediata y en el sitio; no posterior al hecho. Esto permite al ingeniero descubrir y corregir problemas potenciales antes de que se conviertan en defectos catastróficos. (27)

Relaciones entre Clases (RC)

La métrica Relaciones entre Clases se basa en el número de relaciones de uso que se establecen entre una clase y las demás clases existentes. Se evalúa a partir de los siguientes atributos de calidad:

- Acoplamiento: Un aumento del RC implica un aumento del acoplamiento de la clase.
- Complejidad de Mantenimiento: Un aumento del RC implica un aumento de la complejidad del mantenimiento de la clase.
- Reutilización: Un aumento del RC implica una disminución en el grado de reutilización de la clase.
- Cantidad de Pruebas: Un aumento del RC implica un aumento de la cantidad de pruebas necesarias para probar una clase. (33)

En la Tabla 1 se muestran las categorías para clasificar cada uno de los atributos de calidad anteriormente mencionados, así como el criterio de evaluación:

Atributo	Categoría	Criterio
Acoplamiento	Ninguno	RC = 0
	Bajo	RC = 1
	Medio	RC = 2
	Alto	RC > 2
Complejidad de Mantenimiento	Baja	RC <= Promedio[20]
	Media	Promedio <=RC<= 2* Promedio
	Alta	RC >2* Promedio
Reutilización	Baja	RC >2* Promedio
	Media	Promedio< RC <= 2 *Promedio
	Alta	RC <= Promedio
Cantidad de pruebas	Baja	RC <= Promedio
	Media	Promedio<= RC <2*Promedio
	Alta	RC >= 2*Promedio

Tabla 1: Criterios de evaluación para la métrica RC

A continuación se presenta un ejemplo de la aplicación de esta métrica al diseño de la aplicación:

Clases	RC	Acoplamiento	Complejidad	Reutilización	Cant. Pruebas

[20] Se refiere al promedio de asociaciones de uso que presentan las clases de la muestra, este promedio tiene un valor de 4.9 aproximadamente.

			Mantenimiento		
ConfesionJudicialController	2	Medio	Baja	Alta	Baja
DisponerSobrePruebaAntesPruebaConfesionController	2	Medio	Baja	Alta	Baja
DisponerSobreEscritoPruebaConfesionJudicialController	2	Medio	Baja	Alta	Baja
ActaPruebaConfesionController	2	Medio	Baja	Alta	Baja
DisponerSolicitudDeAclaracionDeConfesionController	2	Medio	Baja	Alta	Baja
CrearResolucionTeniendoInformeAclaracionConfesionController	2	Medio	Baja	Alta	Baja
RegistrarEscritoSolicitandoConfesionJudicialPorArticuloController	2	Medio	Baja	Alta	Baja
ConfesionJudicialGtr	11	Alto	Alta	Baja	Alta
DisponerSobrePruebaAntesPruebaConfesionGtr	6	Alto	Media	Media	Media
DisponerSobreEscritoPruebaConfesionJudicialGtr	7	Alto	Media	Media	Media
ActaPruebaConfesionGtr	14	Alto	Alta	Baja	Alta
DisponerSolicitudDeAclaracionDeConfesionGtr	9	Alto	Media	Media	Media
CrearResolucionTeniendoInformeAclaracionConfesionGtr	8	Alto	Media	Media	Media

Tabla 2: Resultados obtenidos luego de aplicada la métrica RC[21]

Luego de analizar los resultados obtenidos para cada uno de los atributos de calidad de esta métrica se evidencia que el 35% de las clases posee un acoplamiento medio, el otro 35% alto y el 30% posee un acoplamiento bajo. Además el 65% de las clases posee una complejidad de mantenimiento baja, el

[21] En esta tabla se muestran los nombres de las clases, para evitar errores de sintaxis del código se evita acentuar las palabras.

20% media y solo un 15% presenta una complejidad de mantenimiento alta. Por otra parte se puede observar que el 65% tiene una reutilización alta, el 20% media y solo un 15% posee una baja reutilización. El 65% de las clases posee una cantidad de pruebas bajas, el 20% media y el 15% de las clases una alta cantidad de pruebas. Los resultados anteriormente expuestos permiten corroborar que la mayoría de las clases presentan un acoplamiento relativamente bajo, una baja complejidad de mantenimiento, una alta reutilización así como una baja cantidad de pruebas que se traduce en menos esfuerzos a la hora de realizar pruebas a la aplicación. Se evidencia de esta forma la calidad del diseño de las clases, lo cual facilita la implementación del sistema. A continuación se muestran gráficamente los resultados obtenidos:

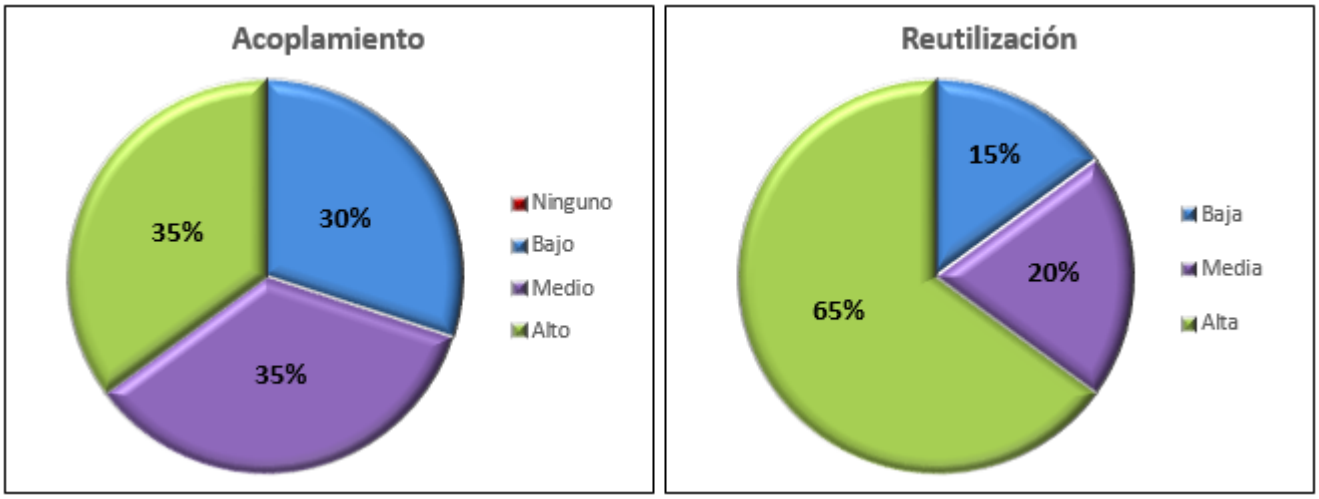


Figura 19: Representación gráfica de los resultados de los atributos de calidad para la métrica RC

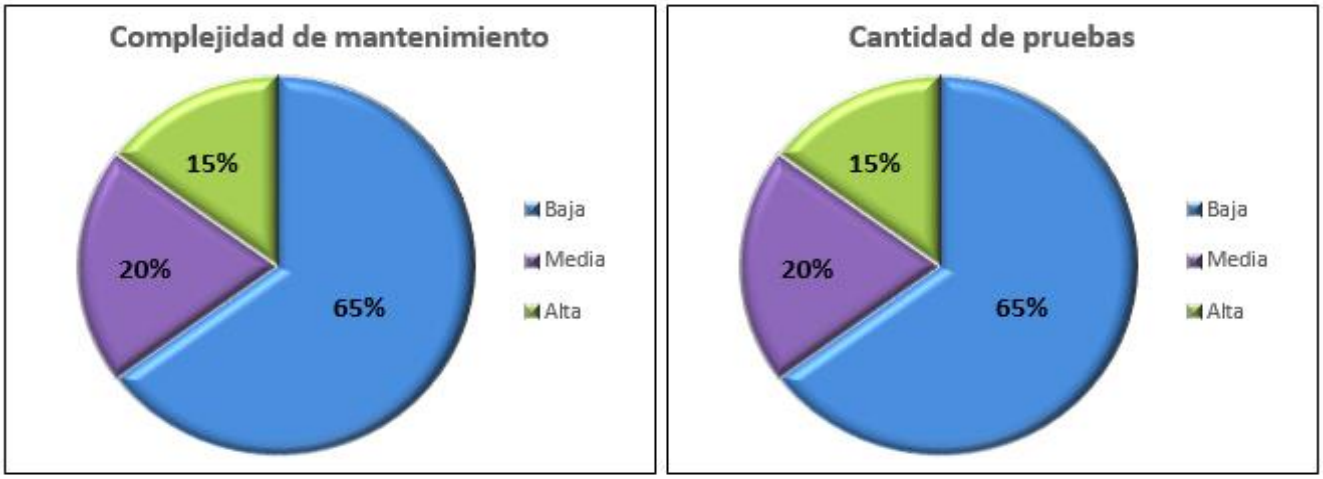


Figura 20: Representación gráfica de los resultados de los atributos de calidad para la métrica RC

Tamaño de Clase (TC)

La métrica Tamaño de Clase calcula el tamaño de una clase a partir de la cantidad de operaciones y atributos que se encuentran contenidos dentro de la misma (33). Se evalúa a partir de los siguientes atributos de calidad:

- Responsabilidad
- Complejidad de las pruebas
- Reutilización

Si el resultado que se obtiene se encuentra por encima del umbral significa que la clase posee un alto grado de responsabilidad, lo que provoca que disminuya la reutilización, se haga más compleja la implementación y se necesite un mayor esfuerzo a la hora de realizar las pruebas. (33)

Para evaluar los resultados obtenidos al aplicar la métrica TC se utilizarán los siguientes umbrales definidos por Lorenz y Kidd:

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

Tabla 3: Umbrales para la métrica TC

La Tabla 4 muestra un ejemplo de las medidas de los parámetros de calidad obtenidas luego de aplicar la métrica TC:

Clases	Procedimientos	Responsabilidad	Complejidad	Reutilización
ConfesionJudicialController	12	Baja	Baja	Alta
DisponerSobrePruebaAntes PruebaConfesionController	8	Baja	Baja	Alta
DisponerSobreEscritoPrueba ConfesionJudicialController	7	Baja	Baja	Alta
ActaPruebaConfesionControl ler	10	Baja	Baja	Alta
DisponerSolicitudDeAclaraci	7	Baja	Baja	Alta

onDeConfesionController				
CrearResolucionTeniendoInformeAclaracionConfesionController	5	Baja	Baja	Alta
RegistrarEscritoSolicitandoConfesionJudicialPorArticuloController	14	Baja	Baja	Alta
ConfesionJudicialGtr	19	Media	Media	Media
DisponerSobrePruebaAntesPruebaConfesionGtr	11	Baja	Baja	Alta
DisponerSobreEscritoPruebaConfesionJudicialGtr	23	Media	Media	Media
ActaPruebaConfesionGtr	9	Baja	Baja	Alta
DisponerSolicitudDeAclaracionDeConfesionGtr	16	Baja	Baja	Alta
CrearResolucionTeniendoInformeAclaracionConfesionGtr	17	Media	Media	Media

Tabla 4: Resultados obtenidos luego de aplicada la métrica RC

Para este ejemplo se aplicó la métrica a un total de 20 clases, obteniéndose un total de 333 procedimientos para un promedio de 16.7 procedimientos. De las clases analizadas se obtiene un total de 15 clases de tamaño pequeño, 3 de tamaño medio y 2 de tamaño grande como se evidencia en la Tabla 5:

Umbral	Tamaño	Cantidad de clases
<=20	Pequeño	15
>20 y <=30	Medio	3
>30	Grande	2

Tabla 5: Cantidad de clases por tamaño según los umbrales definidos por Lorenz y Kidd

Analizando los resultados para cada uno de los atributos de calidad establecidos se evidencia que el 65% de las clases posee una baja responsabilidad, el 25% una responsabilidad media y solo el 10% tiene una responsabilidad alta, comportándose de la misma manera para el atributo complejidad. Se observa además que el 65% de las clases presenta un alto grado de reutilización, el 25% una reutilización media y solo el 10% posee una baja reutilización. Estos resultados demuestran la calidad del diseño de la solución propuesta, ya que al obtener bajos índices de responsabilidad y complejidad, unidos a un alto grado de reutilización de las clases se facilita en gran medida la implementación de las mismas. Los siguientes gráficos ilustran los datos obtenidos:

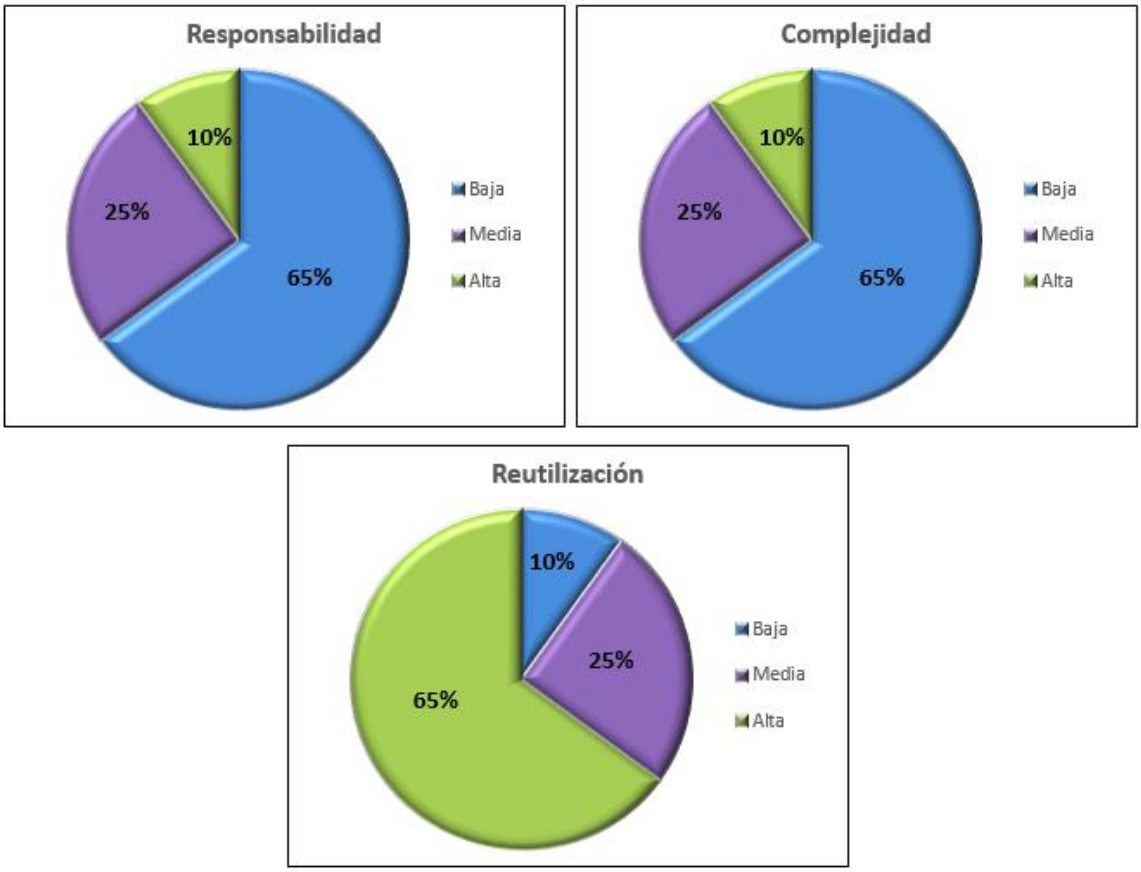


Figura 21: Representación gráfica de los resultados de los atributos de calidad para la métrica TC

Carencia de Cohesión en los Métodos (CCM)

El CCM es la cantidad de métodos que acceden a un mismo atributo dentro de la clase. Cuando el CCM es alto los métodos deben acoplarse a otros mediante los atributos, lo que incrementa la complejidad del diseño de clases. La carencia de cohesión en los métodos de una clase implica que

debe ser separada en dos o más subclases, ya que la existencia de baja cohesión incrementa la probabilidad de cometer errores durante el proceso de desarrollo. (33)

Las tablas Tabla 6 y Tabla 7 muestran el proceso realizado para la aplicación de esta métrica, tomando como ejemplo la clase *Prueba*. Lo primero es asignarle un identificador (una letra) a cada uno de los atributos que posee la clase, luego se selecciona por cada uno de los métodos de la clase, cuáles son los atributos que utiliza, para finalmente determinar como nivel de CCM el mayor número de veces que un mismo atributo fue utilizado por diferentes métodos.

Atributos	Identificadores
Id	a
motivoDisposicionTexto	b
Propósito	c
tipoPrueba	d
tipoEstado	e
documentoPresentacion	f
documentoDisposicion	g
escritoOtroPrueba	h
pruebaInformePrueba	i
informePrueba	j
Acta	k
instanciaSennalamientoPrueba	l

Tabla 6: Atributos de la clase Prueba

Métodos	Atributos	Métodos	Atributos
getId	a	removeEscritoOtroPrueba	h
setMotivoDisposicionTexto	b	getEscritoOtroPrueba	h

getMotivoDisposicionTexto	b	addInformePrueba	j
setTipoPrueba	d	removeInformePrueba	j
getTipoPrueba	d	getInformePrueba	j
setTipoEstado	e	__construct	h,i,j,k
getTipoEstado	e	addPruebaInformePrueba	i
setDocumentoPresentacion	f	removePruebaInformePrueba	i
getDocumentoPresentacion	f	getPruebaInformePrueba	i
setDocumentoDisposicion	g	addActa	k
getDocumentoDisposicion	g	removeActa	k
setProposito	c	getActa	k
getProposito	c	addInstanciaSennalamientoPrueba	l
addEscritoOtroPrueba	h	removeInstanciaSennalamientoPrueba	l

Tabla 7: Métodos de la clase Prueba y sus atributos

Al analizar los resultados obtenidos luego de aplicar la métrica, se puede observar que la clase Prueba presenta un nivel de CCM igual a 4, un valor aceptable según los autores de la métrica que definen un umbral de 5. Se comprueba de esta forma que los métodos no se encuentran acoplados entre sí a través de los atributos, disminuyendo de esta manera la complejidad del diseño. El resto de las clases se comportan de manera similar.

Árbol de Profundidad de Herencia (APH)

La aplicación del árbol de profundidad de herencia asegura cuántas clases antecesoras pueden afectar potencialmente una clase. En la profundidad de herencia mientras mayor sea el número de métodos a heredar por dicha clase, mayor será la complejidad de esta. Es por ello que en árboles grandes mientras más métodos y clases se vean envueltos, mayor será la complejidad de diseño presentado (33). Lorenz y Kidd proponen un umbral de 6 niveles como indicador de un abuso en la herencia.

La siguiente tabla muestra los resultados obtenidos luego de aplicar la métrica APH:

Clases	Procedimientos
ConfesionJudicialController	3
DisponerSobrePruebaAntesPruebaConfesionController	3
DisponerSobreEscritoPruebaConfesionJudicialController	3
ActaPruebaConfesionController	3
DisponerSolicitudDeAclaracionDeConfesionController	3
CrearResolucionTeniendoInformeAclaracionConfesionController	3
RegistrarEscritoSolicitandoConfesionJudicialPorArticuloController	3
ConfesionJudicialGtr	1
DisponerSobrePruebaAntesPruebaConfesionGtr	1
DisponerSobreEscritoPruebaConfesionJudicialGtr	1
ActaPruebaConfesionGtr	1
DisponerSolicitudDeAclaracionDeConfesionGtr	1
CrearResolucionTeniendoInformeAclaracionConfesionGtr	1

Tabla 8: Resultados obtenidos luego de aplicada la métrica APH

A partir de los datos obtenidos de la aplicación de la métrica a las clases seleccionadas se observa que el APH de estas se mantiene entre 0 y 3, lo cual indica que el resultado es óptimo si se tiene en cuenta que el umbral definido es de 6. Se garantiza así una baja complejidad en el diseño y por tanto una alta cohesión, comportándose de manera similar en el resto de las clases que no aparecen en la muestra.

3.3 Validación de la implementación

Se hace necesaria la validación de la solución propuesta con el objetivo de comprobar que cumple con las especificaciones y necesidades requeridas. Existen deficiencias que solo pueden ser detectadas una vez terminada la implementación, para ello resulta de vital importancia llevar a cabo pruebas de software que permita identificar estos errores para poder subsanarlos.

Para validar la implementación de la solución propuesta se aplicaron pruebas funcionales internas por parte del proyecto y calidad de centro CEGEL, haciendo uso del método de prueba de caja negra a través de la técnica de particiones equivalentes. Para ello se diseñaron casos de prueba para cada caso de uso permitiendo de esta manera, a partir de un conjunto de condiciones de entrada, encontrar posibles errores que pueda presentar la aplicación.

Para la elaboración de cada caso de prueba se utilizó un documento que consta de 3 hojas de cálculo:

Presentación: En esta hoja de cálculo se hace la presentación del caso de prueba, incluyendo el nombre del centro de desarrollo y el proyecto al cual pertenece el software que se va a probar. Además aparecen 2 tablas, una para el control del documento y la otra para el control de cambios.

Variables: Es la hoja de cálculo donde se describen las variables que se utilizan en el caso de prueba, estas variables no son más que cada uno de los campos de entrada que posee la interfaz del caso de uso que se desea probar. Está compuesta por una tabla donde se especifican el nombre de la variable, un número de identificación, su clasificación, su valor (puede ser obligatorio u opcional) y la descripción. En el Anexo 1 se muestra la hoja de cálculo Variables, para el caso de prueba correspondiente al caso de uso Gestionar prueba de Confesión Judicial.

Caso de prueba: En esta hoja de cálculo se confecciona el caso de prueba. En ella se realiza una descripción general del caso de uso a probar, las condiciones de ejecución para llevarlo a cabo y una tabla por cada una de sus secciones con los siguientes campos:

- **Escenario:** Son las condiciones bajo las que se prueba el software con el objetivo de comprobar si la respuesta que da el sistema es correcta o si existe algún tipo de error.
- **Descripción:** Se describe brevemente el escenario.
- **Variable 1…n:** Son las variables que se utilizan para probar cada escenario y que ya fueron definidas en la hoja de cálculo Variables.
- **Respuesta del sistema:** Se describe la respuesta del sistema cuando se ejecuta cada clase de equivalencia.
- **Flujo central:** Indica el camino a seguir para ejecutar la sección del caso de uso que se desea probar.

Los anexos Anexo 2, Anexo 3 y Anexo 4 muestran las tablas correspondientes a las secciones Adicionar, Modificar y Eliminar confesión judicial respectivamente, de la hoja de cálculo Caso de prueba para el caso de uso antes mencionado.

Los casos de prueba diseñados fueron aplicados en las 3 iteraciones de pruebas realizadas. En la primera iteración se encontraron un total de 39 no conformidades, relacionadas fundamentalmente con errores de interfaz y formato de algunos documentos que debían ser generados, además de la ausencia de determinadas validaciones que impedían el correcto funcionamiento del sistema. En la segunda iteración se identificaron un total de 12 no conformidades, las cuales tenían que ver con errores ortográficos y datos que no se cargaban de la Base de Datos fundamentalmente. En la tercera iteración no se detectaron no conformidades, quedando listo el sistema para ser probado por el equipo de calidad del centro CEGEL. La Figura 22 muestra gráficamente la información anteriormente expuesta sobre los resultados obtenidos en cada una de las iteraciones realizadas.

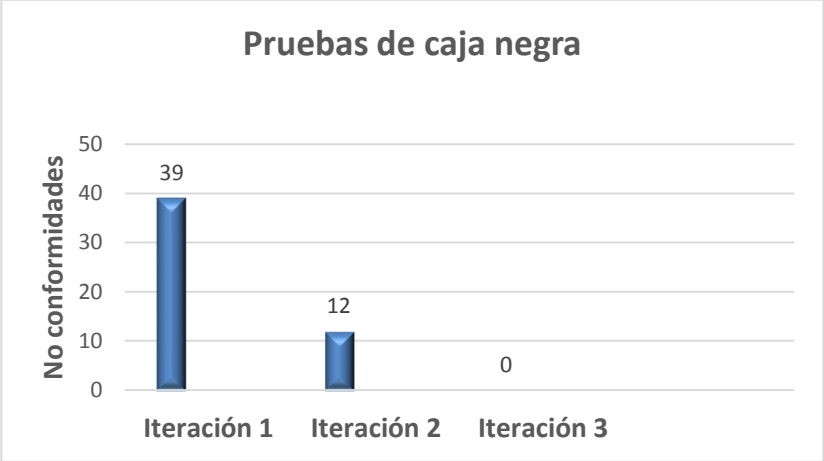


Figura 22: Cantidad de no conformidades obtenidas en las iteraciones realizadas en las pruebas internas

Luego de ser resueltas las no conformidades obtenidas en las pruebas de funcionalidad realizadas internamente en el proyecto, se procedió a probar la solución por parte de calidad del centro CEGEL. Se realizaron 3 iteraciones obteniendo 10 no conformidades en la primera iteración, relacionadas fundamentalmente con errores en la validación de algunos datos, en la segunda iteración se identificaron 3 no conformidades que tenían que ver con determinada información que no se cargaban correctamente de la Base de Datos y en una tercera iteración no se encontraron no conformidades. La Figura 23 muestra gráficamente los resultados de las pruebas realizadas, mostrando la cantidad de no conformidades por cada iteración.

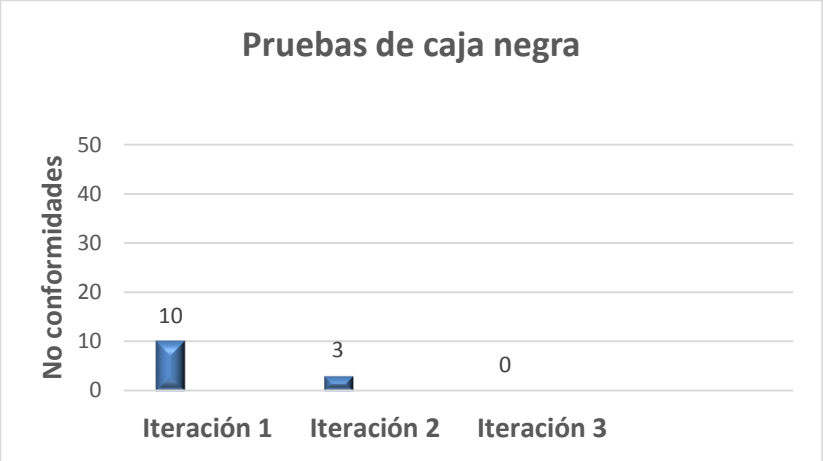


Figura 23: Cantidad de no conformidades obtenidas en la primera iteración realizada en calidad del centro CEGEL

La informatización del subproceso Pruebas de confesión del módulo Común, propició la reutilización de dicho subproceso en los diferentes módulos que se encuentran en proceso de desarrollo actualmente. Luego de una replanificación en el proyecto se logró una disminución considerable del tiempo y personal necesario para la implementación de los módulos Administrativo, Disciplina y Derecho Laboral y Ordinario Civil, quedando disminuido el tiempo en 15 días para cada uno de los módulos, involucrando 20 personas para ello. La Tabla 9 muestra el tiempo y personas necesarias para el desarrollo de los módulos antes y después de la implementación de las Pruebas de Confesión en el módulo Común.

Módulos	Tiempo (inicialmente)	Personas (inicialmente)	Tiempo (replanificado)	Personas (replanificado)
Administrativo	60	35	45	20
Disciplina y Derecho laboral	120		105	
Ordinario civil	90		75	

Tabla 9: Tiempo y personas necesarias para el desarrollo de los módulos antes y después de la implementación de las Pruebas de Confesión

3.4 Conclusiones parciales

- La aplicación de las métricas de diseño demostró que las clases generalmente presentan una baja complejidad, un acoplamiento relativamente bajo, un alto grado de reutilización y una alta cohesión, lo que facilitó en gran medida la implementación.
- Con la aplicación de las pruebas de caja negra se comprobó que se implementaron todas las funcionalidades especificadas y que cada una de ellas funciona correctamente.

CONCLUSIONES

La culminación del presente trabajo permitió obtener el diseño e implementación del subproceso Pruebas de Confesión, contribuyendo de esta manera al desarrollo de los módulos Administrativo, Disciplina y Derecho laboral y Ordinario civil del proyecto de Informatización para la Gestión de los Tribunales Cubanos. A partir de los resultados obtenidos se puede arribar a las siguientes conclusiones:

- El estudio de los conceptos fundamentales asociados al negocio permitió comprender y demostrar la necesidad de continuar con el desarrollo del subproceso Pruebas de Confesión a partir de los requisitos funcionales especificados.
- El modelo de diseño obtenido permitió materializar con precisión los requisitos del cliente y de esta forma facilitar y agilizar la implementación de las funcionalidades correspondientes.
- A partir de la implementación se obtuvo un sistema que satisface las necesidades del cliente y provee una solución a los problemas presentes en los TPC por los cuales se comenzó el proceso de desarrollo del software.
- La validación de la solución a partir de la aplicación de las métricas de diseño y las pruebas de caja negra, posibilitó comprobar la calidad de los artefactos obtenidos.
- La solución obtenida contribuyó al desarrollo de los módulos Administrativo, Disciplina y Derecho laboral y Ordinario civil, disminuyendo el tiempo para cada uno de ellos en un 25%, 12,5% y 11% respectivamente, además se redujo en un 43% la cantidad de programadores necesarios para la implementación de los mismos.

RECOMENDACIONES

Una vez cumplidos los objetivos de la presente investigación se recomienda:

- La reutilización del subproceso Pruebas de Confesión del módulo Común en cada uno de los módulos en los que se practiquen este tipo de pruebas y que aún no ha sido implementados, pertenecientes a los subsistemas Civil, Laboral y Económico.

BIBLIOGRAFÍA

1. **Ley de Procedimiento Civil, Administrativo, Laboral y Económico. Artículo 262.**
2. Gaceta Oficial de la República de Cuba. Ministerio de Justicia. [En línea] [Citado el: 1 de Diciembre de 2013.] http://www.gacetaoficial.cu/html/tribunalespopulares.html. 1.
3. Ivar Jacobson, Grady Booch, James Rumbaugh. *EL Proceso Unificado de Desarrollo de Software.* Madrid : Addison-Wesley, 2000.
4. IBM. IBM. [En línea] [Citado el: 3 de Diciembre de 2013.] http://www.ibm.com/Search/?q=rational+unified+process+tutorial&co=us&lo=any&ibm-submit.x=0&ibm-submit.y=0&sn=&lang=en&cc=US&en=utf&hpp=.
5. James Rumbaugh, Ivar Jacobson y Grady Booch. *The Unified Modeling Language Reference Manual.*
6. Pascual González, Ana A González, Jose A. Gallud. UCLM. *Herramientas CASE. ¿Cómo incorporarlas con éxito en nuestra organización?* [En línea] [Citado el: 11 de Diciembre de 2013.] www.uclm.es/ab/educacion/ensayos/pdf/revista10/10_17.pdf.
7. Knowledge Reuse Group. Universidad Carlos III de Madrid. [En línea] [Citado el: 11 de Diciembre de 2013.] http://www.ie.inf.uc3m.es/grupo/docencia/reglada/Is1y2/PracticaVP.pdf. 5.
8. LibrosWeb. Symfony en pocas palabras. [En línea] [Citado el: 11 de Diciembre de 2013.] http://librosweb.es/symfony/capitulo_1/symfony_en_pocas_palabras.html.
9. Borrillo, Ricardo. Genbeta : dev. Desarrollo y software. [En línea] [Citado el: 11 de Diciembre de 2013.] http://www.genbetadev.com/desarrollo-web/disenando-tu-nuevo-proyecto-web-con-bootstrap-2-0.
10. jQuery. [En línea] [Citado el: 10 de Febrero de 2014.] http://jquery.com/.
11. Swedberg, Karl y Chaffer, Jonathan. *Learning jQuery : better interaction design and web development with simple JavaScript techniques.* Birmingham : Packt Publishing, 2007.
12. Bonanata, Maximiliano. *Programación y Algoritmos.* s.l. : M P Ediciones S.A, 2003.
13. Sebesta, Robert W. *Concepts of programming languages .* s.l. : Addison-Wesley, 2003.
14. PHP. *Manual de php.* [En línea] [Citado el: 10 de Febrero de 2014.] http://www.php.net/manual/es.
15. Pérez, Javier Eguiluz. *Introducción a javascript.* s.l. : Autoeditado, 2008.
16. Pérez, Javier Eguíluz. *Introdución a XHTML .* 2008.
17. W3C. [En línea] http://www.w3.org/TR/html401/intro/intro.html#h-2.2.
18. Bertino, Elisa y Martino, Lorenzo. *Sistemas de bases de datos orientadas a objetos: conceptos y arquitecturas.* s.l. : Ediciones Díaz de Santos, 1995.
19. Riggs, Simon y Krosing, Hannu. *PostgreSQL 9 Administration Cookbook.* s.l. : Packt Publishing, 2011.
20. Object Rol Modeling. The official site for conceptual data modeling. *ORM in Detail.* [En línea] [Citado el: 13 de Febrero de 2014.] http://www.orm.net/.
21. Doctrine. [En línea] [Citado el: 13 de Febrero de 2014.] https://doctrine-orm.readthedocs.org.
22. Doctrine. [En línea] [Citado el: 13 de Febrero de 2014.] http://www.doctrine-project.org/.
23. Mateu, Carles. *Desarrollo de aplicaciones web.* Barcelona : Eureca media, 2004.
24. Ben Collins-Sussman, Brian W. Fitzpatrick, C. Michael Pilato. *Control de versiones con Subversion.* California : s.n., 2004.
25. Stefan Küng, Lübbe Onken, Simon Large. TortoiseSVN. The coolest interface to (Sub)version control. [En línea] [Citado el: 15 de Febrero de 2014.] http://tortoisesvn.net/docs/nightly/TortoiseSVN_es/.
26. desarrolloweb.com. [En línea] [Citado el: 25 de Febrero de 2014.] http://www.desarrolloweb.com/articulos/1622.php.
27. Pressman, Roger S. *Ingeniería del software. Un enfoque práctico.* s.l. : Mc-Graw-Hill, 2001.

28. Libros Web. ***Diseño y programación web.*** **[En línea] [Citado el: 25 de Febrero de 2014.] http://librosweb.es/symfony_1_2/capitulo_2/el_patron_mvc.html.**
29. Larman, Craig. ***UML y Patrones. Introducción al análisis y diseño orientado a objetos.*** **México : Prentice Hall, 1999.**
30. Mena, Ernesto D. Aldas y Cadena, Maritza A. Andrade. Dspace. ***Escuela Politécnica Nacional.*** **[En línea] 11 de Enero de 2011. [Citado el: 25 de Febrero de 2014.] http://bibdigital.epn.edu.ec/handle/15000/2669.**
31. Facultad de informática - Universidad Politécnica de Madrid. Patrones del "Gang of Four". ***Patrones del "Gang of Four".***
32. Johnson, Rod. ***Expert One-on-One J2EE Design and Development.*** **s.l. : Wrox Press, 2003.**
33. ***TOWARDS A METRICS SUITE FOR OBJECT ORIENTED DESIGN.*** **Shyam R. Chidamber, Chris F. Kemerer. Cambridge : s.n.**
34. Rumbaugh, James, Jacobson, Ivar y Booch, Grady. ***El Lenguaje Unificado de Modelado. Manual de Referencia.*** **s.l. : Addison Wesley, 2007. 4.**
35. Pérez, Javier Eguiluz. ***Introducción a XHTML.*** **2008.**

ANEXOS

Anexo 1: Descripción de variables para los casos de prueba correspondientes al caso de uso Gestionar prueba de Confesión Judicial.

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	Litigantes	Campo de selección	Obligatorio	Lista las personas que intervienen en los procesos (demandantes o demandados) que pueden confesar ante el tribunal.
2	Provincia	Campo de selección	Obligatorio	Este campo es obligatorio siempre que se seleccione Reside actualmente en otra dirección. Permitirá seleccionar de un listado la provincia.
3	Municipio	Campo de selección	Obligatorio	Este campo es obligatorio siempre que se seleccione Reside actualmente en otra dirección. De acuerdo a la provincia seleccionada permitirá escoger de un listado el municipio.
4	Dirección	Campo de texto	Obligatorio	Este campo es obligatorio siempre que se seleccione Reside actualmente en otra dirección.
5	Reside actualmente en otra dirección	Campo de múltiple selección	Opcional	Este campo se selecciona en caso que el litigante que se le vaya a practicar la prueba resida en otra dirección.
6	Declarar en domicilio	Campo de múltiple selección	Obligatorio	Este campo es obligatorio siempre que se seleccione Realizar solicitud sobre circunstancias especiales. Es obligatorio que uno de los tres esté seleccionado como mínimo.
7	Declarar con intérprete de idioma	Campo de múltiple selección	Obligatorio	
8	Declarar con intérprete para sordo/mudos	Campo de múltiple selección	Obligatorio	
9	Realizar solicitud sobre circunstancias	Campo de múltiple selección	Opcional	Este campo se selecciona en caso que el litigante que se le vaya a practicar la prueba presente circunstancias especiales.

	especiales			
10	Preguntas	Campo de texto	Obligatorio	Permite insertar las preguntas que conformarán el pliego para la confesión.

Anexo 2: Caso de prueba correspondiente al escenario Adicionar prueba de confesión judicial del caso de uso Gestionar prueba de Confesión Judicial.

Escenario	Descripción	1	5			9			10	Respuesta del sistema	Flujo central
			2	3	4	6	7	8			
EC 1.1 Adicionar prueba de confesión judicial	El sistema permitirá adicionar una prueba de confesión judicial.	V	V	V	V	V	V	V	V	El sistema permite registrar la prueba de confesión judicial	Selecciona la pestaña de Confesión Judicial. Selecciona la opción Adicionar, selecciona quién contestará el pliego, llena los datos correspon-dientes y luego la opción Registrar.
		N/A	N/A	N/A	N/A	N/A	N/A	N/A	¿Estás?		
EC 1.2 Los datos son incorrectos y/o existen campos obligatorios vacíos	Introduce datos incorrectos o deja campos obligatorios vacíos.	I	V	V	V	V	V	V	V	Indica que los datos son incorrec-tos y no guarda los cambios.	
		Seleccio-ne	N/A	N/A	N/A	N/A	N/A	N/A	¿Estás?		
		V	V	I	V	V	V	V	V		
		N/A	N/A	Seleccione	N/A	N/A	N/A	N/A	¿Estás?		
		V	V	V	I	V	V	V	V		
		N/A	N/	N/		N/	N/	N/	¿Estás?		

			A	A		A	A	A			
EC 1.3 La operación es cancelada	La operación es cancelada.	V								Cierra la interfaz y no guarda los cambios.	Selecciona la pestaña de Confesión Judicial, luego la Opción Adicionar, completa o no los datos solicitados y luego selecciona la opción Cancelar.
		N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A		

Anexo 3: Caso de prueba correspondiente al escenario Modificar prueba de confesión judicial del caso de uso Gestionar prueba de Confesión Judicial.

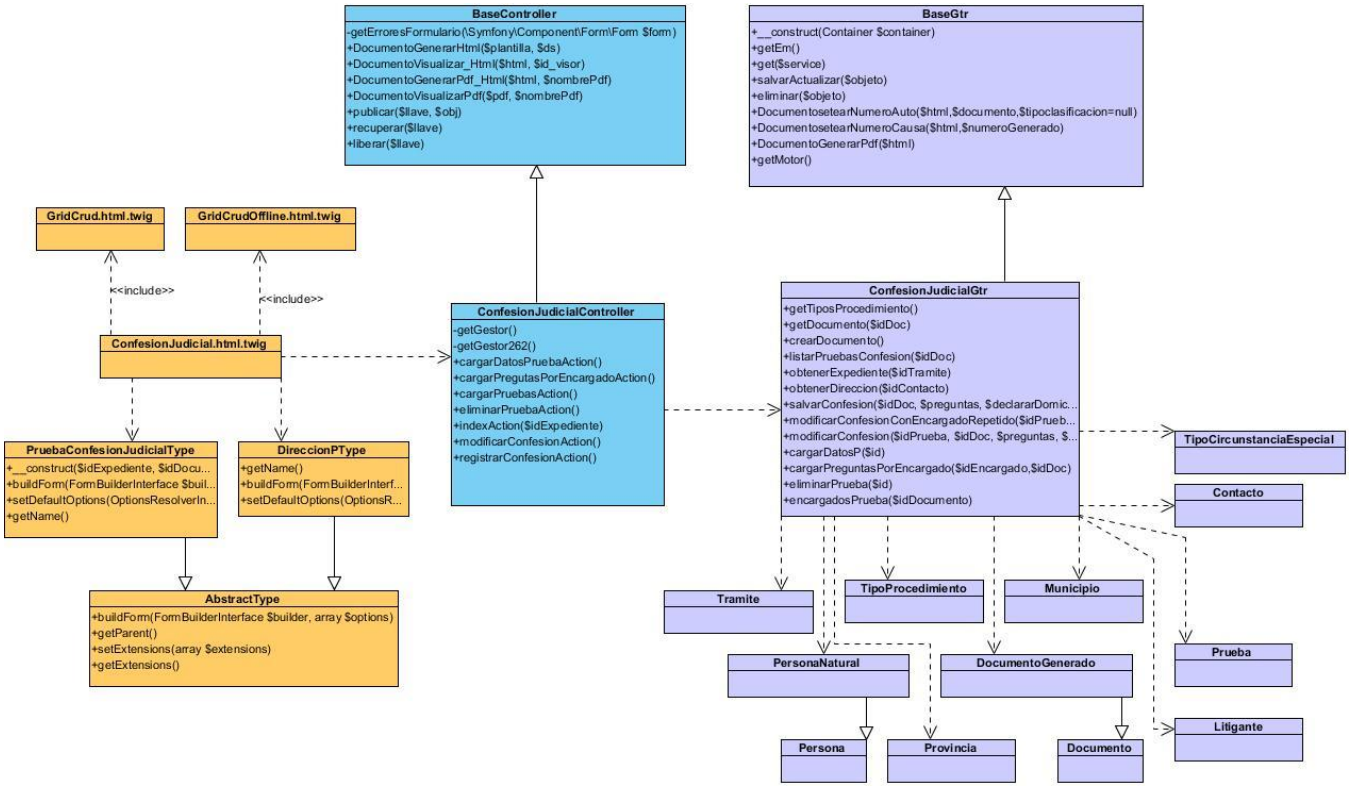
Escenario	Descripción	1	5			9			10	Respuesta del sistema	Flujo central
			2	3	4	6	7	8			
EC 1.1 Modificar prueba de confesión judicial	El sistema permitirá modificar una prueba de confesión judicial.	V	V	V	V	V	V	V	V	Guarda la confesión judicial con los nuevos datos.	Selecciona la pestaña de Confesión Judicial. Selecciona una confesión y la opción Modificar.
		N/A	N/A	N/A	N/A	N/A	N/A	N/A	¿Estás?		
EC 1.2	Introduce	I	V	V	V	V	V	V	V	Indica que	

Los datos son incorrectos y/o existen campos obligatorios vacíos	datos incorrectos o deja campos obligatorios vacíos.	Seleccio-ne	N/A	N/A	N/A	N/A	N/A	N/A	¿Estás?	los datos son incorrec-tos y no guarda los cambios.	Muestra los datos de la confesión y permitiendo modificarlos. Luego selecciona la opción Registrar.
		V	V	I	V	V	V	V	V		
		N/A	N/A	Seleccione	N/A	N/A	N/A	N/A	¿Estás?		
		V	V	V	I	V	V	V	V		
		N/A	N/A	N/A		N/A	N/A	N/A	¿Estás?		
EC 1.3 La operación es cancelada	La operación es cancelada.	V								Cierra la interfaz y no guarda los cambios.	Selecciona la pestaña de Confesión Judicial, luego una confesión y selecciona la opción Modificar, completa o no los datos solicitados y selecciona la opción Cancelar.
		N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A		

Anexo 4: Caso de prueba correspondiente al escenario Eliminar prueba de confesión judicial del caso de uso Gestionar prueba de Confesión Judicial.

Escenario	Descripción	1	2	3	4	6	7	8	10	Respuesta del sistema	Flujo central
EC 1.1 Eliminar prueba de confesión judicial	El sistema permitirá eliminar una prueba de confesión judicial.	N/A								Elimina la prueba de confesión judicial.	Selecciona la pestaña de Confesión judicial, selecciona la confesión y luego la opción Eliminar.

Anexo 5: Diagrama de clases correspondiente al caso de uso Gestionar prueba de Confesión Judicial.



Anexo 6: Diagrama de secuencia correspondiente al caso de uso Gestionar prueba de Confesión Judicial.