Universidad de las Ciencias Informáticas

Facultad 3

Diseño e implementación de los Flujos alternos del proceso Ejecutivo del Sistema de Informatización para la Gestión de los Tribunales Populares Cubanos.

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

Autor(es):

Dévorah Costales Ferrer
Juan Alberto Pereira Delgado

Tutor: Ing. José Miguel Cazorla Santana

Co-tutor(es): Ing. Humberto Arencibia Cruz
Ing. Indira Garcés Pérez

Ciudad de La Habana
Curso 2013-2014

Declaración de Autoría

Declaramos ser autores del presente trabajo de diploma y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____del año _____.

_____
Juan Alberto Pereira Delgado
Autor

_____
Dévorah Costales Ferrer
Autor

_____
Ing. José Miguel Cazorla Santana
Tutor

_____
Ing. Humberto Arencibia Cruz
Co-Tutor

_____
Ing. Indira Garcés Pérez
Co-Tutor

Datos de Contacto

Autores:

Dévorah Costales Ferrer
Correo Electrónico: dcostales@estudiantes.uci.cu

Juan Alberto Pereira Delgado
Correo Electrónico: japereira@estudiantes.uci.cu

Tutor:

Ing. José Miguel Cazorla Santana.

Correo Electrónico: jmcazorla@uci.cu

Graduado de Ingeniero en Ciencias Informáticas en el año 2009 en la Universidad de las Ciencias Informáticas (UCI). Especialista General interno. Ha desempeñado el rol de programador en el proyecto Tribunales. Ha cursado varios diplomados entre los que destacan el de Docencia e Investigación Universitaria. Tiene una experiencia acumulada en el área de desarrollo de software de aproximadamente 4 años.

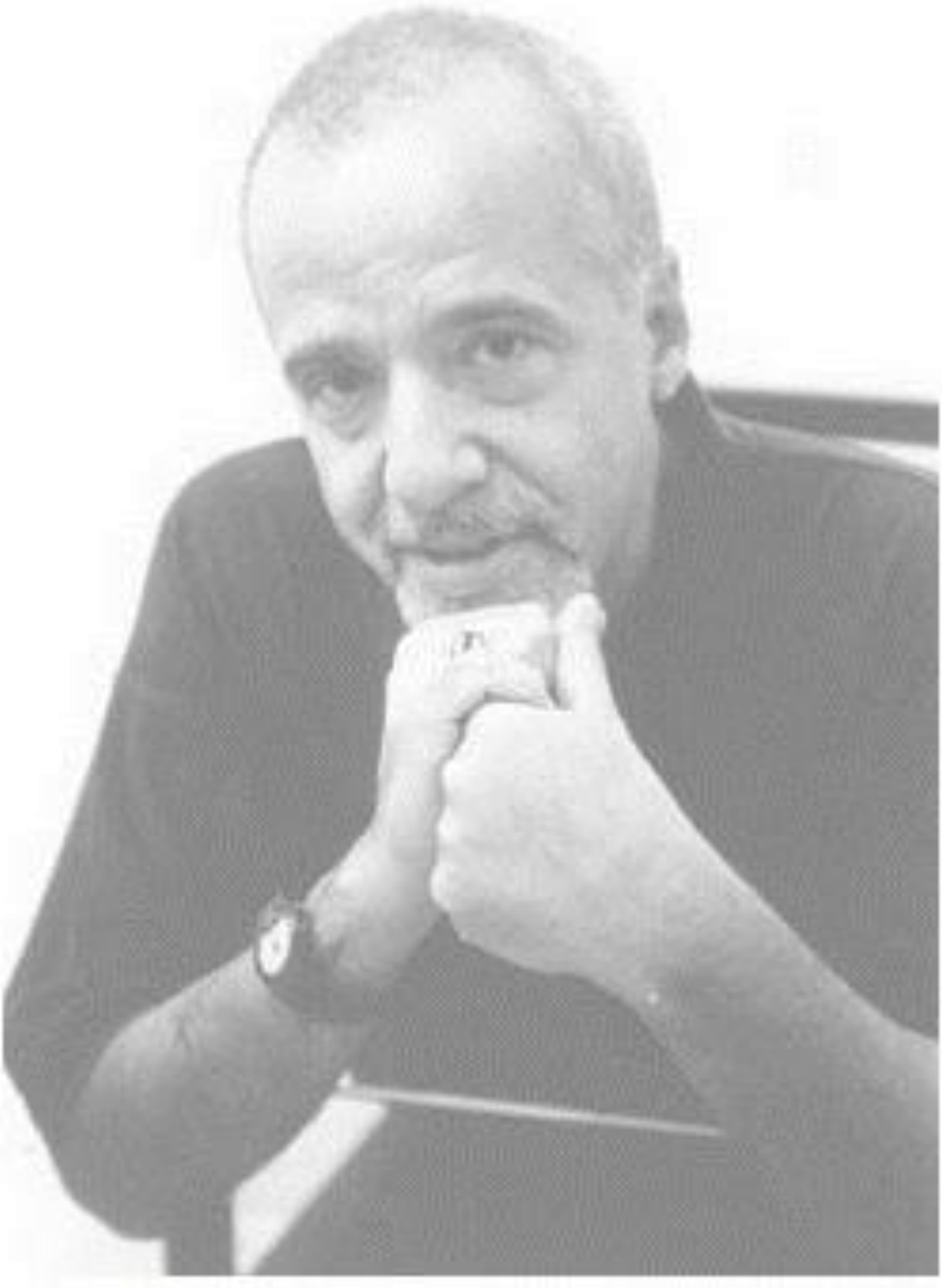
Co-Tutores:

Ing. Humberto Arencibia Cruz

Correo Electrónico: hacruz@uci.cu

Ing. Indira Garcés Pérez

Correo Electrónico: igarces@uci.cu



Carreteras rectas no hacen conductores hábiles.

Paulo Coelho

Dedicatoria

A mi heroína, mi mamá, que siempre ha estado ahí para mí, tanto en las buenas como en las malas, dándome consejos y guiándome por el camino correcto, siendo mi luz en la oscuridad. Sin ti hoy no fuese lo que soy.

A mí querido amigo, confidente, compañero de tesis y de vida, mi novio que ha luchado junto a mí durante todo este tiempo siempre apoyándome y apoyándonos en las buenas y en las malas. A ti mi amor te dedico este trabajo del cual tú has sido el mayor impulsor.

Dévorah.

A mi mejor amiga y compañera de estos cinco años, sin ti esos tantos momentos no hubieran sido los mismos.

Hilda y Gollo, donde quiera que estén ahora, verán mi futuro realizado, es este mi regalo para ustedes.

Para Sarah y Raúl, Yanerys y Riquel, Humbe y Gilbe, mis abuelos y mis tíos, quién sino ellos que me han visto crecer y sabían hasta donde era capaz de llegar.

A mis padres, ellos son mi razón de ser y me han apoyado en todos estos años.

Para mi hermano Gabriel, espero serte de ejemplo y ser parte de tu futuro.

Javier, Gilda, tía Ramona, Noel, Arleya y todas esas personas que desde fuera han seguido junto a mí este camino.

A todos aquellos que de una forma u otra han pasado junto a nosotros este tiempo, mis buenos amigos, compañeros de aula y de laboratorio a todos aquellos que ya no están hoy en la UCI y que abogaron por que llegáramos a ser lo que hoy somos.

Juan.

Agradecimientos

Primero que todo le agradezco a mi gordita, mi mamita, que no me ha dejado sola nunca, desde el día 5 de septiembre del 2009 hasta la actualidad ha sido el bastón que me sostiene y que me ha dado fuerzas para levantarme cada vez que tropezaba, o para recordarme con algunos regaños el camino que debía seguir. Nadie sabe cuánto ha luchado para poder mantenerse y mantenerme con fuerzas durante el transcurso de la carrera, mimi te agradezco sobre todo la confianza que tuviste en mí, sufriendo junto al igual que yo las derrotas en algunos momentos pero también las grandes alegrías.

A Juan, mi prometido, a ti mi vida que siempre estuvimos juntos tanto en los momentos difíciles como los de fiesta, a pesar de las peleas y los regaños has sido durante estos cinco años la alegría cuando alrededor todo era tristeza, has sido testigo a mi lado de lo difícil que fue esta vida universitaria y de los muros que juntos tuvimos que derribar para poder ser lo que hoy somos, para poder lograr el fruto del sacrificio. A ti mi amor gracias por tener la paciencia que tuviste conmigo y por haber entregado tanto de ti para que este trabajo tuviese éxito.

A toda mi familia que siempre se ha preocupado por mí dándome consejos y ayudándome de una manera u otra, en especial a mi prima Arleya que no tengo como agradecerle todo lo que ha hecho por mí y mi compañero. A todos los quiero mucho.

Dévorah.

Gracia a mi personita especial, gracias por esos buenos y malos días a mi lado, por apoyarme en los tiempos difíciles, por compartir las penas y las alegrías; porque, a pesar de todo, no hubiera querido otra compañera de mesa, de aula, de sueños, de vida, y sobre todo de tesis, que no fueras tú.

Gracias a mamá, como tú no hay ninguna, esas llamadas de cada día eran mi inspiración a seguir adelante cuando todo no marchaba bien.

Mi padre, siempre quisiste lo mejor para mí, cada vez que me exigías, hacías que rompiera mis límites y fuera cada vez mejor; me serviste como guía para sobrepasar esas metas que me proponía.

Muchas gracias a mis todos mis abuelos y tíos, a mi familia en general, a mis amigos, y a todos aquellos que nunca dejaron de tener fe en mí.

A Gilda, a tía Ramona y a Arleya, por ese café que me brindaban, por los consejos que me daban, por esas risas que me sacaban cuando las ganas eran de llorar y por esas otras tantas cosas que si menciono no terminaría nunca.

Juan.

A las profesoras Dariela y Vilma, gracias por la ayuda que incondicionalmente nos brindaron; así como para aquellos profesores que sembraron una semilla en nosotros y la dejaron crecer, y hoy nos ven realizados.

Agradecerle a los tutores especialmente a Indira que nos defendió y nos apoyó en todo momento, no sabemos cómo compensar todo lo que hizo por nosotros.

A nuestra amiga Lissette que perdió los ojitos revisando cada versión del documento que le enviamos y que nos ayudó muchísimo con su experiencia como una recién graduada, dándonos consejos para que todo lo hiciéramos lo mejor posible y nos demostró que no son los conocimientos los que valen, sino las ganas de ayudar.

Gracias a todas esas personas que aunque su nombre no están plasmados en esta hoja, fueron de ayuda para sobrepasar las dificultades, gracias por servirnos de base y hacer de nosotros una mejor persona. A todas las tías del comedor que siempre se preocuparon por nosotros, en especial Mercy y Acela.

Gracias al Comandante Fidel, nuestro sueño siempre fue estudiar informática, sin la creación de la UCI eso no hubiera sido posible.

Dévorah y Juan.

La información es uno de los elementos fundamentales para cualquier proceso empresarial, pero cada vez se hace más difícil mantener las herramientas de gestión de la información al margen del desarrollo tecnológico. Sin embargo, la brecha tecnológica no ha sido un impedimento para que el país pueda avanzar en el mundo de la informática; tomando como alternativa la creación de la Universidad de las Ciencias Informáticas, la cual actualmente entre otras actividades desarrolla un sistema de gestión orientado a satisfacer las necesidades informáticas de los Tribunales Populares Cubanos. Una de las materias que se desarrolla en los tribunales es la Económica, en la cual se efectúa, entre otros, el proceso Ejecutivo. De dicho proceso se realizó una primera versión, que no resultó ser eficiente, debido a que solo se implementaba el Flujo básico que debía seguir el mismo.

Por tal motivo, el presente trabajo de diploma pretende documentar cómo se implementaron los Flujos alternos para dar completamiento al módulo Ejecutivo, en aras de obtener una aplicación de calidad. Como resultado de la utilización de herramientas, lenguajes de programación, patrones de diseño y una fuerte arquitectura, elementos imprescindibles para mantener un sistema estructurado y funcional, se obtuvo un módulo íntegro del proceso Ejecutivo. Las pruebas aplicadas para comprobar la excelencia y el buen funcionamiento del sistema resultante, demostraron que el mismo cumplía con las necesidades del cliente, además fue capaz de resolver todas las deficiencias que inicialmente afectaban el correcto desarrollo del módulo.

Palabras Claves: alternos, tribunales, Ejecutivo, Económico, subsistema, tecnologías.

Índice de Contenidos

Introducción

El actual siglo está marcado por un rápido desarrollo de la tecnología en diversas ramas de la actividad humana como: la ciencia, la educación, las comunicaciones, entre otras. El desarrollo tecnológico ha sido un factor determinante de cambio para procesos como la gestión de la información, considerado piedra angular para el éxito de las empresas. Una correcta gestión de la información asegura su integridad, disponibilidad y confidencialidad, contribuyendo a la eficacia, eficiencia y productividad de las organizaciones.

En Cuba, el proceso de gestión de la información ha evolucionado muy lentamente, sin embargo, han sido muchas las alternativas puestas en práctica para mejorar las actividades que tienen una marcada dependencia de la información. Con el objetivo de "*producir aplicaciones y servicios informáticos a través del uso de las nuevas tecnologías, además de servir de soporte a la industria cubana de la informática"* (UCI, 2012), surge la Universidad de las Ciencias Informáticas. En dicha institución radican diversos centros de desarrollo de software, encargados de informatizar servicios que abarcan líneas vitales para el desarrollo del país. Uno de ellos es el Centro de Gobierno Electrónico (CEGEL), el cual tiene la misión de satisfacer necesidades de clientes gubernamentales mediante el desarrollo de productos, servicios y soluciones informáticas de alta confiabilidad (Laboratorio de Gestión de Proyectos, 2010).

Actualmente, una de las entidades a la cual CEGEL le está brindando servicios es al Tribunal Supremo Popular. En las diferentes instancias de esta organización (tribunales municipales y provinciales) se han estado presentando un conjunto de problemas que están afectando la eficiencia en la realización de sus procesos algunos de ellos son: el trabajo manual en los procedimientos provoca la existencia de duplicaciones en la radicación[1] de los expedientes, además los documentos oficiales contienen borrones y tachaduras, efecto que incide negativamente sobre la estética de los mismos; sumando también la desactualización de los registros debido a la demora en la introducción de datos en los libros.

Con la intención de resolver estas deficiencias el Tribunal Supremo decide, en coordinación con el centro CEGEL, desarrollar un sistema de gestión con el objetivo de informatizar los procesos que se realizan en cada una de las materias que componen los tribunales cubanos, surgiendo de esta manera el Sistema de Informatización para la Gestión de los Tribunales Populares Cubanos (SITPC). Dicho sistema desarrolla cinco

1 Proceso en el cual se le asigna un número al expediente y se asienta en el libro.

subsistemas: Administrativo, Económico, Laboral, Penal y Civil; donde cada uno de ellos define internamente un conjunto de módulos. El subsistema Económico, al cual se hará referencia en lo adelante, implementa cuatro módulos: Ordinario, Casación, Revisión y Ejecutivo; siendo este último el centro de la presente investigación.

El módulo Ejecutivo tiene como principal tarea informatizar los trámites concernientes a este proceso dentro de la materia Económica, dicho módulo se compone de la realización del Flujo básico en conjunto con los Flujos alternos; si algunos de los procedimientos que integran ambos flujos no son desarrollados, no se puede obtener un sistema funcionalmente íntegro para el proceso Ejecutivo. Sin embargo, como resultado de la primera iteración realizada, según el plan de iteraciones inicialmente elaborado para el módulo, solo fueron implementados los procedimientos correspondientes al Flujo básico, no siendo así para los Flujos alternos. Al no estar implementadas dichas funcionalidades, el sistema desarrollado hasta el momento no es capaz de realizar las disposiciones[2] sobre los escritos que se insertaban durante el proceso, provocando en adición la interrupción o retraso en la continuación de los trámites posteriores a estos procedimientos; además limitaba a seguir un único flujo la realización de los procedimientos. Por otro lado, no son gestionados en el sistema algunos de los procedimientos con los que finaliza al proceso Ejecutivo, los cuales son determinantes para que el mismo se efectúe correctamente hasta su terminación.

Debido a que la primera iteración implementada del módulo Ejecutivo solo satisface una parte de las necesidades del cliente, puesto que no cuenta con la implementación de las funcionalidades correspondientes a los Flujos alternos, necesarios para la completitud del proceso, se identifica el siguiente **problema de investigación**: la primera iteración del hito de implementación del módulo Ejecutivo no contempla la modelación de la totalidad de las funcionalidades que conforman este proceso de la materia Económica en los Tribunales Populares Cubanos. Luego de analizar el problema planteado se determinó abordar como **objeto de estudio** el proceso Ejecutivo de la materia Económica. Definiéndose como **objetivo general**: desarrollar una segunda iteración del módulo Ejecutivo según el plan de iteraciones inicial, para implementar las funcionalidades relativas a los Flujos alternos que no están presentes en la primera iteración, permitiendo completar el módulo. Para abordar detalladamente la solución de dicho problema se propone como **campo de acción**: los Flujos alternos correspondientes al módulo Ejecutivo de los Tribunales Populares Cubanos.

2 Proceso de admitir, subsanar o rechazar un escrito presentado por una parte.

Se define como **idea a defender** que si se realiza el diseño e implementación de los Flujos alternos se completará el desarrollo del módulo Ejecutivo, obteniéndose una aplicación íntegra del mismo.

Partiendo del objetivo general se definen como **objetivos específicos** los siguientes:

- Consultar y analizar las bibliografías necesarias para elaborar el marco teórico de la investigación.
- Transformar los requisitos funcionales a un lenguaje artefactual para identificar las relaciones entre las partes del sistema, mediante el modelo de diseño.
- Obtener un sistema funcional para dar solución a los requisitos definidos previamente para los Flujos alternos, mediante la implementación de casos de uso del sistema.
- Realizar las pruebas de software necesarias para verificar y validar los resultados obtenidos.

Para darle cumplimiento a los objetivos específicos se plantearon las siguientes **tareas de investigación:**

1. Caracterización del proceso Ejecutivo de la materia Económica.
2. Valoración de la metodología de desarrollo de software.
3. Valoración de las tecnologías a utilizar.
4. Caracterización y selección de los patrones de diseño más factibles para la propuesta de solución.
5. Elaboración de los diagramas de clases y de comunicación para el modelo de diseño.
6. Implementación de casos de uso del sistema.
7. Caracterización y selección de métodos de pruebas de software.
8. Aplicación de las técnicas de pruebas de software y casos de prueba correspondientes.

Durante la realización de la investigación se utilizaron diferentes métodos de investigación, a los cuales se hace alusión a continuación:

Métodos teóricos

El método analítico-sintético tiene como finalidad analizar cada una de las partes del objeto de estudio de manera independiente, para luego de ser analizadas poder determinar sus características generales y las relaciones que existen entre ellas (Hernández, et al., 2011). El método fue empleado para comparar las metodologías de desarrollo estudiadas, y para el análisis de las herramientas empleadas en la investigación, así como para el estudio del estado del arte de la investigación.

Mediante el método de modelación se crean abstracciones con el objetivo de explicar la realidad (Hernández, et al., 2011 p. 60). El mismo fue utilizado para la definición de la estructura de funcionamiento que guía el desarrollo del sistema. El diseño de diagramas de comunicación es una de las técnicas de modelación empleada; pues permite inferir la respuesta de un determinado procedimiento, y la forma en que se relacionan cada una de las clases implicadas.

Métodos particulares

"Son métodos más específicos que están desarrollados en base a las características propias de cada ciencia y para su aplicación están vinculados a técnicas de recolección de datos característicos de ese tipo de investigación" (Hernández, et al., 2011 p. 66). Para verificar el correcto funcionamiento del sistema se utilizó el método prueba de software, como método particular de las ciencias informáticas. El mismo tiene un carácter comprobador y el "*objetivo de diseñar una serie de casos de prueba que tengan una alta probabilidad de encontrar errores*" (Pressman, 2005 p. 418).

Métodos estadísticos o de medición

Métodos descriptivos: "*permiten organizar y clasificar los indicadores cuantitativos obtenidos en las investigaciones empíricas, determinando a través de ellas las propiedades, relaciones y tendencias del fenómeno*" (Hernández, et al., 2011 p. 64). Como métodos de medición utilizados se encuentran las métricas de diseño; las cuales permiten medir de forma cuantitativa la calidad de los atributos del sistema, además de evaluar la calidad del diseño durante el desarrollo del mismo.

El presente documento está estructurado en tres capítulos de la siguiente manera:

Capítulo 1: Fundamentación teórica. Comprende el estudio del estado del arte, necesario para realizar la investigación, garantizándose una buena utilización de los conceptos que serán manejados. Además, se realiza un profundo análisis de las metodologías, herramientas, lenguajes de programación y los patrones que se utilizarán para el desarrollo de la solución.

Capítulo 2: Propuesta de solución. Se realiza un desglose de la propuesta de solución y sus principales funcionalidades. Se describen los casos de uso asociados a las funcionalidades que debe brindar el sistema. Se analiza además la arquitectura sobre la que está soportada la aplicación y se explican detalladamente los patrones de diseño a utilizar; así como la modelación de los diagramas de clases del diseño y de comunicación.

Capítulo 3: Análisis de los resultados. Se analizan y aplican las métricas a utilizar para la validación del diseño. Además, se elaboran y ejecutan los casos de prueba para

verificar el correcto funcionamiento de los componentes implementados, garantizando así la fiabilidad y la calidad de la solución obtenida.

Capítulo 1: Fundamentación Teórica

Introducción

"*En la fundamentación teórica se hace una síntesis de los resultados alcanzados en la revisión bibliográfica relacionados con el tema, se presentan organizadamente los conocimientos científicos acumulados hasta la fecha y los principales autores que trabajan la temática, ubicándolos si es posible en tiempo y espacio*" (Hernández, et al., 2011 p. 23). El presente capítulo abordará aspectos esenciales para el desarrollo de la investigación, especialmente el estudio del proceso Ejecutivo. Este apartado estará enfocado además en analizar las técnicas, arquitectura y patrones de diseño, útiles para la construcción del software. Se analizarán los diferentes lenguajes de programación a utilizar para la implementación del sistema, así como otras herramientas y tecnologías que serán de vital importancia para el desarrollo de la propuesta de solución.

1.1. Descripción del proceso Ejecutivo de la materia Económica

Los Tribunales Populares Cubanos *"son los órganos jurídicos encargados de ejercer la máxima autoridad judicial en la República de Cuba; sus decisiones en este orden son definitivas*" (Tribunal Supremo Popular, 2013). Los tribunales están organizados jerárquicamente: primero el Tribunal Supremo Popular (TSP), seguido por los Tribunales Provinciales Populares (TPP), instancia donde se desarrolla el proceso Ejecutivo y finalmente los Tribunales Municipales Populares (TMP).

El proceso Ejecutivo es el procedimiento responsable de resolver de manera controlada todos los asuntos que le son atribuidos y que versan sobre litigios por incumplimiento, modificación, nulidad, ineficacia o extinción de los contratos económicos. Además, atiende procedimientos como las reclamaciones por daños y perjuicios, de carácter extracontractual, causados a terceros en ocasión del desarrollo de actividad productiva, comercial o de servicios y la presentación de demandas realizadas con motivo de actos o hechos donde intervienen personas naturales o jurídicas. De manera resumida, el proceso Ejecutivo tiene como objetivo presentar tutela inmediata a un acreedor cuyo crédito resulta de una deuda, ya sea líquida, vencida y exigible; además de hacer efectivo el pago de la misma. El pago de la deuda puede asegurarse a través del embargo de los recursos monetarios o bienes del deudor (Ing. Arencibia Cruz, 2012).

El proceso Ejecutivo inicia con la presentación de la demanda, por parte del acreedor o demandante en la Sala de lo Económico del Tribunal Provincial; la misma estará soportada por varios documentos auxiliares, necesarios para presentar ante el Juez Presidente de la sala. Una vez creado el expediente se le asigna a un Juez Ponente para que dé seguimiento a los trámites judiciales del caso.

Posteriormente se da lugar a la admisión de la demanda, acto seguido se notifica al demandante la decisión y se cita al deudor, señalando la fecha para el acto de requerimiento de pago. Si para la celebración del proceso no se cuenta con las evidencias suficientes que prueben la existencia de una deuda, el caso puede anularse y concluir; en caso contrario continúa el procedimiento con la creación del Acta de Requerimiento de Pago.

Cuando el demandado no comparece ante el tribunal después de citado, probando que el mismo tiene una deuda, el tribunal está en el derecho de enviar a la entidad bancaria donde opera el deudor un documento oficial, ordenando el embargo de la cantidad de dinero necesaria para cubrir el monto. Como paso final, el Juez Ponente declara concluso el caso y finaliza con el dictamen de la Sentencia de remate, quedando resuelto el conflicto. Todos los trámites que se realicen en el proceso Ejecutivo cuentan con un límite de tiempo máximo de ocho días y los mismos pueden generar otra cadena de procesos alternos de menor dimensión (Ing. Arencibia Cruz, 2012).

Con la informatización del proceso Ejecutivo se garantiza la sustitución de importaciones en cuanto a sistemas informáticos en el país, de manera que se ayuda a sustentar la economía en cuanto a minimización de costos. Además, se logrará la celeridad de aquellos procesos que tienen una marcada dependencia de la gestión de la información. Por otra parte, son innumerables los beneficios que brindará la realización de este sistema para los Tribunales Populares Cubanos, como por ejemplo: la rapidez en la búsqueda de información, la disminución en el almacenamiento de grandes cantidades de documentos y en la actualización de los mismos, así como la facilidad de trabajo para los jueces y demás trabajadores de dichas entidades. El principal beneficio que se obtendrá es la satisfacción de los clientes a los que los tribunales les prestan servicios, en cuanto a la reducción del tiempo que demora la tramitación de los procedimientos. Al desarrollar esta aplicación se da un paso más de avance en la informatización de la sociedad cubana.

1.2. Metodologías de desarrollo de software

Las metodologías de desarrollo de software son un conjunto de procedimientos, técnicas, herramientas y un soporte documental que ayuda al equipo de proyecto a

realizar un software de calidad. Una metodología puede consistir en la realización de varias tareas o actividades elementales en las cuales se dividen los procesos, donde cada actividad tendrá una forma de ejecutarse y para ello dispondrá de una herramienta como apoyo. La metodología de desarrollo de software responde al cómo se obtendrá el resultado final y los métodos que se aplicarán para lograr tal objetivo.

Pressman[3] define las metodologías de desarrollo de software como "*una herramienta de planificación y control del proyecto que busca de manera precisa obtener los requisitos y el modelo correcto para el software"* (Pressman, 2005 p. 85). Las metodologías de desarrollo de software pueden clasificarse en ágiles o tradicionales.

Las metodologías ágiles están orientadas a entornos donde los sistemas tienen un elevado nivel de variabilidad y exigen reducir considerablemente el tiempo de desarrollo, además de que son asociadas a pequeños sistemas de baja o media complejidad, Sommerville[4] las define como: "*métodos de desarrollo de software dirigidos a la entrega rápida del producto"* (Sommerville, 2005 p. 363). Por otro lado, las metodologías tradicionales se centran en detallar cada uno de los procesos mediante una estricta definición de roles, actividades, artefactos, herramientas y notaciones para el modelado, además de la documentación del producto. Mayormente, este tipo de metodologías son empleadas en los proyectos donde se requiere un alto nivel de organización para gestionar gran cantidad de tiempo y recursos (Sommerville, 2005).

La siguiente tabla comparativa muestra un resumen de algunas de las metodologías de software empleadas en la actualidad:

Metodologías / ***Indicadores***	***RUP*** *(**R**ational **U**nified **P**rocess)*	***XP*** *(e**X**treme **P**rogramming)*	***FDD***[5] *(**F**eature **D**riven **D**evelopment)*
Tamaño de los equipos	Pensada para proyectos y equipos de trabajo grandes	Proyectos con equipos de trabajo pequeños	Depende de las funcionalidades a implementar
Relación con el cliente	Intercambia con el cliente al final de cada fase	Se basa en una fuerte comunicación con el cliente en todo momento (orientada al cliente)	

3 Roger S, Pressman, asesor y autor del libro para estudiantes universitarios: Ingeniería de Software. Un enfoque práctico, que consta de varias ediciones.
4 Ian F. Sommerville, académico británico, investigador y profesor de Ingeniería de Software en la Universidad de St. Andrews en Escocia, autor del popular libro de texto de la misma materia.
5 Desarrollo conducido por características, creada como una alternativa a Scrum.

Nivel de documentación	Demasiada documentación	Apenas genera documentación	Nivel medio, entre RUP y XP
Descripción de requisitos	Casos de uso para describir lo que se espera del software	Historias de usuario y casos de prueba	Basada en funcionalidades
Tipo metodología	Pesada (Tradicional)	Ligera (Ágil)	Varía pesada / ligera
Tiempo duración	Desarrollo a largo plazo (más de un año)	Se implementan mejor para proyectos relativamente cortos donde el tiempo no excede de un año	
Organización y control	Asignación de tareas y responsabilidades por cada iteración	No se tienen metas intermedias sino una general, el producto final	Nivel medio, no tan controlado como RUP ni tan inestable como XP
Vulnerabilidad	Un cambio en las etapas de vida del sistema incrementa el costo en tiempo y dinero	El desarrollo de software es riesgoso y difícil de controlar	Debe de tener en su equipo miembros con experiencia en el desarrollo de software

Tabla 1. Tabla comparativa entre metodologías ágiles y tradicionales.

RUP es la metodología seleccionada por el equipo de desarrollo del SITPC debido a que, entre otros elementos, está integrado por un porciento elevado de personas. Además, las tareas son asignadas en correspondencia con el rol que desempeña cada trabajador, asegurando que cada miembro cumpla con su responsabilidad; debido a que RUP *"provee un enfoque disciplinado en la asignación de tareas y responsabilidades dentro de una organización de desarrollo"* (RUP, 2004 p. 1). Al ser RUP una metodología iterativa e incremental, propicia que sistemas grandes como el SITPC basen su desarrollo en la entrega de artefactos por iteraciones, que de una forma u otra dan solución a una actividad específica y que en cada iteración se incrementan, uniéndolas a las funcionalidades ya implementadas previamente.

1.3. Lenguaje Unificado de Modelado (UML v2.0)

El Lenguaje Unificado para la Construcción de Modelos (UML, Unified Modeling Language) se define como un "*lenguaje que permite especificar, visualizar y construir los artefactos de los sistemas de software. Es un sistema notacional (que, entre otras cosas, incluye el significado de sus notaciones) destinado a los sistemas de modelado que utilizan conceptos orientados a objetos*" (Larman, 1999 p. 15). Desde el

surgimiento de este lenguaje los métodos y técnicas a seguir para el desarrollo del software no han cambiado, sino la manera de realizarse. El nuevo estándar 2.0 del UML está desarrollado sobre la base de creación de un lenguaje de modelado mucho más extensible que el implementado en las versiones anteriores y sobre la validación y ejecución de modelos creados mediante el mismo; favoreciendo a su vez la creación de herramientas que soporten la automatización y generación de código ejecutable, a partir de modelos UML (Lic. Valerio, 2010). Dicho lenguaje será empleado para modelar los diagramas necesarios con el objetivo de estructurar el comportamiento y la relación que debe existir entre los objetos del sistema, de manera que el proceso de modelación se realice sobre la base de un lenguaje estandarizado.

1.4. Arquitectura de software

La arquitectura de un software se caracteriza por utilizar un estilo arquitectónico o la combinación de varios de ellos, que no es más que la organización por la cual estará regido el sistema. La selección y utilización de estilos arquitectónicos no es limitada, pues generalmente sistemas muy grandes aplican más de un estilo para la arquitectura. "*El diseño arquitectónico se centra en la representación de la estructura de los componentes del software, sus propiedades e interacciones"* (Pressman, 1997 p. 238).

1.4.1 Estilos arquitectónicos

Los estilos arquitectónicos también pueden asociarse a la manera en la cual estará organizado el sistema, la estructura y las propiedades de los componentes que el mismo comprende, así como las interrelaciones que tienen lugar entre todos los componentes arquitectónicos del sistema. Cada estilo describe una categoría del sistema que contiene: (1) un conjunto de componentes (una base de datos, módulos computacionales, entre otros) que realizan una función requerida por el sistema; (2) un conjunto de conectores que posibilitan la comunicación, la coordinación y la cooperación entre los componentes; (3) restricciones que definen cómo se pueden integrar los componentes que forman el sistema y (4) modelos semánticos que permiten al diseñador entender las propiedades globales de un sistema, para analizar las propiedades conocidas de sus partes constituyentes (Reynoso, et al., 2004). Dentro de los diversos estilos arquitectónicos que existen, cada uno de ellos cuenta internamente con al menos una clasificación de arquitectura o subestilo:

Estilos Arquitectónicos	
Estilo de flujo de datos	Tuberías y filtros
Estilos centrados en datos	Arquitectura de pizarra o repositorio

Estilos de llamada y retorno	Arquitectura Modelo-Vista-Controlador (MVC) Arquitectura basada en capas Arquitectura orientada a objetos Arquitectura basada en componentes
Estilos peer-to-peer	Arquitectura basada en eventos Arquitectura orientada a servicios Arquitectura basada en recursos
Estilos heterogéneos	Sistemas de control de procesos Arquitectura basada en atributos
Estilos de código móvil	Arquitectura de máquinas virtuales

Tabla 2. Clasificación de los estilos arquitectónicos (Reynoso, et al., 2004).

El equipo de arquitectura del proyecto que desarrolla el SITPC seleccionó, dentro del estilo de llamada y retorno, la arquitectura basada en N-capas y el estilo arquitectónico Modelo-Vista-Controlador (MVC).

Arquitectura basada en N-capas: se define como una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior. Entre sus ventajas se encuentran: (1) la facilidad que ofrece a los desarrolladores de dividir problemas complejos en pequeñas partes, garantizando que la solución del mismo se obtenga sin tener que revisar a profundidad todo el código, también (2) admite mejoras y refinamientos, además de (3) proporcionar amplia reutilización (Reynoso, et al., 2004).

Arquitectura Modelo-Vista-Controlador: es un estilo arquitectónico encargado de separar la lógica de negocio de la interfaz del usuario, tratando al modelo, las vistas y los controladores como entidades separadas (Reynoso, et al., 2004). El estilo MVC divide su estructura en tres componentes centrales: el modelo, encargado de representar la información con la que trabaja la aplicación (lógica del negocio); la vista, dicho componente tiene la tarea de transformar el modelo en una representación visual de fácil interacción para el usuario y por último el controlador, componente que se especializa en procesar los mensajes o eventos que realiza el usuario a través de la vista, realizando los cambios necesarios en la misma (Eguiluz, 2011).

1.4.2 Patrones de diseño orientados a objetos

Un patrón en su definición general es un comportamiento común de varias entidades, características o procedimientos. Dentro del contexto de desarrollo de software un patrón puede definirse como una solución de diseño de software, que se le aplica a un

problema y que puede ser capaz de utilizarse en otros contextos (Alexander, et al., 1977). Son una técnica para flexibilizar el código haciéndolo satisfacer determinados criterios y pueden clasificarse como arquitectónicos, de diseño o de interfaz.

Los patrones de diseño están orientados a describir el cómo se debe construir el software de manera tal que se utilicen correctamente las clases y los objetos que lo componen (Gamma, et al., 1995). Estos patrones se dividen en dos grupos:

- Los patrones GoF[6], los cuales tienen su esencia en el diseño de colaboración entre los objetos. Dichos patrones se clasifican según su propósito en Creacionales, Estructurales y de Comportamiento; según su ámbito en Objeto y de Clase.
- Los patrones GRASP[7], los mismos son expertos en información y tienen como principio básico la asignación de responsabilidades.

Generalmente los patrones de diseño hacen que sea más fácil reutilizar conceptos de diseño y arquitectura. Debido a que ya han sido probados, se convierten en una técnica sencilla y práctica para los desarrolladores de nuevos sistemas.

1.5. Herramientas CASE

Las herramientas de Ingeniería Asistida por Computadores (CASE, Computer Aided Software Engineering) proporcionan ayuda al proceso del software automatizado en algunas de sus actividades como la ingeniería de requisitos, el diseño, el desarrollo de programas y las pruebas; además de brindar información acerca del desarrollo de software (Sommerville, 2005). Existen diversas herramientas CASE que responden a diferentes necesidades de los clientes que las utilizan; entre ellas se pueden encontrar Visual Paradigm, POSEIDON para UML y ArgoUML, por solo mencionar algunas.

Herramientas / ***Indicadores***	***Visual Paradigm***	***POSEIDON para UML***	***ArgoUML***
Interfaz de usuario	Interfaz de usuario bien diseñada, fácil de aprender a usar e intuitiva		Interfaz poco amigable y difícil de comprender
Instalación y uso	Fácil de instalar y usar		Instalación costosa
	Permite la captura de requisitos,		Realiza los diagramas

6 Dado que el libro fue escrito por cuatro autores, a los patrones se les conoce hoy con el nombre de "Gang of Four" (Pandilla de los Cuatro) o por la abreviatura en inglés "GoF".
7 "General Responsibility Assignment Software Patterns", patrones generales de software para asignación de responsabilidades.

Desarrollo	análisis, diseño e implementación a través de un conjunto completo de diagramas		de apoyo a la Ingeniería de Software y aplica la ingeniería inversa a proyectos ya terminados
Integración con otras herramientas	Eclipse/IBM WebSphere, NetBeans IDE, Oracle Jdeveloper, entre otras	_	AndroMDA
Modelación	Crea el esquema de clases a partir de una base de datos y viceversa	Modela cualquier clase de sistema que esté o no relacionada con programación	Carece de soporte completo para algunos tipos de diagramas de secuencia y de colaboración
Vulnerabilidades	-	La grabación de proyectos está limitada a ocho diagramas	Los modelos creados una vez que se cierran no tienen posibilidad de edición

Tabla 3. Comparación entre algunas herramientas CASE.

La tabla describe mediante indicadores, las características esenciales de algunas herramientas útiles para el desarrollo de un sistema informático. Visual Paradigm v8.0 es la herramienta seleccionada por el equipo de desarrollo del SITPC, ya que la misma permite la reutilización del software, la portabilidad y estandarización de la documentación, y el uso de las distintas metodologías propias de la Ingeniería del Software.

Para desarrollar el sistema no solo se emplearon lenguajes para el modelo de diseño como UML v2.0, sino que también se utilizaron diversos lenguajes de programación, base para toda aplicación informática. Algunos de los lenguajes más conocidos y utilizados son: JavaScript, PHP, HTML y otros, debido a las ventajas que proveen al ser basados en códigos abiertos, fáciles y entendibles para el programador; los mismos se describen a continuación.

1.6. Lenguajes de programación

Un lenguaje de programación se utiliza para describir un conjunto de acciones consecutivas que un equipo debe ejecutar. Muchos de los lenguajes a emplear para implementar la solución forman parte de las tecnologías y arquitecturas utilizadas.

1.6.1 PHP v5.3

PHP es un lenguaje de programación del tipo software libre, que trabaja del lado del servidor (server-sided) y es usado para la creación de páginas dinámicas. Su funcionamiento básico está basado en la realización de peticiones que el cliente hace al servidor, el intérprete de PHP traduce las peticiones y genera dinámicamente en una página web la respuesta del servidor. Entre las ventajas que más resaltan, y es importante mencionar, se encuentran que:

- Es un lenguaje multiplataforma, práctico y fácil de usar.
- Es un software de código abierto.
- Está completamente orientado al desarrollo de aplicaciones web dinámicas, con acceso a información almacenada en una base de datos.
- Ofrece una programación segura y confiable, donde el cliente y el navegador nunca tienen acceso al código fuente (The PHP Group, 2001-2014).

1.6.2 HTML5

El Lenguaje de Marcado de Hipertexto (HTML, HyperText Markup Language) permite escribir textos de forma estructurada y se basa en etiquetas que marcan el inicio y fin de cada elemento dentro del documento (Eguiluz, 2011). HTML5 es la versión más reciente del estándar HTML, es la unión y evolución del HTML4 con el API[8] Modelo de Objeto de Documentos 2 (DOM 2, Document Object Model 2), permitiendo la creación de nuevas APIs y un desarrollo de aplicaciones prácticas, ágiles y dinámicas. HTML5 posee características necesarias para la creación de aplicaciones modernas basadas en un navegador sin violar la compatibilidad, gracias al uso de JavaScript, CSS3 y su integración en marcos de trabajo (W3C Nightly, 2013).

1.6.3 CSS3

Las hojas de estilo en cascada (CSS, Cascading Style Sheets) es un lenguaje que surge como complemento para definir la estética de un documento estructurado o HTML (W3C, 2013). El objetivo del lenguaje es separar la estructura de un documento de su presentación. Con CSS3 se obtiene un mayor control de la presentación del sitio al poder tener todo el código CSS reunido en uno, lo que facilita su modificación; también se consigue una mayor legibilidad del código HTML al tener el código CSS

8 Una API (Interfaz de Programación de Aplicaciones) es un conjunto de funciones que permite al programador acceder a servicios de una aplicación a través del uso de un lenguaje de programación.

independiente. Como principales oportunidades ofrece el ahorro de tiempo y la creación de una vista agradable para el cliente, lograda a través del uso de estilos y efectos visuales (CSS, 2012).

1.6.4 JavaScript 3

JavaScript es un lenguaje de programación que se utiliza principalmente para crear páginas web dinámicas. Es un lenguaje interpretado, donde los programas escritos en dicho lenguaje se pueden probar directamente en cualquier navegador sin necesidad de realizar procesos intermedios. Actualmente JavaScript se encuentra en su versión correspondiente a la tercera edición del estándar ECMA-262[9] (Eguiluz, 2007).

El lenguaje JavaScript es idóneo para desarrollar aplicaciones de alta seguridad, por su facilidad de diseño para ejecutarse en entornos muy limitados; de manera que permita a los usuarios confiar en la ejecución de los scripts (Eguiluz, 2007).

1.7. Marco de trabajo

Los lenguajes de programación previamente descritos forman parte de un conjunto estandarizado de conceptos, prácticas y criterios, que unidos en un solo marco de trabajo, pueden complementarse y ser la base para la organización y desarrollo de un software. En la actualidad los sistemas informáticos se están confeccionando sobre la base de técnicas de desarrollo rápido, las cuales se utilizan para construir aplicaciones donde es obligatorio el uso intensivo de datos. Cualquiera de las técnicas usadas para el desarrollo del software, siempre que sea posible, pueden ser ensambladas y formar un conjunto de herramientas capaces de crear, buscar y visualizar datos; así como presentar reportes sobre los mismos (Sommerville, 2005).

Un marco de trabajo contiene un conjunto de herramientas útiles que fusionadas son capaces de mostrar a los usuarios finales el resultado deseado. "*Un marco de trabajo (o marco de trabajo de aplicaciones) es un diseño de un subsistema formado por una colección de clases concretas y abstractas y la interfaz entre ellas. Los detalles particulares del subsistema de aplicación son implementados añadiendo componentes y proporcionando implementaciones concretas de las clases abstractas en el marco de trabajo*" (Sommerville, 2005 p. 390).

1.7.1 Symfony v2.1

Symfony v2.1 es una de las recientes versiones de Symfony, el popular marco de trabajo creado para desarrollar aplicaciones PHP. El marco de trabajo antes mencionado ha sido ideado para aprovechar al límite todas las nuevas características

9 ECMA International es una asociación fundada en 1961 y dedicada a la estandarización de las Tecnologías de Información y las Comunicaciones. ECMA creó el comité TC39 para estandarizar el lenguaje JavaScript 1.1. El primer estándar que se creó fue denominado ECMA-262.

de PHP v5.3 y versiones posteriores, siendo uno de los marcos de trabajo con mejor rendimiento actualmente. Su arquitectura interna está completamente separada por componentes, lo que permite reemplazar o eliminar con gran facilidad aquellas partes del sistema que no se ajusten a lo requerido. Aunque su creador Fabien Potencier aclara que: "*Symfony2 no es un framework MVC. Symfony2 sólo proporciona herramientas para la parte del Controlador y de la Vista. La parte del Modelo es responsabilidad tuya (…)*" (Potencier, 2011), Symfony puede basar su funcionamiento en este clásico estilo de diseño web conocido como arquitectura Modelo-Vista-Controlador (MVC); el mismo se enfoca principalmente en la reutilización de código y la separación de conceptos (Eguiluz, 2011).

Las facilidades de funcionamiento que presenta Symfony v2.1 es una de las razones por las que dicho marco de trabajo fue seleccionado por el proyecto SITPC, para funcionar como plataforma de desarrollo genérica, la cual trabaja internamente con tecnologías avanzadas como el marco de trabajo Doctrine v2.0. Cuenta además con la biblioteca jQuery v1.8 y el motor de plantillas Twig, los cuales unidos al marco de trabajo Bootstrap v2.1, se encargan de gestionar las vistas y permitir una mejor interacción con la aplicación.

1.7.1.1 Twig, motor de plantillas de Symfony

Twig es un motor de plantillas rápido y fácil de usar pensado para PHP. El principal objetivo de Twig es mejorar, en cuanto a tiempo, la elaboración y ejecución de las plantillas hasta código PHP en un número reducido de líneas de código, obteniendo el mismo resultado que si se utilizara código PHP puro. El motor Twig ofrece un nivel de seguridad elevado, pues es capaz de evaluar el código no confiable contenido en la plantilla; lo que permite utilizarlo como un lenguaje de plantillas para aplicaciones donde los usuarios pueden modificar el diseño de la misma (Pacheco, 2011).

Las plantillas en Symfony2 pueden incluir dos extensiones, la primera especifica el formato del contenido que genera la plantilla (.html, .css, entre otras) y la segunda indica el motor de plantillas a utilizar (.twig o .php). Para la generación de las vistas, el equipo de arquitectura del SITPC decidió utilizar el motor de plantillas de Symfony, Twig; dentro de los beneficios que ofrece están: la sencillez de su sintaxis y la implementación de herencia múltiple de plantillas, que es la parte más poderosa de Twig. La herencia permite crear una estructura de la plantilla base, que contenga todos los elementos comunes usados frecuentemente por el sistema; definiendo los bloques que las plantillas descendientes pueden sustituir (Pacheco, 2011).

1.7.1.2 Componente de seguridad AcaXia

Symfony v2.1 implementa un conjunto de políticas de seguridad para garantizar la integridad de aquellos sistemas que se desarrollan sobre la base de este marco de

trabajo. Sin embargo, la seguridad de cualquier sistema informático está en dependencia del nivel de confidencialidad de la información que este maneja; es por ello que depende de la organización desarrolladora determinar si es suficiente o no la seguridad que ofrece el marco de trabajo antes mencionado. Para la implementación del SITPC, el equipo de arquitectura decidió emplear como componente de seguridad el Sistema de Gestión Integral de Seguridad AcaXia, además de la seguridad que ofrece la versión 2.1 de Symfony.

AcaXia es una herramienta de gestión que permite conocer, gestionar y minimizar los posibles riesgos que atentan contra la seguridad de la información, además de garantizar su confidencialidad, integridad y disponibilidad. Dicho componente tiene su fortaleza en sus cinco “A”:

- Autenticación: propiedad que permite identificar si el usuario es realmente quien dice ser. En el proyecto se empleó el mecanismo de autenticación personalizada para todos los usuarios que harán uso de la aplicación, garantizando la fortaleza mediante el uso de contraseñas.
- Autorización: según el nivel de sensibilidad de la información los permisos son asignados a los usuarios a través de roles. Para el sistema, el equipo de desarrollo definió utilizar una jerarquía de usuarios en correspondencia con las acciones que pueden realizar, estos podrán ser activados y desactivados por el administrador. Además, los permisos serán limitados basándose en los roles definidos para cada usuario y en los niveles de acceso a la información.
- Auditoría: evaluar y obtener de manera objetiva las evidencias relacionadas mediante la visualización de informes sobre actividades realizadas en el sistema. Esta propiedad es usada dentro del sistema para registrar las trazas de error, de rendimiento y de acceso, con el objetivo de poder auditar el recorrido histórico de los recursos y los usuarios del sistema, para ello se crea el rol de auditor con los permisos necesarios para examinar y procesar las trazas generadas dentro de la aplicación.
- Administración de perfiles: la configuración de los perfiles de usuarios tiene la función de registrar datos importantes y únicos de los usuarios en el sistema, que luego serán almacenados en la base de datos. Dentro del sistema las cuentas de usuario no pueden ser compartidas, es decir, una cuenta de usuario puede tener una sola sesión activa en el servidor, ello se puede comprobar mediante los datos registrados en el perfil de usuario.
- Administración de conexiones: solo están autorizadas a acceder al sistema aquellas direcciones o rangos de IP (Protocolo de Internet) que sean configuradas como accesibles (Ing. Roque Hernández, 2013).

1.7.2 Bootstrap v2.1

Bootstrap es un marco de trabajo que simplifica el proceso de creación de diseños web, utilizando algunas de las técnicas más modernas de los navegadores para ofrecer estilos de tipografías, formularios, botones, tablas, entre otros. Está basado en los últimos estándares de desarrollo web: HTML5, CSS3 y JavaScript/jQuery. Bootstrap tiene como una de su principales características la creación de interfaces adaptables a diferentes navegadores; asegurando que cualquier aplicación que se realice sea soportada por cualquier tipo de tecnología (Sánchez, 2013).

1.7.3 jQuery v1.8

jQuery es una biblioteca de JavaScript de código abierto que permite: simplificar la manera de interactuar con los documentos HTML, manejar eventos, desarrollar animaciones y agregar interacción a páginas web. La misma ofrece una serie de funcionalidades basadas en JavaScript, las cuales logran grandes resultados en menos tiempo y espacio (ScottGu, 2008).

Dicha biblioteca consiste en un único fichero JavaScript que contiene las funcionalidades comunes de eventos y efectos. La característica principal de la biblioteca es que permite cambiar el contenido de una página web sin necesidad de recargarla, mediante la manipulación del árbol DOM y peticiones AJAX[10] (Murphey, et al., 2013).

1.7.4 Mapeador de Objeto-Relacional (ORM)

Los Mapeadores Objeto-Relacional (ORM, Object-Relational Mapping) consisten en una serie de objetos que permiten acceder a los datos almacenados en un gestor de base de datos y que relacionados entre sí contienen cierta lógica de negocio. Los ORM utilizan una capa de abstracción (DBAL, Data Base Abstraction Layer) para transformar automáticamente las llamadas a los objetos en consultas SQL (Lenguaje de consultas estructurado), evitando la utilización de sintaxis específicas de un sistema de base de datos concreto. La llamada de métodos de un objeto de datos desde varias partes de la aplicación hace de los ORM una herramienta óptima para la reutilización de código (Potencier, et al., 2007).

Doctrine v2.0: es un ORM situado en la parte superior de una poderosa capa de abstracción de base de datos. La principal tarea de Doctrine es la traducción entre objetos (PHP) y las filas relacionales de la base de datos. Además, tiene la característica de escribir las consultas de base de datos en su propio dialecto, llamado Lenguaje de Consulta de Doctrine (DQL, Doctrine Query Language). Su sintaxis es similar al lenguaje SQL, pero su funcionamiento es completamente distinto: DQL

10 ***A**synchronous **J**avaScript **A**nd **X**ML* (JavaScript asíncrono y XML), técnica de desarrollo web usada para crear aplicaciones interactivas.

realiza consultas sobre objetos, mientras que SQL realiza consultas sobre tablas.

Doctrine v2.0 brinda persistencia a los objetos PHP de manera transparente, no enfoca su trabajo en la inserción o búsqueda de filas y/o columnas en la base de datos; sino que trata de hacer persistir y buscar objetos hacia y desde la base de datos. Además, dicho lenguaje permite a los desarrolladores escribir poderosas consultas de una manera sencilla y flexible, puesto que es Doctrine v2.0 internamente quien se encarga de las conversiones para gestionar la información de la base de datos. Las entidades de Doctrine v2.0 son clases escritas en lenguaje PHP puro (Eguiluz, 2011).

1.8. Tecnologías de desarrollo

Las tecnologías de desarrollo son un conjunto de conocimientos técnicos ordenados científicamente, que permiten diseñar y crear bienes y servicios que facilitan la adaptación al medio ambiente y satisfacer tanto las necesidades esenciales como los deseos de la humanidad (Doval, et al., 1995). Para el desarrollo de software se utilizan herramientas que ofrezcan un resultado final de calidad, mientras más avanzadas sean las mismas, más eficiente y eficaz será el producto resultante. Para el desarrollo del SITPC se utilizan una serie de herramientas de desarrollo que a continuación se exponen.

1.8.1 NetBeans IDE v7.4

Un Entorno Integrado de Desarrollo (IDE, Integrated Development Environment) es un entorno de programación que ha sido empaquetado como un programa de aplicación, o sea, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica. Los IDE pueden ser aplicaciones por sí solos o pueden ser parte de aplicaciones existentes (Maldonado, 2007). Entre otras características, ofrecen un marco de trabajo amigable para la mayoría de los lenguajes de programación tales como C++, Python, Java, C#, Delphi, Visual Basic y otros.

NetBeans IDE v.7.4: es un entorno de desarrollo de código abierto integrado con PHP, permite la creación de aplicaciones web y propone una estructura para organizar el código fuente. Es multiplataforma y el editor conjuntamente integra lenguajes como HTML, JavaScript, CSS y PHP. Brinda apoyo al marco de trabajo Symfony2, el cual se emplea en el desarrollo del SITPC, lo que permite dejar a un lado la consola de Symfony centrándose en el desarrollo de la aplicación mediante este entorno de desarrollo. NetBeans IDE v.7.4 puede realizar un conjunto de operaciones desde el mismo editor de código fuente, lo que reduce el tiempo en el desarrollo y el consumo de varios recursos en solo uno, además se integra con el control de versiones SubVersion v1.5 (netbeans.org, 2000).

1.8.2 PostgreSQL v9.2

Un Sistema Gestor de Base de Datos (SGBD) es un sistema de software que permite la definición de bases de datos; así como la elección de las estructuras de datos necesarios para el almacenamiento y búsqueda de los mismos, ya sea de forma interactiva o a través de un lenguaje de programación. Los SGBD relacionales son una herramienta efectiva que permite a varios usuarios acceder a los datos al mismo tiempo. Brindan facilidades eficientes y un grupo de funciones con el objetivo de garantizar la confidencialidad, la calidad, la seguridad y la integridad de los datos que contienen, así como un acceso fácil y eficiente a los mismos (Bertino, et al., 1995). En la actualidad algunos de los SGBD más usados y eficientes son: Oracle, DB2, MySQL, MS SQL Server y PostgreSQL; siendo este último el usado para crear y administrar la base de datos del SITPC.

PostgreSQL v9.2: es el SGBD objeto-relacional de código abierto más potente del mercado, constando ya con una arquitectura suficientemente probada, que se ha ganado una sólida reputación de confiabilidad, integridad de datos y corrección, convirtiéndolo en el líder entre los gestores de bases de datos de código abierto (PostgreSQL, 2010).

1.8.3 SubVersion v1.5

Un sistema de control de versiones es un software que administra el acceso a un conjunto de ficheros y mantiene un historial de cambios realizados. Es útil para guardar cualquier documento que cambie con frecuencia, como código fuente, documentación o ficheros de configuración. Se puede clasificar en distribuido o centralizado (Ministerio de Educación de España, 2008).

SubVersion v1.5: SubVersion (SVN) fue creado con el objetivo de ser una alternativa real y mejorada a un sistema de control de versiones obsoleto. SVN es libre y de código abierto, ya que está distribuido bajo la licencia Apache. La versión 1.5 ofrece mejoras en algunas características ya existentes: opciones de resolución de conflictos más interactiva, revisiones parciales, administraciones parciales de los cambios desde el lado del cliente (client-side), definiciones externas basados en archivos y apoyo operacional de registro para SVNServer. Entre las operaciones más frecuentes que realiza se encuentran: actualizar, confirmar, descargar, ver historial, entre otras (Collins-Sussman, et al., 2008). Para mantener una sincronización rápida, en cuanto a las actualizaciones de cada una de las actividades que se desarrollan dentro del SITPC, fue necesario emplear el SVN de manera centralizada. Estas operaciones se realizan replicando el modelo cliente/servidor, donde todas las versiones se encuentran almacenadas en un único directorio (denominado repositorio de fuentes)

de un ordenador (un servidor); las versiones pueden ser accedidas por clientes instalados en los ordenadores.

1.8.4 Apache v2.2

Un servidor web es un programa informático que procesa aplicaciones realizando conexiones bidireccionales y/o unidireccionales y síncronas o asíncronas con el cliente, generando o cediendo una respuesta en cualquier lenguaje o aplicación del lado del cliente. Es quien gestiona la mayor parte de las funciones de lógica de negocio y de acceso a los datos de la aplicación. Los principales beneficios de la aplicación de la tecnología de servidores web son la centralización y la disminución de la complejidad en el desarrollo de aplicaciones (Apache, 2012).

Servidor web Apache v2.2: Apache es un servidor web HTTP muy potente, gratuito, de código abierto, flexible, fácil de mantener y altamente configurable debido a su diseño modular. Es multiplataforma y presenta abundante documentación (Apache, 2012), adicionalmente tiene dentro de sus principales ventajas el funcionamiento con varios lenguajes, incluyendo PHP.

Conclusiones parciales del capítulo

En el capítulo se han analizado los conceptos fundamentales de desarrollo de software, de manera que se pueda seguir el curso de la investigación, llegándose a las siguientes conclusiones:

- El estudio del proceso Ejecutivo proporcionó sólidos conocimientos sobre cómo debe funcionar dicho proceso y sus principales características, las cuales son útiles para complementar la solución.
- Después de caracterizar las tecnologías, técnicas y metodologías a utilizar se pudo comprobar que estas eran las adecuadas para implementar un sistema de calidad, acorde a las nuevas tecnologías de la información y las comunicaciones.
- Los patrones de diseño seleccionados garantizan la correcta comunicación entre los objetos del sistema, asegurando la excelencia del diseño y de las fases posteriores.

Capítulo 2: Propuesta de solución

Introducción

En el presente capítulo se abordarán los aspectos técnicos de la investigación, así como las principales funcionalidades que el sistema debe implementar y la arquitectura utilizada en el desarrollo del mismo. Como aplicación de buenas prácticas para el desarrollo de software se tuvo en cuenta el uso de patrones de diseño aplicados en la solución propuesta y la utilización de estándares de codificación en la implementación. Además, para el modelo de diseño se obtendrán los diagramas de clases y de comunicación correspondientes a los Flujos alternos.

2.1. Descripción de los requisitos funcionales

En la iteración preliminar, realizada por los analistas del módulo, fueron definidos los requisitos funcionales y no funcionales que se debían implementar para el mismo. Como resultado del desarrollo de esta etapa se concluyó que el proceso de ingeniería de requisitos[11] se realizó correctamente (Ing. Lamorú Marciel, 2013). En una segunda iteración del módulo Ejecutivo, se pretende realizar el diseño e implementación de los requisitos funcionales correspondiente a los Flujos alternos. El módulo cuenta para este flujo con 16 requisitos funcionales, los cuales se enumeran a continuación:

1. Crear ANHLA[12] por falta de subsanación.
2. Crear comunicación sobre no admisión de demanda.
3. Remitir oficio de Instrucción 215.
4. Disponer sobre contestación.
5. Declarar no presentada la contestación.
6. Disponer sobre escrito de oposición.
7. Crear providencia abriendo a pruebas.
8. Disponer sobre modificación de pretensiones.
9. Visualizar expedientes decursando término[13].
10. Crear providencia uniendo respuesta de banco.
11. Declarar firmeza de ANHLA por falta de subsanación.
12. Declarar firmeza de sentencia.
13. Disponer sobre personería de abogado.

11 Definido por Sommerville como: "Proceso de descubrir, analizar, documentar y verificar" (Sommerville, 2005 p. 108) los requisitos funcionales y no funcionales del sistema.
12 ANHLA: Auto No Haber Lugar a Admitir.
13 Expedientes que están esperando días para la celebración del acto.

14. Declarar contestación extemporánea[14].
15. Declarar oposición extemporánea.
16. Declarar subsanación extemporánea.

Después de haber sido analizados los requisitos funcionales, estos fueron transformados en 16 casos de uso (CU) para cumplir con las especificaciones del sistema. En la tabla que se presenta a continuación se describen brevemente los casos de uso a implementar en el segundo paquete del módulo Ejecutivo.

No.	Caso de uso	Funcionalidad
1	Declarar subsanación extemporánea	Permite crear la providencia declarando extemporánea la subsanación.
2	Declarar contestación extemporánea	Permite crear la providencia declarando extemporánea la contestación.
3	Declarar oposición extemporánea	Permite crear la providencia declarando extemporánea la oposición.
4	Declarar no presentada la contestación	Permite crear la providencia declarando la contestación como no presentada, luego de haber sido enviado a subsanar el escrito.
5	Declarar firmeza de sentencia	Permite crear la providencia declarando la firmeza de la sentencia.
6	Declarar firmeza de ANHLA por falta de subsanación	Permite crear la providencia de firmeza de ANHLA por falta de subsanación.
7	Crear providencia uniendo respuesta del banco	Permite crear la providencia uniendo la respuesta del banco.
8	Crear ANHLA por falta de subsanación	Permite crear el ANHLA la demanda por no cumplir con el tiempo establecido para subsanar los errores.
9	Crear providencia abriendo pruebas	Permite crear la providencia solicitando las pruebas a las partes.
10	Crear oficio de contraloría a la fiscalía	Permite remitir el oficio de contraloría a la fiscalía correspondiente.
11	Crear comunicación	Permite crear el oficio de Instrucción 215.

14 Escritos fuera de tiempo, no se cumplió con los días hábiles para presentar dichos documentos.

	sobre no admisión de demanda	
12	Disponer sobre escrito de modificación de pretensiones	Permite admitir o subsanar el escrito modificando las pretensiones.
13	Disponer sobre escrito de personería de abogado	Permite admitir o disponer subsanación del escrito de personería.
14	Disponer sobre escrito de oposición	Permite disponer sobre el escrito de oposición, admitiendo o solicitando la subsanación del mismo. Además se especifica si se solicitan o no pruebas para la oposición.
15	Visualizar expedientes decursando término	Lista los expedientes que decursan término.
16	Disponer sobre escrito de contestación	Permite disponer sobre el escrito de contestación, admitiendo o solicitando la subsanación del mismo.

Tabla 4. Breve descripción de la función de los casos de uso.

Para ejemplificar cada uno de los pasos a realizar como parte de la solución se tomó como referencia el caso de uso Disponer sobre escrito de oposición, en la tabla 5 se muestra la información relevante del caso de uso (Anexo 1 muestra la descripción completa).

Objetivo	El CU permite admitir o disponer la subsanación del escrito de oposición.
Actores	Juez Ponente (inicia)
Resumen	El CU se inicia cuando el juez accede al sistema para disponer sobre el escrito de oposición. Para ello selecciona el expediente y una de las opciones disponibles.
Complejidad	Alta
Prioridad	Media
Precondiciones	El expediente ha pasado al estado “pendiente a disponer sobre oposición”.
Post-condiciones	Si se admitió el escrito sin solicitar pruebas, el estado del expediente cambia a “concluso”.

	Si se admitió el escrito solicitando pruebas, el estado del expediente cambia a “abriendo pruebas”. Si se dispuso subsanación del escrito, el expediente cambia a “pendiente de subsanación”.

Tabla 5. Descripción del CU Disponer sobre escrito de oposición.

2.2. Arquitectura del sistema

Como parte importante del desarrollo de la solución se encuentra la definición de la arquitectura del sistema, en este caso el equipo de desarrollo del SITPC ha seleccionado la arquitectura basada en N-capas en conjunto con el estilo arquitectónico Modelo-Vista-Controlador (MVC). Este último se emplea no solo porque el marco de trabajo Symfony2 lo implementa internamente, sino por las facilidades de trabajo que ofrece y por la simplicidad que presenta en su estructura. Además detalla las responsabilidades exactas de cada capa y la forma en que estas se relacionan, intentando separar la capa visual gráfica de su correspondiente programación y acceso a datos, propiciando la reutilización de código.

La arquitectura utilizada para el diseño de la aplicación está estructurada en cuatro capas (N=4): de presentación, de negocio, de abstracción de datos y de datos. Una regla importante que debe cumplir esta arquitectura es que cada capa debe actuar como cliente de la capa inferior a ella y como servidor de la capa superior; de esta forma se garantiza no violar la seguridad y la lógica del sistema.

De manera transversal a las capas se encuentran las entidades del dominio y los componentes útiles para el tratamiento de la seguridad (AcaXia), así como las excepciones, inyección de dependencias, entre otros. Todo lo descrito anteriormente es posible observarlo en la siguiente figura:

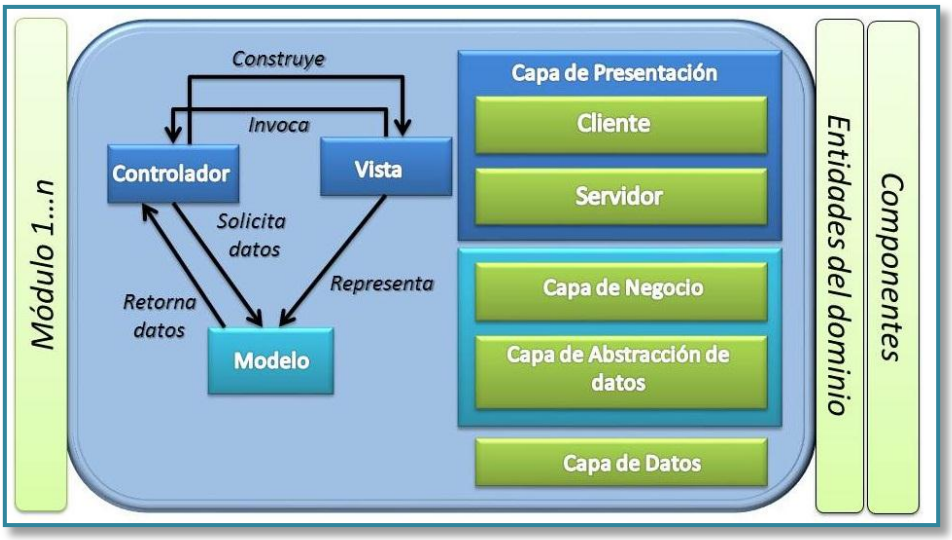


Figura 1. Arquitectura del sistema.

Capa de presentación: se encuentran en esta capa las interfaces de los usuarios y la lógica de control. Mediante las interfaces se muestra o capta la información utilizando componentes propios de las vistas (formularios, grids, mensajes de estado o confirmación, entre otros); a la vez que recibe las funcionalidades que le son enviadas desde el controlador. La misma implementa internamente dos subcapas, denominadas cliente y servidor.

- Cliente: es la interfaz que observa el cliente. Utiliza los lenguajes HTML5, CSS3, el marco de trabajo Bootstrap v2.1 y la biblioteca jQuery v1.8.
- Servidor: tiene la responsabilidad de gestionar la lógica de control y la conversión de páginas mediante el motor de plantillas Twig.

Capa de negocio: gestiona y procesa las peticiones recibidas desde el nivel superior, para su posterior envío de la respuesta. Esta capa es la encargada de proveer a las interfaces la información que soliciten a través de las clases gestoras.

Capa de abstracción de datos: tiene la función de gestionar las peticiones de consultas a la base de datos, recibidas desde la capa de negocio. En esta capa se encuentran las clases repositorios y entidades.

Capa de datos: ofrece las herramientas necesarias para manipular y almacenar la información con la que trabajará la aplicación (SGBD PostgreSQL v9.2); almacenando los esquemas, tablas, funciones y procedimientos necesarios para su posterior utilización.

En necesario aclarar que las capas de negocio y de abstracción de datos tienen la particularidad de operar junto a la capa de modelo que implementa el estilo arquitectónico MVC, al contrario de la capa de datos, la cual se encuentra independiente de las demás capas.

El sistema está compuesto por diferentes subsistemas según la materia judicial que se requiera implementar. Cada materia desarrolla diferentes procesos, los cuales representan módulos o bundles[15] del subsistema; en este caso la materia Económica contiene al módulo Ejecutivo. Dicho módulo cuenta con una arquitectura organizativa claramente detallada, donde cada archivo es guardado en la carpeta correspondiente a su clasificación, como se muestra en la siguiente figura:

15 Bundle (paquete): Es un conjunto estructurado de archivos que implementa una sola característica y que fácilmente se puede compartir con otros desarrolladores; todo el código de la aplicación está organizado en paquetes.

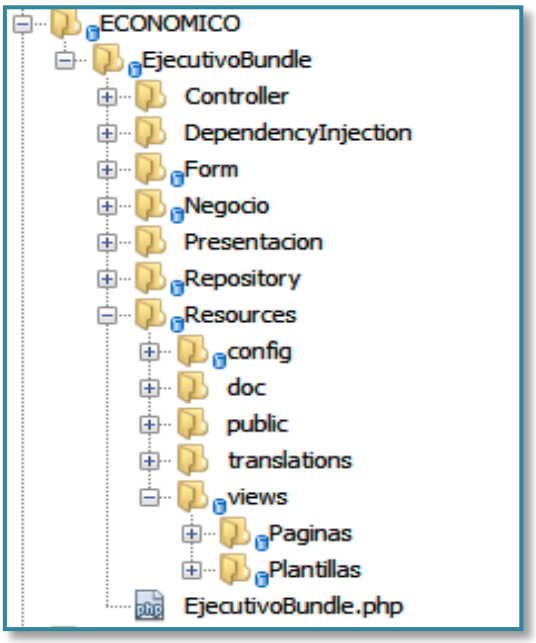


Figura 2. Estructura organizativa propuesta por Symfony2.

En el directorio **EjecutivoBundle\Resources\views** se encuentran las vistas, encargadas de representar al modelo, estas pueden acceder a dicho componente sin tener que cambiar su estado. Las vistas son ficheros de extensión *.html.twig* que tienen su fortaleza en la herencia entre plantillas. Para lograr mostrar una vista agradable al usuario se utilizan diferentes herramientas como el marco de trabajo Bootstrap v2.1, la biblioteca jQuery v1.8 y el motor de plantillas Twig.

En la dirección **EjecutivoBundle\Controller** se encuentran todas las clases controladoras, las cuales tienen la responsabilidad de recibir los eventos de entrada desde la vista, reaccionando a la petición del cliente; para luego crear e invocar cualquier objeto del modelo para que procese la información, sin necesidad de establecer una conexión directa entre la vista y los objetos de datos.

En el directorio **EjecutivoBundle\Negocio\Gestor** las clases gestoras representan el núcleo de la aplicación, las mismas trabajan independientes de la vista y del controlador. Las clases gestoras tienen la función de: (1) gestionar el acceso a la información, mediante el uso de consultas, (2) enviar a la clase controladora aquella parte de la información obtenida de las entidades, que en cada momento solicita la vista y por último, (3) en ellas se encuentra toda la lógica del negocio.

La ruta **EjecutivoBundle\Repository** contiene las clases repositorios, encargadas de realizar las consultas a la base de datos mediante el marco de trabajo Doctrine v2.0 utilizando su propio dialecto (lenguaje DQL), como se puede observar en la figura 3. Este tipo de clases pueden estar ubicadas también en "*SIT\vendor\Base\ComunBundle\Repository*" para uso general. Por último las clases entidades son definidas de manera general dentro del sistema.

```
//obtiene TODAS las deudas de un expediente(activas, pdtes de eliminar y pdtes de activar)
public function obtenerDeudasExpediente($idexpediente, $resultType = ResultType::ObjectType)
{

      $em = $this->getEntityManager();
      $qb = $em->createQueryBuilder()
              ->select('d')
              ->from('ComunBundle:Economico\Deuda', 'd')
              ->join('d.documento', 'docG')
                              ->join('docG.escrito', 'esc')
              ->where('esc.expediente = :idexp')
              ->setParameter('idexp',$idexpediente);

      return UtilRepository2::doResult($qb, $resultType);

  }
```

Figura 3. Ejemplo de consulta realizada con lenguaje DQL en la clase DeudaRepository.

2.2.1 Modelo de datos

El modelado de datos responde a una serie de interrogantes que se pueden responder mediante el procesamiento de información; por ejemplo, el Diagrama Entidad-Relación (DER) representa las relaciones entre los objetos de datos, es la notación que se usa para realizar la actividad de modelado de datos. Mediante dicho diagrama el ingeniero de software identifica objetos de datos y sus relaciones a través de una notación gráfica, con el objetivo de definir los datos que serán de utilidad para desarrollar la aplicación. *"A nivel de aplicación, la traducción de un modelo de datos (…) en una base de datos es el punto clave para alcanzar los objetivos de negocio del sistema"* (Pressman, 1997 p. 239). El modelo de datos está conformado por tres elementos fundamentales:

- ✓ Entidades (objetos con existencia física o conceptual).
- ✓ Atributos (características que definen o identifican a una entidad).
- ✓ Relaciones (dependencia o asociaciones entre las entidades).

El modelo de datos correspondiente al módulo Ejecutivo consta de un total de 17 tablas, de ellas 5 son nomencladores y las 12 restantes son de información referente a los objetos.

Las entidades del modelo representan, en su mayoría, una tabla en la base de datos que permiten manejar la información de los objetos. Cada tabla tiene una forma de nombrarse con el objetivo de diferenciar cada una de ellas, así como su funcionalidad. Las tablas cuyos nombres comienzan con la letra **-d-** en minúscula representan una agrupación de datos, por ejemplo, **dDeuda**.

Las tablas que comienzan con la letra **-n-** representan nomencladores. Un nomenclador se define como una forma de clasificar o agrupar diversas prácticas, actividades o cualquier conjunto de palabras técnicas propias de una ciencia. En este caso la entidad **nClasificacionDeuda** almacena los diferentes tipos de deudas; las

cuales pueden ser principales, de intereses moratorios o de costas procesales. Por último, se encuentran en el modelo las relaciones; las mismas comienzan su nombre con la letra **-r-**, pueden ser entre dos o más tablas y contienen los atributos necesarios para gestionar las relaciones de mucho a mucho, por ejemplo **rCuentaLitigante**. Lo anteriormente explicado se puede evidenciar en la siguiente figura:

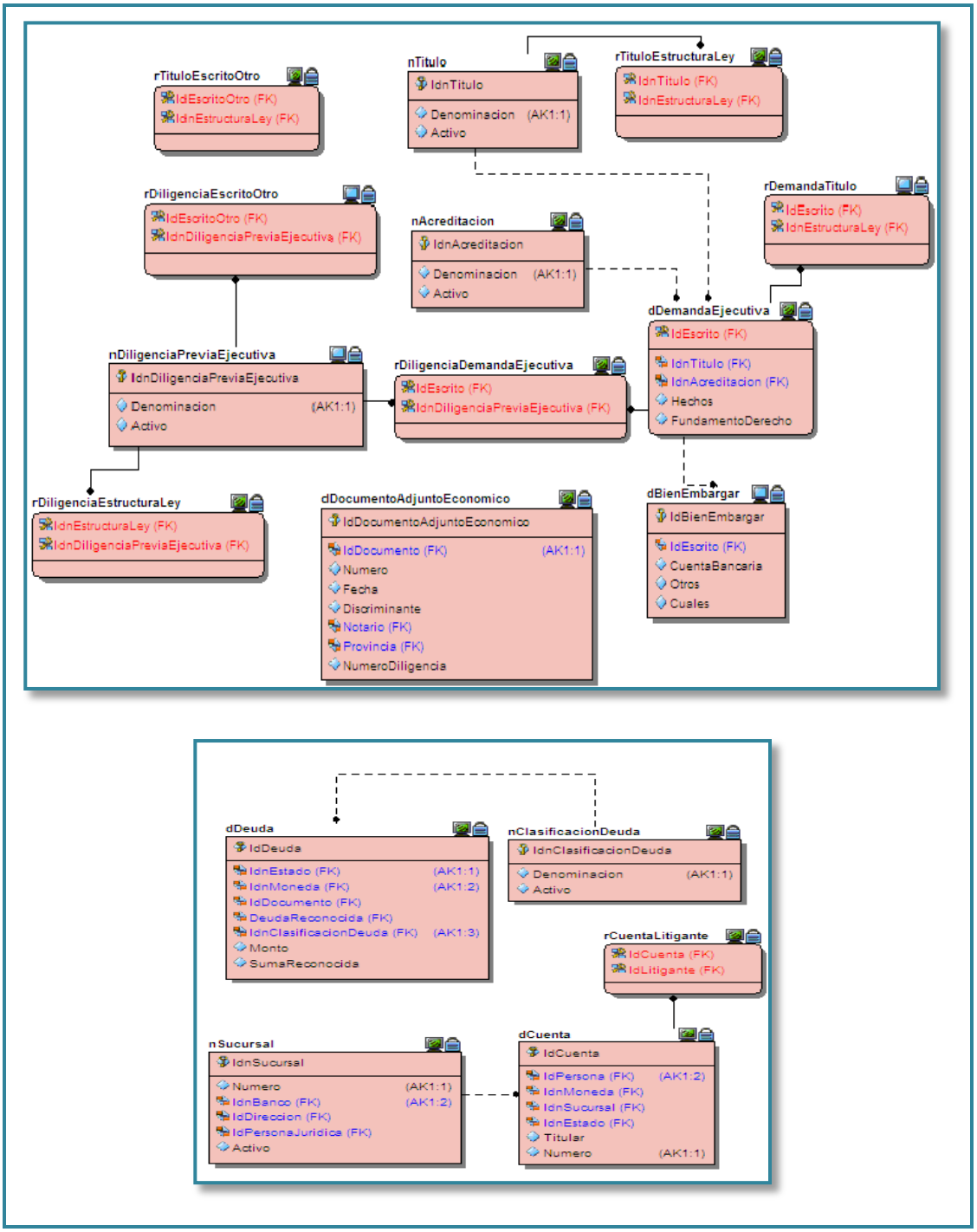


Figura 4. Modelo de datos del subsistema Económico.

Tratamiento de las entidades con Doctrine

En el capítulo anterior fueron analizadas las tecnologías y herramientas que servirían de soporte para desarrollar la propuesta de solución al problema planteado, entre las

que se encontraba el marco de trabajo Doctrine v2.0. En este acápite se analizará la manera en que Doctrine v2.0 mantiene sincronizada la base de datos de la aplicación a partir de los objetos construidos, es decir, las clases PHP denominadas entidades. Es importante mencionar que este ORM no es exclusivo de Symfony2, sino que se puede utilizar en cualquier proyecto que se desarrolle con PHP. Doctrine v2.0 realiza la traducción entre los objetos PHP y cada una de las tuplas[16] de la base de datos a través del mapeo de datos. Existen distintas maneras de mapear la información de las clases: mediante archivos XML, archivos YAML o directamente en la clase con anotaciones Docblock o bloque de documentación, este tipo de información se incluye dentro de las clases entidades del sistema, las cuales representan la mayoría de las veces una tabla en la base de datos.

Un Docblock es un tipo especial de comentario que proporciona información detallada acerca de los elementos del código fuente. Dicha información puede ser utilizada por los desarrolladores para comprender la función de un elemento dado; también los IDEs se basan en ella para proporcionar, entre otras funciones, la conversión de las clases en entidades persistentes mediante un tipo de ORM, en este caso Doctrine v2.0 (Doctrine Team, 2006) como muestra la siguiente figura:

```
use Doctrine\ORM\Mapping as ORM;

/**
 *  Base\ComunBundle\Entity\Economico\Deuda;
 *
 * @ORM\Table(name="economico.ddeuda")
 * @ORM\Entity(repositoryClass="SIT\ECONOMICO\EjecutivoBundle\Repository\DeudaRepository")
 */
class Deuda {
```

Figura 5. Anotaciones Docblock utilizadas para el mapeo de la información con Doctrine v2.0.

2.2.2 Patrones de diseño

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes de diseño de software; soluciones que han sido probadas e implementadas en diferentes contextos (Pressman, 2005). A continuación se presentan los patrones de diseño que se utilizaron en la solución de la aplicación.

Patrones GoF

Dentro de los patrones GoF, se seleccionaron: el Decorador, como patrón estructural y el Método fábrica (Factory method) como patrón creacional.

Decorador: patrón que "*añade funcionalidades a objetos dinámicamente; proporcionando una alternativa flexible a la especialización mediante herencia*"

16 Una tupla representa un objeto único de datos implícitamente estructurados en una tabla, también es comúnmente denominada fila.

(Gamma, et al., 1995 p. 196). El patrón Decorador es utilizado en el trabajo con el motor de plantillas Twig, dicho componente define una plantilla base con contenido en lenguaje HTML, de la cual heredan las demás plantillas de la aplicación. La plantilla "base.html.twig" describe la estructura principal del diseño que tendrán las demás plantillas descendientes, es decir, posee un conjunto de bloques que decoran las demás plantillas del árbol de herencia.

En la aplicación la plantilla nombrada "plantilla_HOME.html.twig" hereda directamente de "base.html.twig" tomando todos los valores que por defecto tiene la misma y agregando valores propios. Esta plantilla define diferentes componentes que serán comunes para cada una de las vistas de los módulos como: menú superior, menú de procedimientos, datos de usuario y botón de salida, el bloque de ayuda unido a la opción de cambio de contraseña y el área de trabajo.

Además, se cuenta con la plantilla denominada "plantilla_SIT.html.twig" que se divide en dos secciones: el menú lateral y el bloque de contenido. Cada plantilla de la aplicación que herede de "plantilla_SIT.html.twig" puede redefinir el menú lateral izquierdo con los procesos correspondientes a cada procedimiento y el área de contenido con las plantillas pertenecientes a los casos de uso. Lo anteriormente descrito es posible observarlo en la figura 6:

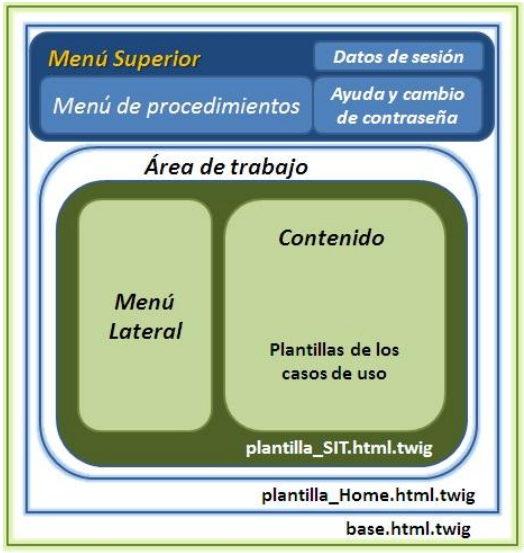


Figura 6. Evidencia del patrón Decorador en la creación de las plantillas de la aplicación.

Método fábrica (Factory method): patrón caracterizado por definir una interfaz para la creación de objetos, permitiéndole a las subclases decidir qué clase instanciar. (Gamma, et al., 1995). Una forma de evidenciar este patrón es mediante el uso que hace Doctrine v2.0 de su administrador de entidades (EntityManager[17]). Cada vez que una clase gestora necesite hacer una consulta sobre la base de datos, solo debe

17 Interfaz pública que presenta Doctrine v2.0.

llamar a los métodos de las clases repositorios (Repository), las cuales están asociadas a una entidad (Entity). El EntityManager solo debe hacer uso del patrón para proporcionar una instancia de la clase repositorio en cuestión, como se puede observar en la siguiente figura:

```
public function getTramite($idtramite) {
    return $this->getEm()->getRepository('ComunBundle:AG\Tramite')->find($idtramite);
}
```

Figura 7. Ejemplo de consultas realizadas en la clase gestora del CU Disponer sobre escrito de oposición.

Patrones GRASP

Como patrones de diseño GRASP se seleccionaron los siguientes:

Experto: patrón empleado para asignar una responsabilidad al experto en información, la clase que cuenta con la información necesaria para cumplir con una determinada responsabilidad (Larman, 1999). Con la aplicación del patrón experto se logra disminuir el acoplamiento y aumentar la cohesión del diseño. Un ejemplo en el sistema son las clases gestoras, debido a que en ellas se encuentra toda la información necesaria para la creación de un objeto o la implementación de un método, para responder a una operación determinada. En dicha clasificación también se pueden incluir las clases entidades, las cuales tienen la información necesaria referente a un objeto del negocio.

Creador: ayuda a identificar qué clase debe ser la responsable de la creación o instanciación de nuevos objetos o clases. La principal ventaja que ofrece el empleo de este patrón es el bajo acoplamiento, lo cual supone facilidad de mantenimiento y reutilización (Larman, 1999). El patrón creador es utilizado para la creación de las vistas en las clases controladoras, pues la responsabilidad recae solo sobre las mismas. Se emplea también en las clases gestoras, en la creación de objetos de las clases entidades para su posterior manipulación.

Controlador: patrón que funciona como intermediario en la comunicación entre la interfaz de usuario y el algoritmo que la implementa; establece que la responsabilidad del manejo de los eventos de un sistema pertenece a una clase (Larman, 1999). Además sugiere que la capa de presentación debe estar separada del código que la implementa, aumentando el nivel de reutilización de código y logrando un mayor control sobre el sistema.

Dentro de las cuatro clasificaciones que propone Larman, para el diseño de la aplicación se empleó el tipo de controlador “manejador artificial de casos de uso”. Esta clasificación garantiza que exista al menos una clase controladora por caso de uso,

dividiendo los eventos del sistema en un mayor número de controladores, favoreciendo la alta cohesión y el bajo acoplamiento (Larman, 1999). La clase controladora tiene la función de recibir los datos que llegan desde la vista y enviarlos a la clase que contiene la información necesaria para responder a dicha petición: la clase gestora.

Bajo acoplamiento: el objetivo de este patrón es lograr la menor dependencia entre clases, de tal forma que las modificaciones en alguna de ellas afecten en menor medida el funcionamiento del sistema. El grado de acoplamiento no puede considerarse de manera aislada de otros patrones, como Experto y Alta Cohesión (Larman, 1999). En el sistema un ejemplo de clases que presentan bajo acoplamiento son las entidades, pues estas no guardan relación alguna con las vistas y los controladores. Además, las clases gestoras también evidencian el uso de este patrón, puesto que se crea, al menos, un gestor por controlador; asegurando con ello que las modificaciones en la programación solo afecten un determinado CU y no el funcionamiento del sistema completo o gran parte de él. Un diseño de software eficiente intenta conseguir el acoplamiento más bajo posible entre los componentes, para obtener un diseño entendible y menos propenso a tener un "*efecto ola*"[18] (Stephen, 2002).

Alta cohesión: el patrón de alta cohesión sugiere que cada clase del sistema debe realizar solo las operaciones necesarias para cumplir una tarea dentro de un área funcional, manteniendo un número equilibrado de responsabilidades (Larman, 1999). Pressman afirma que *"un módulo cohesivo lleva a cabo una sola tarea dentro de un procedimiento de software, lo cual requiere poca interacción con los procedimientos que se llevan a cabo en otras partes de un programa"* (Pressman, 1997 p. 230). Por consiguiente a un mayor grado de cohesión, menores serán los esfuerzos de mantenimiento y el nivel de acoplamiento será mínimo.

Inyección de dependencias: patrón implícito en el marco de trabajo Symfony2. La inyección de dependencias consiste en suministrarle a las clases todos los objetos (dependencias) que necesitan, una vez que hayan sido creados y configurados. Permite extraer responsabilidades de un objeto para delegarlas en otro, a través de un servicio, sin necesidad de volver a crearlo (symfony.es, 2009). Por el tamaño que las aplicaciones pueden llegar a tomar, Symfony2 propone una herramienta de gran utilidad: el contenedor de dependencias[19]; el cual resuelve el problema del exceso de servicios. Esta herramienta es un objeto PHP que simplifica y facilita el uso de los

18 Denominación dada por Stevens, W., G. Myers y L. Constantine en el libro "Structured Design", este efecto ocurre cuando aparece algún error que se propaga por el sistema, según la dependencia que exista entre las clases.

19 También puede llamarse contenedor de servicios o solamente contenedor.

servicios, de manera que gestiona la creación de sus instancias. Un contenedor de servicios conoce todas las relaciones entre las clases y la configuración necesaria para instanciar correctamente a cada una de ellas (Rodríguez, Juan David, 2013).

En el sistema se encuentra el fichero "*Resources\config\services.yml*" dentro de la carpeta del módulo Ejecutivo. En el fichero *"services.yml"* se describen los servicios pertenecientes al módulo y sus dependencias. El mismo se divide en dos secciones; parameters[20] y services, siendo esta última donde se declaran los servicios (Figura 8). La clase Controller que implementa Symfony2 proporciona en un atributo público llamado container, una instancia del contenedor de dependencias; de esta manera, cualquier clase controladora que herede de dicha clase podrá obtener una instancia del atributo, y con ello cualquier servicio existente mediante el método get() del mismo (Figura 9).

```
parameters:
  #-----------------------variables Gestores --------------------#
     #economico.ejec.gtr.example.class: SIT\ECONOMICO\EjecutivoBundle\Example
     economico.ejec.gtr.DisponerSobreEscritoOposicion.class:
     SIT\ECONOMICO\EjecutivoBundle\Negocio\Gestor\DisponerSobreEscritoOposicionGtr

services:
#-----------------------GESTORES --------------------#
    economico.ejec.gtr.DisponerSobreEscritoOposicion:
          class: %economico.ejec.gtr.DisponerSobreEscritoOposicion.class%
          parent: arquitectura.basegtr

    economico.ejec.providenciaRespuestaBancoGtr:
       class: %economico.ejec.providenciaRespuestaBancoGtr.class%
       parent: arquitectura.basegtr
```

Figura 8. Ejemplo de servicios en el fichero "services.yml".

```
class DisponerSobreEscritoOposicionController extends BaseController {

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

Figura 9. Uso del atributo "container" en el CU Disponer sobre escrito de oposición.

2.3. Modelo de diseño

El modelo de diseño es la etapa de desarrollo del software que representa el puente entre los requerimientos y la implementación del sistema. Como elemento importante para lograr un modelo de diseño de calidad es necesario seguir una correcta notación, que cumpla con el objetivo de esta etapa: crear modelos o prototipos que describan cómo deben interactuar las clases, para el posterior desarrollo de la aplicación. A

20 En esta sección del fichero se declaran las configuraciones de los servicios.

continuación se describe la notación utilizada para desarrollar los diagramas de comunicación y de clases, así como el diseño de los mismos, mediante el uso del lenguaje UML v2.0.

2.3.1 Diagrama de comunicación

El diagrama de comunicación es un tipo de diagrama de interacción que busca graficar cómo se estructuran organizacionalmente el envío y recepción de mensajes entre los objetos. Dicho diagrama muestra un conjunto de objetos enlazados entre sí, además de los mensajes que deben intercambiar (Larman, 1999). Los diagramas de comunicación son representados mediante un grafo, donde los nodos representan objetos y las aristas los enlaces entre los mismos. La principal diferencia entre el diagrama de secuencia y el diagrama de comunicación es que el primero hace énfasis en el orden en que deben ocurrir los mensajes, siguiendo una línea temporal.

Notación para el diagrama de comunicación

Para la elaboración de los diagramas de comunicación se debe tener en cuenta una serie de principios que sean capaces de representar cada operación del sistema antes de ser implementado. El diseño de los diagramas de comunicación se realizó según los siguientes aspectos de notación básica para este tipo de diagrama:

- Las clases e instancias son representadas a través de rectángulos, con la particularidad de que las instancias tienen un nombre, el cual es separado por dos puntos del nombre de la instancia.
- Opcionalmente se puede indicar el número de secuencia con el cual un mensaje es enviado de una clase a otra.
- Los mensajes son representados por una flecha etiquetada con el nombre de la funcionalidad que indica la dirección del mensaje, entre objetos pueden pasar un número indefinido de mensajes.
- Los parámetros de un mensaje se anotan dentro de paréntesis después del nombre del mensaje, siendo opcional la especificación del tipo de dato del mismo.
- El valor de retorno (opcional) se pone seguido de la funcionalidad que se envía en el mensaje separado por dos puntos (Hernández Orallo, 2013).

De forma general, los mensajes que se envíen entre las clases que se diseñan en el diagrama de comunicación para UML v2.0, seguirán una sintaxis estándar identificada por la expresión: *numeroMensaje: mensaje (parámetro: tipoparametro): tipoRetorno*. La siguiente figura muestra la manera en que se realizaron todos los diagramas de comunicación para los CU correspondientes a los Flujos alternos:

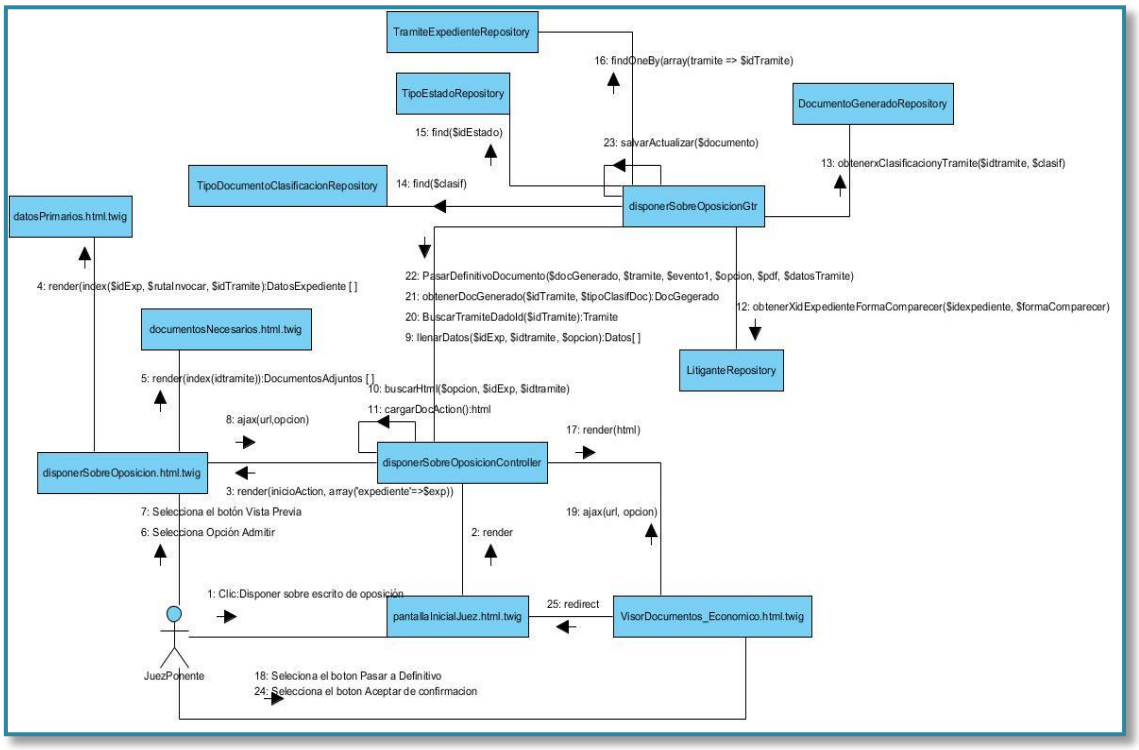


Figura 10. Diagrama de comunicación para el CU Disponer sobre escrito de oposición.

2.3.2 Diagrama de clases

Luego de finalizar el diseño de los diagramas de comunicación, se realizó como paso siguiente la modelación de los diagramas de clases. Este tipo de diagramas describen gráficamente las entidades que intervienen en el desarrollo de la aplicación, así como la relación que existe entre ellas. Cada entidad o clase refleja las características de un objeto, así como su comportamiento a través de operaciones o métodos.

Notación para el diagrama de clases

Para la elaboración de los diagramas de clases se aplicaron las siguientes reglas:

- Identificar las clases que intervienen en la solución del software y representarlas a través de rectángulos divididos en dos secciones: atributos y métodos.
- Las clases (Controller, Gestor, Repository) y ficheros (Twig) terminan con la palabra que especifica su tipo.
- Para representar las vistas y las clases repositorios se utilizarán los prototipos definidos por los analistas del SITPC.
- Cada método contendrá los parámetros necesarios para operar y el tipo de dato que devuelve.
- Los métodos de las clases controladoras que significan eventos que se realizan en la interfaz se identifican con la palabra “Action”.

- Las relaciones entre clases se realizan a través de conexiones, especificando el tipo de relación que existe entre las mismas (link, use, redirect, include, dependencia o generalizaciones) (Hernández Orallo, 2013).

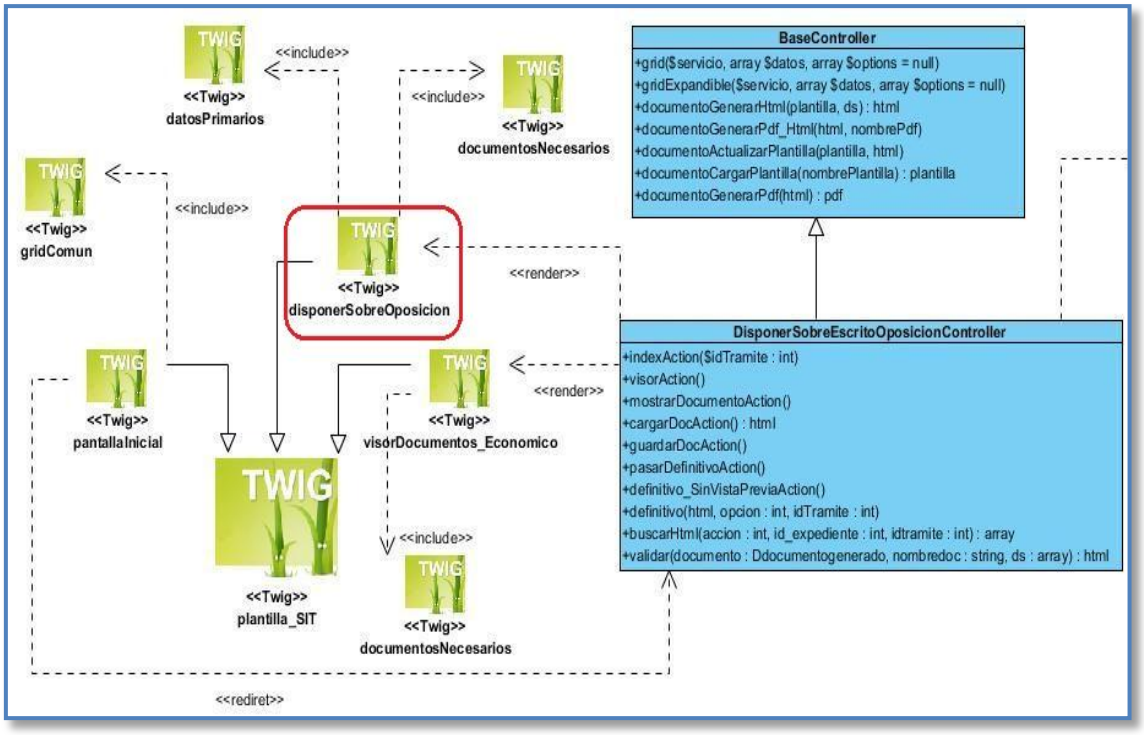


Figura 11. Fragmento del diagrama de clases del CU Disponer sobre escrito de oposición.

En el diagrama que se presenta en la figura 11 (los anexos 2 y 3 muestran el diagrama completo), se puede observar cómo interactúan las interfaces del sistema, la interfaz principal del caso de uso (enmarcada en rojo) y la clase controladora, así como la relación que existe entre esta y la clase de la que hereda.

2.3.3 Diagrama de despliegue

El modelo de despliegue es *"un modelo de objetos que describe la distribución física del sistema en términos de cómo se distribuye la funcionalidad entre los nodos de cómputo. (…) se utiliza como entrada fundamental en las actividades de diseño e implementación debido a que la distribución del sistema tiene una influencia principal en su diseño"* (Booch, et al., 2000 p. 217). Para el SITPC se definió, específicamente orientados al modelo de despliegue, los requisitos no funcionales que se citan a continuación:

*"**RnF.44.** Para el despliegue del sistema se utilizará un servidor web por cada Tribunal municipal y provincial y un servidor de base de datos separado en dos servidores físicos, además de un servidor web y un servidor de base de datos en la sede nacional.*

***RnF.45.** Las computadoras de las estaciones de trabajo serán clientes ligeros con una distribución de Linux compatibles con el navegador Firefox 10.x o superior. Los*

servidores usarán como distribución Ubuntu 10.x o superior en su versión estable según los recursos disponibles" (Ing. Roque Hernández, 2013 pág. 12).

Además de los requisitos no funcionales definidos se agrega al modelo de despliegue el uso de impresoras conectadas a las estaciones de trabajo, con el propósito de brindar servicios básicos internos y externos a la entidad. El servidor de base de datos del Tribunal Supremo Popular almacenará la información de todas las instancias, preferiblemente en horario no laboral; también existirá otro servidor que mantiene una réplica de la misma. Los servidores web de las instancias municipales y provinciales contendrán todo lo relacionado con la aplicación, lo descrito anteriormente se muestra en la figura 12:

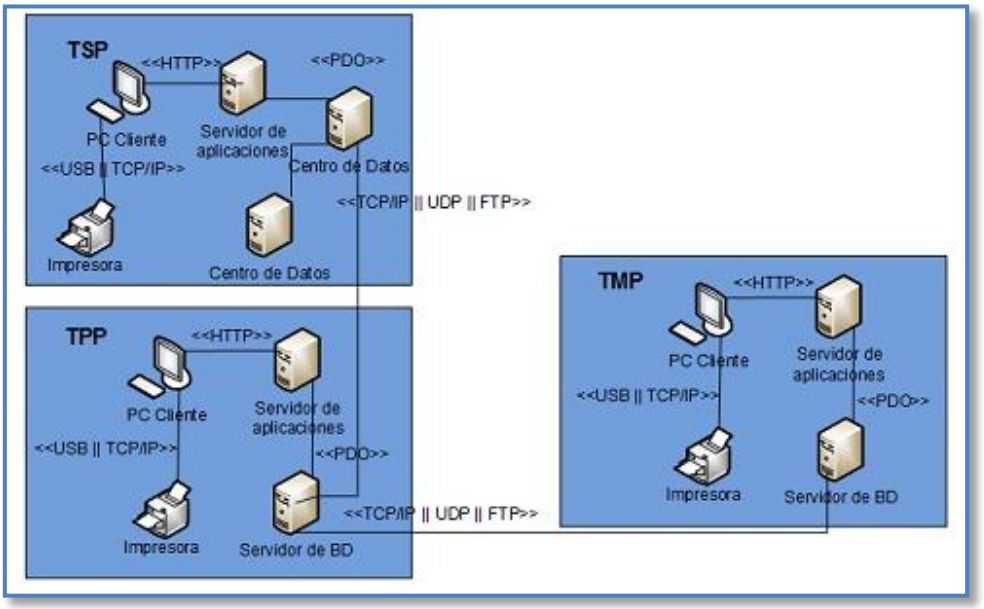


Figura 12. Diagrama de despliegue propuesto para SITPC.

Algunos de los protocolos más importantes a utilizar para las conexiones que se realizarán entre las instancias de los tribunales y dentro de las mismas son:

TCP/IP: su función principal es el uso bidireccional, garantizando la entrega de los datos al destino sin errores y en el mismo orden en que se transmitieron. También proporciona un mecanismo para distinguir distintas aplicaciones dentro de una misma máquina (Stallings, 2000).

UDP: protocolo orientado al control de flujo, control de errores y entrega en secuencia (Stallings, 2000).

HTTP**:** es un protocolo cliente/servidor orientado a transacciones y utilizado para transmitir información con la eficiencia necesaria. Los datos transferidos por el protocolo pueden ser texto, hipertexto, audio, imágenes o cualquier información accesible a través de Internet (Stallings, 2000).

2.4. Estándares de codificación

Los estándares de codificación son normas que deben seguirse durante todo el ciclo de implementación del sistema. Los mismos fueron seleccionados por el equipo de arquitectura y diseño del proyecto (Ing. Almora Galvez, et al., 2013). A partir de la siguiente figura se enumeran dichos estándares:

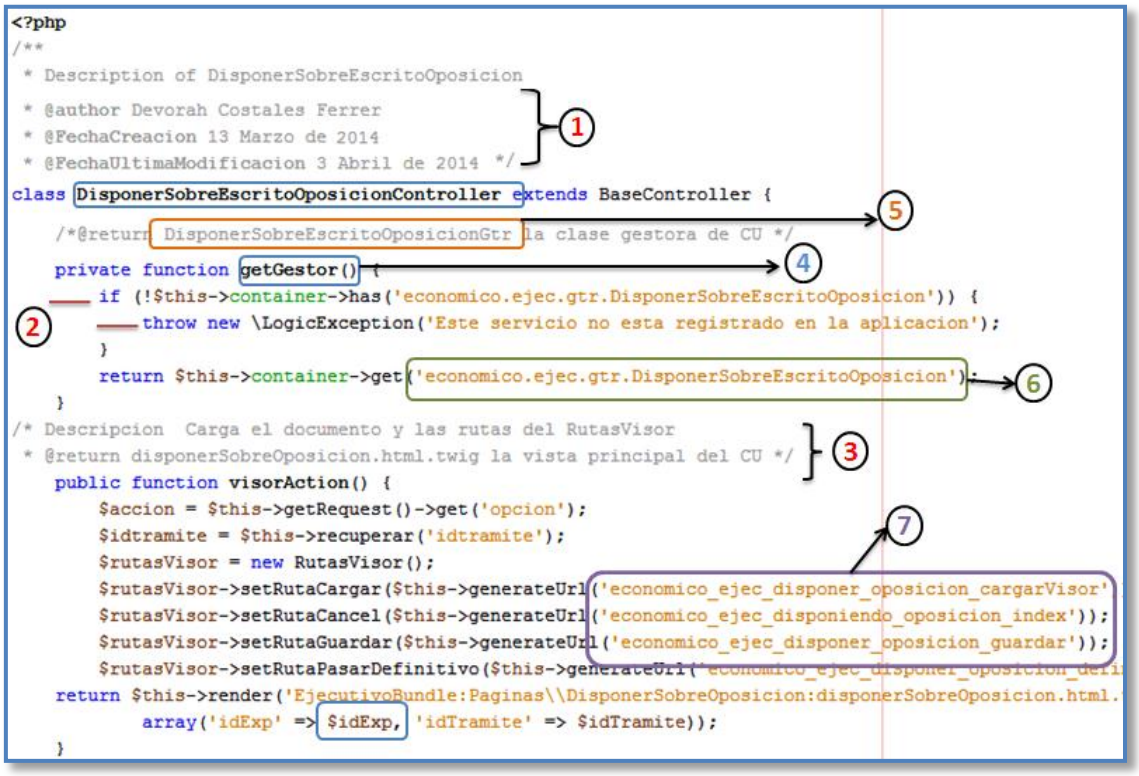


Figura 13. Estándares de codificación a seguir en la implementación de la aplicación.

(1) Cabecera del archivo

Es importante que todos los archivos *.php* inicien con una cabecera específica que indique información del autor y fecha de los últimos cambios realizados, entre otros aspectos. Depende de cada módulo decidir si agregar o no más datos.

(2) Indentación[21]

El contenido siempre se indentará con tabulaciones, nunca utilizando espacios en blanco.

(3) Comentarios

Los métodos deben ser comentados con el objetivo de describir la función que realizan, sin necesidad de que el programador ajeno al código tenga que estudiarlo a profundidad para su comprensión. Además, debe ofrecer información referente a los parámetros y valores que debe devolver cada método comentado.

(4) Denominación de archivos.

21 Espacio en blanco al inicio de línea o sangría antes del código.

Las funciones y variables comienzan con letra minúscula, en el caso de las clases y archivos con letra mayúscula. Cuando son palabras compuestas, estas no deben exceder de cuatro cadenas de letras, siguiendo además la notación Camello[22].

(5) Denominación de las clases

Las clases gestoras deben llamarse igual que las clases controladoras, solo se diferenciarán por la terminación de cada nombre que identifica el tipo de clase al que se hace referencia. Ejemplo: *DisponerSobreEscritoOposicion**Gtr***, clase gestora.

(6) Denominación de los servicios

Los parámetros deben estar en minúscula utilizando como separador el carácter punto (.); igual para los servicios, pero sin el sufijo ".class". La estructura que se propone para los parámetros es: *materia.procedimiento.gestor|tableModel.categoria.class.*

(7) Denominación de las rutas.

Las rutas deben escribirse en minúscula utilizando como separador el carácter guión bajo (_) de la siguiente forma: *materia_procedimiento_controlador_accion.* El patrón de la ruta debe tener relación con la acción que se llama y ser lo más entendible posible, en caso de que el nombre esté compuesto por más de una palabra serán separadas por el signo de menos (-) y no debe ponerse la materia ni el procedimiento en el mismo.

Otras Consideraciones:

- Los nombres deben ser descriptivos y concisos.
- No usar ni grandes frases ni pequeñas abreviaciones para las variables, de manera que con solo conocer su nombre se sepa la función que realizan. Esto aplica para los nombres de variables, funciones, argumentos de funciones y clases.
- Las constantes deben de escribirse siempre en mayúsculas y tanto estas como las variables globales, deben de tener como prefijo el nombre de la clase a la que pertenecen.
- Las llaves de apertura irán al final de la sentencia que delimitan, y las llaves de cierre alineadas con el inicio de la sentencia en una nueva línea.

2.5. Interfaces de la aplicación

"El diseño de la interfaz de usuario es la categoría de diseño que crea un medio de comunicación entre el hombre y la máquina" (Pressman, 1997 p. 259). Las interfaces de usuario deben ser diseñadas según las habilidades y experiencias, además de las capacidades físicas y mentales, del usuario al que está orientada la aplicación. Por

22 La notación Camel o Camello se representa con la primera letra en minúscula y la primera letra de las palabras siguientes con mayúscula (***n**otacion**C**amello*).

tanto, para el diseño de las interfaces se tuvieron en cuenta algunos principios de interacción con el usuario y de presentación de la información, a pesar de que es difícil encontrar un equilibrio entre ambos estilos.

Para lograr una fluida interacción con el usuario se aplicaron principios como: (1) la selección de menús, mediante el cual se puede asegurar que el usuario introduzca la menor cantidad de datos erróneos en la aplicación y que el tecleo de datos sea mínimo. (2) El relleno de formularios, lo que garantiza la introducción de datos de manera sencilla para el usuario.

En cuanto al estilo de presentación, se analizaron detenidamente los colores y cuadros de diálogos a utilizar para el diseño de la interfaz. Se emplean colores claros en el diseño del fondo contrastando con el color del texto; dichos colores varían entre azul, gris, negro y blanco; de manera que se le pueda ofrecer al usuario una vista agradable, como se muestra en la siguiente figura:

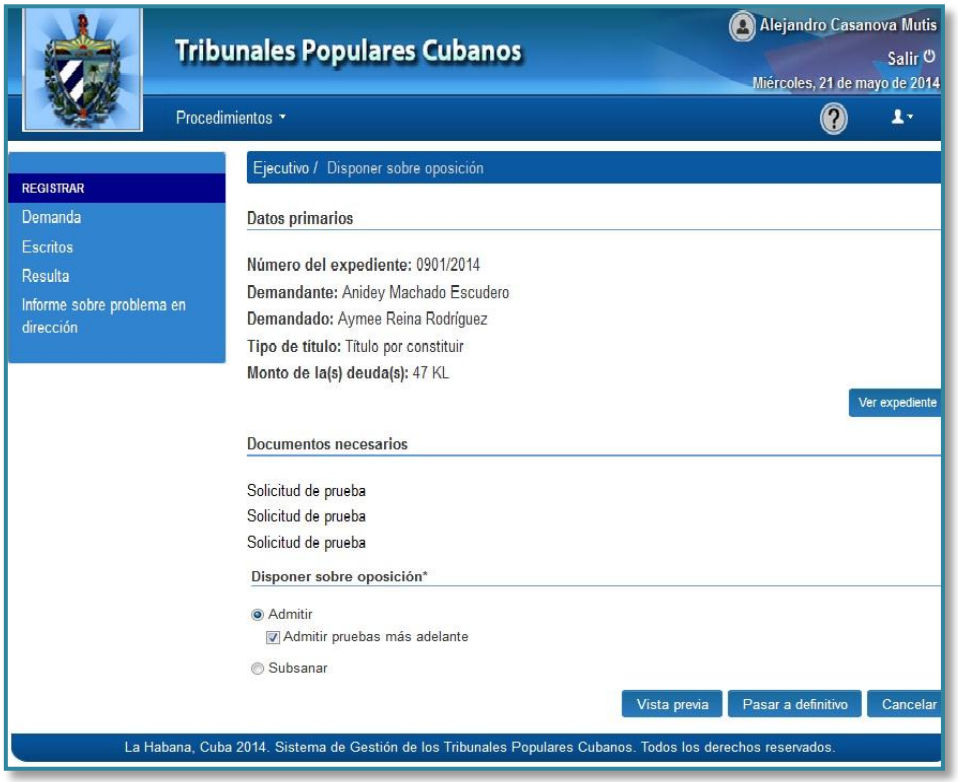


Figura 14. Vista principal del CU Disponer sobre escrito de oposición.

La imagen antes presentada muestra la vista principal del CU Disponer sobre escrito de oposición, donde se empleó el uso de formularios y acciones de botones. Cuando el usuario selecciona el botón “Vista previa” se genera una nueva página, que presenta el documento correspondiente a la opción seleccionada por el usuario. En esta nueva sección el usuario puede seleccionar los botones “Pasar a definitivo” o “Guardar”, acciones que se realizan sobre el documento; en ambos casos luego de realizada la acción se pasa a la pantalla inicial del módulo. Además puede seleccionar

también el botón “Cancelar”, mediante el cual se suspende o anula la acción, mostrándose nuevamente la vista inicial del CU. Es importante conocer que este CU puede generar tres documentos distintos según la opción que se seleccione.

Otro de los estilos de presentación del diseño de interfaces es el uso de cuadros de diálogo (Figura 14). Estos se utilizan en la aplicación para ofrecer mensajes de error, advertencias, preguntas o confirmaciones. El sistema es capaz de responder a través de mensajes relacionados con la actividad actual que está realizando el usuario, además el texto que se muestra en cada cuadro de diálogo está escrito en un lenguaje natural, entendible para cualquier tipo de usuario.

Figura 15. Cuadro de diálogo que verifica la opción de guardar un documento, generado en el CU Disponer sobre escrito de oposición.

Conclusiones parciales del capítulo

En el capítulo abordado se presentaron elementos necesarios para lograr eficazmente la implementación del sistema, para su posterior validación durante la etapa de prueba.

- Se describieron los requisitos funcionales y casos de uso asociados a estos, de manera que se cumplan apropiadamente las necesidades del cliente.
- Mediante la arquitectura definida para el sistema y los patrones de diseño utilizados se han podido definir los diagramas de comunicación y de clases, necesarios para un correcto desarrollo del producto.
- Se definieron los estándares de codificación a utilizar, lográndose así que el código quedase estructurado y legible, asegurándose de esta manera una mayor reutilización del mismo.

Capítulo 3: Análisis de los resultados

Introducción

El presente capítulo aborda el tema referente al análisis de los resultados obtenidos después de haber implementado el sistema, mediante la aplicación de métricas de diseño y técnicas de prueba. Las métricas de diseño serán las encargadas de evaluar la calidad de los atributos del software de forma cuantitativa. Las técnicas de prueba, en especial las de caja blanca y caja negra, serán de utilidad para detectar errores tanto en la aplicación como en su código fuente.

Antes de comenzar el desarrollo de este capítulo es necesario conocer cuándo se está en presencia de un software o producto informático de calidad, para ello es fundamental tener conocimiento sobre qué es la calidad del software, interrogante que responde Pressman como: *"el cumplimiento de los requisitos de funcionalidad y desempeño explícitamente establecidos, (…) que se esperan de todo software desarrollado profesionalmente"* (Pressman, 2005 p. 463). Este concepto puede tratarse además como la relación de factores que se desean controlar y asegurar para obtener un software de resultados exitosos.

3.1. Factores de calidad del software

Para obtener un software de calidad es necesario tener en cuenta un conjunto de factores[23] o atributos, que en un momento determinado pueden afectar la excelencia de la aplicación en cuanto a su funcionamiento. Con el proceso de validación de los resultados se obtiene una medida que debe ser comparada con algún indicador, para poder llegar a conclusiones sobre la realidad del sistema. Por tal motivo McCall[24] y otros autores (McCall, et al., 1977) propusieron un conjunto de atributos del software que pueden afectar la calidad del mismo.

Algunos de los atributos propuestos por los autores (Anexo 4) son difíciles de medir debido a que exigen, en ocasiones, mediciones complejas y poco transparentes. Para seleccionar los atributos a analizar en cada una de las métricas de calidad, que en el siguiente epígrafe se desarrollarán, se tuvo en cuenta que estos fueran: (1) simples y calculables, en otras palabras, deben ser relativamente fáciles de calcular sin exigir demasiado tiempo ni esfuerzo. (2) Consistentes y objetivos, donde los resultados obtenidos al aplicar la métrica no sean ambiguos. (3) Independientes del lenguaje de

23 Condición o característica que activamente contribuye a la calidad del software.
24 Jim A. McCall, ingeniero de software que propuso en 1977, junto a otros autores, una categorización útil de los factores de calidad de software en el libro "Factors in Software Quality".

programación, dicho en otro modo, deben centrarse en el modelo de diseño o en la estructura del propio programa y por último, (4) el cálculo matemático de la métrica debe emplear medidas que no lleven a combinaciones extrañas o difíciles de manejar (Pressman, 2005).

Cada atributo responde a una pregunta específica que puede relacionar varios atributos, por ejemplo: el nivel de mantenimiento está relacionado con el nivel de acoplamiento y cohesión de las clases del sistema. En la siguiente tabla se describen los atributos a medir de manera general en las métricas a aplicar para validar la calidad de la aplicación:

Atributo	Descripción	Interrogante
Mantenimiento	Esfuerzo necesario para localizar y corregir un determinado error en el programa.	¿Cuán difícil es el arreglo del sistema?
Reutilización	Grado en que un programa (o partes de él) pueden reutilizarse en otras aplicaciones (la cantidad de clases del sistema y sus responsabilidades).	¿Se podrá reutilizar el software o al menos una parte de él?
Acoplamiento	Están relacionados con el nivel de mantenimiento y reutilización del sistema y pueden variar el comportamiento de dichos atributos.	¿Cuán dependientes son unas clases de otras?
Cohesión		¿Cuántas responsabilidades tiene una clase?
Responsabilidad	Grado o nivel con que una clase realiza un conjunto de operaciones.	¿Cuán compleja puede ser una clase?
Complejidad de implementación	Cantidad de código necesario para que un programa realice su función, está en relación con la cantidad de responsabilidad que se le asigne a una clase.	¿Cuál es el tamaño en líneas de código que puede tener una clase?

Tabla 6. Atributos de calidad a analizar en la aplicación de las métricas.

3.2. Métricas para validar el diseño

Existen diferentes tipos de métricas, las que pueden clasificarse en: métricas para el modelo de análisis, de diseño, de código fuente y de pruebas; esta sección del capítulo hará referencia a las métricas de diseño, las cuales se especializan en analizar el diseño orientado a objetos y a clases. El glosario de estándares de la IEEE[25] define métrica como "*una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo determinado*" (IEEE Computer Society, 1993). Las métricas que se aplican para validar el diseño del sistema tienen la tarea de medir los atributos de esta fase, ofreciéndole al ingeniero de software la posibilidad de validar y evaluar la calidad del sistema antes, durante y después de su desarrollo; además de demostrar cualitativa y cuantitativamente cuán eficiente es la aplicación.

La implementación del módulo Ejecutivo está sustentada en 16 casos de uso, de los cuales se tomará una muestra de 10 CU representando el 62,5% del total. Teniendo en consideración dicha muestra, se puede garantizar la veracidad de los resultados obtenidos de la aplicación de las métricas que serán analizadas en lo adelante.

3.2.1 Métricas de diseño orientado a objetos: colección de métricas LK[26]

Dentro de las métricas especializadas en el diseño orientado a objetos se encuentra la colección de Lorenz y Kidd (LK). A continuación se analiza la validez del diseño del sistema mediante la aplicación de las métricas LK referentes al tamaño de clases y las relaciones entre estas.

Tamaño de clase (TC)

La métrica tamaño de clase consiste en medir el tamaño general de una clase concentrándose en la cuantificación de atributos y métodos de una clase individual. Analiza también el promedio de la cantidad de operaciones y el número de atributos que se encuentran dentro del sistema de manera general. Una vez analizado el indicador tamaño de clase, si el valor resultante tiende al crecimiento, es probable que la clase esté saturada de responsabilidades; en consecuencia, el nivel de reutilización sería mínimo y la implementación altamente compleja (Pressman, 2005).

Para realizar el análisis de la métrica TC es necesario conocer los atributos a medir y la categoría (baja, media, alta) en la que se debe clasificar cada clase, según los resultados obtenidos de aplicar el criterio de clasificación:

25 Instituto de Ingenieros Eléctricos y Electrónicos, asociación técnico-profesional dedicada a promover, estandarizar y aplicar los avances en las tecnologías de la información, electrónica y ciencias en general.
26 Mark Lorenz posee su propia compañía de consultoría en desarrollo orientado a objetos (OO), trabajó junto a Jeff Kidd en el Centro de Tecnología de IBM donde posteriormente desarrollaron una herramienta para recolectar métricas OO.

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

Tabla 7. Criterios para categorizar las clases a analizar en la métrica TC.

La variable *Prom (promedio)* representa el promedio de operaciones que tienen de manera general las clases analizadas en la métrica en cuestión. Dicho promedio es empleado para comparar el valor correspondiente a la cantidad de operaciones para cada clase, con cada uno de los atributos de calidad a evaluar en la misma. Para un total de 53 clases examinadas (Anexo 5) entorno a la cantidad de procedimientos que realiza y el nivel de cada atributo en la clase, se pudo comprobar que las clases del sistema se diseñaron manteniendo la menor responsabilidad y complejidad posible, así como un elevado nivel de reutilización. Los gráficos siguientes muestran de manera general el comportamiento de cada uno de los factores de calidad analizados en la métrica TC:

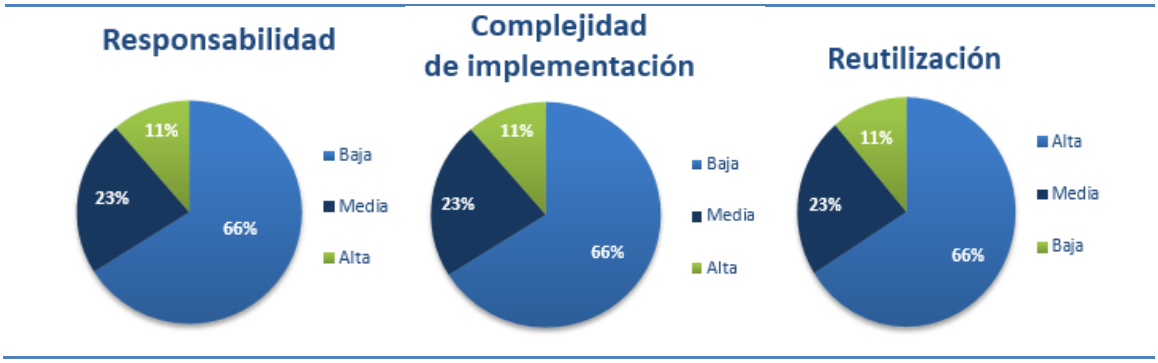


Figura 16. Resultado obtenido de aplicar la métrica TC.

Para lograr un tamaño de clase óptimo, el valor que se obtiene de aplicar la métrica debe ser directamente proporcional al valor resultante de la complejidad de implementación y el nivel de responsabilidad de las clases; de igual forma debe ser inversamente proporcional al nivel de reutilización de código.

En este caso se obtuvo que el 66% de las clases analizadas presentan un bajo nivel de responsabilidad y de complejidad de implementación, lo que significa que: (1) existe alta cohesión entre clases, (2) que el valor obtenido de aplicar la métrica de manera general tiende al mínimo, (3) el nivel de reutilización representado por el 66% sigue una trayectoria de crecimiento y por último, (4) la implementación tiene un bajo nivel

de complejidad, debido a que la clase posee la menor cantidad de métodos y atributos posibles, necesarios para su implementación.

Los niveles medio y alto en los atributos de responsabilidad y complejidad no son relevantes para el análisis de la métrica, puesto que los valores obtenidos para estas clasificaciones no influirán en gran medida en el resultado. Sin embargo, en el factor reutilización el valor alcanzado para el nivel alto es significativo y representa que cada clase tendrá un mayor grado de reutilización. Como resultado complementario, se muestra en el anexo 6 la cantidad de procedimientos por clases; donde el mayor porcentaje está representado por las clases que contienen entre 6 y 10 métodos.

Relaciones entre clases (RC)

La métrica RC se utiliza para validar el nivel de relación que debe existir entre cada una de las clases del sistema, asegurando la alta cohesión y el bajo acoplamiento en las mismas (Stephen, 2002). Los factores de calidad empleados para validar los resultados obtenidos fueron: nivel de acoplamiento, complejidad del mantenimiento y reutilización del código.

Para obtener un nivel óptimo de relación entre clases, el valor obtenido al aplicar dicha métrica debe ser directamente proporcional al acoplamiento y a la complejidad de mantenimiento; además debe ser inversamente proporcional al nivel de reutilización del código. Para aplicar la métrica RC es necesario categorizar cada una de las clases según la cantidad de relaciones que esta contenga, para ello se debe visualizar la tabla del anexo 7.

Con el propósito de lograr un resultado altamente veraz en la aplicación de la métrica RC, esta fue aplicada de manera independiente a cada uno de los 10 casos de uso de la muestra. Después de terminada dicha actividad, se tomaron todos los resultados obtenidos individualmente y se agruparon para ser analizados de manera general, promediando los valores obtenidos por categoría en cada atributo.

Figura 17. Resultado obtenido de aplicar la métrica RC de manera general.

Luego de concluida la aplicación de la métrica y analizados los resultados se pudo concluir que: (1) más del 75% de las clases del sistema poseen un nivel bajo de

acoplamiento, lo cual (2) favorece el proceso de mantenimiento; debido a que (3) no existe un alto nivel de dependencia entre las clases. Como otro factor favorable se encuentra que (4) las modificaciones a realizar son sencillas y no ocasionarán un efecto de encadenamiento de errores hacia otras clases. Se comprobó que el 76,5% de las clases presentan un alto nivel de reutilización, lo que propicia (5) realizar un menor esfuerzo en la implementación de casos de uso similares.

3.2.2 Métricas de diseño orientado a clases: colección de métricas CK[27]

Las métricas de diseño orientado a clases se encargan de medir el grado de especialización de una clase, planteando una relación entre la cantidad de métodos y la complejidad de las mismas (Pressman, 1997). A continuación se enuncian las métricas de este tipo que fueron aplicadas para validar el sistema.

Falta de cohesión en métodos (FCM)

Esta métrica indica "*el número de métodos que pueden acceder a uno o a más de los mismos atributos*" (Pressman, 2005 p. 485), o sea, la cohesión de una clase indica cuán relacionados se encuentran los métodos locales con las instancias de las variables locales en dicha clase. Cuando ningún método accede a los mismos atributos se puede afirmar que el valor de FCM es el mínimo (FCM=0), en caso contrario tiende a ascender. Si el valor de la métrica FCM es relativamente alto, los métodos son acoplados mediante atributos, produciendo un diseño de clases complejo y difícil de mantener. Lo ideal para lograr un diseño de clases eficiente es mantener baja la FCM, garantizando así una alta cohesión entre los métodos y entre las clases del sistema (Stephen, 2002). Las siguientes tablas muestran la aplicación de la métrica FCM para la entidad "Deuda":

Atributos	Identificadores
id	A1
monto	A2
sumaReconocida	A3
tipoMoneda	A4
deudaReconocida	A5
tipoEstado	A6
tipoClasificacionDeuda	A7
documento	A8

Tabla 8. Atributos locales de la entidad "Deuda".

27 Dr. Shyam R. Chidamber y Dr. Chris F. Kemerer son profesores adjuntos al Dpto. de Negocio Internacional de la Universidad de Pittsburgh.

Métodos	Atributo							
	A1	A2	A3	A4	A5	A6	A7	A8
getId	x							
setMonto		x						
getMonto		x						
setSumaReconocida			x					
getSumaReconocida			x					
setTipoMoneda				x				
getTipoMoneda				x				
setDeudaReconocida					x			
getDeudaReconocida					x			
setTipoEstado						x		
getTipoEstado						x		
setTipoClasificacionDeuda							x	
getTipoClasificacionDeuda							x	
setDocumento								x
getDocumento								x
Cantidad de métodos que acceden a un atributo	1	2	2	2	2	2	2	2

Tabla 9. Métodos locales de la entidad "Deuda" relacionados con los atributos de la tabla 8.

Los atributos se agruparon siguiendo una codificación combinando letras en orden alfabético con números del 1 al 9; dicho proceso se realizó para todas las clases que son utilizadas en la implementación de los 10 CU tomados como muestra. Una vez concluido el análisis se pudo obtener como resultado que el valor promedio de la métrica FCM es 2, sin embargo puede alcanzar un valor máximo de 4. El valor resultante de FCM indica que: (1) existe una buena subdivisión de clases, manteniéndose lo menos acopladas posibles por atributos; por consiguiente (2) facilita el mantenimiento de las mismas, en tanto que (3) disminuye su complejidad y las oportunidades de que ocurran errores durante el proceso de desarrollo.

Árbol de profundidad de herencia (APH)

La métrica APH se basa en analizar la longitud del camino de la herencia de clases, desde una clase nodo terminal (hoja) hasta la raíz del árbol de herencia. Una jerarquía de clases profunda puede llegar a tener demasiados métodos o variables de instancia en una clase, de manera que es posible que existan dificultades a la hora de predecir el comportamiento de la misma (Pressman, 2005).

Los ingenieros Chidamber y Kemerer recomiendan que un buen indicador del valor de APH sea una longitud menor de 6, comenzando por las clases del marco de trabajo o la clase raíz. En este caso se tomaron como clases raíces aquellas que fueron creadas por el equipo de arquitectura del proyecto y que son la base de la cual

heredarán las clases descendientes. Es válido aclarar que puede existir el caso de que altos valores de APH puedan favorecer la reutilización de métodos, a pesar de que también puede complejizar el diseño de las clases (Stephen, 2002).

Para poner en práctica dicha métrica, fue necesario analizar por cada clase utilizada en la implementación de los CU tomados como muestra, la longitud existente entre la raíz y el nodo terminal. Luego de concluir dicho procedimiento se obtuvieron los valores que se muestran en la siguiente figura:

Figura 18. Resultado obtenido de aplicar la métrica APH.

Concluido el análisis de la métrica APH los resultados demuestran que (1) el valor de la métrica APH es aceptable, pues la longitud de las clases está muy por debajo del valor óptimo que proponen Chidamber y Kemerer para esta métrica. Por otra parte, se puede afirmar que (2) la probabilidad de que las clases descendientes hereden la mayor cantidad de métodos posibles es baja; (3) provocando que el diseño tenga un alto nivel de facilidad de mantenimiento. De manera general, en el anexo 8 se muestra la distribución de los diferentes tipos de clases por nivel de profundidad de herencia en el sistema.

3.3. Proceso de verificación de la aplicación

A menudo se tiende a confundir los procesos de validación y verificación, sin embargo cada uno tiene objetivos distintos. La validación tiene como finalidad asegurar que el sistema cumple con las necesidades del cliente. Por otra parte, el proceso de verificación comprueba que el sistema funciona correctamente, además de monitorear el cumplimiento de los requisitos funcionales y no funcionales (Sommerville, 2005). A continuación se describe la estrategia de prueba definida para comprobar que el software cumple con las funcionalidades especificadas y su correcta implementación.

Estrategia de prueba

La estrategia de prueba es un conjunto de pasos que guían los procesos de verificación y validación del software; mediante esta técnica se seleccionan los tipos

de pruebas idóneos para comprobar la calidad del sistema, así como los métodos y técnicas necesarias. Para desarrollar el proceso de verificación de los Flujos alternos se propone la siguiente estrategia de prueba:

Proceso de verificación		
Nivel de la prueba	Componente	Sistema
Tipo de prueba	Pruebas de unidad o componente	Pruebas funcionales
Método de prueba	Caja blanca	Caja negra
Técnica de prueba	Camino básico	Partición equivalente

Tabla 10. Estrategia de prueba a seguir para verificar la aplicación.

Es importante mencionar que las pruebas de software no garantizan con plena seguridad que el sistema estará limpio de errores y que siempre funcionará según las especificaciones; sin embargo, no se debe caer en un ciclo interminable de pruebas, puesto que durante el proceso de depuración de errores se pueden introducir otros nuevos. Es por ello que una de las interrogantes más difíciles de responder dentro de la Ingeniería de Software es cuándo se deben detener las pruebas, ya que no hay forma de saber si el error detectado en cada iteración es el último.

Debido a esta cuestión, no existe un criterio que defina una condición de parada para la fase de prueba; aunque existen criterios basados en estimaciones sobre la cantidad de errores que se pueden encontrar en un sistema. Uno de los criterios más importantes es declarar los requerimientos de completitud en cuanto a *"un estimado de que porcentaje de errores pueden factiblemente ser encontrados a través de las pruebas"* (Myers, et al., 2012 p. 136), es decir, la prueba debe detenerse cuando se encuentre un porciento predefinido de errores en el sistema; a pesar de que este criterio *"implica una adivinación un tanto arbitraria, teniendo en cuenta la naturaleza del programa y de las consecuencias de los errores detectados"* (Myers, et al., 2012 p. 137). Haciendo uso de estas estimaciones se propone, para detener las pruebas, realizar tantas iteraciones sean necesarias hasta que se encuentren las no conformidades (NC) que satisfagan la siguiente fórmula:

$$\left(\sum_{i=1} X_{i-1}\right) * 0{,}3 \geq X_i$$

El valor de la cantidad de no conformidades encontradas en la última iteración debe ser menor o igual que el 30% de la sumatoria de la cantidad de NC detectadas hasta la penúltima iteración, siendo ***"i"*** (i = 1, 2, 3) la cantidad de iteraciones y ***"X"*** la

cantidad de NC encontradas en cada iteración. Lo anteriormente explicado será la base para determinar la completitud de los métodos de caja blanca y caja negra respectivamente.

3.3.1 Pruebas de unidad

Pruebas de unidad o componente: las pruebas de componente se encargan de comprobar el correcto funcionamiento de la unidad más pequeña del software de manera individual. Este tipo de prueba realiza un análisis de los diferentes caminos que existen en una estructura de control, de manera que se puedan detectar errores en el código fuente de la aplicación (Pressman, 1997).

Para realizar las pruebas de unidad se emplea el **método** de caja blanca o estructural, dicho método realiza una evaluación del código fuente, de manera que pueda efectuar un minucioso examen de los detalles procedimentales del código a evaluar; para ello es necesario conocer la lógica del programa. Las pruebas de caja blanca se realizan sobre las funcionalidades internas de un módulo y se encargan de comprobar los caminos lógicos, ciclos (bucles) y condiciones que debe cumplir el programa. Mediante la utilización de este método es posible desarrollar casos de prueba que garanticen la ejecución, al menos una vez, de los caminos independientes (camino que recorre por lo menos una nueva arista en el grafo de flujo).

Técnica de prueba: camino básico

La técnica del camino básico es empleada en las pruebas de caja blanca, la misma tiene como objetivo comprobar que cada camino se ejecute independiente de un componente o programa, obteniéndose una medida de la complejidad lógica del diseño. Esta técnica de prueba debe ser utilizada para evaluar la efectividad de los métodos asociados a una clase, con el objetivo de asegurar que cada camino independiente sea ejecutado por lo menos una vez en el sistema (Sommerville, 2005).

Para una correcta aplicación de la prueba de caminos se debe encontrar el número de rutas independientes, calculando la complejidad ciclomática del grafo de flujo, desarrollada por McCabe[28]. "*El término ciclomático (…) proporciona el número de caminos independientes mediante grafos dirigidos fuertemente conectados*" (McCabe, et al., 1996 p. 10).

Pressman propone como estrategia para aplicar la prueba de camino básico, realizar un análisis de la complejidad ciclomática de cada procedimiento que componen las clases del sistema; una vez concluido este paso se selecciona el método con valor de complejidad más elevado. El procedimiento de mayor valor tiene una alta probabilidad

28 Thomas J. McCabe, fundador y visionario de la calidad de software de la corporación McCabe Software, Inc. Ganó el Premio al Impacto por la métrica "Una medida de complejidad".

de contener errores, además de que ofrece una medida del número de pruebas que deben diseñarse para validar la correcta implementación de una determinada función (Pressman, 2005).

Un ejemplo de cómo se realizó el proceso de selección de las funciones más complejas en una clase se muestra en el anexo 9. Dos de los procedimientos implementados en la clase controladora "DisponerSobreEscritoOposicionController" cuentan con una complejidad ciclomática de valor 9, siendo este el mayor resultado que aparece. Por tanto, se deben diseñar nueve casos de pruebas según la función que se seleccione, ya sea, "guardarDocAction" o "definitivo".

Esta métrica se calcula sobre un grafo y se puede realizar mediante tres formas distintas:

1. El número de regiones contribuye a estimar el valor de la complejidad ciclomática.
2. v(G)= E – N + 2.
3. v (G)= P + 1.

Conociendo que:

G: gráfica de flujo (grafo) **E**: número de aristas

v(G): complejidad ciclomática **N**: número de nodos del grafo

v: número ciclomático en teoría de grafos.

P: número de nodos predicados[29] incluidos en el grafo.

Una vez calculada la complejidad ciclomática, el valor obtenido representa el límite superior de pruebas que deberán aplicarse (Pressman, 2005). A continuación, se realiza el procedimiento previamente descrito para el método "definitivo()" de la clase controladora antes mencionada. Dicho método se encarga de pasar a definitivo[30] un documento generado, el cual se clasifica en dependencia de la opción que se haya seleccionado desde la interfaz del usuario. Si existe el documento, este se pasa a definitivo; en caso contrario se busca otro tipo de documento existente. Una vez obtenido el tipo de documento se procede a cambiar su clasificación y se pasa a definitivo. En caso de no existir ningún documento para el trámite en cuestión, se inserta uno nuevo con la clasificación correspondiente, al que se hace referencia en los distintos casos de prueba y posteriormente se pasa a definitivo. En los anexos 10 y 11 respectivamente es posible observar el código del método seleccionado, del cual se obtuvo el siguiente grafo:

29 El nodo predicado representa las condiciones en el grafo de camino básico (*if-then, while, for y otros*).
30 Proceso mediante el cual se dicta como invariable una decisión tomada sobre un proceso judicial.

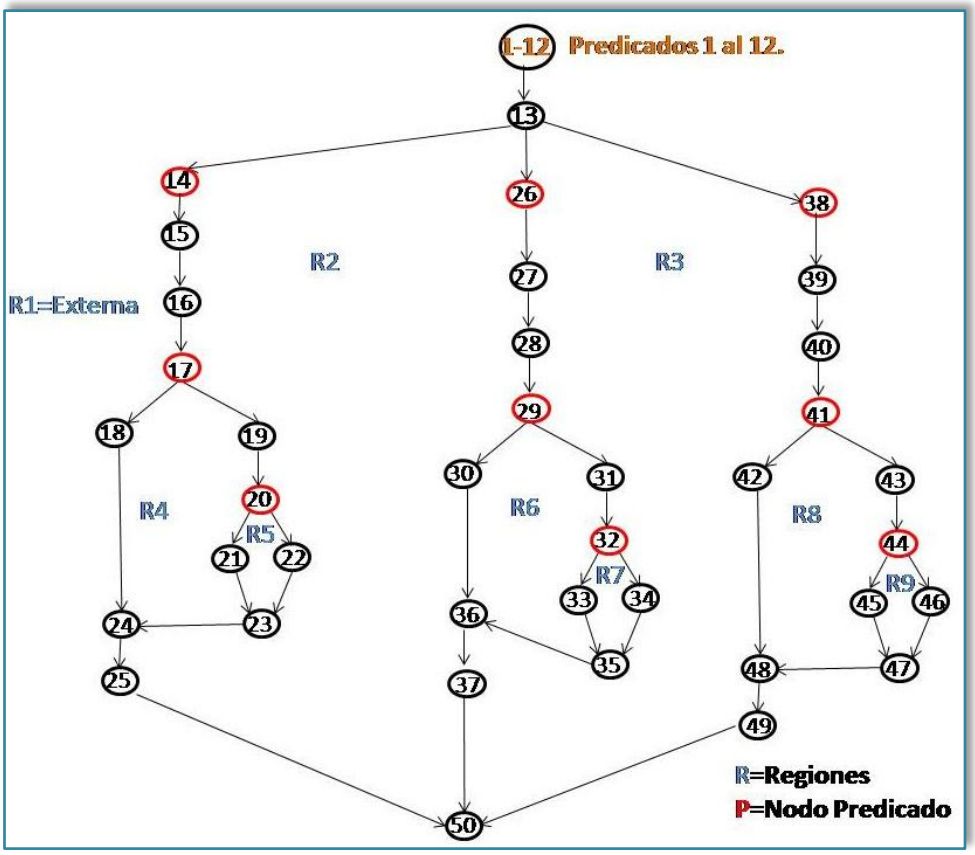


Figura 19. Grafo de flujo del método "definitivo()" de la clase "DisponerSobreEscritoOposicionController".

A dicho grafo de flujo se le aplicó la métrica de complejidad ciclomática, calculada por las tres vías conocidas, obteniéndose los siguientes resultados:

1. Número de regiones = 9.
2. v(G)= 46 – 39 + 2 = 9 .
3. Existen en el grafo 9 nodos predicados o condiciones, las cuales se descomponen en 3 "case" y 6 "if". McCabe plantea que para calcular la complejidad, se cuenta el número de sentencias "case" como 1 menos que el número de aristas que salen del nodo de decisión ("switch") (McCabe, et al., 1996). Cumpliéndose entonces que v(G)= 8 + 1 = 9 .

Después de calcular la complejidad del grafo, se pudo comprobar que los resultados obtenidos son iguales; por tanto se deben realizar 9 casos de pruebas, uno por cada camino independiente. Los caminos resultantes fueron:

a) 1-12-13-14-15-16-17-18-24-25-50.

b) 1-12-13-14-15-16-17-19-20-21-23-24-25-50.

c) 1-12-13-14-15-16-17-19-20-22-23-24-25-50.

d) 1-12-13-26-27-28-29-30-36-37-50.

e) 1-12-13-26-27-28-29-31-32-33-35-36-37-50.

f) 1-12-13-26-27-28-29-31-32-34-35-36-37-50.

g) 1-12-13-38-39-40-41-42-48-49-50.

h) 1-12-13-38-39-40-41-43-44-45-47-48-49-50.

i) 1-12-13-38-39-40-41-43-44-46-47-48-49-50.

Debido a que los predicados desde el 1 hasta el 12 siempre deben ejecutarse de manera continua antes de llegar a la sentencia “switch”, para evitar extender los caminos se pondrá solamente 1-12, dicha abreviatura representa el camino: 1-2-3-4-…-11-12.

La ejecución de este caso de uso debe generar tres documentos distintos, para asegurar la creación de dichos documentos se procederá a mostrar en las siguientes tablas los casos de prueba para tres caminos distintos (caminos 1, 5 y 9), con el fin de lograr este propósito:

Entrada	Se reciben los parámetros de entrada $html, $opcion e $idTramite. La variable $opcion de entrada debe ser igual a 1 y debe existir un documento generado.
Resultados Esperados	Se pasa a definitivo el documento generado del tipo “Providencia Concluso admitiendo pruebas”.
Condiciones	La variable $opcion recibida desde la entrada debe ser igual a 1 y la cantidad de documentos debe ser mayor que 0 [count($documento)>0] .

Tabla 11. Caso de prueba para el camino básico número 1.

Entrada	El parámetro de entrada $opcion debe ser igual a 2. A su vez, no debe existir ningún documento generado y sí existir un documento otro.
Resultados Esperados	Se cambia la clasificación y se pasa a definitivo el documento generado del tipo “Providencia Concluso solicitando pruebas”.
Condiciones	La variable $opcion recibida desde la entrada debe ser igual a 2. Además la cantidad de documentos debe ser menor o igual a 0 y la cantidad de documentos otros debe ser mayor que 0 [count($doc) > 0].

Tabla 12. Caso de prueba para el camino básico número 5.

Entrada	El parámetro de entrada $opcion debe ser igual a 3. A su vez, no deben existir ningún documento generado y ningún documento otro.
Resultados Esperados	Se inserta y se pasa a definitivo el documento generado del tipo “Providencia Subsanación de oposición de demanda”.
Condiciones	La variable $opcion recibida desde la entrada debe ser igual a 3. Además, la cantidad de documentos y de documentos otros deben ser menor o igual a 0.

Tabla 13. Caso de prueba para el camino básico número 9.

Resultado de aplicar el método de caja blanca

Para realizar el proceso de verificación a nivel de componentes, se ejecutó la técnica del camino básico sobre los métodos de las clases controladoras; dichas clases fueron seleccionadas debido a que engloban las funcionalidades del sistema. La siguiente figura muestra la cantidad de no conformidades encontradas en cada una de las iteraciones realizadas:

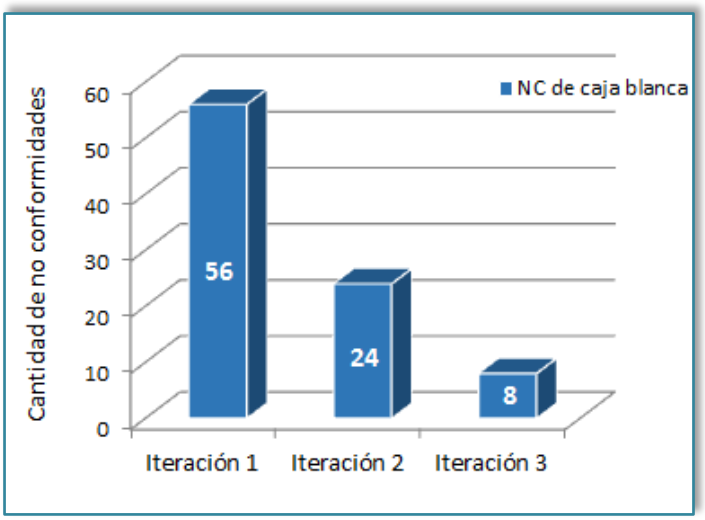


Figura 20. Cantidad de no conformidades encontradas en la prueba de caja blanca.

Para comprobar la fiabilidad del código fuente se realizaron tres iteraciones completas en busca de errores de codificación, como condición de parada se tuvo en cuenta el criterio de estimación anteriormente expuesto. Una vez que se diseñaron y ejecutaron los casos de prueba adecuados para recorrer todos los posibles caminos, se culminó con la tercera iteración, donde se encontraron 8 no conformidades; dicha cantidad representa menos del 30% de la sumatoria de las no conformidades encontradas en las iteraciones anteriores. Se puede concluir que luego de realizar la prueba de caja blanca, donde se diseñaron y ejecutaron los casos de prueba correspondientes,

además de corregir todas las no conformidades encontradas, se logró asegurar el cumplimiento del proceso de mejora del código.

3.3.2 Pruebas funcionales

Pruebas funcionales: las pruebas funcionales se centran en comprobar que el sistema funcione acorde a los requisitos funcionales y no funcionales, detectando posibles defectos derivados de errores en la fase de programación; dichas pruebas se llevan a cabo a través de la interfaz gráfica de la aplicación. Mediante estas pruebas se demuestra que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce una salida correcta, además de mantener íntegra la información externa del sistema (Pressman, 1997).

Para efectuar las pruebas funcionales se seleccionó el **método** de prueba caja negra o de comportamiento. Estas pruebas son aplicadas sin ningún tipo de suposición acerca de la estructura interna del sistema, permitiendo a los ingenieros del software obtener conjuntos de condiciones de entrada que ejerciten los requisitos funcionales de un programa, demostrando mediante los resultados el posible comportamiento (correcto o incorrecto) de la aplicación. Las pruebas de caja negra pretenden encontrar diversos tipos de errores, por ejemplo: funciones incorrectas o ausentes, errores en la interfaz, en estructuras de datos o en accesos a bases de datos externas, además de fallos en el rendimiento, entre otros (Pressman, 2005).

Técnica de prueba: partición equivalente

La técnica partición equivalente es aplicada en las pruebas de caja negra con el propósito de probar cada una de las funcionalidades del sistema, mediante la definición de casos de prueba que sean capaces de encontrar diferentes tipos de errores a nivel de interfaz. El concepto de partición equivalente ha sido definido por Pressman como *"…un conjunto de estados válidos y no válidos para las condiciones de entrada. Por lo general, una condición de entrada es un valor numérico específico, un rango de valores, un conjunto de valores relacionados o una condición booleana"* (Pressman, 1997 p. 437).

A continuación se muestra un caso de prueba aplicado para validar la funcionalidad del CU "Disponer sobre escrito de oposición", donde la variable "Disposición" no debe de ser nula:

Escenario	Descripción	Datos Primarios	Documentos necesarios	Disposición	Respuesta del sistema	Flujo central
EC 1.1 Disponer sobre escrito de oposición. Escenario correcto.	*Se seleccionan los datos para poder disponer sobre el escrito de oposición satisfactoriamente.*	*Ver DCP Visualizar datos primarios.*	*N/A*	*V*	*Se dispone sobre el escrito de oposición*	*1. Clic en Disponer sobre oposición. 2. Clic en Vista Previa o Pasar a definitivo.*
EC 1.2 Disponer sobre escrito de oposición. Escenario incompleto.	*Se dejan campos vacíos.*	*Ver DCP Visualizar datos primarios.*	*N/A*	*/* *Debe estar seleccionada*	*Muestra la sección señalado en rojo y un tool tipo text "Debe seleccionar este campo"*	*1. Clic en Disponer sobre oposición. 2. Clic en Vista Previa o Pasar a definitivo.*
EC 1.3 La operación es cancelada.	*La operación es cancelada.*	*Ver DCP Visualizar datos primarios.*	*N/A*	*V*	*Cierra la interfaz y no guarda los cambios.*	*1. Clic en Disponer sobre oposición. 2. Clic en Cancelar.*

Tabla 14. Diseño de caso de prueba del CU Disponer sobre escrito de oposición.

Matriz de trazabilidad de requisitos frente a casos de prueba

La matriz de trazabilidad es una técnica que tiene como objetivo verificar si todas las partes del producto desarrollado se pueden identificar o relacionar con cada uno de los requisitos definidos en la fase de análisis, es decir, si a partir de cualquier elemento del software se puede llegar hasta los requisitos o viceversa. La trazabilidad es la forma más común de representar enlaces entre requisitos y otros elementos del sistema, de manera que se pueda seguir una representación del diseño o un componente real del programa hasta los requisitos funcionales o no funcionales (Softqatest, 2010).

Una matriz de trazabilidad puede definir diferentes tipos de enlaces entre requisitos y otros elementos empleados en el desarrollo de la aplicación, algunos de los enlaces son:

- Requisito-Requisito.
- Requisito-Interfaces del sistema.
- Requisito-Casos de uso (Anexo 12).
- Requisito-Caso de Prueba.

Después de conocidas algunas de las relaciones sobre las que se puede realizar una matriz de trazabilidad, se debe seleccionar el tipo de matriz que se desea utilizar, así como las partes del producto para las cuales se desea mantener la información traceable (Slideshare, 2010). Para verificar el cubrimiento de cada requisito definido para los Flujos alternos del módulo Ejecutivo, se empleará la matriz de trazabilidad de requisitos funcionales frente a casos de prueba (relación de casos de prueba en anexo 13). Dicha matriz tiene como finalidad comprobar si los casos de prueba diseñados y/o ejecutados cubren cada una de las especificaciones o requerimientos con los que debe cumplir la aplicación. La siguiente tabla muestra la matriz resultante de analizar

los requisitos funcionales identificados para los Flujos alternos, así como los casos de pruebas diseñados para verificar la correcta implementación de los mismos.

Requisito/Caso de prueba	CP1	CP2	CP3	CP4	CP5	CP6	CP7	CP8	CP9	CP10	CP11	CP12	CP13	CP14	CP15	CP16
R1								X								
R2	X															
R3		X														
R4			X													
R5																X
R6				X												
R7									X							
R8						X										
R9							X									
R10										X						
R11														X		
R12															X	
R13					X											
R14											X					
R15												X				
R16													X			

Tabla 15. Matriz de trazabilidad de requisitos funcionales frente a casos de prueba.

De la elaboración de la matriz se obtuvo como resultado que (1) cada requisito funcional perteneciente a los Flujos alternos se encuentra asociado con al menos un caso de prueba. Además (2) existe una alta probabilidad de que algún caso de prueba se afecte como consecuencia de la modificación o eliminación de algunos de los requisitos definidos; de igual forma (3) se verifica en qué medida influye un diseño de un caso de prueba incorrecto en un determinado requisito. Por otra parte se puede decir que (4) cuando una prueba arriba a un resultado inesperado, los enlaces entre las pruebas, los requisitos y el código apuntan directamente hacia la parte o funcionalidad del sistema que se debe examinar para buscar los defectos; (5) disminuyendo a su vez el esfuerzo necesario para encontrar el error.

Resultado de aplicar el método de caja negra

Luego de validar que cada requisito funcional se asocia con al menos un caso de prueba, se realizaron varias iteraciones para verificar la correcta ejecución de las funcionalidades correspondientes a los Flujos alternos. Primeramente, se probó cada caso de uso como un componente independiente, y a partir de la segunda iteración se integraron aquellos casos de uso que debían ejecutarse de manera continua, es decir, siguiendo un orden lógico de sucesiones entre los procedimientos.

Las pruebas de integración fueron realizadas para verificar la correcta coherencia entre los diferentes módulos o casos de uso, asegurando que un módulo recibe de otro lo que esperaba. Dichas pruebas se realizaron por etapas, englobando progresivamente más y más casos de uso en cada prueba. A continuación se

muestran los resultados obtenidos de probar las funcionalidades de los Flujos alternos, así como los resultados de las pruebas de integración:

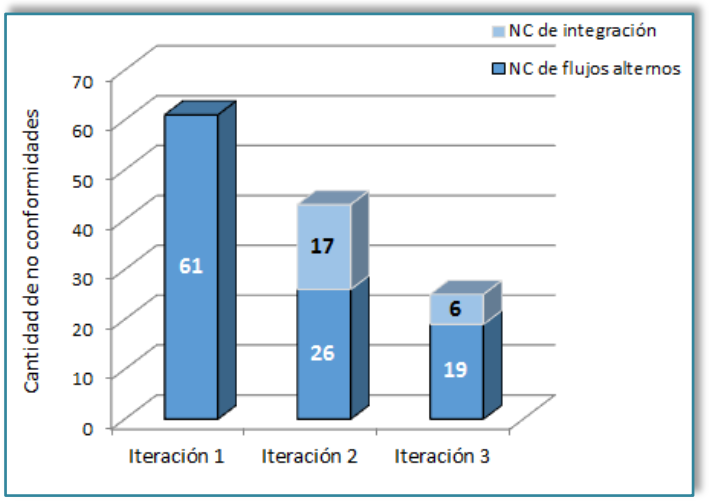


Figura 21. Cantidad de no conformidades encontradas por iteración, en la fase de prueba interna de los Flujos alternos.

Como se puede observar, la gráfica muestra los resultados obtenidos en cada una de las iteraciones realizadas; donde la última iteración culmina con 25 no conformidades, las cuales fueron resueltas inmediatamente. El resultado de dicha iteración demuestra que el valor encontrado es menor que el 30% de la sumatoria de las no conformidades encontradas en las dos iteraciones anteriores, cumpliéndose el criterio de parada propuesto.

Posteriormente como apoyo al proceso de verificación del sistema, el Departamento de Calidad del centro CEGEL realizó las pruebas necesarias para comprobar el correcto funcionamiento de las operaciones implementadas, según la especificación de requisitos, obteniéndose los siguientes resultados:

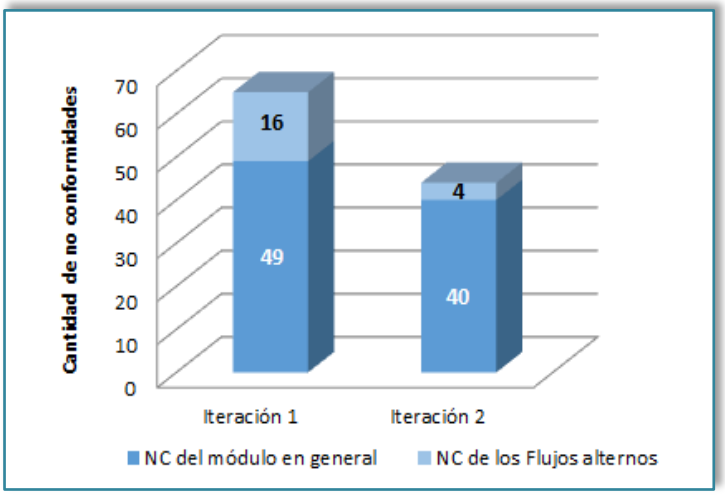


Figura 22. Cantidad de no conformidades encontradas por iteración, en la fase de prueba externa de los Flujos alternos.

Luego de realizadas dos iteraciones, se encontraron en cada una de ellas un determinado número de NC, las mismas representan los Flujos alternos y su integración con el Flujo básico (Ing. Maza Oval, et al., 2014). Aunque el proceso en el Departamento de Calidad no se concluyó debido a que el módulo debía iniciar la fase de prueba a mayor escala en el Centro Nacional de Calidad de Software (Calisoft)[31], se puede observar que la cantidad de errores encontrados en el sistema, correspondientes a las funcionalidades de los Flujos alternos, no ofrecen valores relevantes con respecto a los resultados obtenidos de todo el módulo.

Después de haber sido resueltas todas las NC encontradas en la última revisión, se puede concluir que las funcionalidades implementadas para los procedimientos alternos del proceso Ejecutivo cumplen con los requisitos especificados en la fase de análisis. Además, la cantidad de NC encontradas mediante el método de caja negra, tanto a nivel interno como externo, demuestra un elevado nivel de efectividad en cuanto al diseño y ejecución de los casos de prueba, y la implementación de las funcionalidades a las que estos se asocian

Conclusiones parciales del capítulo

De manera general, el capítulo estuvo referido al desarrollo de las métricas necesarias para validar el diseño de la aplicación, así como las pruebas empleadas para verificar el correcto funcionamiento del sistema, obteniéndose los siguientes resultados:

- Las métricas aplicadas para la validación del diseño demostraron que el sistema es funcionalmente correcto, puesto que los resultados obtenidos por cada una de las métricas fueron exitosos.
- Después de analizar el resultado obtenido de aplicar las pruebas de caja blanca, se pudo concluir que el código fuente de la aplicación ejecuta todas las líneas de código al menos una vez, garantizando que no existe ninguna línea innecesaria.
- Luego de aplicar las pruebas de caja negra y resolver las no conformidades se obtuvo como resultado que las funcionalidades del sistema para los Flujos alternos fueron implementadas correctamente.

31 Centro responsable de la evaluación de la conformidad a partir de normas nacionales e internacionales.

Conclusiones generales

Mediante el presente trabajo de diploma se ha documentado cómo se realizó la implementación de los Flujos alternos del módulo Ejecutivo del subsistema Económico. Luego de finalizar la investigación de manera exitosa, se arribaron a las siguientes conclusiones:

- La metodología utilizada como herramienta de planificación y control del proyecto condujo durante todo el proceso las tareas y actividades necesarias para dar cumplimiento al desarrollo de los Flujos alternos.
- El uso de patrones de diseño permitió transformar los requisitos funcionales a un lenguaje artefactual obteniéndose un modelo de diseño con calidad.
- En la etapa de implementación se codificaron los requisitos establecidos para los Flujos alternos, obteniéndose un sistema funcional e íntegro del módulo Ejecutivo.
- Mediante el uso de las métricas de diseño seleccionadas se pudo validar que la aplicación posee un elevado nivel de reutilización, propicio para la realización sencilla del mantenimiento.
- Las pruebas realizadas al sistema fueron satisfactorias, de manera que se comprobó que las funcionalidades implementadas cumplen con los requisitos definidos.

Glosario de términos

- **Acreedor**: el que tiene derecho a exigir alguna cosa o servicio de acuerdo con su fuente o causa legítima. En la denominación de acreedor, son comprendidas no solamente los que prestan dinero sino todas aquellas personas a quienes por cualquier causa se les debe.
- **Auto**: resolución judicial que se emite en forma fundada, destinada a decidir incidentes o puntos esenciales que afecten de manera directa la personería o la competencia y en general, a las resoluciones que de acuerdo con su naturaleza deban dictarse en forma razonada.
- **Deuda líquida, vencida y exigible:** el acreedor sólo puede requerir al deudor cuando la obligación sea exigible (esto ocurre cuando la deuda esté vencida) y líquida (su cuantía debe ser determinada con una simple operación aritmética y que sean reclamables en juicio).
- **Document Object Model (DOM):** es una interfaz de programación de aplicaciones para documentos válidos y bien construidos HTML y XML. Define la estructura lógica de los documentos y el modo en que se acceden y manipulan los mismos.
- **Extracontractual:** la distinción entre responsabilidad contractual y extracontractual parte de la existencia o no de un vínculo previo entre las partes. Así, puede definirse la responsabilidad extracontractual como la que nace de un daño producido a otra persona sin que exista una relación jurídica convenida entre el autor del daño y el perjudicado.
- **Indicador:** mecanismo empleado para localizar con precisión las deficiencias de un software o las características medibles del mismo, además puede ser una o la unión de varias métricas.
- **Lenguaje interpretado:** lenguaje de programación diseñado para ser ejecutado por medio de un intérprete. Se debe tener en cuenta que el modo de ejecución del programa escrito en el lenguaje es independiente del propio lenguaje. A ciertos lenguajes interpretados también se les conoce como lenguajes de *script*.
- **Litigio:** es un conflicto de intereses calificado y elevado a una autoridad jurisdiccional, por un sujeto de derecho con una intención o pretensión contra otro, que manifiesta una resistencia o que se opone al planteamiento del primero.
- **Mapeo:** técnica de programación utilizada para convertir datos que intervienen entre un sistema que utiliza un lenguaje de programación orientado a objetos y una base de datos relacional, utilizando un motor de persistencia.

- **Metadatos**: significa literalmente «sobre datos», son datos que describen otros datos. En general, un grupo de metadatos se refiere a un grupo de datos, llamado *recurso* Por ejemplo, en una biblioteca se usan fichas (metadatos) que especifican autores, títulos, casas editoriales y lugares para buscar libros (datos).
- **Motor de plantillas:** los motores de plantillas permiten separar el código PHP del código HTML. Las plantillas tienen como objetivo combinar la labor del diseñador gráfico y el de programador en una sola persona y dentro del mismo código.
- **Remate:** es una actuación judicial, dentro del proceso Ejecutivo, donde se formaliza la venta, al mejor postor, de los bienes embargados en cumplimiento de la deuda.
- **Personería**: en Derecho, personalidad que puede adquirir una organización para contratar.
- **Subsanar**: Reparar, remediar un defecto o resarcir un daño. En la materia jurídica la subsanación es la corrección de errores en documentos oficiales donde no se cumple con los parámetros o reglas legales.

Bibliografía

Alexander, Christopher, Ishikawa, Sara y Silverstein, Murray. 1977. *A Pattern Language.* s.l. : Oxford University Press, 1977. ISBN: 0-19-501919-9.

Apache, Software Foundation. 2012. What is Apache. [En línea] 17 de Enero de 2012. [Citado el: 2 de Diciembre de 2013.] http://wiki.apache.org/httpd/FAQ#What_is_Apache.3F.

Bertino, E. A y Martino, L. A. 1995. *Sistemas de bases de datos orientadas a objetos.* s.l. : Ediciones Díaz de Santos, 1995.

Booch, Grady, Jacobson, Ivar y Rumbaugh, James. 2000. *El Proceso Unificado de Desarrollo de Software.* Madrid : Addison Wesley, 2000. ISBN: 84-7829036-2.

Collins-Sussman, Ben, W. Fitzpatrick, Brian y Pilato, C. Michael. 2008. Version Control with Subversion. [En línea] 1 de Febrero de 2008. [Citado el: 10 de Diciembre de 2013.] http://svnbook.red-bean.com/en/1.5/.

CSS, Manual de. 2012. Introducción a CSS – Manual de CSS. [En línea] 2012. [Citado el: 16 de Noviembre de 2013.] http://www.manualdecss.com/manualdecss/introduccion-a-css/.

Doctrine Team. 2006. 4. Basic Mapping. Sitio oficial de Doctrine. [En línea] Doctrine Proyect, 2006. [Citado el: 20 de 2 de 2014.] http://docs.doctrine-project.org/en/2.0.x/reference/basic-mapping.html.

Doval, Luis y Gay, Aquiles. 1995. *Tecnología: finalidad educativa y acercamiento didáctico.* Buenos Aires : s.n., 1995.

Eguiluz, Javier. 2011. *Desarrollo web ágil con Symfony2. 1ra edición.* 2011.

—. 2007. Introducción a JavaScript. *LibrosWeb.* [En línea] Agosto de 2007. [Citado el: 4 de Noviembre de 2013.] http://librosweb.es/javascript/capitulo_1.html.

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

Hernández Orallo, Enrique. 2013. *El Lenguaje Unificado de Modelado (UML).* Valencia,España : s.n., 2013.

Hernández, León, Rolando, Alfredo y Coello González, Sayda. 2011. *El proceso de investigacion científica.* La Habana : Universitaria, 2011.

IEEE Computer Society. 1993. IEEE Standard Glossary of Software Enginneering Terminology. [En línea] 1993. [Citado el: 20 de 03 de 2014.] https://standards.ieee.org/findstds/standard/610.12-1990. 610.12-1990.

Ing. Almora Galvez, Yeleny y Ing. Mulen Llorente, Yoelvis. 2013. *Estándares de Codificación.Sistema de informatización de los Tribunales Populares Cubanos.* La Habana : Universidad de las Ciencias Informáticas, 2013.

Ing. Arencibia Cruz, Humberto. 2012. *Modelo de Procesos de Negocio con BPM.Subsistema Económico.Proceso Ejecutivo.* La Habana : Universidad de las Ciencias Informáticas, 2012.

Ing. Lamorú Marciel, Daimi. 2013. *Especificación de requisitos de software. Subsistema Ecónomico. Módulo Ejecutivo.* La Habana : Sistema de Informatización de los Tribunales Populares Cubanos, 2013.

Ing. Maza Oval, Doris y Ing. Pupo Leyva, Iliannis. 2014. *Documento de No Conformidades v1.0. Subsistema Económico. Módulo Ejecutivo.* Habana : Sistema de Informatización de los Tribunales Populares Cubanos, 2014.

Ing. Roque Hernández, Yoslenys. 2013. *Especificación de Requisitos no Funcionales de Software. SITPC.* La Habana : Universidad de las Ciencias Informáticas, 2013.

Laboratorio de Gestión de Proyectos. 2010. CEGEL, Centro de Gobierno electrónico. [En línea] UCI, 2010. [Citado el: 13 de Diciembre de 2013.] http://gespro.cegel.prod.uci.cu.

Larman, Craig. 1999. *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* México : Prentice Hall, 1999. ISBN: 970-17-0261-1.

Lic. Valerio, Adrián Anacleto. 2010. *Deploying ideas.* s.l. : Epidata Consulting, 2010.

Maldonado, Daniel. 2007. Qué son los IDE de programación. *El CoDiGo K.* [En línea] 3 de Septiembre de 2007. [Citado el: 24 de Septiembre de 2013.] http://www.elcodigok.com.ar/2007/09/que-son-los-ide-de-programacin.html.

McCabe, Thomas J. y Watson, Arthur H. 1996. *Structured Testing: A Testing Methodology.* Gaithersburg, Maryland : Dolores R. Wallace, 1996. NIST-43NANB517266.

McCall, Jim A, Richards, Paul K y Walters, Gene. 1977. *Factors in Software Quality. Preliminary Handbook on Software Quality for an Acquisiton Manager.* s.l. : GENERAL ELECTRIC CO SUNNYVALE CA, 1977. Vols. Volume-III. ADA049055.

Ministerio de Educación de España. 2008. Sistema de control de versiones: SubVersion. *Observatorio Tecnológico.* [En línea] 17 de Enero de 2008. [Citado el: 2 de Diciembre de 2013.] http://recursostic.educacion.es/observatorio/web/es/software/software-general/548-luis-garcia. NIPO: 820-10-289-9.

Murphey, Rebecca y D'Onofrio, Leandro. 2013. Fundamentos de Jquery. [En línea] Agosto de 2013. [Citado el: 1 de Marzo de 2014.] http://librojquery.com/.

Myers, Glenford J., Sandler, Corey y Badgett, Tom. 2012. *The art of software testing.* New Jersey : John Wiley & Sons, Inc., 2012. ISBN 978-1-118-03196-4.

netbeans.org. 2000. Sitio oficial de NetBeans. [En línea] Junio de 2000. [Citado el: 14 de Enero de 2014.] https://netbeans.org/about/index.html.

Pacheco, Nacho. 2011. Manual de Twig Release 1.2.0. 2011.

PostgreSQL, Equipo desarrollo. 2010. *Tutorial de PostgreSQL.* s.l. : Thomas Lockhart, 2010.

Potencier, Fabien. 2011. FabienPotencier.org. *What is Symfony2?* [En línea] 25 de Octubre de 2011. [Citado el: 24 de Noviembre de 2013.] http://fabien.potencier.org/article/49/what-is-symfony2.

Potencier, Fabien y Zaninotto, François. 2007. Symfony 1.2, la guía definitiva. *LibrosWeb.* [En línea] Agosto de 2007. [Citado el: 27 de Febrero de 2014.] http://librosweb.es/symfony/capitulo_1/conceptos_basicos.html.

Pressman, Roger S. 1997. *Ingenieria de Software. Un enfoque práctico.5ta Edición.* Madrid : McGraw- Hill, 1997. ISBN: 9708121410.

Pressman, Roger S. 2005. *La ingenieria del software, un enfoque práctico. 6ta Edición.* Madrid : McGraw- Hill, 2005. ISBN: 9701054733.

Reynoso, Carlos y Kiccillof, Nicolás. 2004. *Estilos y patrones en la estrategia de arquitectura de Microsoft.* s.l. : Universidad de Buenos Aires, 2004.

Rodríguez, Juan David. 2013. Tutorial: inyección de dependencias en Symfony2. [En línea] Blog independiente, 2013. [Citado el: 12 de 2 de 2014.] http://juandarodriguez.es/tutoriales/inyeccion-de-dependencias-en-symfony2/.

RUP. **Mendoza Sanchez, Ing. María A. 2004.** Perú S.A.C : s.n., 2004.

Sánchez, Álvaro Fontela. 2013. OpenWebCMS. [En línea] 21 de Mayo de 2013. [Citado el: 13 de Febrero de 2014.] http://openwebcms.es/2013/que-es-bootstrap/.

ScottGu. 2008. Jquery and Microsoft. [En línea] 28 de Diciembre de 2008. [Citado el: 12 de Noviembre de 2013.] http://weblogs.asp.net/scottgu/archive/2008/09/28/jquery-and-microsoft.aspx..

Slideshare. 2010. Slideshare.net. [En línea] 27 de Octubre de 2010. [Citado el: 15 de Mayo de 2014.] http://www.slideshare.net/iscrquinter/03-administracion-de-requisitos?qid=7144a6c4-f561-4a11-bb06-6917eba1f456&v=default&b=&from_search=1.

Softqatest. 2010. Softqatest.es. [En línea] 15 de Junio de 2010. [Citado el: 15 de Mayo de 2014.] http://www.softqatest.net/?p=77.

Sommerville, Ian. 2005. *Ingeniería del Software. 7ma Edición.* Madrid : Pearson Educación, S.A., 2005. ISBN: 84-7829-074-5.

Stallings, William. 2000. *Comunicaciones y Redes de Computadores. 6ta Edición.* s.l. : Prentice Hall, 2000.

Stephen, H. Kan. 2002. *Metrics and Models in Software Quality Engineering, Second Edition.* s.l. : Addison Wesley, 2002. ISBN: 0-201-72915-6 .

symfony.es. 2009. Inyección de dependencias en Symfony 2.1.0. [En línea] 2 de Abril de 2009. [Citado el: 3 de Noviembre de 2013.] http://www.symfony.es/noticias/2009/04/02/inyeccion-de-dependencias-en-symfony-2.

The PHP Group. 2001-2014. PHP: Historia de PHP y Proyectos Relacionados – Manual. [En línea] 2001-2014. [Citado el: 5 de Noviembre de 2013.] http://es.php.net/manual/es/history.php.php.

Tribunal Supremo Popular. 2013. Sitio Oficial del Tribunal Supremo Popular de la República de Cuba. [En línea] 2013. [Citado el: 24 de Noviembre de 2013.] http://www.tsp.cu/tribunal_supremo_popular_cuba.

UCI. 2012. Portal de la Universidad de las Ciencias Informáticas. *Misión.* [En línea] UCI, 2012. [Citado el: 13 de Diciembre de 2013.] http://www.uci.cu/?q=mision.

W3C Nightly. 2013. Introduction — HTML 5.1. HTML 5.1 Nightly A vocabulary and associated APIs for HTML and XHTML. [En línea] 2013. [Citado el: 5 de Noviembre de 2013.] http://www.w3.org/html/wg/drafts/html/master/introduction.html#html-vs-xhtml.

W3C. 2013. Sobre el W3C - W3C España. [En línea] 2013. [Citado el: 17 de Octubre de 2013.] http://www.w3c.es/Consorcio/.

Anexos

Anexo 1

Objetivo	El CU permite admitir o disponer subsanación del escrito de oposición.
Actores	Juez Ponente (inicia)
Resumen	El CU se inicia cuando el juez accede al sistema para disponer sobre el escrito de oposición. Para ello selecciona el expediente y selecciona una de las opciones disponibles.
Complejidad	Media
Prioridad	Media
Precondiciones	El expediente ha pasado al estado “pendiente a disponer sobre modificación de pretensiones”.
Postcondiciones	Si se admitió el escrito sin solicitar pruebas, el estado del expediente cambió a “concluso”. Si se admitió el escrito solicitando pruebas, el estado del expediente cambió a “abriendo pruebas”. Si se dispuso subsanación del escrito, el expediente cambió a “pendiente de subsanación”.
Flujo normal de eventos	
Flujo básico Admitir escrito oposición	

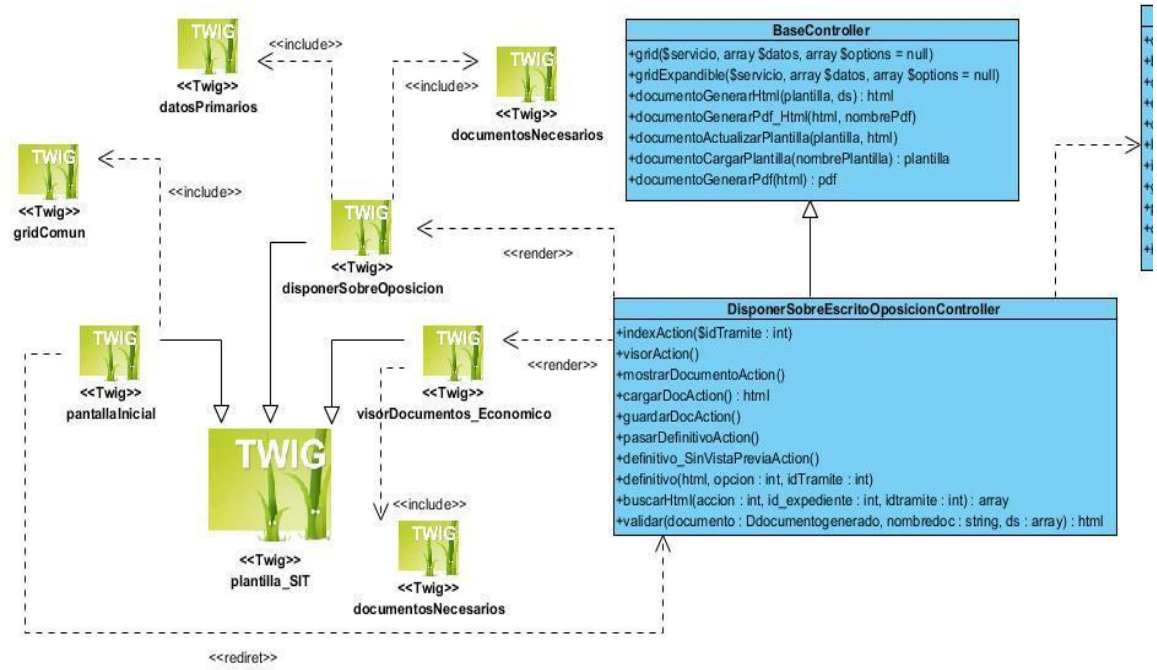
	Actor	**Sistema**
1.	Accede al expediente en el trámite Disponer sobre oposición.	
2.		Invoca el CU Visualizar datos primarios. Muestra los documentos necesarios. Muestra las disposiciones: • Admitir • Admitir solicitando pruebas *(ver Sección 1)* • Subsanar *(ver Sección 2)* Brinda las opciones: • Vista Previa • Pasar a definitivo • Cancelar
3.	Marca la opción Admitir.	
4.	Selecciona la opción Pasar a definitivo.	
5.		Muestra un mensaje de confirmación con las opciones: • Si • No
6.	Selecciona la opción Sí.	
7.		Verifica que los datos

		introducidos son correctos y guarda. El documento queda en estado definitivo. (*Termina el caso de uso)*
Sección 1 Disponer Admitir Solicitando Pruebas		
	Actor	**Sistema**
	Selecciona la opción Solicitar Prueba.	
		Invoca el CU Solicitar Pruebas. *(continúa en el paso 4 del flujo normal de eventos)*
Sección 2 Disponer Subsanación		
	Actor	**Sistema**
1.	Selecciona la opción Subsanar.	
2.		Solicita los motivos de la subsanación. *(continúa en el paso 4 del flujo normal de eventos)*
Flujos alternos		
4c Cancelar		
	Actor	**Sistema**
1.	Selecciona la opción Cancelar.	
2.		Muestra un mensaje de confirmación con las opciones: • Si • No
3.	Selecciona la opción Sí.	
4.		No guarda los datos y regresa a la interfaz anterior.
4a Vista Previa		
	Actor	**Sistema**
1.	Selecciona la opción Vista Previa.	
2.		Muestra la resolución y las opciones: • Pasar a Definitivo *(ver a partir del paso 4 del Flujo normal de eventos)* • Guardar • Cancelar
3.	Selecciona la opción Guardar.	
4.		Verifica que los datos introducidos son correctos y guarda. El documento queda en estado pendiente. (*Termina el caso de uso)*
Relaciones	**CU incluidos**	Visualizar datos primarios Documentos necesarios Solicitar Pruebas.

	CU extendidos	

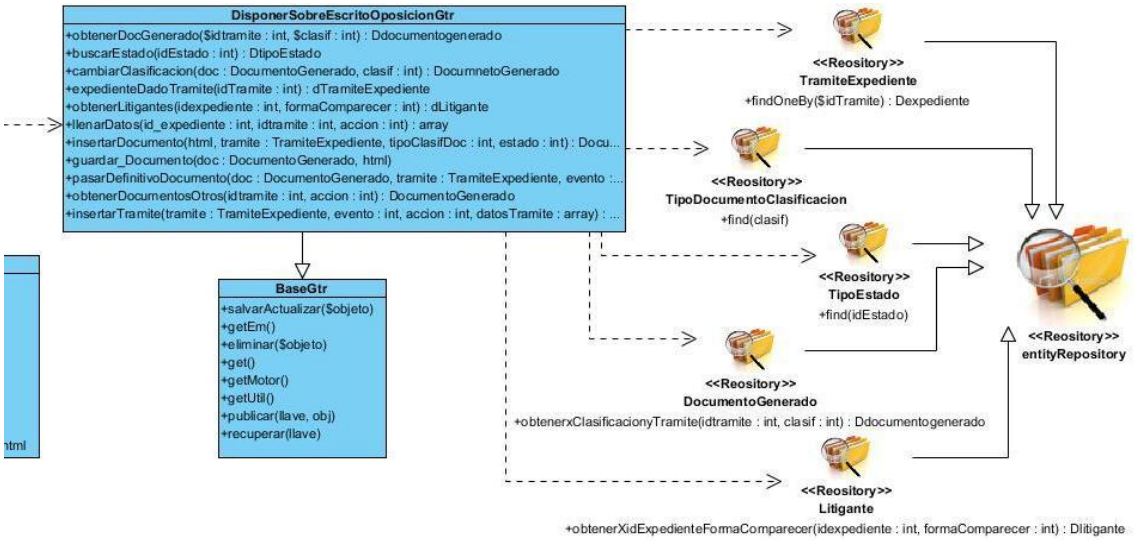
Descripción extendida del CU Disponer sobre escrito de oposición.

Anexo 2



Parte 1 del diagrama de clases del CU Disponer sobre escrito de oposición.

Anexo 3



Parte 2 del diagrama de clases del CU Disponer sobre escrito de oposición.